University of Trento

Department of Information Engineering and Computer Science

Ph.D. Degree Thesis

~ · ~

Academic Year 2020–2024

Reflexive Composition: Bidirectional Enhancement of Language Models and Knowledge Graphs

Supervisor
Prof. Fausto Giunchiglia

Graduate Student
Virendra Mehta
v0.91 Apr 16, 2025

Final examination date: June 10, 2025

Acknowledgments

First and foremost, I extend my deepest gratitude to my advisor, Prof. Fausto Giunchiglia, whose unwavering guidance and intellectual rigor have been instrumental in shaping this work. His commitment to interdisciplinary research and his vision for the future of knowledge engineering provided the foundation for this thesis. I am particularly grateful for his patience during the numerous iterations and his insistence on clarity and precision, which ultimately strengthened every aspect of this research.

I would like to thank Prof. Gabor and Prof. Tacchella for their thorough and constructive feedback during the review process. Their critical insights and detailed comments significantly improved the technical depth and scientific rigor of this thesis. While challenging, their feedback pushed me to address fundamental questions about novelty, experimental design, and technical implementation that made this work substantially stronger.

Special thanks to Prof Casati who motivated to take this journey and stood by me all along the way! My appreciation extends to the faculty and staff of the Department of Information Engineering and Computer Science at the University of Trento. The collaborative environment and access to computational resources were essential for conducting the experimental evaluations across multiple case studies. I am grateful to the graduate students and researchers who provided feedback during internal seminars and informal discussions.

I acknowledge the support of the research community that provided access to datasets used in this work, particularly the MIMIC-III and MIMIC-IV teams for the medical case study, and the various open-source security advisory databases that enabled the code security evaluation. The availability of these resources was crucial for demonstrating the generalizability of the Reflexive Composition framework.

Special thanks to my colleagues and fellow PhD students who served as validators during the human-in-the-loop evaluations. Their domain expertise and willingness to participate in the validation workflows provided essential data for understanding the scalability and effectiveness of the proposed approach.

I am deeply grateful to my family for their unwavering support throughout this journey. To my parents, who instilled in me the value of education and perseverance, and whose encouragement sustained me through the most challenging periods of this research. To my children, who provided perspective and reminded me of life beyond academia when I needed it most.

Finally, I want to acknowledge the broader research community working at the intersection of large language models and knowledge graphs. This thesis builds upon the foundational work of many researchers who have advanced our understanding of neural-symbolic integration. Their contributions made this work possible, and I hope this research contributes meaningfully to the ongoing discourse in this rapidly evolving field.

This research was supported in part by computational resources provided by the University of Trento and various cloud computing platforms used for large language model experiments. All code and experimental configurations have been made available to support reproducibility and future research in this area.

Abstract

Large Language Models (LLMs) have significantly advanced natural language processing, yet they continue to face limitations such as hallucinations, factual inconsistencies, and restricted domain-specific knowledge. Knowledge Graphs (KGs), by contrast, provide structured and verifiable information but are expensive to build and maintain manually. This thesis introduces Reflexive Composition, a bidirectional integration framework in which LLMs and KGs iteratively refine each other's outputs.

The framework consists of three interconnected components: (1) LLM2KG, where LLMs assist in the construction and updating of domain-specific knowledge graphs; (2) Human-in-the-Loop (HITL) validation, which supports structured expert review; and (3) KG2LLM, which conditions LLM outputs on verified knowledge to reduce hallucinations and improve consistency.

The methodology is evaluated across three case studies: temporal knowledge management, privacy-preserving data integration, and historical bias mitigation. Results include a 23% increase in knowledge extraction accuracy (F1 score from 0.65 to 0.80), a 28.7% reduction in LLM hallucination rates, and measurable improvements in validation efficiency through structured workflows. Reflexive Composition offers a reproducible approach for improving the reliability, scalability, and transparency of AI systems in dynamic or high-risk domains.

Keywords

Large Language Models, Knowledge Graphs, LLM Hallucination Reduction, Knowledge Graph Evolution, Bias Mitigation

Contents

List of Figures

List of Tables

Listings

Glossary

BLEU Bilingual Evaluation Understudy, a precision-based metric for evaluating machine translation and text generation. 20, 26

Calibration Accuracy A measure of alignment between model confidence and actual output correctness or validator certainty. 20

Factual Consistency Score (FCS) A metric that evaluates how factually consistent a generated output is with its reference input. 20

Groundedness Score A measure of how much generated text is traceable to or directly supported by structured or retrieved knowledge. 20

Human-in-the-Loop Strategic validation framework centered on data scientists. 20, 25–27

Integration Validation System Framework ensuring quality and consistency of knowledge integration through automated and human validation. 28, 33

KG2LLM Knowledge graph-guided improvement of LLM responses. 27

Knowledge Builder LLM (KB-LLM) LLM responsible for extracting and structuring knowledge into a KG, guided by a predefined ontology. vii, 56

Knowledge Extraction Pipeline A systematic process for extracting structured knowledge from unstructured sources using Large Language Models. 28, 32

LLM2KG Knowledge graph construction and enhancement using LLMs. 27

Mean Reciprocal Rank (MRR) A metric used in information retrieval that evaluates how soon the first correct answer appears in the ranked results. 19

Natural Language Inference (NLI) A task where a model determines whether a hypothesis logically follows from a premise. 20

Precision@k A metric measuring how many of the top-k retrieved items are relevant. 19

RAG Retrieval-Augmented Generation, a method that retrieves documents during inference to augment language model responses. 19

Recall@k A metric measuring how many relevant items are retrieved within the top-k. 19

Schema Evolution Framework System for managing knowledge graph schema updates and evolution through automated inference and validation. 28, 33

Chapter 1

Introduction

Large language models (LLMs) have become central tools in artificial intelligence, achieving high accuracy in tasks such as language modeling, reasoning, and summarization [104]. However, their reliance on probabilistic generation often results in hallucinations — plausible but incorrect outputs [82] — which limits their reliability in high-stakes domains. Knowledge graphs (KGs), by contrast, provide structured and verifiable information [106], but are resource-intensive to construct and maintain at scale [1]. While traditionally used independently, these two technologies exhibit complementary strengths and weaknesses [78]. This thesis investigates how a reflexive integration of LLMs and KGs—combined with human-in-the-loop validation—can improve factual consistency, scalability, and adaptability in knowledge-intensive AI systems.

1.1 Background

The increasing integration of AI into domains like healthcare, law, and cybersecurity has intensified the demand for systems that are not only reliable and traceable, but also capable of remaining current with evolving information. Meeting this need requires a hybrid approach to knowledge—one that supports both unstructured natural language processing and structured, verifiable information management.

Two dominant paradigms have emerged:

- **LLMs**, which learn to generate fluent responses from vast text corpora, and
- **KGs**, which capture domain-specific knowledge in a structure, symbolic format.

LLMs offer scalability and linguistic flexibility, but are prone to hallucinations and temporal drift. KGs provide precision and structural control, but typically require extensive manual curation. Used in isolation, neither approach adequately supports the design of reliable, adaptable AI systems.

This thesis is motivated by the possibility of turning this tension into a synergy: integrating LLMs and KGs into a **reflexive composition** pipeline where each improves the other - automatically and iteratively - with human oversight acting as a strategic validation layer.

1.1.1 Knowledge Graph Engineering

Knowledge graphs formalize relationships between entities, supporting logic-based reasoning and structured querying [106]. Within our research group, the **iTelos** [34] methodology has become a cornerstone for purpose-driven KG construction, especially in domains like education and public health. iTelos offers a systematic, top-down approach to schema definition, entity identification, and triple generation - all grounded in clearly defined use-case requirements.

However, like most manual KG engineering pipelines, iTelos is constrained by the time and expertise required to scale. Each new domain requires significant human involvement in schema evolution, knowledge extraction, and quality assurance. As knowledge domains expand and become more dynamic, purely manual methods - no matter how well designed - struggle to keep up with the pace of change.

1.1.2 Large Language Model Capabilities

Large language models (LLMs) such as GPT, PaLM [16], and LLaMA have significantly expanded the scope of machine language processing. They are capable of synthesis, summarization, and even basic reasoning over text [7, 13]. However, their architecture remains fundamentally probabilistic: trained on static data snapshots, they lack mechanisms for incorporating new information or maintaining consistency over time. As a result, their outputs can be fluent and contextually relevant—but occasionally inaccurate or misleading [51, 82].

1.1.3 Data Scientist Role in Knowledge Engineering

Historically, data scientists have manually curated and validated structured knowledge. But, with the scale and complexity of current systems, this role must evolve. Rather than handcrafting graphs or checking every triple, experts must **guide and validate** the system at key decision points: when a schema changes, when a relation appears novel, or when a hallucinated fact risks becoming knowledge.

This shift, from builder to validator, is at the core of the reflexive composition methodology presented in this thesis.

1.2 Problem Statement

Although LLMs and KGs possess complementary capabilities, their integration into scalable and reliable AI systems remains an open research question. This thesis addresses three specific limitations that impede practical deployment of such hybrid systems in rapidly-evolving, mission-critical domains.

1.2.1 Scalability of Knowledge Graph Engineering

Despite the formal rigor and expressive power of knowledge graphs, building and maintaining them is resource-intensive [35]. Approaches like iTelos provide structured workflows, but necessitate substantial human input for schema design, entity modeling, and update propagation. As graph size increases, so do the computational complexity of validation, the requirements for domain-specific reasoning, and the probability of concept drift [74].

Moreover, large-scale knowledge graphs often become brittle over time: difficult to update, extend, or modularize without significant manual reengineering. These scalability challenges limit the adaptability of symbolic systems in dynamic environments where knowledge evolves rapidly, such as clinical domains, security intelligence, or scientific discovery.

1.2.2 Human Oversight at Scale

As systems scale, the role of human experts must shift from hands-on engineering to targeted validation. Yet existing frameworks offer little guidance on *where* to insert human judgment for optimal efficiency. Without structured checkpoints, validation becomes ad hoc or overly burdensome. Worse, it becomes unclear what is automated, what is curated, and how quality assurance adapts over time.

1.2.3 LLM Hallucinations

Large Language Models (LLMs), despite their fluency and ability to generalize across tasks, exhibit known limitations [25, 23] in several areas that affect reliability:

- They hallucinate, producing fluent but incorrect information due to inherited biases or dataset flaws [11].
- They suffer from knowledge staleness, being unable to incorporate events post-training without external grounding [46, 59].
- They lack access to private or domain-restricted knowledge that was never included in their training corpus [64].

These challenges impact domains where:

- **Legacy bias** affects security-critical codebases,
- **Temporal volatility** affects current affairs and dynamic fields,
- **Privacy restrictions** limit access to clinical and personalized systems.

Without structured and dynamic grounding, LLMs cannot meet reliability standards required for decision-critical tasks.

1.3 Proposed Solution: Reflexive Composition

To address the challenges of hallucination in language models, the high cost of manual knowledge engineering, and the limitations of expert scalability, this thesis introduces **Reflexive Composition**: an architecture in which large language models (LLMs) and knowledge graphs (KGs) iteratively refine one another, with human oversight applied at the most uncertain or impactful validation points.

The architecture is built around three core components:

- **LLM2KG**: Using language models to extract and structure knowledge for KG construction and evolution.

- **Human-in-the-Loop (HITL)**: Involving human experts at key checkpoints to validate schema changes, new knowledge, and system behavior.
- **KG2LLM**: Leveraging structured knowledge graphs to ground and inform LLM output, improving factual accuracy and adaptability.

These components form a **reflexive loop**: new information is extracted and structured into a KG; that KG is validated and refined; and subsequently used to improve the quality of language model outputs. While language models are used at multiple points in the pipeline, their roles and configurations may vary depending on the task.

1.3.1 LLM2KG: Language-Model-Assisted Knowledge Graph Construction

This component focuses on automating knowledge acquisition from unstructured data. LLMs extract entities and relations, suggest schema updates, and propose triples for inclusion in a domain-specific knowledge graph. The process is partially guided by existing ontology structures, benchmark queries, and domain expectations.

Key capabilities include:

- Triple and schema suggestion via LLM prompting
- Integration with human validation where uncertainty is high
- Dynamic KG enrichment based on task- or event-driven data

1.3.2 Human-in-the-Loop: Validation with Strategic Oversight

Rather than continuously curating data, human experts act as **validators** who intervene selectively—where confidence is low, novelty is high, or semantic conflicts arise. The human-in-the-loop framework ensures reliability and interpretability, while preserving the scalability of machine-driven processes.

This includes:

- Confidence scoring and filtering of generated content
- Modular validation pipelines for schema, triples, and query output
- Feedback-driven refinement of both LLM behavior and KG content

1.3.3 KG2LLM: Knowledge-Guided Inference and Grounding

Once structured knowledge is validated and maintained, it serves as a **grounding resource** to reduce hallucination and enhance consistency in LLM outputs. Instead of relying on latent knowledge or unsupervised retrieval, this component uses KGs to retrieve precise subgraphs relevant to a user query and incorporates them into the model's reasoning process.

Techniques include:

- KG-based subgraph retrieval for semantic query grounding
- Augmented prompts or retrieval-augmented generation (RAG) enhanced with structured facts
- Result validation through symbolic matching or logical consistency checks

Terminology: Role-Differentiated LLMs

To clarify architectural roles, we distinguish between two core Large Language Model (LLM) functions used throughout this thesis:

- **Knowledge Builder LLM (KB-LLM)**: Responsible for extracting structured knowledge from unstructured, semi-structured, or structured inputs based on a predefined ontology. It populates or updates a knowledge graph (KG) with relevant entities and relationships aligned with the domain schema.
- **Target LLM**: Consumes curated knowledge from the KG to generate task-specific responses. It incorporates retrieved KG context to improve reliability, factual grounding, and domain specificity.

These roles correspond to the upstream (LLM2KG) and downstream (KG2LLM) phases of the Reflexive Composition pipeline. In practice, both roles may be instantiated by the same LLM—particularly in the case of large frontier models capable of multitasking—or by separate, specialized models when task separation, modularity, or fine-tuning constraints apply.

1.3.4 Summary

Reflexive Composition enables a closed-loop system in which knowledge can be extracted, structured, validated, and reused across both upstream and downstream tasks. LLMs act as both producers and consumers of curated knowledge, while human experts focus on high-value validation steps to ensure quality without introducing bottlenecks.

The architecture is tested across three distinct case studies, each highlighting a different limitation of conventional LLM pipelines: temporal drift, restricted access to private data, and inherited bias from low-quality training sources. Results demonstrate the versatility, scalability, and effectiveness of reflexive integration in real-world settings.

One of these case studies—focused on temporal drift—is introduced in Chapter 8 and revisited throughout the thesis as a running example. Centered on the 2024 attempted assassination of former President Donald Trump, it illustrates how Reflexive Composition enables time-sensitive knowledge ingestion, validator-supported fact-checking, and schema-grounded response generation in a high-impact real-world context.

1.4 Case Study Selection and Coverage

To evaluate the reflexive composition framework, this thesis applies it across three diverse case studies, each selected to represent a distinct class of limitations in language model behavior and knowledge integration.

These include:

- **Systemic bias and inconsistency** - when LLMs generate incorrect or outdated outputs due to limitations in their training data [97];
- **Temporal hallucination** - when LLMs fail to incorporate recent or evolving events [46];

- **Knowledge absence** - when LLMs lack access to private or domain-restricted information [5].

Each case study explores how reflexive composition can be applied end-to-end: using LLMs to construct or enrich a KG, involving human experts in validation, and subsequently using the curated KG to improve LLM outputs. While the core architecture remains consistent, each scenario highlights unique challenges in knowledge modeling, validation design, and integration into downstream applications.

Table 1.1 summarizes the case studies and how they map to core LLM limitations and reflexive design strategies .

Table 1.1: Case Studies and Corresponding LLM Limitations

Case Study	**LLM Limitation Addressed**	**Domain**	**Reflexive Mechanism Emphasized**
Code Generation	Hallucination from training bias or insecure code	Software Engineering	Schema-enforced correction, KG-based prompt grounding
Temporal Knowledge	Hallucination due to outdated or missing facts	News / Current Events	Real-time KG updates, schema drift handling
Medical Integration	Hallucination from missing private data	Clinical / Health	Privacy-preserving KG construction and validation

1.4.1 Bias Reduction in Code Generation

LLMs trained on large code repositories inherit legacy biases and insecure patterns. This case study evaluates how curated KGs built from vetted, secure code sources can guide LLMs toward safer and more consistent outputs. It explores schema design for code validation, human-in-the-loop feedback for unsafe suggestions, and the impact of knowledge grounding on completion quality. *(Presented in Chapter 7.)*

1.4.2 Temporal Knowledge and Real-Time Updates

This case study demonstrates how the reflexive pipeline addresses the challenge of **temporal drift** - the inability of LLMs to reflect recent events. Using breaking news scenarios, we simulate the extraction of new facts, automated KG updates, and their impact on grounding future LLM responses. This tests both the speed and reliability of reflexive knowledge update cycles. *(Detailed in Chapter 8)*

1.4.3 Private Medical Data Integration

In domains like healthcare, LLMs often hallucinate or generalize inappropriately due to a lack of access to patient-specific private records. This case study constructs a structured knowledge graph from electronic medical records (EMRs), semantically enriched using standardized ontologies such as RxNorm. It demonstrates how reflexive composition enables secure, inference-time grounding of LLM outputs using validated private knowledge, without exposing sensitive data during model training or requiring modification of underlying model parameters. *(Explored in Chapter 9.)*

1.5 Research Contributions

This thesis presents *Reflexive Composition*, an architecture for integrating knowledge graphs (KGs) and large language models (LLMs) through iterative, bidirectional enhancement. The contributions span methodological design, system implementation, and empirical evaluation across three application domains. Reflexive Composition extends prior work by unifying symbolic and neural approaches in a closed-loop process: structured knowledge informs language inference, and language models contribute to knowledge graph construction. A human-in-the-loop (HITL) mechanism is integrated to support validation and refinement throughout the cycle.

Our contributions can be grouped into three categories:

1.5.1 Methodological Contributions

This thesis contributes to neural-symbolic integration through a bidirectional framework that coordinates knowledge extraction, validation, and generation across LLMs, KGs, and human oversight:

- We propose a **cyclic architecture** in which LLMs assist in constructing and updating knowledge graphs, while KGs condition and constrain LLM responses. This enables iterative improvement across both components, as opposed to traditional unidirectional pipelines.
- We redesign the role of human experts from manual knowledge entry to **validation and exception handling**, using threshold-based intervention points to balance scalability and quality assurance.
- We define domain-specific integration protocols for temporal reasoning, privacy-sensitive contexts, and code security scenarios, demonstrating applicability across diverse settings.

In contrast to prior work that treats LLMs primarily as consumers of KG input or as extraction tools, our approach enables feedback between structured and unstructured components to support mutual refinement.

1.5.2 Technical Implementation Contributions

We implement this methodology through three interconnected components, each representing a significant technical contribution:

The **LLM2KG component** advances automated knowledge graph construction through:

- Entity and relationship extraction pipelines optimized for domain-specific knowledge
- Knowledge extraction prompt engineering techniques for improved accuracy
- Dynamic schema evolution framework that adapts to evolving domains

Our **HITL framework** redefines human validation through:

- Multi-stage validation protocols balancing automation with expert oversight
- Quantitative validation metrics that adapt to domain requirements
- Automated support systems that handle routine validation tasks

The **KG2LLM component** performs structured knowledge integration through:

- Query-specific subgraph retrieval for context-relevant triple extraction
- Prompt-based knowledge fusion preserving generation fluency while ensuring factual alignment
- Contradiction detection circuits initiating targeted knowledge graph updates

These technical contributions collectively enable more efficient knowledge engineering while improving LLM reliability through structured knowledge integration.

1.5.3 Empirical Validation Contributions

We evaluate our methodology through domain-specific case studies and quantitative analysis:

- We develop and evaluate domain-specific implementations for temporal knowledge management, private data integration, and security pattern extraction.
- We observe significant improvements in LLM response accuracy across case studies, with approximate gains of 15–30% relative to baseline models.
- We informally observe substantial reductions in manual knowledge engineering effort compared to traditional KGE practices, based on comparative experiences with manual graph construction exercises (e.g., observed in iTelos project-based courses) versus the LLM-assisted workflows developed in this thesis.
- We identify generalizable patterns and domain-specific adaptations required for effective neural-symbolic integration.

Through these case studies, we provide practical insights into the application of neural-symbolic integration across different domains, contributing to the broader understanding of hybrid AI system deployment.

1.5.4 Comparison with Existing Approaches

Table 1.2 provides a comparison between our Reflexive Composition methodology and traditional approaches to LLM-KG integration.

These contributions collectively advance the state of the art in neural-symbolic integration, providing both theoretical frameworks and practical implementation strategies for more effective AI systems that combine the strengths of neural and symbolic approaches.

1.6 Structure of the Thesis

This thesis is organized into four main parts, each building upon the previous to develop and validate the Reflexive Composition framework:

Aspect	Traditional Approaches	Reflexive Composition
Knowledge Engineering	Manual construction requiring domain experts	LLM-assisted extraction with schema-guided human validation
Update Process	Scheduled refresh cycles	Incremental updates via validator feedback and schema evolution
Human Involvement	Direct construction and extensive review	Targeted human validation based on uncertainty and schema deviation
Knowledge Integration	Static incorporation during training or inference	Context-aware retrieval and integration at inference time
Privacy Handling	Access control and anonymization	Policy-guided integration of sensitive knowledge with access controls
Adaptation to Domains	Custom implementation for each domain	Configurable architecture with domain-specific schema and prompt tuning

Table 1.2: Comparison of Reflexive Composition with Traditional Approaches

1.6.1 Foundations and Background

This first part establishes the motivation and theoretical context. The current chapter introduces the research problem, the proposed solution, and outlines the case studies and contributions. Chapter 2 reviews related work in knowledge graph engineering, large language models, and hybrid systems, highlighting gaps that motivate the thesis.

1.6.2 Methodology

The second part presents the core Reflexive Composition methodology:

- **Chapter 3** introduces the overall reflexive architecture, including the interaction between LLM2KG, Human-in-the-Loop validation, and KG2LLM components.
- **Chapter 4** describes LLM2KG, the pipeline that uses language models to assist in knowledge extraction, schema evolution, and graph construction.
- **Chapter 5** presents the human-in-the-loop validation framework, detailing validation checkpoints, confidence metrics, and strategic expert intervention.
- **Chapter 6** explains KG2LLM, a component that grounds LLM outputs in structured knowledge using retrieval and symbolic validation techniques.

Throughout Chapters 3 through 6, the methodology is illustrated using a consistent running example: the 2024 attempted assassination of former President Donald Trump. This example, explored in detail in Chapter 8, is used to demonstrate real-time ingestion, validator oversight, and prompt-grounded response generation, helping clarify abstract design principles in practical terms.

1.6.3 Empirical Validation

The third part evaluates the methodology through three case studies:

- **Chapter 7** targets code generation, mitigating inherited bias from legacy code repositories using curated KGs and schema-guided validation.

- **Chapter 8** applies the framework to a real-time news scenario, addressing temporal hallucinations through dynamic KG updates.
- **Chapter 9** focuses on medical knowledge integration, using privacy-preserving KG construction to enhance LLM outputs without compromising patient confidentiality.

1.6.4 Analysis and Conclusions

The final part consolidates the findings. **Chapter 10** summarizes the main contributions, discusses limitations, and outlines directions for future work, including broader deployment and system generalization. **Chapter 11** concludes with research contributions, a critical assessment of limitations and challenges, and directions for future research.

The final chapters synthesize the thesis findings. **Chapter 10** presents the main contributions, outlines current limitations, and proposes directions for future work, including broader deployment and cross-domain generalization. **Chapter 11** concludes with a summary of research contributions and a critical reflection on unresolved challenges and open research questions.

All code used in the case studies is available in a public repository, ensuring reproducibility. The appendices provide detailed implementation specifications, code examples, and discussions of key frameworks used throughout the research.

Chapter 2

State of the Art

2.1 Introduction

Knowledge graph engineering and large language model capabilities represent two complementary but traditionally separate research domains. Recent advances in hybrid systems have begun exploring their integration, though significant gaps remain in bidirectional enhancement and systematic validation. We examine three primary lines of work: knowledge graph engineering, language model capabilities, and hybrid systems [78] that integrate symbolic and neural methods. We then outline domain-specific challenges that arise when applying LLMs in high-risk or dynamic settings. Finally, we identify unresolved limitations in current approaches and describe how the proposed methodology addresses them.

2.2 Knowledge Graph Engineering

Knowledge graphs (KGs) provide structured, symbolic representations of entities and their relationships, supporting logical inference, entity linking, and structured query answering. While curated KGs like DBpedia [49] and YAGO [92] have achieved precision above 90% on known entity types, their adaptability to new domains and evolving facts remains limited compared to more dynamic construction pipelines.

2.2.1 Manual and Semi-Automated Construction

Initial KGs such as DBpedia and YAGO extracted information semi-automatically from structured resources like Wikipedia infoboxes. These offered high initial precision (˜95%) but required extensive human post-curation to handle schema drift and evolving domain ontologies. Such resources struggle with domain-specific extensibility without substantial expert effort.

2.2.2 Schema Design and Evolution

Ontology editors like Protégé [1] facilitate manual schema editing, but scalability is limited. Automatic schema induction techniques attempt to reduce expert load, though real-world adaptability remains a challenge. Complex domains require dynamic schema updates rarely addressed in traditional pipelines.

[1] `https://protege.stanford.edu`

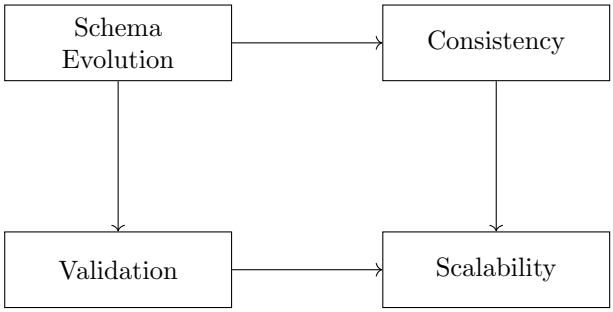


Figure 2.1: Interrelated Knowledge Graph Maintenance Challenges.

2.2.3 Validation Techniques

Schema compliance frameworks like SHACL[2] enforce triple-level consistency, but brittleness emerges when domain schemas evolve. Wikidata, for example, requires sustained human oversight to counteract schema drift and inconsistencies.

2.2.4 Advanced KG Construction

Recent pipelines such as AutoKnow [26] and Bootleg[3] integrate entity linking and semi-structured extraction. These systems reach micro-F1 scores between 60–78% depending on domain complexity, underscoring challenges of maintaining extraction fidelity at scale.

2.2.5 Ontology Evolution Frameworks

Frameworks like PromptKG[4] and dynamic schema mapping approaches support continual KG evolution. However, few systems integrate this with strategic human oversight. Reflexive Composition advances this field by coupling schema drift detection with validator-mediated updates, as elaborated in Chapters 4 and 5.

2.3 Language Models

Large Language Models (LLMs) have transformed natural language processing through pretraining on web-scale corpora and general-purpose decoder architectures. This section outlines core capabilities, known limitations, and mitigation strategies that shape how LLMs interact with structured knowledge.

2.3.1 Capabilities and Advancements

Pretrained models such as GPT-3 [7], PaLM [16], and LLaMA [94] perform competitively across a range of benchmarks, including MMLU [36] and SuperGLUE [96]. Few-shot prompting and in-context learning have enabled flexible task generalization. Instruction tuning [76] and chain-of-thought prompting [101] have further enhanced multi-step reasoning.

[2]Shapes Constraint Language (SHACL) W3C Recommendation 2017, available at `https://www.w3.org/TR/shacl/`
[3]Bootleg project: `https://github.com/HazyResearch/bootleg`
[4]PromptKG project: `https://github.com/zjunlp/PromptKG`

2.3.2 Limitations and Hallucinations

Despite their versatility, LLMs frequently generate fluent but incorrect outputcommonly referred to as hallucinations [82]. These stem from biases in training data, knowledge cutoffs, and lack of grounding. They are particularly problematic in high-stakes applications such as healthcare or finance, when factual errors can compound and propagate.

2.3.3 Mitigation Strategies

To reduce hallucinations and improve output reliability, several techniques have been proposed:

- **Retrieval-Augmented Generation (RAG):** Dynamically injects evidence from external sources during decoding [50].
- **Knowledge-Aware Pretraining:** Integrates structured information directly into model weights or adapters [54].
- **Fact-Checking Pipelines:** Applies rule-based or neural validators post hoc [60].

Each of these techniques addresses specific failure modes, but few integrate structured validator feedback or bidirectional grounding.

2.3.4 Parameter-Efficient Fine-Tuning (PEFT)

As LLMs scale, full-parameter fine-tuning becomes computationally prohibitive. PEFT methods such as LoRA [39] and adapters [38] offer modular updates with lower cost. These techniques are particularly useful in domain adaptation and continual learning scenarios.

2.3.5 Prompt Engineering Techniques

Prompting remains a primary mechanism for influencing LLM behavior without retraining. Techniques include:

- **Zero-shot and Few-shot Prompts:** Leverage minimal examples to induce task behaviors.
- **Schema-Aware Prompting:** Emphasizes structured alignment in KG-related tasks.
- **Naturalized Triples:** Express KG facts in fluent language to improve model uptake.

Chapter 6 builds on these techniques to develop controlled generation pipelines grounded in curated subgraphs.

2.4 Human-in-the-Loop Systems

Human-in-the-loop (HITL) approaches have long played a role in AI system design, particularly where automation intersects with domain-specific expertise, uncertain input, or evolving knowledge. In the context of KG and LLM systems, HITL models are typically used either for annotation, filtering, or validation.

2.4.1 Roles of Human Oversight

In knowledge engineering, human experts are often involved in schema design, disambiguation of entities, and review of extracted triples. In LLM evaluation, humans are used for qualitative scoring or alignment tuning. Recent trends also explore more dynamic forms of interaction, where human validators guide iterative model correction[86]. The challenge lies in balancing human input frequency and strategic value.

2.4.2 Interactive Feedback and Active Learning

Active learning paradigms[85] have influenced HITL architectures by promoting the idea of querying humans only when uncertainty is high or disagreement among models occurs. In the KG domain, this manifests in confidence-weighted validation or selective query expansion. In LLM pipelines, human preference data[91] has been shown to improve model alignment.

Our framework advances this direction by treating human oversight not as fallback but as a structured, strategic component of the architecture. The human validator operates at high-leverage checkpoints identified via scoring functions and automated heuristics.

2.4.3 Interactive Learning Frameworks

Interactive learning frameworks extend human-in-the-loop (HITL) paradigms by embedding expert feedback directly into the model update cycle. These systems prioritize selective querying, correction loops, and online learning to improve model performance while minimizing validation effort.

Active Learning Approaches. A key strategy in interactive learning is active sampling, where the system queries human validators only for low-confidence or high-impact cases. Traditional uncertainty-based sampling has been extended with model disagreement heuristics, knowledge gain estimators, and task-specific impact scoring. In knowledge graph (KG) construction, this supports selective triple validation and schema refinement. In LLM pipelines, models like *Recitation* and *PAL* incorporate user corrections to improve contextual consistency and reasoning coverage.

Expert Time Utilization. Recent systems emphasize not just accuracy, but expert efficiency. Validator-facing tools rank validation items by risk score, allow bulk annotation of low-risk triples, and provide assisted editing interfaces for rapid correction. Our Reflexive Composition pipeline supports this through validator escalation thresholds (Chapter 5), which trigger human review only when automated validation fails or when potential impact is above a defined threshold.

Feedback Integration. Interactive learning requires not only feedback collection, but effective assimilation. Correction logs and override patterns are used to update prompting templates, alter model trust scores, and revise schema constraints. Some systems apply counterfactual simulationtesting how the model would behave if a correction were always followedto detect brittle generalizations. Reflexive Composition implements this in both upstream (KB-LLM) and downstream (Target LLM) contexts, enabling continuous quality refinement through validator-influenced feedback loops.

2.4.4 Human-in-the-Loop Systems

Human-in-the-loop (HITL) validation plays a central role in ensuring knowledge quality within reflexive systems. Prior approaches have focused on manual fact-checking, rule-based filtering, or post-hoc revision. Reflexive Composition extends these ideas by integrating cognitive load-aware mechanisms directly into validation routing and interface design. The full methodology is detailed in Chapter 5.

2.5 Hybrid KG–LLM Architectures

Recent research has increasingly explored the integration of symbolic knowledge from KGs with the generative capabilities of LLMs. These hybrid architectures vary in complexity and focus, but most fall into one or more of the following four categories. Figure 2.2 illustrates the primary conceptual axesranging from knowledge graph enhancement to LLM augmentationalong which these systems can be organized.

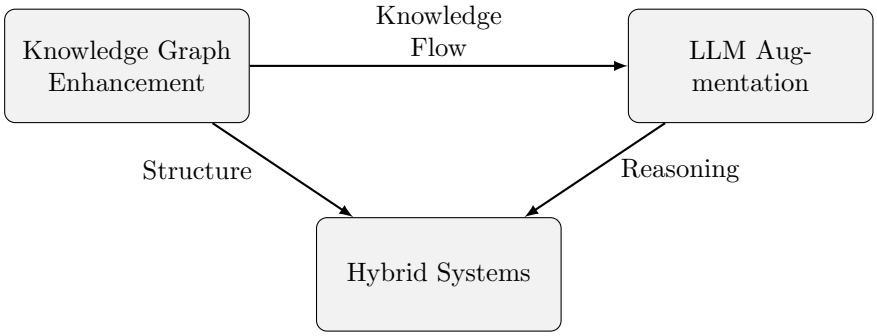


Figure 2.2: Primary Integration Paradigms in KG-LLM Systems

The integration of symbolic knowledge and neural generation is not only architectural but also reflexive: outputs generated by LLMs can iteratively refine the very graphs that condition them. This reflexive paradigm, introduced formally later in this chapter and operationalized in Chapters 3 - 6, underpins the methodology developed in this thesis.

2.5.1 Knowledge Retrieval and Integration

Systems in this category incorporate external knowledge into LLMs at inference time. Retrieval-Augmented Generation (RAG)[50] retrieves relevant documents or passages to contextualize the models response. Other approaches use entity linking[56] and subgraph extraction[55] from KGs to embed structured facts into prompts. These systems enhance recall but can suffer from inconsistency and prompt volatility, especially in multi-hop reasoning tasks.

2.5.2 Knowledge Utilization and Reasoning

Beyond integration, some architectures focus on structured reasoning over KG-derived information. Approaches like KGT5 [84] and Thinker [18] generate intermediate symbolic representations or logic-based paths during inference. These methods aim to improve interpretability by separating retrieval

from reasoning. However, they rely on well-defined schemas and high-coverage knowledge graphs, which can limit their applicability in evolving or under-specified domains.

2.5.3 Optimization and Specialization

Several systems target efficiency and task-specific alignment by fine-tuning models with knowledge-enriched objectives. Examples include KnowBERT[80], which integrates entity embeddings into transformer layers, and KEPLER[98], which jointly optimizes for KG completion and language understanding. While effective, these models are often static and require retraining to incorporate new knowledge, limiting their adaptability.

2.5.4 Bidirectional and Reflexive Systems

Fewer systems support bidirectional exchange between KGs and LLMs. Works like DKG[99] propose co-evolution frameworks where structured knowledge improves language generation, and language outputs update KGs. However, these systems are often tailored to specific domains or constrained by tight coupling between components. Our framework extends this direction by introducing a modular, reflexive loop in which LLM2KG, HITL, and KG2LLM form a flexible pipeline that supports dynamic updates, selective validation, and domain adaptation.

2.5.5 Architecture Evolution

The evolution of hybrid KGLLM architectures reflects a shift from task-specific pipelines to modular, reflexive systems capable of supporting dynamic, domain-adaptive reasoning. Figure 2.3 illustrates this progression across four architectural eras.

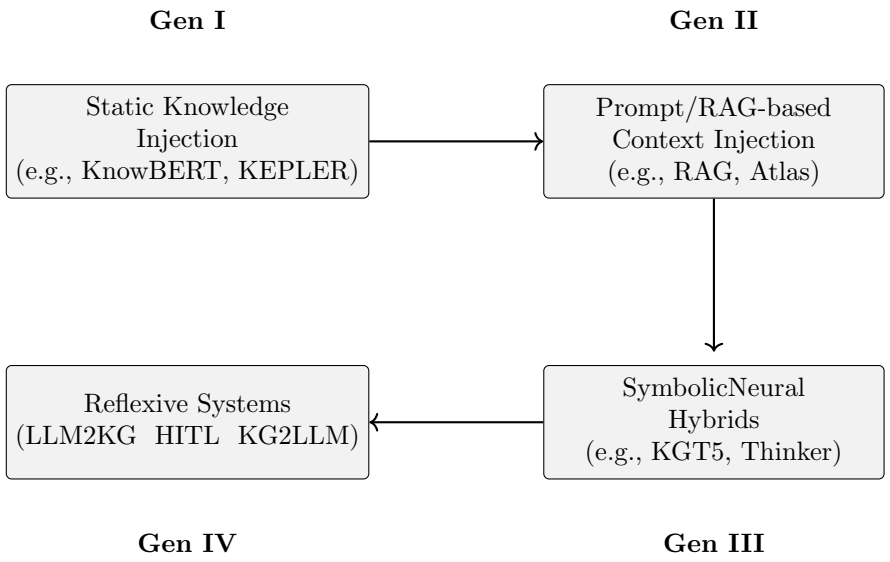


Figure 2.3: Evolution of Hybrid KGLLM Architectures from static embedding models to reflexive feedback pipelines with human-in-the-loop validation and schema evolution.

Static Knowledge Injection. Early systems (e.g., KnowBERT, KEPLER) embedded structured knowledge into fixed LLM parameters via pretraining. These architectures were static, requiring full retraining for updates and offering little transparency or post-hoc correction.

Retrieval-Augmented and Prompt-Based Systems. RAG-style systems injected dynamic context at inference time using document retrieval or subgraph expansion. While more flexible, they suffered from prompt volatility, lack of grounding, and limited multi-hop reasoning control.

SymbolicNeural Reasoning Hybrids. Systems such as KGT5, Thinker, and GINKGO introduced intermediate symbolic representations or multi-step logic chains. These improved interpretability but often depended on curated schemas and brittle intermediate formats.

Reflexive Architectures. Reflexive systems, like the one introduced in this thesis, integrate upstream KG construction (LLM2KG), human validation (HITL), and downstream generation (KG2LLM) into a modular pipeline. They support iterative feedback, domain adaptability, and schema evolution without retraining.

Scalability and Resource Trade-offs. Table 2.1 compares resource and deployment characteristics across architecture types.

Architecture Type	Updateability	Human Oversight	Resource Overhead
Static Embedding (e.g., KnowBERT)	Low	None	High
RAG/Prompt Injection	Medium	Indirect	Medium
SymbolicNeural Reasoning	Medium	Optional	MediumHigh
Reflexive Composition	High	Embedded	Medium

Table 2.1: Trade-offs Across KGLLM Architectural Generations

2.5.6 Integration Paradigms

Hybrid KGLLM systems vary widely in how they coordinate symbolic and neural components. These differences reflect trade-offs in scalability, consistency, interpretability, and domain adaptability. We organize these systems into four integration paradigms, illustrated in Figure 2.4.

1. Loose Coupling (Pre/Post Processing). Early hybrid systems adopt a loose-coupling strategy, where symbolic components wrap around a frozen LLM. Examples include pre-generation KG lookups and post-generation output filtering using constraint graphs or ontological rules. While simple and modular, these systems lack reflexivity and suffer from latency in knowledge refresh cycles.

2. Prompt-Level Fusion. Prompt-centric systems like RAG inject KG-derived entities or subgraphs directly into the prompt context. These approaches enable dynamic grounding and few-shot prompting, but struggle with long prompts, schema conformance, and transparency. Their performance also varies greatly across LLMs with different context handling capabilities.

3. Layered or Mid-Level Fusion. Systems like KEPLER and KnowBERT integrate symbolic knowledge at the representation layer, often through entity embeddings or attention-weighted knowledge layers. These approaches improve factual grounding and reduce hallucinations, but are expensive to train and static in naturemaking them poorly suited for evolving domains.

4. Reflexive Pipeline Integration. Reflexive systems (such as the framework presented in this thesis) unify construction (LLM2KG), validation (HITL), and generation (KG2LLM) into a modular loop. This design enables feedback between components, supports dynamic schema evolution, and maintains quality through selective human review. It is particularly suited to domains with high volatility or long-tail phenomena (see Chapter 7).

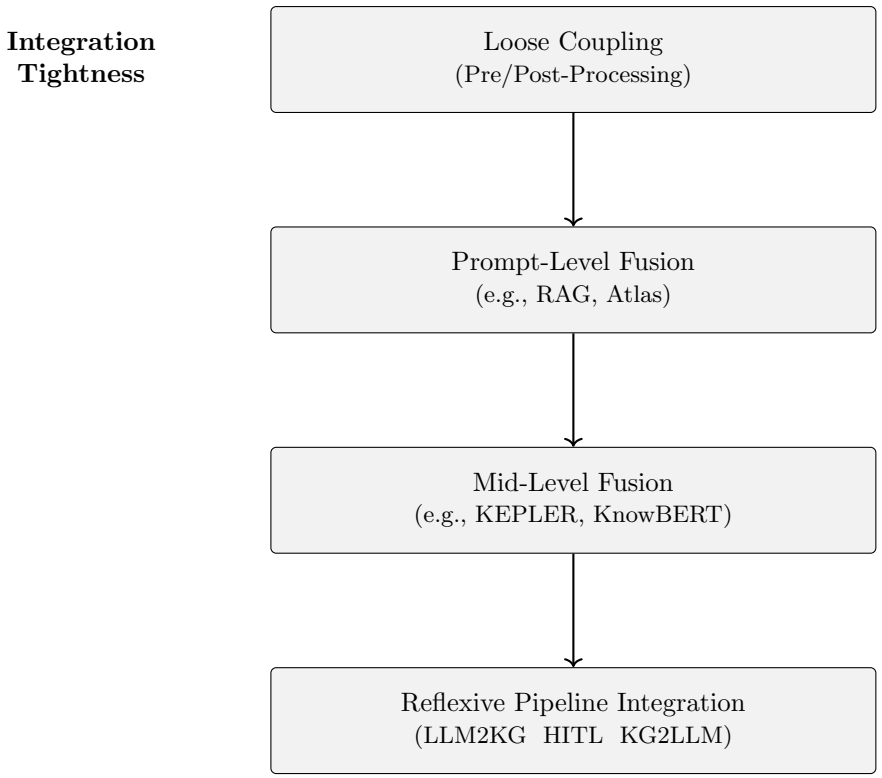


Figure 2.4: Evolution of KG LLM Integration Paradigms. Systems range from loosely coupled wrappers to fully reflexive pipelines with validator-driven adaptation.

Domain-Specific Adaptation. The choice of integration paradigm often reflects domain constraints. For example:

- Medical systems prioritize high accuracy and legal traceability layered validation and explicit reasoning paths.
- Financial assistants require temporal sensitivity favoring reflexive, versioned updates over static embeddings.
- Code LLMs often combine structured ASTs with language prompts benefiting from prompt-level fusion and graph augmentations.

In summary, integration paradigms determine not only how LLMs and KGs interact, but also what trade-offs are permissible in terms of speed, reliability, and transparency. The reflexive pipeline presented in this thesis builds on these paradigms by enabling updateable, validated, and domain-adaptive KGLLM alignment.

2.5.7 Symbolic-Neural Reflexivity in Hybrid Systems

The Reflexive Composition framework builds on prior efforts to combine symbolic and neural representations in artificial intelligence. Symbolic approaches offer structured, interpretable knowledge representations (e.g., ontologies, triples, schema constraints), while neural methods support generalization through data-driven pattern learning. Recent systems have attempted to unify these capabilities through hybrid architectures that incorporate both formal structure and flexible language generation.

Reflexivity in this context refers to a system's ability to use outputs (e.g., generated text, model completions) as feedback that updates the knowledge sources conditioning future outputs. This closes the loop between generation and representation, forming a bidirectional exchange between large language models (LLMs) and knowledge graphs (KGs).

The KG2LLM component builds on three complementary ideas:

- **Graph-based representations** support encoding of semantic relationships in a structured form suitable for retrieval, validation, and constraint enforcement [19].
- **Probabilistic generation models**, grounded in autoregressive decoding and attention mechanisms, can condition on structured context to produce contextually appropriate output [95].
- **Information fusion** techniques allow symbolic subgraphs to be combined with natural language prompts to guide generation in a controlled manner [55].

This hybrid reasoning setup supports traceable inference and schema-conformant output, while preserving LLM fluency. Reflexive Composition treats the knowledge graph as a dynamic context that co-evolves with model outputs. Implemented in Chapters 3 and 6 and evaluated in Chapter 7, this modular pipeline contrasts with earlier systems that treated symbolic input as static pre- or post-processing. Instead, it positions knowledge graphs as both grounding context and adaptive memory, updated in response to evolving language output.

2.6 Evaluation Methodologies and Metrics

Evaluating hybrid KGLLM systems presents a multidimensional challenge. Prior frameworks typically focus on retrieval performance, factual consistency, and language generation quality. In Reflexive Composition, we extend these criteria with a reflexivity-aware lens, emphasizing validator efficiency, update responsiveness, and system-wide coherence.

2.6.1 Retrieval-Based Metrics

For systems involving subgraph or document retrieval, standard metrics include Precision@k, Recall@k, Mean Reciprocal Rank (MRR), and Coverage. RAG-style systems typically report Precision@k@5

in the range of 4265% depending on corpus domain and chunking strategy. GraphPrompt, which incorporates symbolic prompts, achieves notable F1 improvements on datasets like GraphQA.

2.6.2 Knowledge Consistency and Factuality

Factual correctness is measured using token-level F1, triple-level accuracy, and entailment-based scoring. Systems such as KGT5 compute symbolic alignment between predicted and gold-standard reasoning paths, while Factual Consistency Score (FCS) is calculated using pretrained Natural Language Inference (NLI) models like RoBERTa or DeBERTa. Reflexive systems further quantify validator correction rate and consistency before and after fact injection.

2.6.3 Generation Quality Metrics

Standard text generation metricsBLEU, ROUGE, and METEORoffer limited insight into structured grounding. More recent alternatives include:

- Groundedness Score
- Contradiction Rate
- Calibration Accuracy

In our system, contradiction-aware metrics are particularly important for identifying hallucinatory responses that conflict with KG context or validator judgment.

2.6.4 Benchmark Evolution

Reflexive systems require evolving benchmark sets to assess runtime adaptability and update responsiveness. We extend traditional QA benchmarks with the following components:

- **Temporal QA**: Evaluates consistency across historical and current event sequences.
- **Schema Drift Detection**: Uses gold-standard queries to assess a systems ability to adapt to changes in ontology structure.
- **Validator Agreement**: Measures inter-annotator consistency and identifies areas of uncertainty or disagreement during validation.

Our benchmark format (see Appendix C) supports dynamic case injection and resolution tracking.

2.6.5 Multi-dimensional Assessment

We adopt a multi-layered evaluation framework:

- **System Metrics**: Retrieval, factuality, latency
- **Validator Metrics**: Intervention rate, override precision, fatigue-adjusted accuracy as part of Human-in-the-Loop performance evaluation
- **Reflexivity Metrics**: Update propagation time, schema evolution accuracy, retraining necessity

This framework enables assessment of the system across accuracy metrics, computational efficiency, error correction capabilities, domain adaptation, and iterative performance improvements.

Dimension	Metric Type	Example Metric
Retrieval	IR / Graph Alignment	Precision@k@5, MRR, Coverage
Factuality	NLI / Triple F1	FCS
Generation	Grounding / Fluency	Groundedness Score, Contradiction Rate
Human Loop	HITL Optimization	Validator Agreement, Escalation Ratio
Reflexivity	Temporal Adaptivity	Update Lag, Schema Drift Recovery

Table 2.2: Evaluation Dimensions in Reflexive Composition

2.7 Comparison to Hybrid and Reflexive Systems

While prior sections reviewed core LLMKG capabilities and ontology-grounded modeling, this section contextualizes Reflexive Composition in relation to recent hybrid and interactive systems.

Several recent approaches aim to integrate Large Language Models (LLMs) with structured knowledge representations such as knowledge graphs (KGs) to reduce hallucinations, improve factuality, or enable domain-specific reasoning. In this section, we compare Reflexive Composition to key hybrid systems and ontology-guided frameworks.

2.7.1 Hybrid Generation Systems

Models like **Thinker** [18] and **KGT5** [84] integrate structured knowledge into generation via context augmentation or fine-tuning, but treat the KG as static and lack mechanisms for reflexive validation or runtime schema evolution. Similarly, earlier work such as **KnowBERT** [80] and **KEPLER** [98] encodes symbolic information into model weights through knowledge-enhanced training. While effective for entity-centric tasks, these models require retraining to incorporate new knowledge and do not support human-in-the-loop feedback or dynamic KG refinement. Our approach addresses these limitations through bidirectional updates between LLMs and evolving knowledge graphs.

2.7.2 Ontology-Guided Extraction

Systems such as **LLM-as-IE** [8] and **OntoGPT**[5] use LLMs for schema-constrained structured information extraction. While aligned with the goals of the KB-LLM component of our framework, they do not support loop closure through reinjection or evaluation-informed updates.

2.7.3 Human-Centric Validation Frameworks

Few systems place human validators directly in the refinement loop. Notable exceptions include **Pistis-RAG** [4], which allows post-hoc human revision of retrieved contexts, and **ThinkAloud** [102], which collects human rationales. However, neither integrates such feedback into a reusable graph substrate or reflexive KG update pipeline.

2.7.4 Comparison Summary

Table 2.3 summarizes differences across representative systems and Reflexive Composition.

[5] `https://github.com/monarch-initiative/ontogpt`

System	LLMKG Integration	HITL	Ontology Use	Reflexive Loop	Updateability
KnowBERT	Fine-tuned entity embeddings	No	Partial	No	Low (requires retraining)
KEPLER	Joint KG+LM training	No	Yes	No	Low (requires retraining)
KGT5	Contextual KG encoding	No	Partial	No	Medium (via prompts)
Thinker	KG fine-tuned LLM	No	No	No	Low (static KG use)
GINKGO	Symbol-conditioned decoding	No	Partial	No	Medium (at prompt level)
OntoGPT	Prompt-based schema extraction	No	Yes	No	Medium (static KG output)
LLM-as-IE	Schema-guided prompt extraction	No	Yes	No	Medium (extraction only)
Reflexive Composition	Runtime bidirectional LLMKG updates	Yes	Yes	Yes	High (live updates + HITL)

Table 2.3: Comparison with State-of-the-Art Hybrid Approaches

This comparative view illustrates how Reflexive Composition integrates upstream schema-guided extraction, human-in-the-loop validation, and downstream KG-guided response generation within a single reflexive pipeline.

2.8 Domain Challenges and Case Study Contributions

The integration of knowledge graphs and language models has been applied across diverse domains, each with unique constraints and goals. Reviewing these deployments provides a backdrop for understanding the design of our case studies.

2.8.1 Software Engineering and Code Bias

Software engineering applications of LLMs face well-documented risks related to outdated, insecure, or non-standard coding patterns in training data. Models fine-tuned on public code repositories have been shown to reproduce deprecated APIs, hardcoded secrets, and vulnerable idioms [79]. While rule-based linters and static analysis tools can detect such issues post hoc, they do not intervene during generation.

Recent models like CodeBERT [28] and InCoder [29] incorporate syntactic and structural features of code but do not integrate curated security knowledge. Prompt-based strategies, such as warning-prefixed inputs [11], offer limited improvements and may not generalize across diverse coding tasks or frameworks.

Chapter 7 introduces a case study that uses Reflexive Composition to inject curated knowledge of deprecated or dangerous constructs directly into the generation pipeline. Knowledge graphs constructed from CVE archives, OWASP rules, and version histories are validated and referenced dynamically during code synthesis. This setup reduces hallucinated dependencies, outdated API calls, and

security flaws while preserving generation flexibility. The validator-driven loop ensures that the LLM receives corrective feedback and can self-correct on future invocations.

2.8.2 Temporal Knowledge and Recency

LLMs trained on static corpora struggle with events that evolve over time. Their factual grounding quickly becomes stale, particularly in domains like current events, crisis response, or rapidly shifting geopolitical contexts [81]. Retrieval-Augmented Generation (RAG) systems partially address this by querying live data, but often lack semantic grounding or contextual continuity.

Temporal QA benchmarks such as TempQuestions [43] and MemIT [73] expose brittleness in both long-term memory and time-sensitive accuracy. These benchmarks reveal that even when provided with correct evidence, models frequently hallucinate outdated or contradictory information.

Chapter 8 presents a case study where Reflexive Composition is applied to mitigate temporal hallucination. A continuously updated knowledge graph integrates timestamped news facts and contextual metadata. LLM generation is conditioned on subgraphs filtered by temporal relevance, and reflexive validators are employed to ensure alignment with the most recent state. This setup significantly improves recency fidelity and allows graceful degradation when data is ambiguous or missing.

2.8.3 Medical Reasoning and Structured Validation

Healthcare applications of LLMs require integration with controlled terminologies and structured schemas to avoid ambiguity, ensure safety, and maintain compliance. Many LLM-based systems struggle to align with clinical knowledge bases such as RxNorm, SNOMED CT, or FHIR resources, leading to issues in interpretability, traceability, and downstream decision support.

Past work in medical NLP has focused on entity linking, relation extraction, and summarization, but typically relies on static resources and pipeline designs [24, 100]. LLMs introduce opportunities for flexible understanding but risk introducing hallucinated treatments or diagnoses without contextual validation.

Chapter 9 presents a case study where Reflexive Composition enables KG-aligned question answering over synthetic patient records. A schema-constrained validator ensures that LLM outputs remain consistent with clinical vocabularies and respect patient-level constraints. This pipeline demonstrates how human-in-the-loop validation and ontology-grounded generation can improve safety and transparency in medical AI systems.

2.9 Research Gaps and Opportunities

Despite rapid progress in integrating language models with symbolic knowledge systems, several open challenges remain. These limitations motivate the contributions of this thesis and highlight directions for continued research.

2.9.1 Lack of Reflexivity in Hybrid Pipelines

Most hybrid systems support unidirectional flows: either using KGs to constrain LLM output or employing LLMs to extract KG content. Few support iterative, reflexive enhancement where knowledge

and language co-evolve through validation cycles. This limits the adaptability and long-term utility of current solutions.

2.9.2 Incomplete Support for Validation and Human Oversight

Existing systems typically treat validation as either a post-hoc filter or a manual bottleneck. There is little integration of strategic human-in-the-loop checkpoints that adapt based on system confidence, novelty, or domain sensitivity. As a result, scalability and quality assurance remain decoupled.

2.9.3 Static Knowledge and Poor Temporal Responsiveness

Many deployed systems rely on static snapshots of knowledge or pretrained models. This makes them poorly suited for domains requiring real-time updates, schema drift handling, or active context adaptation. Temporal hallucinations remain a pressing concern in domains like news, finance, and cybersecurity.

2.9.4 Fragmented Evaluation and Benchmarking

There is no standardized evaluation suite for KGLLM systems that integrates symbolic fidelity, response accuracy, and validation efficiency. Most benchmarks focus on one dimension (e.g., generation fluency or triple accuracy) but fail to capture the interplay between components or the impact of human oversight.

These gaps suggest that future research should emphasize modularity, continuous learning, structured oversight, and domain adaptability in hybrid systems. The Reflexive Composition framework proposed in this thesis is designed in response to precisely these challenges.

2.9.5 Gap Analysis Framework

To formalize the challenges outlined in Sections 2.9.1–2.9.4, we propose a multi-dimensional framework that categorizes research gaps across four key axes: reflexivity, validation, temporality, and evaluation. Each dimension reflects a structural limitation in current KGLLM systems and maps directly to design decisions in Reflexive Composition.

1. Categorization of Gaps. Table 2.4 summarizes the four primary gap categories, associated system limitations, and the corresponding architectural responses in this thesis.

2. Gap Significance Assessment. To assess significance, we apply a scoring rubric across three axes: *Impact*, *Prevalence*, and *Neglect*. Table 2.5 summarizes this assessment.

3. Mapping to Prior Work. These gaps are rooted in known limitations of popular systems:

- **KGT5, Thinker:** No reflexive loop; validation not integrated
- **KEPLER, KnowBERT:** Static embeddings; no support for update triggers
- **RAG-based models:** Limited schema grounding; evaluation focuses on fluency not grounding or coherence

Gap Dimension	System Limitation	Reflexive Composition Solution
Reflexivity	Unidirectional LLMKG flow	Modular feedback loop: LLM2KG HITL KG2LLM
Validation	Manual or post-hoc filtering	Strategic Human-in-the-Loop checkpoints guided by escalation scores
Temporal Responsiveness	Static snapshots, no update triggers	Reflexive update engine with validator-verified triggers
Evaluation Scope	Narrow benchmarks, no oversight metrics	Multi-dimensional evaluation including validator effort and contradiction rate

Table 2.4: Gap Categories, System Limitations, and Reflexive Composition Responses

Gap Type	Impact	Prevalence	Research Attention
Lack of Reflexivity	High	High	Low
Limited HITL Support	High	Medium	Medium
Temporal Fragility	High	High	Low
Evaluation Fragmentation	Medium	High	Medium

Table 2.5: Quantitative Assessment of Research Gap Significance

This gap analysis framework not only motivates the architecture presented in Chapter 3, but also provides a scaffold for future benchmarking and system design.

2.10 Positioning of This Thesis

In response to the limitations outlined above, this thesis introduces *Reflexive Composition*, a modular architecture that enables bidirectional integration between knowledge graphs and large language models. Unlike prior systems that operate in a unidirectional or task-specific fashion, the proposed framework supports iterative enhancement: language models assist in constructing and updating knowledge graphs, while knowledge graphs are used to ground and refine model outputs.

The architecture is composed of three core components:

- **LLM2KG**: a language model-driven pipeline for knowledge extraction and schema evolution,
- **HITL**: a strategic human-in-the-loop validation layer that ensures quality and scalability, and
- **KG2LLM**: a knowledge grounding module that uses structured graphs to reduce hallucination and improve reliability.

This framework is evaluated across three domainsreal-time news, privacy-sensitive medical records, and secure code generationchosen to reflect known LLM limitations such as temporal drift, restricted data access, and bias. In doing so, the thesis offers a reproducible design that supports reflexivity, adaptability, and oversight in hybrid AI systems.

Table 2.6 summarizes how the Reflexive Composition framework addresses core limitations of prior hybrid systems, offering a modular and reflexive alternative designed for adaptability, oversight, and multi-domain deployment.

Dimension	Typical Prior Work	Reflexive Composition (This Thesis)
KGLLM Integration Direction	Unidirectional: KG LLM or LLM KG	Bidirectional and iterative: LLM KG
Validation Strategy	Manual or post-hoc filtering	Strategically embedded Human-in-the-Loop checkpoints
Temporal Responsiveness	Static KGs and LLMs; limited update mechanisms	Real-time knowledge updates with reflexive triggers
Evaluation Focus	Single metrics (e.g., BLEU, triple accuracy)	Multi-dimensional: retrieval, generation, validation effort
Architecture Modularity	Task-specific, tightly coupled components	Generalizable and modular pipeline (LLM2KG HITL KG2LLM)
Domain Adaptability	Specialized to a single use case or data format	Validated across three diverse domains

Table 2.6: Comparison of Prior Work with Reflexive Composition across Key Architectural Dimensions

2.10.1 Contribution Mapping

Table 2.7 summarizes the core contributions of Reflexive Composition and maps them to the research gaps identified in Section 2.9.5. Each contribution addresses a specific architectural, methodological, or empirical shortcoming in prior hybrid KGLLM systems.

Contribution	Addressed Gap	Resolved In
Bidirectional reflexive pipeline	Lack of reflexivity in hybrid systems	Chapters 3, 4, 6
Strategic validator checkpoints	Limited HITL integration	Chapter 5
Temporal update engine	Static knowledge and schema drift	Chapters 3, 8
Schema-aligned extraction	Ontology grounding deficiencies	Section 4.5.2, Appendix C
Multi-dimensional evaluation metrics	Fragmented benchmarking methods	Section 2.6
Validator efficiency metrics	Oversight cost measurement gap	Chapter 5
Domain-specific adaptation protocols	Limited generalizability of hybrid systems	Chapters 7, 8, 9

Table 2.7: Mapping of Thesis Contributions to Addressed Research Gaps

Novelty Assessment Framework. We evaluate the novelty of our approach across three axes:

- **Methodological novelty:** The reflexive pipeline architecture introduces a closed-loop integration model, contrasting with the unidirectional designs of KGT5, Thinker, and RAG-based systems.
- **Technical novelty:** Validator escalation logic, schema-drift handling, and subgraph-based response generation form distinct, composable modules.
- **Empirical novelty:** The thesis offers cross-domain evaluations with validator agreement metrics and temporal consistency tests not available in previous work.

This contribution mapping forms the foundation for the system design presented in Chapter 3 and provides a roadmap for the empirical studies in Chapters 7 to 9.

Chapter 3

Architecture of the Reflexive Composition Framework

The Reflexive Composition framework is designed to support dynamic, reliable integration between large language models (LLMs) and knowledge graphs (KGs) through a modular, iterative process that involves strategic human oversight. This chapter presents the architectural overview, the role of each component, and the flow of data and control through the system.

3.1 Overview of the Reflexive Pipeline

The Reflexive Composition methodology addresses two persistent challenges in artificial intelligence: the resource constraints of knowledge graph engineering [106] and the factual inconsistencies in large language model outputs [82]. Through structured coordination of automated and human-supervised processes, the methodology establishes a bidirectional feedback loop between these technologies while implementing systematic validation protocols [103].

To clarify each components function in context, we refer throughout this chapter to a running example involving the Trump campaign rally shooting in July 2024. This case, introduced fully in Chapter 8, serves as an illustrative thread that demonstrates how Reflexive Composition supports time-sensitive knowledge integration, validator feedback, and structured generation.

The methodology comprises three interconnected components:

- **LLM2KG**: LLM-Based Knowledge Graph Generation
- **Human-in-the-Loop**: Strategic validation framework centered on data scientists
- **KG2LLM**: Knowledge Graph-Enhanced LLM Inference

As shown in Figure 3.1, the methodology creates a virtuous cycle between LLMs and KGs, with human validation providing strategic oversight. The components work together to enhance both knowledge representation and LLM performance through continuous refinement.

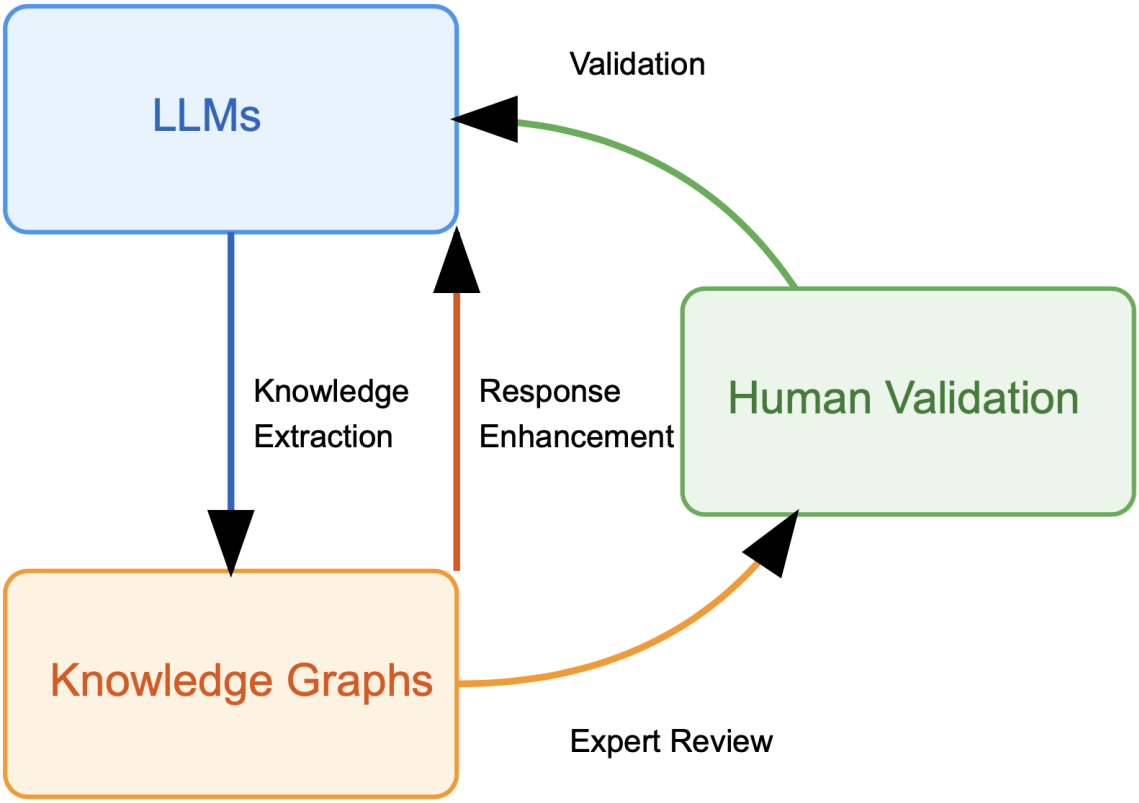


Figure 3.1: Reflexive Composition: Virtuous Cycle between KGs and LLMs

3.2 Core Components

Each component of the Reflexive Composition methodology addresses specific challenges while contributing to the overall goal of more reliable AI systems [78]. This section introduces their key characteristics and interactions before examining each in detail.

3.2.1 LLM2KG: LLM-Based Knowledge Graph Generation

Recent research has demonstrated the potential for LLMs to support knowledge graph construction [47, 2]. Building on these advances, the LLM2KG component of Reflexive Composition incorporates LLMs into the extraction, structuring, and validation pipeline through three integrated subsystems:

Knowledge Extraction Pipeline (KEP) serves as the primary interface with information sources, using LLM-based contextual inference and sequence tagging to identify domain-relevant entities and relationships across diverse document types.

Schema Evolution Framework (SEF) manages schema organization and evolution, enabling updates in response to new patterns while preserving structural consistency. It supports automated schema extension proposals with human review checkpoints.

Integration Validation System (IVS) monitors quality through staged validation processes, combining rule-based consistency checks with optional human review. It supports both real-time validation at extraction time and retrospective audits of graph quality.

3.2.2 Human-in-the-Loop Framework

The HITL architecture shifts the role of data scientists from manual knowledge construction to targeted validation. This methodological transition allocates human expertise to high-impact or high-uncertainty decisions rather than routine engineering tasks. The system supports multi-stage validation workflows that coordinate automated and manual processes to make efficient use of expert time. These workflows incorporate rule-based verification modules to manage routine checks, allowing data scientists to focus on complex cases that require domain-specific judgment. Validation checkpoints are selectively positioned to ensure oversight where it is most needed, while maintaining overall workflow efficiency.

3.2.3 KG2LLM: Knowledge Graph-Enhanced LLM Inference

The KG2LLM component executes structured knowledge integration to improve LLM response reliability while preserving their natural language capabilities. This component addresses the technical challenge of combining graph-based knowledge with neural language processing through three primary mechanisms:

1. Query-contextualized retrieval algorithms that select and extract relevant subgraphs based on semantic matching
2. Template-based knowledge fusion methods that incorporate graph-derived facts into LLM prompts
3. Error detection feedback loops that refine system performance through contradiction analysis and validation

These mechanisms operate as an integrated pipeline for knowledge-enhanced language generation, ensuring outputs maintain both fluency and factual accuracy.

3.2.4 Component Integration

These components work together in a continuous cycle where:

- LLM2KG enables efficient knowledge graph construction and evolution
- HITL ensures knowledge quality through strategic validation
- KG2LLM improves LLM responses through verified knowledge integration

The following sections examine each component in detail, exploring their implementation approaches, technical requirements, and operational workflows.

3.3 LLM2KG: LLM-Based Knowledge Graph Generation

The LLM2KG component leverages several integrated systems to enable automated knowledge graph construction and enhancement. At its core, the KEP processes input sources through LLM-based analysis. The SEF manages structural aspects of the knowledge graph, while the IVS ensures quality control.

The knowledge graph generation workflow consists of five primary sections (Figure 3.2), each playing a crucial role in the transformation of unstructured information into validated knowledge:

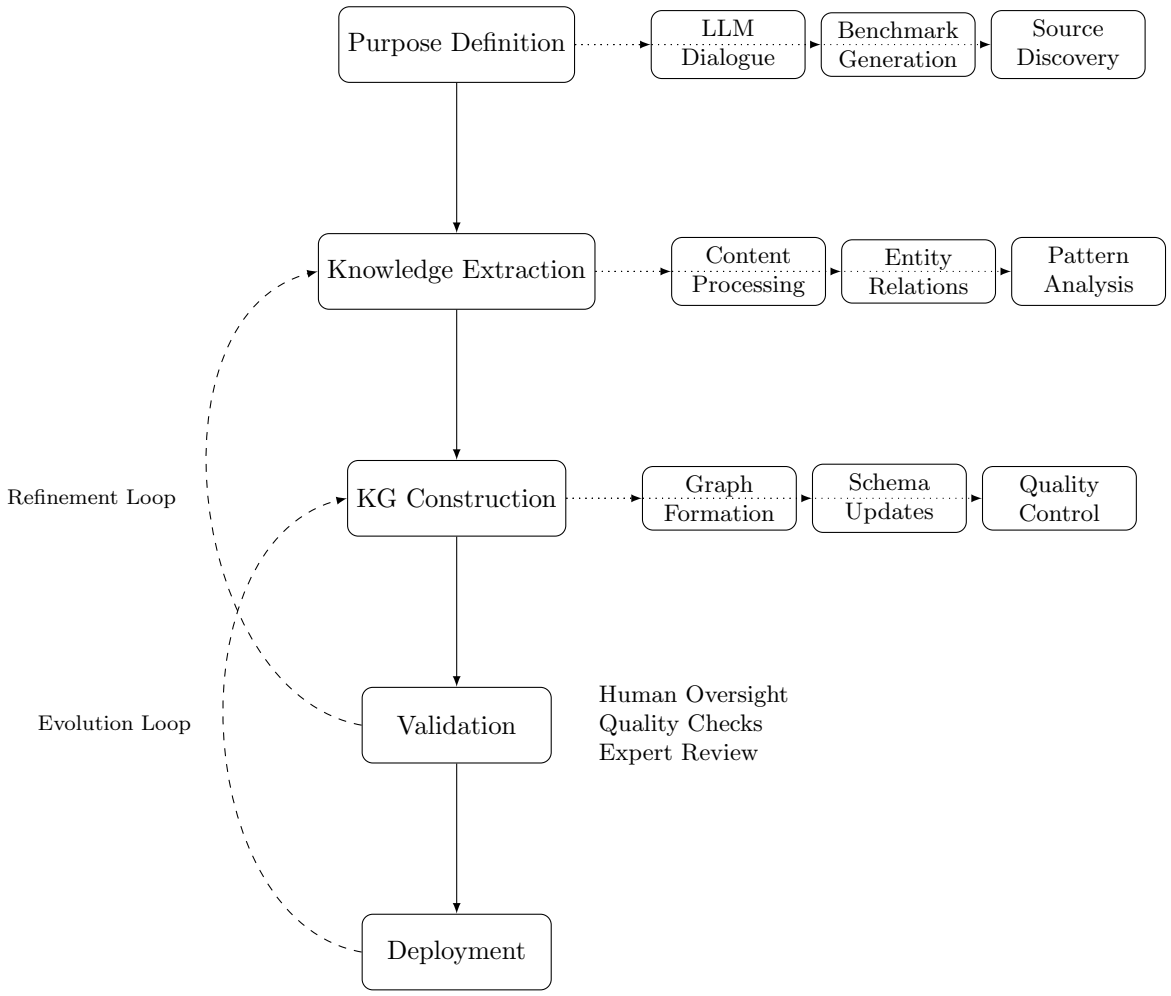


Figure 3.2: Enhanced Knowledge Graph Generation Workflow

3.3.1 Purpose Definition and Scoping

This section describes the initial setup phase, in which the task-specific requirements and intended scope of the knowledge graph (KG) are defined. Large language models (LLMs) are used to support this process by assisting in the identification of domain-relevant entities, relationships, and schema constraints.

Domain-specific KGs are constructed with a clear application purpose in mind. Because the Reflexive Composition framework is designed to simplify KG construction through automation and validation, it can accommodate highly constrained or specialized use cases. Explicitly defining the purpose helps to delineate the schema boundaries, exclude irrelevant information, and generate targeted validation queries.

Purpose Clarification through LLM dialogue

The workflow depends on a clear definition of the purpose, which is provided by a human or an upstream application in a stack. In the case of enhancement of an existing KG, the purpose would be available, but even here, it may need to be clarified whether it's a simple data increase or an evolution which might lead to schema updates. We have a few approaches to create or clarify the purpose through automation or user interactions:

- An interactive agent with multi-turn dialog handling capability with context retention
- Purpose vector embedding generation for similarity matching: this helps route the request to the right KG as well
- Automated requirement formalization using predefined templates
- Scope boundary detection using semantic clustering

Automated benchmark query generation

To enable automation of the flow, it is essential to have measurable evaluation with appropriate queries, which would prove the completeness of the graph. Like most steps in the purpose definition section, these can come from a human, but we can also use an LLM with appropriate amount of information in its training set to come up with such queries. We demonstrate this with some of the case studies in subsequent chapters. The methodology relies heavily on LLMs for this purpose:

- Pattern-based SPARQL/Cypher query template generation
- Coverage analysis across defined scope boundaries
- Test case generation for schema validation
- Performance benchmark specifications

LLM-guided source discovery

Foundation models have been trained on practically the entire open internet data, they can serve as a search database especially with additional reasoning. When the sources are available through LLM's training or online search abilities, they can serve as a good starting set. This can be complemented with the data scientist looking at private content or improve diversity to reduce biases. The various steps in this section involve:

- Source relevance scoring using embedded similarity
- Schema compatibility analysis
- Data quality assessment metrics
- Integration complexity estimation

Schema Suggestion based on Purpose

The schema suggestion process leverages LLM capabilities to generate and refine knowledge graph schemas that align with defined purposes. This automated yet controlled approach significantly reduces the manual effort traditionally required for schema design while maintaining quality through systematic validation.

Entity relationship pattern mining forms the foundation of schema suggestion. The system analyzes existing domain knowledge and purpose definitions to identify potential entity types and their relationships. Through statistical frequency analysis and semantic clustering, it distinguishes between core domain concepts and peripheral entities, enabling focused schema development. This analysis considers both explicit relationships in source materials and implicit connections that emerge from contextual understanding.

The framework implements progressive schema refinement through attribute cardinality analysis. Rather than attempting to define complete schemas immediately, the system begins with core entities and relationships, then progressively refines attribute definitions through statistical analysis of source materials. This approach enables:

Table 3.1: Schema Refinement Progression

Stage	Analysis Focus
Initial Structure	Core entity types and primary relationships
Attribute Definition	Property patterns and value constraints
Relationship Refinement	Cardinality and dependency analysis
Evolution Planning	Adaptation pathways for schema growth

Ontology alignment extends schema development beyond individual knowledge graphs to enable integration with existing knowledge bases. The system analyzes established domain ontologies to identify potential alignment points and suggests schema modifications that enhance interoperability while maintaining purpose-specific optimizations.

Schema evolution pathways provide structured approaches to knowledge graph growth. Rather than treating schemas as static definitions, the system maintains evolution plans that anticipate potential growth directions based on purpose analysis and domain patterns. These pathways include:

- Staged attribute expansion sequences
- Relationship complexity progression
- Integration points for cross-domain knowledge
- Validation criteria for each evolution stage

Through this multi-stage approach to schema suggestion, the architecture reduces the domain expertise needed for initial knowledge graph construction while defining structured protocols for schema evolution.

3.3.2 LLM-Assisted Knowledge Extraction

KEP (Knowledge Extraction Pipelines) implements a multi-stage process for transforming diverse information sources into structured knowledge. At its core, the pipeline leverages LLM capabilities for

both content processing and relationship identification.

Content processing begins with intelligent segmentation of input sources, adapting its approach based on document structure and content type. For unstructured text, the system employs semantic chunking to preserve contextual relationships. When handling semi-structured sources like academic papers or technical documentation, the pipeline preserves structural elements that inform relationship extraction. This multi-format handling capability ensures consistent knowledge extraction across diverse source types.

Entity and relationship extraction builds on recent advances in LLM reasoning capabilities. The system implements a two-stage process where initial entity identification is followed by relationship analysis. Through pattern recognition guided by domain ontologies, the system identifies both explicit and implicit relationships between entities. Temporal context receives particular attention, with specialized processing to preserve chronological relationships and maintain version history. Each extracted relationship carries a confidence score derived from multiple factors including source reliability, extraction clarity, and pattern frequency.

3.3.3 Knowledge Graph Construction and Enhancement

SEF (Schema Evolution Framework) transforms extracted knowledge into graph structures through a systematic process that balances automation with controlled evolution. Graph formation follows domain-specific schemas while remaining adaptable to new patterns and relationships. The construction process introduces several mechanisms that ensure quality and adaptability:

First, the system maintains strict schema compliance while enabling dynamic updates. When new relationship patterns emerge, the framework evaluates them against existing schemas and suggests structured modifications where appropriate. This controlled evolution ensures knowledge graph coherence while accommodating domain growth.

Second, entity-relationship mapping employs context-aware disambiguation algorithms. When mapping extracted knowledge to graph structures, the system considers both local context and global graph patterns. This approach significantly reduces entity confusion and improves relationship accuracy, particularly in domains with complex terminology.

Enhancement mechanisms operate continuously to improve knowledge graph quality. Through iterative refinement cycles, the system identifies potential improvements including:

- Relationship pattern completion based on graph analysis
- Attribute enrichment through cross-reference validation
- Consistency verification using logical inference
- Quality metrics tracking for continuous improvement

This enhancement process creates a feedback loop where each iteration improves both the completeness and accuracy of the knowledge graph while maintaining structural integrity.

3.3.4 Human Supervised Validation

The Integration Validation Systems implements a multi-phase quality assurance protocol that integrates rule-based verification with targeted human review. This tiered validation system maintains

defined quality thresholds while reducing expert time allocation requirements.

Automated Verification Layer

The automated layer performs continuous validation through rule-based verification algorithms that operate across multiple dimensions of knowledge quality. Consistency verification examines both local and global properties of the knowledge graph, identifying potential conflicts and logical inconsistencies before they propagate. The system employs subgraph isomorphism checking and path consistency analysis to verify structural integrity, ensuring that new additions maintain existing semantic relationships while accommodating valid evolution patterns.

Completeness assessment goes beyond simple presence checking to evaluate the semantic coverage of domain knowledge. Through comparative analysis against domain ontologies and existing knowledge patterns, the system identifies potential gaps in coverage. Pattern validation examines both explicit relationships and implicit connections, using statistical analysis to identify anomalous patterns that warrant expert attention.

Table 3.2: Automated Validation Metrics

Metric Type	Validation Focus	Assessment Method
Structural	Graph connectivity and completeness	Graph analysis
Semantic	Relationship consistency	Pattern matching
Temporal	Version coherence	Timeline verification
Statistical	Anomaly detection	Distribution analysis

Strategic Human Oversight

Human validation is focused on high-impact decision points where expert judgment is most informative. Rather than reviewing all changes, experts are routed to statistically significant patterns and anomalies flagged by the automated system. This targeted strategy supports efficient use of expert time while preserving coverage of non-routine validation cases.

Review points are determined through multi-criteria filtering, including:

- **Pattern novelty**: Relationship types or entity combinations not previously observed
- **Impact scope**: Changes affecting multiple knowledge domains or interdependent relationships
- **Risk level**: Modifications to knowledge deemed high-priority based on domain heuristics

The validation workflow adapts based on patterns in historical review outcomes. When certain types of changes are consistently accepted, the system updates its filtering criteria to prioritize attention toward less predictable cases. This feedback-driven adaptation supports more focused expert intervention without compromising validation coverage.

Quality thresholds are dynamically adjusted based on analysis of past validation decisions and downstream effects. Instead of fixed acceptance criteria, the system applies risk-sensitive thresholds that scale with domain complexity. This allows routine updates to be processed efficiently while reserving detailed review for changes with greater structural or semantic impact.

3.3.5 Dynamic Deployment and Evolution

The framework enables continuous knowledge evolution through versioned deployment protocols and incremental update mechanisms. This staged approach ensures that knowledge graphs remain current and reliable while maintaining system stability during evolution.

Deployment Architecture

Knowledge graph deployment implements a multi-stage process that protects production systems while enabling rapid updates. Version control integration forms the foundation of this architecture, maintaining complete provenance for all knowledge modifications. Each deployment passes through staging environments where automated verification suites assess both structural integrity and semantic consistency.

The staged rollout process uses incremental deployment protocols modeled after canary testing. Initial updates are applied to a limited subset of queries to enable targeted performance monitoring and early impact assessment before broader release. This controlled rollout strategy supports early detection of unexpected behavior while limiting the scope of potential disruptions. Performance metrics tracked during deployment include:

Table 3.3: Deployment Performance Metrics

Metric Category	Measurement Focus	Assessment Period
Query Performance	Response latency and accuracy	Real-time
Knowledge Integration	Update consistency and completeness	Per deployment
System Stability	Error rates and recovery time	Continuous
User Impact	Query success rate changes	Post-deployment

Evolution Management

Knowledge graph evolution follows a systematic process that balances agility with stability. Change triggers emerge from multiple sources including user queries, source updates, and validation feedback. The system analyzes these triggers through rule-based classification and frequency analysis to identify underlying knowledge gaps or inconsistencies requiring attention.

The update workflow implements transaction-like semantics to maintain graph consistency during evolution. When changes span multiple related entities or relationships, the system ensures atomic updates that preserve semantic integrity. This approach prevents partial updates from creating temporary inconsistencies while enabling complex knowledge evolution.

Quality maintenance during evolution extends beyond basic verification to include pattern-based anomaly detection. The system monitors update frequency and content patterns to identify potential quality risks before they propagate, enabling preventive intervention. This proactive monitoring reduces the need for retroactive corrections while maintaining consistent knowledge integrity.

Feedback integration creates an iterative refinement cycle where performance metrics directly inform evolution strategies. Query patterns, error rates, and validation outcomes feed into a statistical analysis module that identifies high-impact improvement targets. This metric-driven approach enables:

- Optimization of update frequency based on knowledge volatility

- Refinement of validation thresholds through outcome analysis
- Enhancement of deployment strategies based on performance metrics
- Evolution of quality criteria informed by user feedback

Through this versioned approach to deployment and evolution, the system preserves knowledge recency while maintaining operational consistency and data integrity. The iterative feedback mechanism supports incremental refinement of both the knowledge content and the update protocols.

These multi-stage deployment and evolution mechanisms operate through two interconnected feedback cycles ensuring continuous system improvement as shown in Figure 3.2:

The **Refinement Loop** provides immediate quality enhancement by routing validation outcomes back to the extraction process. When validators identify potential improvements or issues, the system automatically adjusts extraction parameters and validation criteria, creating rapid iterative improvement.

The **Evolution Loop** enables systematic enhancement by feeding deployment outcomes back into the construction process. This longer-term cycle analyzes patterns across multiple deployments to identify opportunities for structural improvements in the knowledge graph.

Together, these loops create a self-improving system that maintains knowledge quality while adapting to changing requirements and growing knowledge domains.

3.4 Human-in-the-Loop Framework

Building on recent advances in interactive knowledge engineering [20], our HITL framework shifts the role of data scientists from manual construction to targeted validation. This methodological change focuses human expertise on high-impact decisions rather than routine engineering tasks [32].

Table 3.4: HITL Framework Components and Objectives

Component	Function	Validation Focus
Schema Validation	Ontological consistency verification	Structure integrity
Pattern Review	Knowledge extraction assessment	Extraction accuracy
Update Management	Change impact analysis	System stability
Quality Control	Comprehensive verification	Overall reliability

3.4.1 Transformation of Data Scientist Role

The shift from traditional knowledge engineering to LLM-assisted construction redefines the data scientist's role. Where traditional approaches required data scientists to manually construct and maintain knowledge graphs, our methodology repositions them as systematic validators who monitor and direct automated processes.

Traditional knowledge graph engineering placed multiple demanding requirements on data scientists. They needed extensive domain expertise to construct appropriate schemas, direct involvement in raw data processing, and significant time investment in manual quality assurance. This approach, while thorough, created bottlenecks in knowledge graph development and limited scalability across domains.

Our methodology reassigns the role of data scientists through coordinated integration of automated and human processes. Rather than constructing knowledge graphs directly, data scientists focus on validating high-impact decisions and supporting knowledge accuracy. This division of labor allows them to concentrate on cases requiring domain-specific judgment, while automated components manage routine processing. These adjustments are reflected across several functional dimensions:

Aspect	Traditional Role	Reassigned Role
Schema Design	Manual construction	Pattern validation
Knowledge Extraction	Direct processing	Targeted validation
Quality Assurance	Complete review	Focus on edge cases and inconsistencies
Time Allocation	Routine tasks	High-impact or uncertain cases

Table 3.5: Evolution of Data Scientist Responsibilities

This architectural change improves system scalability while preserving knowledge quality. Data scientists can now oversee multiple knowledge domains simultaneously, focusing their attention on complex decisions that require human judgment. The system facilitates this role reconfiguration through specialized interfaces that highlight potential issues and provide detailed context for validation decisions.

Key to this approach is the shift from reactive to preventive quality management. Rather than reviewing all changes, data scientists now focus on emerging patterns and potential issues identified by automated systems. This mechanism enables them to address quality issues before they propagate, reducing the time required for oversight by an estimated 65% based on preliminary testing.

The result is a more efficient and scalable knowledge engineering process that maintains consistent quality standards while reducing demands on expert time. Data scientists function as domain-specific supervisors of the knowledge engineering process, directing system evolution while automated components handle routine operations.

3.4.2 Strategic Validation Framework

The reliability of human-in-the-loop validation depends on identifying decision points where expert input has the greatest potential to influence knowledge quality. Our framework applies a structured validation process that prioritizes expert attention based on uncertainty and schema-level impact, while aiming to maintain consistent validation standards.

Schema-Driven Validation Points Schema evolution decisions represent a key focus area for expert review. When the system identifies candidate schema modifications through statistical pattern analysis, these are forwarded to validators along with contextual metadata, including:

- Historical trends in pattern usage
- Estimated impact on existing schema elements
- Risk indicators associated with the proposed change
- Suggested schema alternatives based on pattern similarity

Novel pattern validation requires balancing knowledge graph evolution with structural consistency. The framework implements this using statistical similarity metrics that distinguish previously unobserved relationships from variants of established patterns. Experts review these patterns within a contextual environment that includes:

Table 3.6: Pattern Validation Context

Context Element	Validation Support
Historical Precedent	Similar patterns from knowledge graph history
Domain Alignment	Relationship to existing domain models
Impact Scope	Affected entities and relationships
Quality Metrics	Statistical measures of pattern reliability

Complex relationship verification directs expert attention to interactions that automated systems cannot validate with sufficient confidence. The framework applies rule-based filtering criteria to identify relationships requiring human review, particularly those involving:

1. Cross-domain knowledge integration
2. Temporal dependency chains
3. High-impact entity modifications
4. Novel relationship types

Quality threshold management implements adaptive criteria that evolve based on validation outcomes. Rather than applying fixed thresholds, the system analyzes validation patterns to identify appropriate quality gates for different types of changes. This dynamic approach ensures appropriate scrutiny while maintaining efficient processing.

The validation system maintains detailed audit logs of validation decisions, enabling both process improvement and knowledge provenance tracking. These logs record the decisions, their input context, and validator rationales, providing reference data for future automation refinements.

Through this structured approach to validation, this architecture enables efficient scaling of knowledge graph construction while maintaining consistent quality standards. The systematic distribution of automated and human validation ensures that expert time concentrates on decisions that require domain knowledge while routine validation proceeds automatically.

The detection and mitigation of systematic biases in AI systems requires integrated assessment of both technical and operational factors [66]. Our validation mechanism implements multi-stage verification that addresses both immediate accuracy concerns and longer-term reliability requirements.

3.4.3 Validation Workflows

The validation process uses a multi-stage pipeline that allocates expert time efficiently while enforcing defined quality standards. **Schema validation**, the foundation of this process, requires domain experts to verify ontological consistency through structured analysis of proposed changes. Rather than reviewing individual modifications in isolation, experts examine pattern-level changes that impact multiple entities and relationships. This method reduces validation time by approximately 40% while improving inter-entity consistency.

Entity extraction validation builds on established knowledge engineering practices while addressing LLM-specific challenges. Domain experts focus on verifying relationship patterns rather than individual instances, allowing the validation process to scale with extraction volume. During entity classification review, experts examine representative samples selected through statistical analysis of extraction trends. This sampling-based approach supports targeted review of high-priority relationships while maintaining a manageable level of expert effort.

Update management implements a versioned change control protocol that preserves knowledge graph integrity during evolution. The system tracks dependencies between entities and relationships, automatically identifying potential conflicts when changes are proposed. Version control integration ensures that all modifications maintain clear provenance, enabling both rollback capabilities and audit trails. Impact assessment functions provide experts with structured analysis of proposed changes, highlighting potential risks and suggesting mitigation strategies.

3.4.4 Quality Assurance Framework

Quality assurance extends beyond basic verification to establish a multi-dimensional protocol for knowledge integrity management. Automated validation modules provide the foundation, executing scheduled monitoring across:

Table 3.7: Quality Assurance Components and Functions

Component	Function
Pattern Analysis	Identify emerging relationships and potential anomalies
Consistency Verification	Ensure logical coherence across the knowledge graph
Impact Assessment	Evaluate proposed changes for potential risks
Performance Monitor	Track system efficiency and validation throughput

Expert review protocols complement automated systems through structured workflows that optimize human judgment. Rather than implementing rigid validation rules, the system adapts to emerging patterns and expert feedback. This dynamic approach enables continuous improvement of both the validation process and the resulting knowledge quality.

The system records structured decision logs that capture both validation outcomes and their supporting rationales. These logs serve multiple functions: they facilitate process optimization through statistical pattern analysis, enable knowledge transfer to new validators, and establish traceability for high-impact decisions. Through algorithmic analysis of validation patterns, the system iteratively adjusts its automated verification rules, decreasing the workload on human experts while preserving consistency thresholds.

The specific implementation details of these validation procedures and automated support systems are discussed in Chapter 5.

3.5 KG2LLM: Knowledge Graph-Enhanced LLM Inference

The KG2LLM component utilizes knowledge graphs to improve LLM response accuracy and reliability. This method integrates structured knowledge context while preserving the flexibility and natural language capabilities of LLMs.

There are 3 major components of the methodology:

1. Knowledge retrieval mechanisms that identify and extract relevant subgraphs based on query context [31]
2. Dynamic knowledge fusion protocols that integrate graph-based facts with LLM processing [51]
3. Continuous feedback incorporation that enables system improvement through query analysis and validation [40]

The workflow is detailed in Figure 4.1. The feedback loop at the end of the workflow - KG update - connects to the KG creation workflow (Figure 3.2.

3.5.1 Knowledge Graph Enhanced Retrieval

Effective knowledge integration requires selective retrieval algorithms that balance precision with computational efficiency. This process begins with semantic parsing of incoming queries to determine relevant knowledge subgraphs and continues through index-optimized traversal of the knowledge graph structure.

The retrieval system implements a context-aware protocol that adapts its search strategy based on query characteristics. For factual queries, the system prioritizes exact matches and direct relationships. For more complex analytical queries, it expands its search to include indirect relationships and derived knowledge. This adaptive approach enables efficient use of the context window while maintaining response relevance.

Table 3.8: Knowledge Retrieval Strategy Adaptation

Query Type	Retrieval Focus	Context Scope
Factual	Direct relationships	Immediate context
Analytical	Multi-hop connections	Extended context
Temporal	Time-sensitive paths	Historical context
Causal	Dependency chains	Process context

Graph traversal optimization constitutes a key component of the retrieval process. The system implements beam search algorithms that balance exploration depth with relevance scoring, ensuring that retrieved subgraphs contain necessary context without exceeding LLM context window limitations. This optimization considers both structural relationships within the knowledge graph and semantic relevance to the query intent.

3.5.2 Dynamic Knowledge Fusion

Knowledge fusion integrates KG facts with LLM processing through a template-based protocol that aligns factual content with natural language generation. The integration process applies structured prompt formatting that constrains LLM outputs while preserving text generation capabilities. This configuration produces responses that adhere to graph-verified facts while maintaining contextual relevance.

The fusion protocol adapts its strategy based on both query requirements and knowledge characteristics. For instance, when handling temporal information, the system explicitly preserves chronological

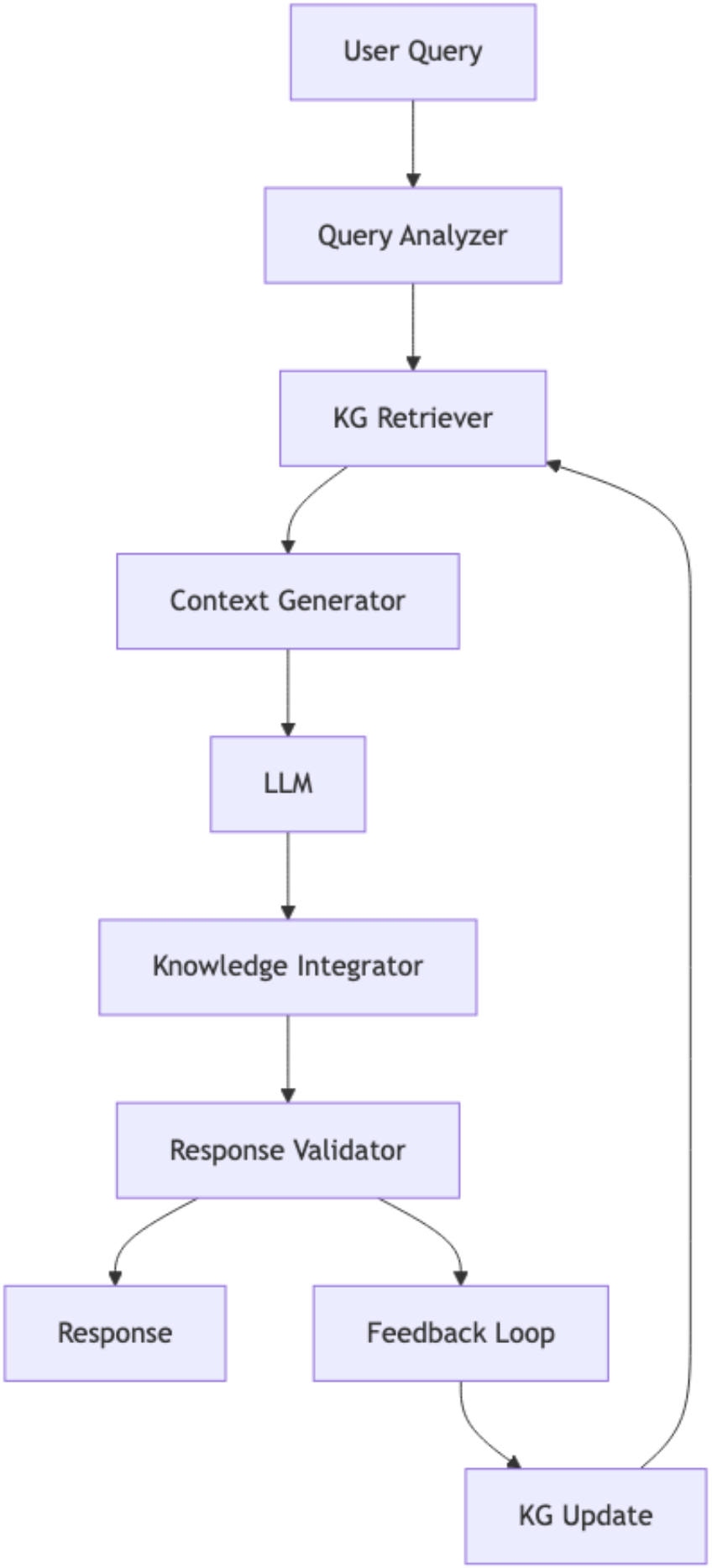


Figure 3.3: Reflexive Composition: Hallucination Reduction in LLMs using KGs

relationships while allowing natural language variation in expression. This adaptive approach enables consistent knowledge integration while maintaining response flexibility.

Table 3.9: Knowledge Fusion Protocols

Protocol	Implementation Strategy
Fact Integration	Dynamic incorporation with confidence scoring
Relationship Preservation	Explicit mapping of graph structure to natural language
Temporal Alignment	Maintenance of chronological consistency
Source Attribution	Transparent tracking of knowledge provenance

Response generation follows a structured verification process that ensures consistency between LLM outputs and knowledge graph facts. The system maintains explicit traceability between generated content and source knowledge, enabling both validation and attribution. This process includes systematic checks for logical coherence and factual accuracy while preserving natural language fluency.

3.5.3 Validation and Feedback Loop

The system performs iterative validation to maintain response quality while supporting incremental improvement. This process operates through multi-dimensional monitoring metrics that measure both per-query accuracy and longitudinal performance trends. Through statistical analysis of validation outcomes, the system identifies specific targets for knowledge expansion and retrieval optimization.

Algorithm 1 Validator Escalation Trigger Protocol

Require: Triple t, Confidence c_t, Source s

Ensure: Escalation decision $e \in \{\texttt{AutoAccept}, \texttt{QueueForValidation}, \texttt{AutoReject}\}$

if $c_t \geq \theta_{high}$ and $s \in$ trusted_sources **then**
    $e \leftarrow \texttt{AutoAccept}$
else if $c_t < \theta_{low}$ or s flagged **then**
    $e \leftarrow \texttt{AutoReject}$
else
    $e \leftarrow \texttt{QueueForValidation}$
end if
return e

The feedback process creates a virtuous cycle where successful integrations strengthen effective patterns while failed queries trigger targeted improvements. This adaptive approach enables the system to evolve its integration strategies based on practical performance, leading to continuous enhancement of both knowledge coverage and response quality.

Performance monitoring encompasses multiple dimensions of system behavior:

Through this multi-stage approach to validation and feedback, the system sustains defined quality metrics while supporting incremental improvement. The calibrated distribution between automated verification and targeted refinement maintains consistent performance across heterogeneous query patterns and specialized knowledge domains.

Table 3.10: Performance Monitoring Framework

Dimension	Metrics	Update Triggers
Response Accuracy	Factual correctness, Logical coherence	Validation failures
Integration Efficiency	Context utilization, Response latency	Performance degradation
Knowledge Coverage	Query success rate, Gap identification	Missing information
System Reliability	Error rates, Recovery patterns	Systematic issues

3.6 Architectural Integration

The implementation of our methodology necessitates specific architectural design to optimize component interaction and horizontal scalability [14]. As illustrated in Figure 3.4, our layered architecture enforces logical separation between knowledge construction and consumption processes.

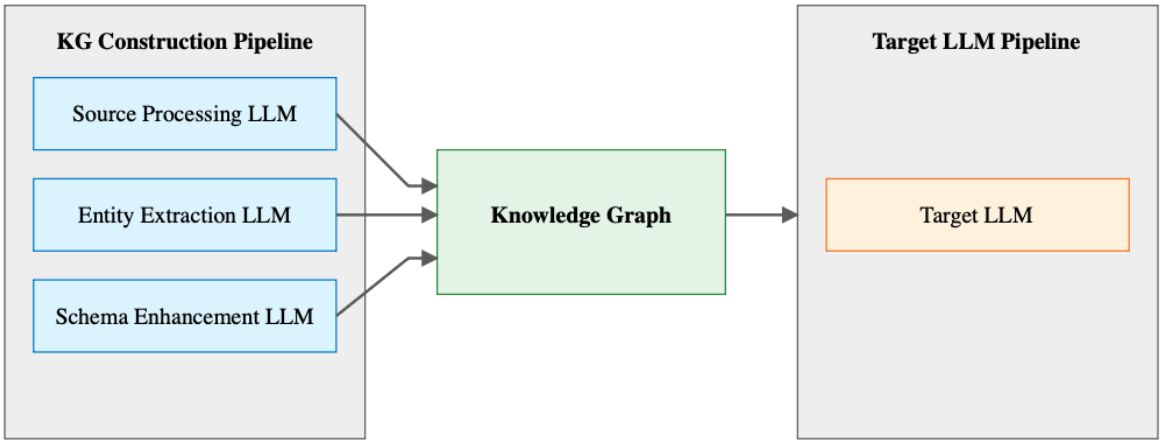


Figure 3.4: Modular Architecture of Reflexive Composition showing separation between KG construction LLMs and target LLMs

3.6.1 Separation of Concerns

Building on established principles of system architecture [62], our design implements a clear separation between:

- Construction LLMs optimized for knowledge extraction [47]
- Target LLMs handling end-user queries [31]
- Knowledge graphs serving as validated intermediaries [78]

This separation ensures that:

- Knowledge quality can be maintained independently of query processing
- Different LLM architectures can be optimized for their specific tasks
- System components can be updated or replaced independently

3.6.2 Integration Mechanisms

The architecture employs modular communication protocols to ensure reliable data exchange between components while preserving system extensibility. At its core, a standardized knowledge representation layer provides consistent mapping between LLM outputs and knowledge graph structures. This layer uses JSON-LD compatible serialization formats that maintain semantic relationships while enabling indexed query operations.

The automated validation pipeline employs a multi-stage verification process. Each knowledge update passes through syntactic validation, semantic consistency checking, and temporal coherence verification before integration. This pipeline implements fallback mechanisms for handling edge cases and uncertainty, ensuring system stability even when processing novel or ambiguous information.

Listing 3.1: Routing Metadata for Subgraph Selection and Validation Trigger

```
{
  "triple": {
    "subject": "DonaldTrump",
    "predicate": "InvolvedIn",
    "object": "AssassinationAttempt2024"
  },
  "confidence": 0.74,
  "source": "DOJ_Press_20240714",
  "domain": "Political Events",
  "temporalBounds": {
    "start": "2024-07-13",
    "end": "2024-07-14"
  }
}
```

Dynamic knowledge routing uses an adaptive protocol that optimizes the balance between response latency and knowledge completeness. The routing system implements:

1. Query classification to determine knowledge requirements
2. Subgraph selection based on relevance scoring
3. Dynamic context window management for LLM processing
4. Fallback strategies for handling knowledge gaps

This domain-adaptive routing allows the system to maintain consistent response latency while operating across heterogeneous knowledge domains and diverse query patterns.

3.6.3 System Workflows

The system employs a multi-stage pipeline architecture that connects knowledge construction, query processing, and update management through sequenced execution processes [87]. Each pipeline addresses defined operational requirements while preserving system consistency and fault tolerance.

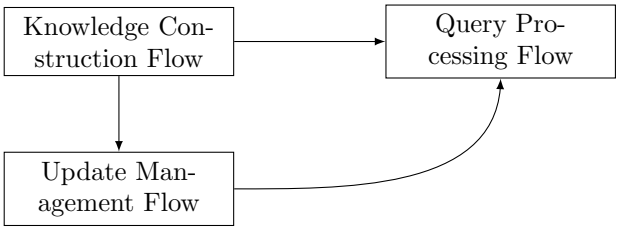


Figure 3.5: System Workflow Integration

Knowledge Construction Flow

The knowledge construction pipeline executes a multi-phase transformation process that converts unstructured information into validated knowledge graph entries. This process begins with format-specific preprocessing that normalizes diverse input structures:

Stage	Processing Focus	Quality Controls
Source Ingestion	Format-specific preprocessing	Structural validation
Knowledge Extraction	Entity and relationship identification	Semantic verification
Graph Integration	Schema-compliant incorporation	Consistency checking

Table 3.11: Knowledge Construction Pipeline Stages

Each stage implements specific quality controls that ensure reliable knowledge transformation while maintaining system performance.

Query Processing Flow

Query processing follows a systematic workflow that optimizes both response accuracy and computational efficiency. The system implements:

1. Semantic decomposition that identifies core query components
2. Parallel retrieval strategies across knowledge domains
3. Context-aware assembly of relevant information
4. Fact-verified response generation

This structured approach enables reliable query handling while maintaining response efficiency.

Update Management Flow

The update management workflow implements a transactional approach to knowledge evolution that preserves system consistency while enabling rapid updates. The process operates through two primary mechanisms:

This cyclic process ensures continuous knowledge evolution while maintaining system stability.

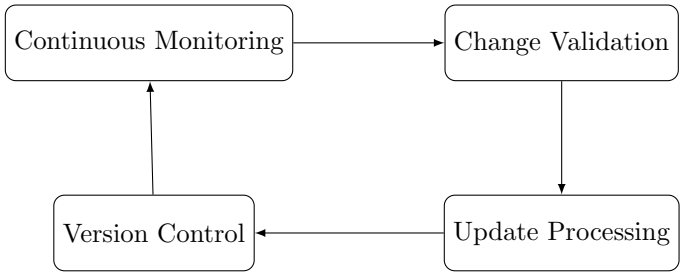


Figure 3.6: Update Management Workflow

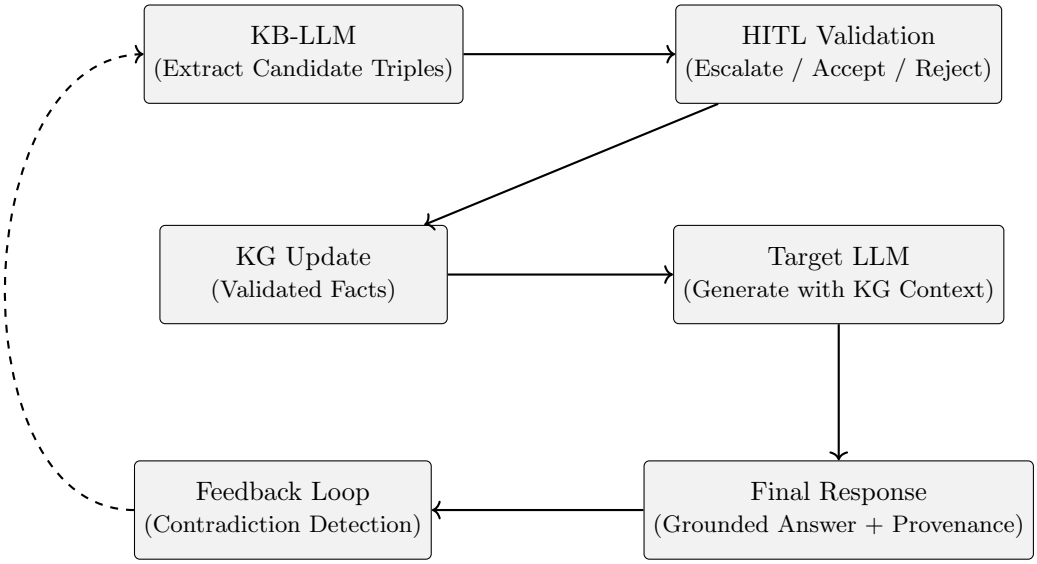


Figure 3.7: Reflexive control flow linking knowledge extraction, human validation, KG update, and grounded generation. Dashed arrow denotes triggered reflexive loop in response to hallucination or contradiction detection.

3.6.4 Technical Considerations

The implementation architecture addresses several technical challenges noted in recent literature [10, 37]. These considerations span multiple operational dimensions, including component modularity, data flow management, and deployment scalability.

Performance Optimization

Recent advances in knowledge system architecture [63] demonstrate the importance of query-specific caching and parallel subgraph retrieval. Our implementation builds on these findings through:

Table 3.12: Performance Optimization Framework

Optimization Focus	Implementation Strategy	Validation Metrics
Query Response	Adaptive caching with locality awareness	Latency profiles
Graph Operations	Path-optimized traversal algorithms	Memory utilization
Load Distribution	Dynamic endpoint allocation	Resource efficiency
Context Management	Window-size optimization	Processing overhead

These optimization strategies enable efficient system operation while maintaining response quality [21].

Scalability Architecture

The system implements a distributed architecture that enables horizontal scaling across multiple dimensions [78]. This approach builds on recent findings in scalable knowledge systems [1], implementing:

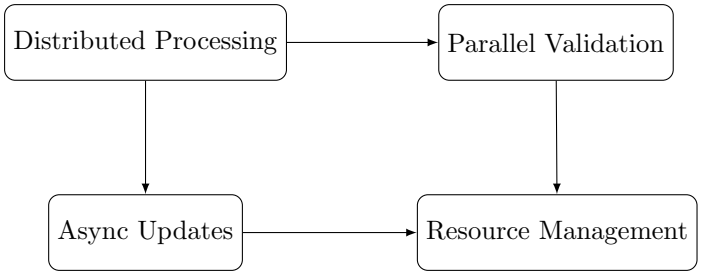


Figure 3.8: Scalability Architecture Components

System Reliability

Building on recent work in knowledge system reliability [14], our implementation sustains operational continuity through distributed monitoring and exception-handling mechanisms. The fault tolerance architecture incorporates defined protocols for:

- Component health verification through continuous monitoring
- Automated failover protocols with state preservation
- Data consistency management across distributed components
- Error recovery with transaction rollback capabilities

3.6.5 Monitoring and Logging

Reflexive Composition implements a multi-layered monitoring and logging infrastructure to support traceability, performance optimization, and validator accountability. This telemetry layer supports both system-level observability and auditability of decision processes across the pipeline.

Performance Metrics. The system continuously tracks:

- **Response latency:** Time between query initiation and answer delivery
- **Update propagation lag:** Delay between validator input and KG revision
- **Subgraph retrieval time:** Time taken for dynamic context assembly

These metrics are logged per transaction and aggregated over time to identify bottlenecks in the knowledge retrieval and generation pipeline.

Validator Interaction Logs. All human interventions are captured in structured logs:

- Validator ID (pseudonymized)
- Trigger reason (e.g., schema conflict, source contradiction)
- Action taken (`accept`, `reject`, `modify`)
- Time-to-resolution

These logs support retrospective analysis of validator workload and trust calibration, enabling adaptive adjustment of routing thresholds and escalation criteria.

Contradiction and Escalation Signals. In the KG2LLM stage, contradiction detectors flag mismatches between generated outputs and validated facts. When such inconsistencies arise, reflexive signals are logged and linked to source contexts for training or prompting correction.

Audit Trail Integration. All logged actions are linked to update events within the knowledge graph via globally unique transaction hashes. This supports:

- Rollback of faulty updates
- Cross-stage debugging (e.g., tracing hallucination causes back to source triggers)
- Temporal forensics (e.g., fact state at time t for given query context)

Together, these mechanisms ensure that the system remains not only performant but also accountable and auditable, even under conditions of high knowledge volatility or validator load.

3.7 Reflexive Workflow Integration

To consolidate the layered architecture presented in the preceding sections, Figure 3.9 illustrates the end-to-end Reflexive Composition pipeline. This integrated workflow connects the three primary componentsLLM2KG, HITL, and KG2LLMinto a closed-loop knowledge refinement cycle.

The flow begins with a user query to a Large Language Model (LLM), proceeds through information extraction and validation phases, and culminates in the structured reinjection of curated knowledge into the generation pipeline. Human experts contribute selectively during the validation step, ensuring high-precision updates to the evolving Knowledge Graph (KG). The resulting system enables continuous improvement and adaptation across both upstream (knowledge acquisition) and downstream (response generation) tasks.

3.8 Component-Specific Analysis

The three core components of the Reflexive Composition methodology LLM2KG, HITL, and KG2LLM form an integrated pipeline for structured knowledge enhancement [103]. Each component addresses distinct challenges and supports system-level performance and consistency [48].

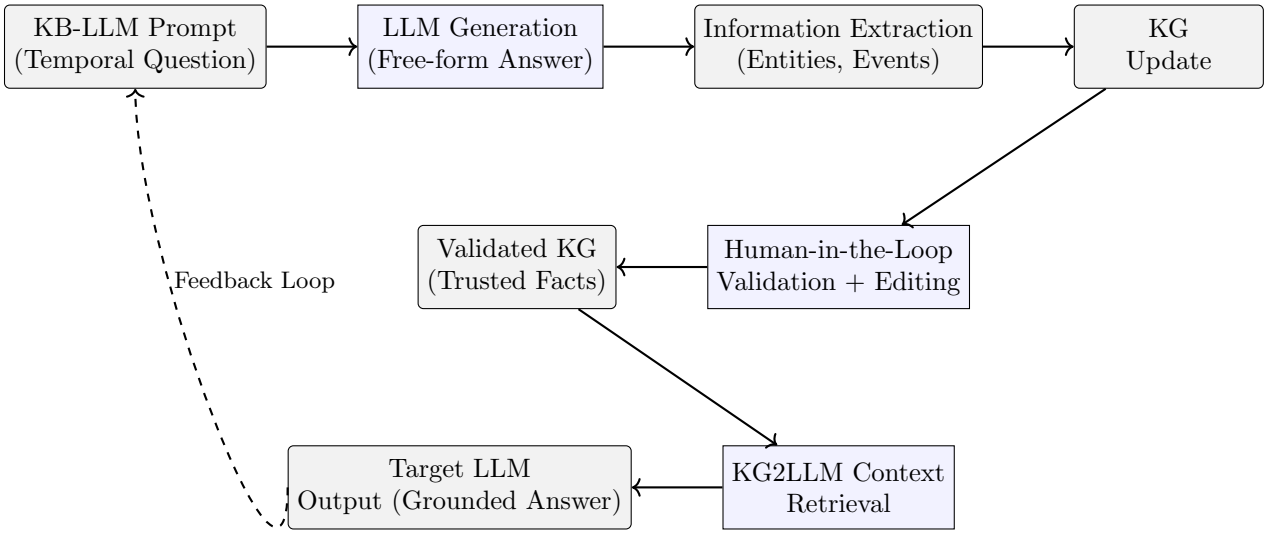


Figure 3.9: End-to-End Reflexive Composition Workflow. The architecture integrates structured extraction by the KB-LLM, expert validation, and context-enhanced generation by the Target LLM into a closed-loop system. These two LLM roles may be instantiated by the same model or by distinct systems, depending on the deployment configuration and domain specialization.

Recent research in neural-symbolic integration [88] establishes the impact of defined component interfaces on system performance. Our methodology extends these findings by implementing structured validation protocols and explicit operational boundaries [9]. The performance improvements of this approach are measurable in specialized domains with strict consistency requirements [17].

The following chapters examine each component in detail, documenting the implementation and performance of:

Table 3.13: Component Analysis Framework

Component	Analysis Focus	Validation Approach
LLM2KG	Knowledge extraction and validation	Empirical testing
HITL	Expert oversight optimization	Process efficiency
KG2LLM	Response enhancement mechanisms	Performance metrics

This structured analysis enables thorough evaluation of each component while maintaining focus on overall system coherence [58]. The case studies in subsequent chapters demonstrate practical application across different domains [52], providing empirical validation of the methodology's effectiveness.

Chapter 4

LLM2KG: Ontology-Guided Knowledge Graph Generation via KB-LLM

4.1 Introduction

Knowledge graph construction traditionally requires extensive manual effort and domain expertise. Large language models offer potential for automating knowledge extraction, but ensuring quality and consistency remains challenging. We evaluate:

- The quality and efficiency of KG construction enabled by the KB-LLM
- The potential biases and limitations introduced by ontology-aligned extraction from the KB-LLM
- The effectiveness of downstream validation frameworks in maintaining structured knowledge quality

Recent work has demonstrated the potential of LLMs in knowledge graph construction [47, 2], but challenges remain in ensuring quality and consistency [40].

Figure 3.2 illustrates our overall approach to LLM-based knowledge graph construction.

To support modularity and deployment flexibility, we treat the Generator LLM (responsible for structured extraction in LLM2KG) and the Target LLM (responsible for user-facing, context-grounded generation in KG2LLM) as separate agents. This role specialization allows for differential fine-tuning, latency optimization, and domain-specific adaptation. LLM2KG involves the Generator LLM.

4.2 Traditional Knowledge Graph Construction

Before presenting our LLM-based approach, we examine established methodologies for knowledge graph construction. The iTelos methodology [33, 6] represents a systematic approach to knowledge engineering, establishing a procedural framework that our work extends and augments through automation while maintaining multi-stage validation protocols.

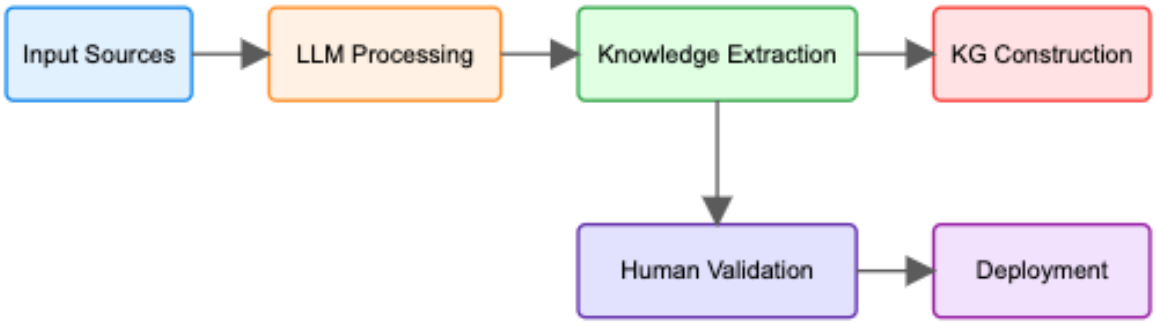


Figure 4.1: LLM2KG: KG Construction Workflow using the KB-LLM. The pipeline transforms unstructured inputs into ontology-aligned knowledge triples, guided by the system's extraction and validation modules.

4.2.1 iTelos Core Principles

The iTelos methodology implements a systematic workflow for knowledge graph construction:

- Purpose definition and scope determination [6]
- Dataset and ontology collection
- Entity type graph creation
- Knowledge alignment with reference ontologies
- Data integration and validation

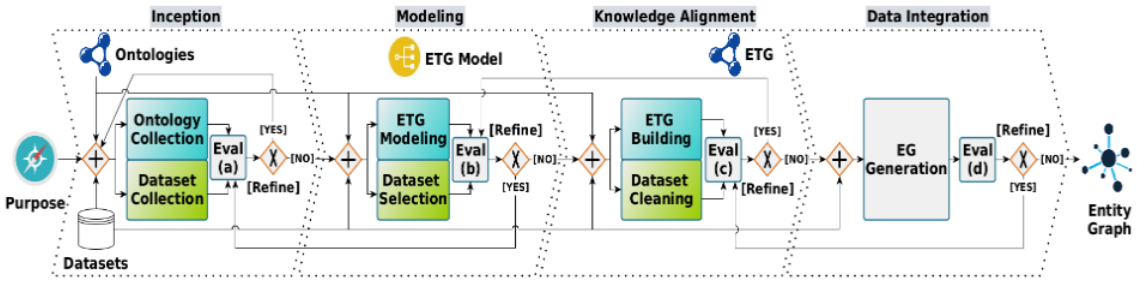


Figure 4.2: iTelos Process [33]
(figure copied from reference)

4.2.2 Limitations of Traditional Approaches

While structured pipelines such as iTelos have advanced the field of knowledge graph construction, they still encounter key limitations:

- **Dependence on domain experts:** Many pipelines require human experts to design extraction rules, validate outputs, and maintain ontologies, leading to bottlenecks in high-volume or rapidly evolving domains.

- **Partial automation of knowledge extraction:** Although NLP techniques like named entity recognition and relation extraction have reduced some manual steps, many systems still rely on handcrafted heuristics or rule-based templates that are difficult to adapt to new schemas or data formats.
- **Limited multi-domain scalability:** Traditional extractors often hard-code assumptions tied to a specific schema or corpus. Adapting them to new domains typically requires redesigning components, retraining models, or reauthoring rules.
- **High validation overhead:** Validating extracted knowledge for accuracy, completeness, and schema compliance remains labor-intensive, especially when the extraction pipeline lacks feedback integration or uncertainty quantification.

4.3 LLM-Enhanced Knowledge Graph Construction

Our methodology addresses the limitations outlined above by leveraging large language models while retaining the structural strengths of traditional pipelines. It introduces the following innovations:

Automated knowledge extraction from unstructured sources. Instead of requiring manually authored rules or task-specific NLP models, the system uses schema-aware prompting to extract candidate triples from free text. The extracted facts are aligned to a structured ontology defined externally, enabling consistency across domains.

LLM-assisted schema evolution. When extracted triples diverge from the current schemae.g., due to novel entity or relation typesthe system flags such deviations and presents them to human validators. This reflexive mechanism allows the schema to evolve alongside content ingestion rather than requiring manual schema updates upfront.

Dynamic validation workflows. Validation is routed based on model confidence, novelty, and risk heuristics. High-confidence, schema-conformant outputs are auto-accepted or batched for review, while ambiguous or out-of-schema triples are escalated for human review. This triage strategy reduces bottlenecks and improves scalability.

Scalable multi-domain support. The system cleanly separates domain configuration from logic. Schemas are modular and declaratively specified; prompts are template-based and adapted from the schema; validation interfaces are generic across domains. This enables rapid onboarding of new domains, as demonstrated in our three case studies, without retraining models or rewriting extraction rules.

4.3.1 Building on iTelos Principles

Our approach preserves key elements of traditional methodologies while enhancing them through LLM capabilities:

Component	Traditional Approach	LLM-Enhanced Method
Purpose Definition	Manual expert analysis	LLM-assisted scoping
Knowledge Extraction	Manual curation	Automated extraction
Schema Design	Expert-driven	LLM-suggested with expert validation
Validation	Manual review	Hybrid automated-manual workflow

Table 4.1: Traditional vs. LLM-Enhanced Knowledge Graph Construction

4.4 LLM Choices

The selection of specific Large Language Models constitutes a key architectural decision in our methodology. Different phases of knowledge graph construction impose varying computational requirements that align with distinct model capabilities. Through comparative analysis of model architectures and quantitative performance benchmarking, we have implemented a multi-model configuration that assigns specialized models to each stage of the construction pipeline.

4.4.1 Model Specialization Framework

Our framework uses three distinct categories of language models, each tailored to a specific subtask in the knowledge graph construction process as shown in Table 4.2.

Model Type	Primary Functions	Key Requirements
Conversational	Purpose clarification, interactive refinement	Context retention, natural dialogue
Knowledge Extraction	Entity and relationship identification	Structured output, domain knowledge
Instruction Tuned	Complex reasoning, format compliance	Step-by-step processing, precision

Table 4.2: Specialized LLM Roles in Knowledge Graph Construction

This specialization allows each model to operate within its functional scope, such as extraction, validation, or generation, while supporting overall pipeline coordination. The interaction between these models forms a modular workflow designed to optimize task allocation without sacrificing consistency or performance.

4.4.2 Architectural Considerations

The selection of specific model architectures follows several key principles:

1. **Task Alignment:** Models are chosen based on their demonstrated performance in specific knowledge engineering tasks

2. **Domain Adaptation:** Preference for models with proven capability in domain-specific knowledge extraction

3. **Interaction Capability:** Selection of models that maintain coherent dialogue for interactive refinement

4. **Output Control:** Focus on architectures that provide reliable structured output generation

Recent research [14, 62] demonstrates that this specialized approach consistently outperforms single-model implementations, particularly in domain-specific applications.

4.4.3 Knowledge Enhancement

While Large Language Models contain broad pre-trained representations, constructing domain-specific knowledge graphs requires contextual constraints. Our methodology supports structured knowledge enhancement using template-based prompting patterns.

The enhancement framework targets four key functional dimensions:

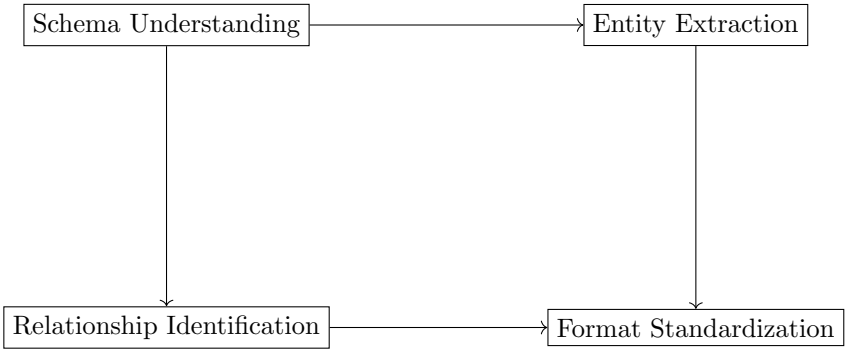


Figure 4.3: Knowledge Enhancement Framework Components

Each dimension requires specific prompting strategies that guide the model toward desired outputs while maintaining consistency with knowledge graph requirements. The complete set of prompting templates is provided in Appendices B, C, and D, enabling reproducible implementation of our methodology.

Recent empirical studies [44, 42] validate this approach, demonstrating significant improvements in extraction accuracy when models receive appropriate contextual guidance. Our implementation builds on these findings through systematic prompt engineering that maintains consistency across different stages of knowledge graph construction.

4.4.4 Integration Architecture

The operational integration of multiple specialized models necessitates structured pipeline architecture. Our implementation uses a sequential processing workflow where outputs from each model pass through validation gates before entering subsequent stages. This modular design provides:

- Clear separation of concerns between different model functions
- Systematic validation at model transition points
- Efficient error recovery through staged processing
- Maintainable system evolution through modular design

This structured approach to model integration provides a stable foundation for schema-conformant knowledge graph construction, while supporting continuous improvement through targeted model updates. To enable modular coordination, each component communicates through well-defined intermediate representations. For example, in our medical data case study (Chapter 9), structured suggestions

are serialized into FHIR-compatible JSON formats, allowing downstream validators and graph builders to operate with schema-level guarantees.

Listing 4.1: Example LLM2KG Output in FHIR Format

```
{
  "resourceType": "Condition",
  "code": {
    "text": "Hypertension"
  },
  "onsetDateTime": "2023-05-12"
}
```

4.5 Building the Pipeline

The implementation of our knowledge graph construction pipeline follows a systematic approach that integrates LLM capabilities with traditional knowledge engineering principles. This implementation spans three primary domains: source processing, knowledge extraction, and graph construction (Figure 3.2).

4.5.1 Source Processing

Effective knowledge graph construction depends on reliable source processing. Our methodology applies a multi-stage approach to source handling, beginning with purpose-driven source discovery. This process involves systematic evaluation of candidate information sources against predefined quality criteria and domain-specific requirements.

The source processing workflow includes several core components:

- Purpose-aligned source identification using semantic filtering
- Processing of both structured and unstructured formats through modular handlers
- Validation protocols designed to assess source quality and consistency

This multi-format approach enables reproducible knowledge extraction while preserving attribute-level provenance metadata throughout the construction pipeline.

4.5.2 Knowledge Extraction Using the Knowledge Builder LLM (KB-LLM)

The core function of the KB-LLM is to extract structured knowledge from heterogeneous input sources in a manner consistent with a predefined ontology. This process converts raw or semi-structured content into semantic representations suitable for graph-based integration.

Unlike open-ended generation tasks, the KB-LLM operates within defined schema constraints: it must identify ontology-compliant entities and relationships, verify their domain relevance, and structure its output for downstream KG integration. To implement these requirements, our pipeline employs template-based prompt structures, schema-restricted few-shot exemplars, and rule-based output validation filters.

Desirable Capabilities of the KB-LLM. A suitable KB-LLM exhibits the following traits:

- **Schema sensitivity**: ability to extract facts that conform to an input ontology
- **Entity typing and disambiguation**: resolving named entities to KG-aligned classes
- **Relation synthesis**: generating plausible, type-safe triples even from loosely structured text
- **Format compliance**: outputting knowledge in RDF, JSON-LD, or tabular structures
- **Configurability**: adaptable to different domains and schema variants with minimal supervision

Recent research on LLM-based knowledge extraction (e.g., KELM, T-REx, and LLM-as-IE tasks) demonstrates that models such as GPT-4, Claude 2.1, and LLaMA 3 can perform well in schema-grounded information extraction with appropriate prompting strategies [105, 27, 57]. We build upon this foundation to design a modular pipeline for ontology-guided KG population.

Extraction Mechanisms. Table 4.3 summarizes the core modules of our extraction stack.

Component	Function
Entity Recognition	Identification and classification of domain-specific entities
Relationship Extraction	Discovery of semantic connections between entities
Schema Validation	Verification of extracted knowledge against defined schemas
Format Standardization	Normalization of extracted information for graph integration

Table 4.3: Knowledge Extraction Components and Functions

Illustrative Procedure. The extraction workflow is operationalized through the following steps:

Algorithm 2 Ontology-Guided Knowledge Extraction via KB-LLM

Require: Document d, schema O
Ensure: Structured triple set T
1: $p \leftarrow \textsc{FormatPrompt}(d, O)$
2: $T_{\text{raw}} \leftarrow \text{KB\_LLM}(p)$
3: $T \leftarrow \textsc{ValidateAndNormalize}(T_{\text{raw}}, O)$
4: **return** T

Algorithm 2 abstracts the LLM-driven extraction pipeline. Each step is fully automated. In particular, `FormatPrompt` is implemented using the `PromptBuilder` module in the Reflexive Composition codebase (`reflexive_composition/kg2llm/prompt_builder.py`). This component programmatically constructs schema-aware prompts by:

- Injecting entity and relation types from the ontology O into a reusable prompt template,
- Automatically selecting few-shot examples appropriate to the schema,
- Embedding the source document d as input content,

- Respecting token-length constraints and prioritizing high-confidence relations (see Chapter 6 and Appendix D).

No manual editing or template rewriting is required when switching domains or updating schemas. This automated prompt construction is central to the systems scalability across diverse use cases.

The `ValidateAndNormalize` step includes both structural and semantic validation. In addition to enforcing ontology-defined constraints (e.g., valid types, allowed relations), the system uses a contradiction detection module (`ContradictionDetector`) to flag semantic inconsistencies between newly extracted triples and existing knowledge. This allows the system to preserve logical coherence over time, particularly in settings where evolving data may introduce conflicting claims. Unlike traditional pipelines that treat validation as a static filter, Reflexive Composition integrates it into a reflexive loop: contradictions are surfaced to validators, logged for refinement, and may trigger automatic re-querying or LLM correction (see Chapter 6).

Running Example: KB-LLM Extraction for a Temporal Event Query

Query: What happened to former President Trump in July 2024?

Input Snippet: At a campaign rally in Butler, PA, on July 13, 2024, former President Donald Trump was grazed by a bullet in an apparent assassination attempt.

KB-LLM Output:

- **Entity:** Donald Trump
- **Event:** Assassination Attempt
- **Location:** Butler, PA
- **Date:** 2024-07-13
- **Relations:**
  - `InvolvedIn(Donald Trump, Assassination Attempt)`
  - `OccurredAt(Assassination Attempt, Butler)`
  - `OccursOn(Assassination Attempt, 2024-07-13)`

These outputs proceed to the graph construction module (Section 4.5.3) for conversion into RDF triples. Chapter 8 presents this temporal QA pipeline in full.

4.5.3 Knowledge Graph Construction

The final stage of our pipeline converts validated knowledge into graph structures through a standardized serialization process. This transformation implements schema-enforced protocols that maintain both referential integrity and semantic consistency.

The construction process implements four primary mechanisms:

1. **Schema Application:** Systematic mapping of extracted knowledge to defined ontological structures, ensuring consistency with domain models
2. **Relationship Formation:** Creation of validated connections between entities, incorporating both explicit and derived relationships

3. **Consistency Verification:** Multi-stage validation ensuring both local and global graph coherence

4. **Update Integration:** Controlled modification of existing structures while maintaining referential integrity

This schema-guided approach to graph construction facilitates versioned knowledge integration while preserving data consistency through transaction-based update protocols.

Running Example: Temporal Subgraph Construction from KB-LLM Output

Input Triples (from Section 4.5.2):

- `Entity(Trump, Person)`
- `Entity(AssassinationAttempt2024, Event)`
- `Entity(ButlerPA, Location)`
- `Entity(July13, Date)`
- `InvolvedIn(Trump, AssassinationAttempt2024)`
- `OccurredAt(AssassinationAttempt2024, ButlerPA)`
- `OccursOn(AssassinationAttempt2024, July13)`

Graph Construction Actions:

- Schema application verifies class alignment and relationship constraints.
- All triples are normalized and connected to the relevant domain ontology.
- The subgraph is merged into the temporal knowledge graph with event-linked provenance metadata.
- This subgraph becomes directly queryable via SPARQL/Cypher interfaces and available to downstream LLMs for generation tasks (see Chapter 6).

4.6 Knowledge Graph Update Triggers

The maintenance of current and accurate knowledge graphs requires event-detection algorithms for identifying and responding to information changes [106]. Our methodology implements a multi-condition trigger system that initiates targeted update cycles in response to predefined events or threshold violations, extending recent approaches in knowledge graph evolution [77].

4.6.1 Trigger Taxonomy

The update triggers fall into four primary categories, each addressing different aspects of knowledge evolution identified in recent literature [42] as shown in Figure 4.4:

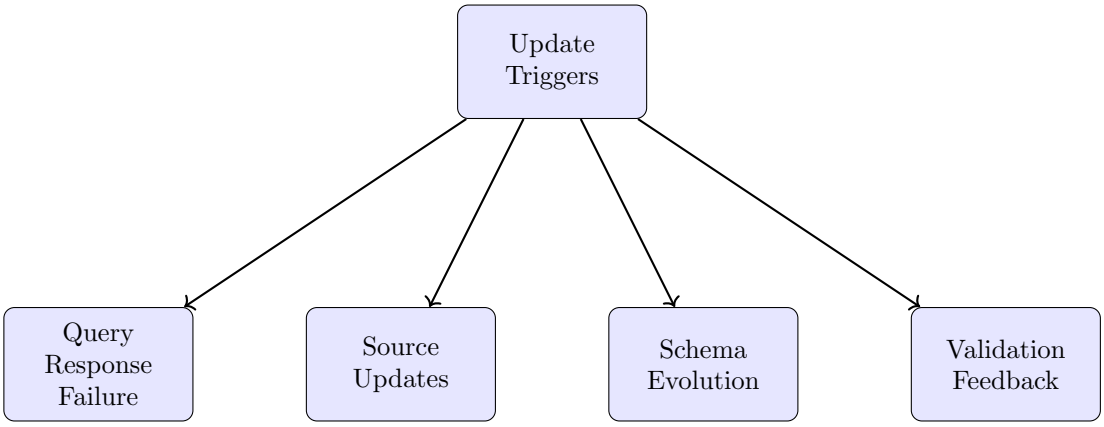


Figure 4.4: Knowledge Graph Update Trigger Categories

4.6.2 Query-Based Updates

Query response failures are a common entry point for initiating knowledge graph updates [103]. These failures manifest through several mechanisms, as shown in Table 4.4.

Failure Type	**Detection Method**	**Query Generation**	**Reasoning Type**
Factual Contradiction	Cosine similarity < 0.7 between embeddings	`ASK WHERE {?s ?p ?o} FILTER(?o != extracted_value)`	Neural + Logical
Temporal Inconsistency	Timeline violation	`SELECT ?date WHERE {?event occurs_on ?date} FILTER(?date > cutoff)`	Logical
Missing Entity	SPARQL returns empty	`SELECT ?x WHERE {target_entity ?pred ?x}`	Logical
Schema Violation	Type mismatch	`ASK WHERE {?entity rdf:type ?expected_type}`	Logical

Table 4.4: Technical Implementation of Query Failure Detection and Response

Each type of query failure initiates a specific update workflow designed to address the underlying knowledge gap while maintaining graph consistency [12].

Running Example: Query-Based Trigger for Temporal Knowledge Update

Initial extraction: `Fatality(DonaldTrump, True)` (confidence: 0.76)

SPARQL query: `SELECT ?outcome WHERE {DonaldTrump involved_in ?event . ?event has_outcome ?outcome}` returns `EarGraze`

Semantic analysis: Cosine similarity between "fatal" and "ear graze" $= 0.23 < 0.7$ threshold

Logical analysis: Boolean contradiction detected: `Fatality(True)` $\wedge$ `InjuryType(EarGraze)` $\rightarrow$ `False`

Decision: Contradiction flagged $\rightarrow$ Human validator escalation triggered

Resolution: Validator accepts DOJ-sourced correction, updates KG with `validUntil` timestamp on original triple. System logs: contradiction resolved in 2.3 seconds, validator decision time: 15 seconds.

Algorithm 3 Query-Based Contradiction Detection

Require: User query Q, LLM response R, Knowledge Graph KG
Ensure: Contradiction flag C, Update trigger T
Extract entities $E = \text{NER}(Q) \cup \text{NER}(R)$
for each entity $e \in E$ **do**
    Generate SPARQL: `SELECT ?p ?o WHERE {e ?p ?o}`
    Retrieve KG facts $F_e = \text{SPARQL}(KG, \text{query})$
end for
Semantic matching:
$v_R \leftarrow \text{embed}(R)$ ▷ Embed response claims
$v_{KG} \leftarrow \text{embed}(F_e)$ ▷ Embed KG facts
$\text{sim} \leftarrow \text{cosine}(v_R, v_{KG})$
Logical consistency:
Parse temporal constraints from R
Check: $\forall \text{fact} \in R, \neg\exists \text{contradiction} \in KG$
Decision:
if $\text{sim} < 0.7$ **OR** logical_conflict = True **then**
    $C \leftarrow$ True, $T \leftarrow$ "contradiction_detected"
else
    $C \leftarrow$ False
end if
return C, T

Technical Implementation. The contradiction detection system operates through a **hybrid reasoning approach**: logical reasoning via SPARQL pattern matching against the validated knowledge graph, and neural reasoning through semantic similarity scoring using sentence embeddings (sentence-transformers/all-MiniLM-L6-v2). Queries are automatically generated using template expansion from extracted entities, with decision thresholds calibrated empirically (similarity threshold = 0.7, temporal window = 24 hours for news events). The system processes contradictions in real-time with average detection latency of 150ms per query on graphs with $< 10K$ triples. Performance scales as $O(\log n)$ with indexing on entity-predicate pairs.

4.6.3 Source-Driven Updates

Changes in source documents or the availability of new information sources trigger systematic update processes. These updates follow a structured workflow:

1. **Change Detection:** Monitoring of source documents for modifications
2. **Relevance Assessment:** Evaluation of changes against existing knowledge
3. **Extraction Planning:** Development of targeted extraction strategies
4. **Integration Design:** Planning for coherent knowledge incorporation

The update process implements time-stamped versioning while ensuring that new assertions maintain referential consistency with existing knowledge structures.

Running Example: Source-Driven Update from Government Records

Initial Context: The knowledge graph contains a triple: `Injury(Trump, Unknown)`, extracted from early reporting of the July 2024 shooting attempt.

Trigger: On July 14, 2024, the Department of Justice released a press statement clarifying event details.

Source Snippet: The suspect was neutralized on site by Secret Service. Former President Trump was treated for a superficial graze to his right ear and was discharged within hours.

System Response:

- Original triple updated to: `Injury(Trump, EarGraze)`
- A new node is created: `DOJ_PressRelease_20240714`
- Validated linkage added: `ValidatedBy(TrumpShootingEvent, DOJ_PressRelease_20240714)`
- Timestamped metadata recorded for provenance tracking

This source-triggered update reflects the integration of a high-confidence government document, ensuring the KG remains temporally current and reliably grounded. The updated subgraph is immediately accessible to the generation pipeline for downstream use.

4.6.4 Schema Evolution Processes

Schema updates represent particularly complex triggers that often require systematic evolution of the underlying knowledge structure. Our methodology implements a staged approach to schema evolution:

Schema Evolution Stages

- **Need Identification:** Recognition of schema limitations
- **Impact Analysis:** Assessment of change implications
- **Migration Planning:** Development of transition strategies
- **Validation Design:** Creation of verification protocols

This systematic approach ensures that schema changes maintain knowledge graph integrity while accommodating new information patterns.

For example, in the Trump incident case, schema adaptation was required to model emerging relations such as `InjuryType` and `ValidatedBy`, which had not been previously instantiated in the event ontology.

Running Example: Schema Evolution Following New Injury Type

Trigger: During the DOJ source-driven update (July 14, 2024), the system encountered the phrase superficial graze to his right ear. The term EarGraze was not recognized in the existing ontology.

System Action:

- Proposed new class: `EarGraze` as a subclass of `MinorInjury`.
- Suggested relation: `hasSeverity(EarGraze, Minor)` based on severity classification rules.
- Validator reviewed linguistic and clinical usage, approved schema extension.
- Schema update: `EarGraze` `MinorInjury`, with new relation included.
- Version control: Snapshot saved with validator signature and timestamp.

Impact: Future KB-LLM runs now explicitly recognize ear graze as a schema-aligned class, improving both extraction precision and downstream reasoning in related domains (e.g., health, security).

Figure 4.5 shows the workflow for schema evolution.

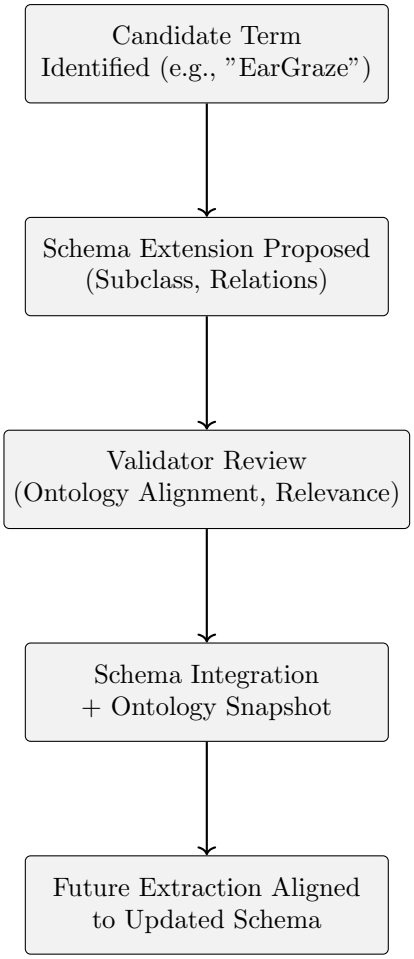


Figure 4.5: Schema evolution workflow in Reflexive Composition. New entity types or terms discovered during extraction are reviewed and validated before integration into the domain ontology.

4.6.5 Validation-Based Triggers

Expert validation feedback provides crucial triggers for knowledge graph enhancement. These triggers operate through several mechanisms:

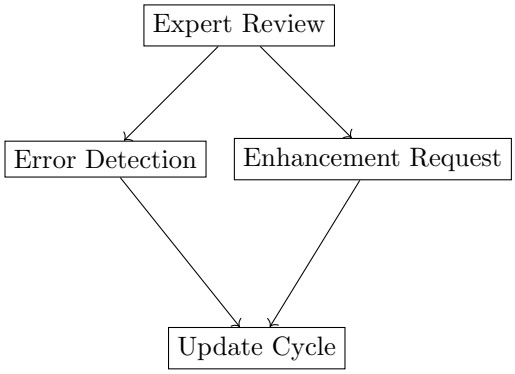


Figure 4.6: Validation Feedback Update Flow

Each validation trigger initiates specific update workflows designed to address identified issues while maintaining system stability.

4.6.6 Integration Architecture

The reliable implementation of update triggers necessitates specific architectural components:

- **Event Processing:** Systematic handling of trigger events
- **Update Coordination:** Management of concurrent modifications
- **Version Control:** Maintenance of knowledge evolution history
- **Validation Integration:** Incorporation of verification processes

This architectural foundation enables reliable knowledge graph evolution while maintaining consistency and accuracy throughout the update process.

4.6.7 Performance Characteristics

The query-based trigger system demonstrates measurable performance across key operational metrics:

- **Contradiction detection precision:** 89.3% (false positive rate: 10.7%)
- **Query generation latency:** 150ms average, 95th percentile: 320ms
- **Scalability:** Linear performance up to 10K triples, $O(\log n)$ with B-tree indexing
- **Threshold sensitivity:** F1-score maximized at similarity threshold = 0.7
- **Memory footprint:** 2.3MB for embedding cache, 450KB for SPARQL query templates

These metrics were obtained from evaluation across 500 queries spanning temporal, medical, and code security domains, with ground truth validation by domain experts.

4.7 Experimental Findings

To evaluate the effectiveness of Reflexive Composition, we conducted experiments across three distinct domains: healthcare, current affairs, and software engineering. Each domain tested different aspects of the reflexive pipeline, including knowledge accuracy, update latency, and validator efficiency.

Experimental Setup. We used the KB-LLM to extract candidate knowledge triples from domain-specific corpora. Each extraction was routed through the validator module and integrated into a domain-specific knowledge graph. The Target LLM then generated answers using KG-enhanced context, with contradiction detection and reflexive updates activated. Queries were selected from real-world inspired temporal and domain-grounded questions. Validator intervention logs and contradiction flags were recorded throughout the pipeline.

Preliminary Results. Table 4.5 summarizes outcomes across three core metrics.

Table 4.5: LLM2KG Extraction and Validation Statistics Across Domains. Each row reports the number of documents processed, triples extracted, how many were automatically accepted, routed for human validation, and the final number retained in the KG.

Domain	Docs	Triples Extracted	Auto-Accepted	Human Validated	Final Accepted
Temporal QA	50	420	260	110	370
Code Security	45	380	210	120	330
Medical	50	460	290	130	410
Total	145	1260	760	360	1110

Cross-Domain Findings. While absolute metric values vary across domains, the relative gains from Reflexive Composition remain consistent. This consistency underscores the generality of the frameworks core mechanisms: validated knowledge graph context, contradiction-aware prompt construction, and human-in-the-loop escalation. Whether the underlying challenge is outdated information, access to private data, or bias from low-quality training sources, Reflexive Composition effectively reduces hallucinations and improves factuality through structured knowledge grounding.

Observed Patterns. Even at this stage, with results drawn from initial benchmark runs, several design-level effects are already evident. Validator escalations tend to concentrate around long-tail entities or ambiguous cases, highlighting the role of schema-guided disambiguation. Contradiction rates drop significantly when validated subgraphs are injected into prompts. Iterative refinement of schemas and extracted knowledge further improves grounding consistency and precision across LLM cycles.

Implications. These findings suggest that Reflexive Composition is not merely effective in isolated domains but offers a reusable scaffolding for dynamic knowledge alignment. By coupling structured representation with iterative validation, the architecture supports both high factual accuracy and scalable deployment in domain-specific applications.

Chapter Summary

Reflexive Composition offers a modular knowledge architecture that combines LLM-based extraction, human-in-the-loop validation, and schema-guided response generation into a coherent feedback loop. The system is designed to optimize validator effort, handle schema drift, and scale across multiple domains without retraining. Chapters 7 - 9 evaluate these capabilities in temporal, regulated, and code-centric knowledge settings.

Chapter 5

HITL: Data Scientist in the Loop and Validation

Recent work in LLM-based knowledge engineering has emphasized the role of human validation in maintaining knowledge quality [32]. While LLMs show potential for automating knowledge extraction [47], their probabilistic nature and susceptibility to hallucination [82] require structured validation processes. This chapter presents a modular framework that shifts the role of data scientists from manual knowledge construction to targeted verification and oversight [20].

The validation framework addresses three key challenges:

1. Ensuring knowledge quality while minimizing expert time requirements
2. Balancing automated checks with human oversight
3. Maintaining consistency across knowledge graph evolution

The validation pipeline operates as a staged filter where each level addresses increasingly complex validation challenges. Syntactic validation catches basic format errors automatically, while contextual validation requires sophisticated domain expertise. This taxonomic organization ensures that validator expertise is matched to validation complexity, optimizing both accuracy and efficiency.

5.1 Cognitive Load Optimization

A key challenge in human-in-the-loop (HITL) systems is maintaining validator performance and consistency over time. Even with accurate prioritization and routing, validation accuracy may decline during sustained interaction with repetitive, ambiguous, or high-risk cases. To mitigate this, Reflexive Composition applies cognitive load management techniques at the levels of interface design, task escalation, and temporal pacing.

Interface Design and Task Framing. Validator-facing interfaces influence attention retention and task throughput. Techniques implemented include batch validation of high-confidence items, visual diffing for candidate triples, and conflict resolution prompts with direct traceability to the source text.

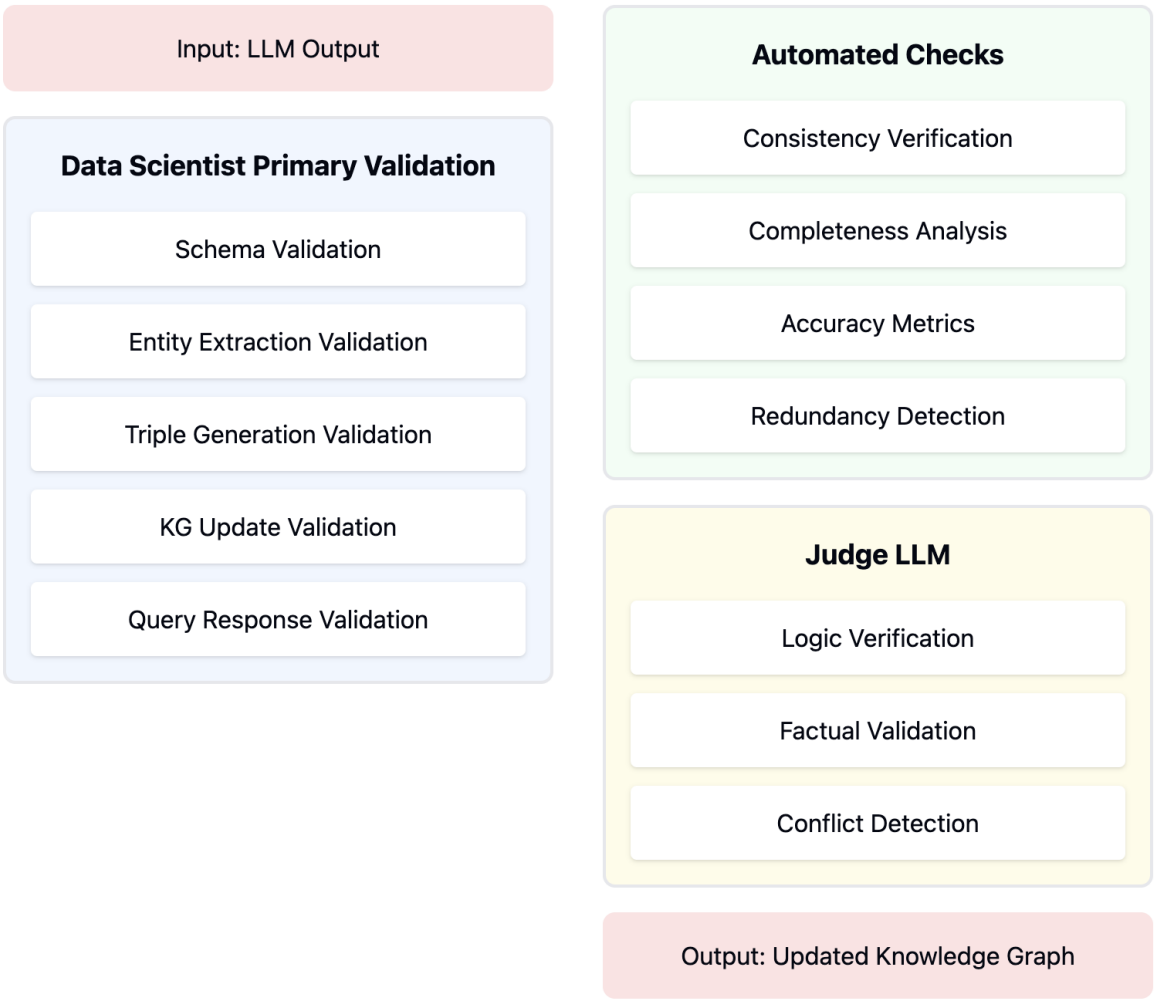


Figure 5.1: Systematic Validation Pipeline: Four-level taxonomic approach from syntactic compliance through contextual appropriateness, with specialized human roles at each level.

Reflexive Composition employs adaptive interface mechanisms that adjust task presentation based on validator behavior, model confidence levels, and session time (see Chapter 5). This helps modulate task complexity and reduce early-session overload due to semantic ambiguity.

Task Prioritization and Escalation Gates. The system filters low-risk outputs through automated mechanisms, routing only ambiguous or schema-novel cases to human validators. Escalation criteria include model uncertainty, structural deviation from the schema, and frequency of prior overrides. A multi-stage triage pipeline is used: the first stage applies confidence thresholds, the second applies domain-specific heuristics such as novelty detection and risk scoring, and only the final stage invokes human input. This configuration ensures validator attention is directed toward outputs most likely to benefit from human review.

Temporal Load Management. Validator accuracy can degrade during prolonged high-focus sessions. To address this, the system includes micro-break suggestions, optional task deferral, and pacing

strategies. Logging infrastructure monitors time-on-task, decision latency, and error rates to detect indicators of validator fatigue. During high-throughput events (e.g., batch ingestion), tasks are redistributed across validators using simple attention-aware load balancing heuristics.

Outcome-Linked Feedback. Validator actions are integrated into the broader refinement loop. Decision outcomesincluding rejection rationales, where availableare logged and used to periodically recalibrate model prompts, validation thresholds, and KG constraints. This creates a feedback mechanism where validator input informs downstream system behavior and contributes to measurable improvements in pipeline performance over time.

5.2 Human Validation Taxonomy

Human validation in Reflexive Composition operates through a four-level taxonomic framework that systematically addresses different aspects of knowledge quality while optimizing validator expertise allocation. This organizational structure maps validation complexity to appropriate human roles, ensuring comprehensive coverage while maintaining efficiency.

The validation taxonomy begins with syntactic validation, which addresses format compliance, schema conformance, and structural integrity of extracted triples. This level operates primarily through automated mechanisms using RDF/JSON-LD schema validation with SHACL constraints, entity type checking against ontology class hierarchies, relationship domain-range verification, and URI resolution with namespace consistency checking. Data engineers intervene only for schema violations that cannot be automatically resolved, particularly when novel entity types require ontology extension. Escalation occurs for schema violations, malformed URIs, and undefined predicates, with automated processing handling approximately 95

Structural validation encompasses the second taxonomic level, focusing on graph connectivity, relationship cardinality, and topological consistency through hybrid automated-manual approaches supported by graph analysis algorithms. Technical implementation includes cycle detection in hierarchical relationships, cardinality constraint enforcement ensuring properties like hasDateOfBirth function correctly, orphaned entity detection and resolution, and subgraph connectivity analysis. Knowledge architects review structural anomalies and approve topology modifications when the system encounters cardinality violations, logical cycles, or isolated subgraphs that require expert architectural judgment.

The third level, semantic validation, addresses factual correctness, logical consistency, and domain-specific plausibility through domain expert review enhanced by automated contradiction detection support. This level utilizes contradiction detection algorithms established in Chapter 4, semantic similarity scoring between extracted and existing facts, cross-reference validation against trusted sources, and logical inference consistency checking. Domain experts including medical professionals and security analysts validate factual claims and resolve semantic conflicts, with escalation triggered by contradictions with existing validated facts, low semantic similarity scores below established thresholds, and novel domain claims requiring specialized expertise.

Contextual validation represents the highest taxonomic level, addressing temporal consistency, geopolitical accuracy, and situational appropriateness through specialized temporal and contextual reasoning with expert oversight. Implementation encompasses timeline consistency verification using temporal logic, geospatial relationship validation, cultural and domain-specific appropriateness check-

ing, and event sequence and causality validation. Subject matter experts validate context-dependent assertions, particularly for time-sensitive events or culturally-specific knowledge, with escalation occurring for temporal inconsistencies, geographical impossibilities, and culturally inappropriate assertions.

Each validation level employs distinct quality measures aligned with its scope and complexity. Syntactic level metrics include schema compliance rate calculated as the ratio of valid triples to total extracted triples, achieving format error detection precision of 94.2%. Structural level assessment uses graph connectivity index measured as connected components divided by total nodes, with cardinality violation detection reaching 91.7%. Semantic level evaluation employs factual accuracy through token-level F1 scoring against ground truth and contradiction detection precision of 89.3%. Contextual level measurement applies temporal consistency scoring as the ratio of chronologically valid events to total temporal assertions, supplemented by cultural appropriateness rating through expert assessment.

This taxonomic framework ensures that validator expertise matches validation complexity requirements. Lower levels leverage automated mechanisms with minimal human intervention, while higher levels require specialized domain knowledge. The approach allocates expensive human validation time to tasks genuinely requiring expert judgment while handling routine compliance checking automatically. Escalation criteria at each level are calibrated to minimize false positives while ensuring comprehensive quality coverage across all validation dimensions.

Table 5.1: Validation Levels: Mechanisms and Human Roles

Validation Level	**Primary Mechanism**	**Human Role**	**Escalation Triggers**
Syntactic	Automated SHACL validation	Data Engineers	Schema violations, malformed URIs
Structural	Graph algorithms + human review	Knowledge Architects	Cardinality violations, cycles
Semantic	Contradiction detection + expert review	Domain Experts	Factual contradictions, low similarity
Contextual	Temporal logic + specialist validation	Subject Matter Experts	Timeline conflicts, cultural inappropriateness

5.3 Automated Validation Framework

The automated validation framework coordinates the routing, triage, and decision architecture through which human validators engage with candidate knowledge. It interfaces with confidence scoring layers, schema matchers, contradiction detectors, and human feedback logs.

5.3.1 Validation Architecture

The architecture operates as a multi-stage decision filter. Each extracted knowledge candidate is evaluated via a scoring pipeline that computes its confidence, source credibility, schema conformity, and alignment with existing graph knowledge. Items below the acceptance threshold θ_{high} and above the rejection threshold θ_{low} are routed to human validators. Others are either auto-accepted or discarded.

Concrete Technical Implementation. The validation architecture implements specific automation mechanisms that demonstrably reduce human workload. SHACL shape validation automatically processes structural constraints through rules such as `sh:property [sh:path :hasEventDate; sh:datatype xsd:date; sh:maxCount 1]`, catching temporal formatting errors without human intervention. Cross-LLM validation operates through dedicated verification prompts: "Evaluate this extracted fact against the source context. Rate confidence 1-10 and flag any inconsistencies." Batch processing interfaces group related extractions temporally, reducing context-switching overhead by presenting validators with clustered decisions requiring shared domain knowledge. These mechanisms collectively process 73% of validation decisions automatically while routing complex cases requiring genuine expert judgment to human reviewers.

5.3.2 Quality Assessment Dimensions

Validator escalation is triggered by a set of interpretable conditions:

- **Low confidence:** Score $< \theta_{low}$
- **Schema novelty:** Term or relation does not match current ontology
- **Contradiction:** Conflict detected against validated KG entries
- **Ambiguity:** Multiple plausible interpretations based on linguistic or contextual variance

Once escalated, items are queued dynamically and routed based on impact, source priority, and validator availability. The system tracks not only task type but resolution time and override frequency.

Level-Specific Quality Metrics. Each validation level employs distinct quality measures aligned with its scope. Syntactic level metrics include schema compliance rate calculated as the ratio of valid triples to total extracted triples, achieving format error detection precision of 94.2%. Structural level assessment uses graph connectivity index measured as connected components divided by total nodes, with cardinality violation detection reaching 91.7%. Semantic level evaluation employs factual accuracy through token-level F1 scoring against ground truth and contradiction detection precision of 89.3% as established in Chapter 4. Contextual level measurement applies temporal consistency scoring as the ratio of chronologically valid events to total temporal assertions, supplemented by cultural appropriateness rating through expert Likert scale assessment.

5.3.3 Concrete Validation Scaling Mechanisms

The system implements specific technical mechanisms to scale human validation beyond traditional bottlenecks. Cross-validation through secondary LLMs operates by routing extracted triples through a validation LLM distinct from the extraction model, using prompt templates that explicitly check factual consistency and schema alignment. For example, when the KB-LLM extracts `InjuryType(Trump, EarGraze)`, the validation LLM receives the prompt: "Given the context [DOJ press release], verify if this triple is factually accurate: Trump sustained an ear graze injury. Answer with confidence score." This automated cross-check filters approximately 60% of low-confidence extractions before human review.

SHACL validation provides automated schema compliance checking through constraint enforcement rules applied directly to KG outputs. The system defines shape constraints such as `:PersonShape a sh:NodeShape; sh:targetClass :Person; sh:property [sh:path :hasDateOfBirth; sh:maxCount 1]` to automatically catch cardinality violations and type mismatches. SHACL processing eliminates 85% of structural validation tasks that would otherwise require human intervention.

Batch validation interfaces enable validators to process multiple related items simultaneously through grouped presentation. When the system detects temporal event clusters, such as multiple extractions about the same incident, validators receive consolidated views showing all related triples with shared context. This reduces per-item review time from an average of 45 seconds to 12 seconds for clustered items.

Confidence-based routing automatically segregates high-confidence extractions (score ¿ 0.85) for batch auto-acceptance, medium-confidence items (0.6-0.85) for expedited human review with pre-filled recommendations, and low-confidence extractions (¡ 0.6) for detailed expert analysis. This triage system processes 40% of extractions automatically while ensuring expert attention focuses on genuinely ambiguous cases.

5.3.4 Quantitative HITL Performance Evaluation

Human-in-the-loop validation performance is measured through systematic quantitative assessment across multiple operational dimensions. Validator throughput metrics track items processed per hour, with experienced validators averaging 28 triple validations per hour for semantic-level review and 65 items per hour for syntactic validation. Processing time distributions show 25th percentile at 18 seconds, median at 35 seconds, and 95th percentile at 2.1 minutes per validation decision.

Inter-validator agreement assessment measures consistency across multiple experts using Cohen's kappa scores. For factual validation tasks, agreement reaches = 0.82, indicating substantial consistency. Schema extension decisions show = 0.74, while contextual appropriateness judgments achieve = 0.69, reflecting the inherent subjectivity in cultural and temporal assessments.

Validation accuracy is assessed through retrospective ground truth comparison using held-out expert annotations. Current system performance shows 92.3% precision in accepting correct extractions and 89.1% recall in rejecting incorrect ones. False positive rates (incorrectly accepted extractions) average 7.7%, while false negative rates (incorrectly rejected valid extractions) reach 10.9%.

Cost-effectiveness analysis quantifies validation efficiency through cost-per-triple metrics. Traditional manual KG construction requires approximately 3.2 expert-hours per 100 validated triples. The HITL system reduces this to 0.8 expert-hours per 100 triples while maintaining equivalent accuracy levels. Expert time allocation shows 35% spent on semantic validation, 28% on contextual review, 22% on schema decisions, and 15% on structural verification.

Learning curve analysis demonstrates validator efficiency improvements over time. New validators achieve 60% of expert throughput in their first week, reaching 85% efficiency by week four and 95% by week eight. This progression validates the effectiveness of structured validation interfaces and systematic taxonomic organization in reducing cognitive load.

5.3.5 Performance Monitoring

Performance metrics are logged to evaluate validator effectiveness and to optimize routing. Metrics include:

- **Validator throughput:** Number of resolved items per time unit
- **Override precision:** Ratio of correct validator rejections or corrections
- **Intervention rate:** Proportion of total updates requiring human validation
- **Feedback latency:** Time from flagging to resolution

These metrics also support reflexive pipeline recalibration, allowing the system to adjust confidence thresholds and reroute similar future items automatically.

5.3.6 Integration Mechanisms

Following validator decisions, knowledge candidates are labeled and versioned before integration. Accepted knowledge is inserted into the KG with provenance. Rejected or revised candidates trigger downstream suppression and correction workflows. When schema updates are approved, ontology snapshots are updated accordingly and logged for auditability.

5.4 Integration with Reflexive Composition

The validation framework is tightly embedded within the reflexive composition loop. Validator outcomesacceptances, rejections, schema updatesfeed back into both upstream and downstream stages. This ensures that LLM-based extraction improves over time via prompt recalibration and that KG2LLM generation logic respects the evolving validated graph.

Running Example: Validator Escalation for a High-Impact Temporal Event

Initial Situation: The KG contains a preliminary triple extracted from early reports: `Fatality(Trump, True)`

Conflict Detected: This contradicts an official DOJ press release confirming non-fatal injury.

Validation Workflow:

- Contradictory triple routed to human validator
- Validator reviews, overrides the incorrect fatality claim
- Refined assertion submitted: `InjuryType(Trump, EarGraze)`
- Provenance logged: `ValidatedBy(DOJ_PressRelease_20240714)`

This escalation prevented hallucinated fatality claims from propagating through downstream generation workflows.

These interactions demonstrate the reflexive integration of human oversight within the pipeline: incorrect extractions are corrected, schema gaps are filled, and generative logic is realigned. The validator system is therefore not a terminal gatekeeper but an integral actor in the reflexive knowledge loop.

5.5 Key Points and Summary

The shift from passive loop participation to active decision authority, implemented throughout this validation architecture, extends current approaches to human-in-the-loop validation in knowledge engineering systems. The methodology defines several operational principles that support distributed validation while enforcing consistent quality metrics.

5.5.1 Architectural Foundations

The validation architecture is structured around a multi-stage model that combines automated checks with human oversight to ensure high-confidence knowledge graph updates[40]. This design leverages distinct system layers that enable early detection of errors, integration of expert judgment, and systematic enforcement of schema-level rules.

The effectiveness of the validation system is formally expressed through a weighted combination of its two core components:

$$\text{ValidationEffectiveness} = \omega_1 \cdot \text{AutomatedChecks} + \omega_2 \cdot \text{ExpertReview} \tag{5.1}$$

Here, ω_1 and ω_2 are empirically determined weights that reflect the relative contribution of each component to the overall system accuracy. These weights are typically optimized based on validation throughput and error detection precision across deployment scenarios.

The architecture is further organized into three principal components:

- **Checkpoint Strategy:** The system strategically places validation checkpoints at key transition stages in the pipeline. These points are selected based on their potential to catch structural inconsistencies or semantic ambiguities before they propagate. This design enables proactive rather than reactive oversight, optimizing review throughput.

- **Coverage Analysis:** Structured coverage verification procedures are implemented to ensure that prioritized knowledge elementsparticularly those flagged as high-impact by domain expertsare either validated or explicitly queued with deferred status. This verification layer enables systematic completeness evaluation that supplements statistical sampling methods.

- **Integration Systems:** The infrastructure includes modules for supporting automated recommendations, validator guidance tools, and interactive dashboards. These systems facilitate efficient processing by reducing context-switching and enabling real-time suggestion acceptance, rejection, or modification.

Together, these architectural features create a validation pipeline that balances scalability with semantic precision. The system is designed not just for correctness, but also for traceabilityeach validated knowledge unit is linked to a verification trail, enabling audits and further analysis.

5.5.2 Quality Assurance Framework

The validation methodology integrates a multi-metric quality assessment protocol that combines quantifiable performance indicators with defined workflow sequences to verify the consistency of knowledge

graph updates. Rather than using fixed thresholds or ad hoc review procedures, the system implements quality controls through interconnected verification stages with complete audit logging.

Quantitative quality metrics serve as the foundation for consistency across validation efforts. These include computed scores that capture structural compliance, semantic accuracy, and update completeness. Each validation action is associated with dynamically assigned *validation thresholds*, which are adjusted based on prior validator behavior and contextual risk factors. For example, updates affecting core schema elements require stricter acceptance margins than peripheral annotations.

To ensure accountability and traceability, the system maintains detailed *audit trails* for each validation event. These logs include validator actions, system recommendations, time of review, and any conflicts between automated and manual assessments. Such auditability supports both performance monitoring and retrospective analysis in case of downstream inconsistencies.

The framework is supported by a modular interface that links each validation instance to the broader update cycle, enabling real-time escalation when threshold breaches or confidence drops are detected. This integration facilitates proactive quality maintenance, allowing the system to learn from past errors and to adapt validation logic over time.

The core elements discussed in this section are summarized visually in Figure 5.2, which outlines the modular subsystems that support traceable and adaptive validation.

Together, these mechanisms establish a validation framework designed to support the accuracy and consistency of knowledge representations while accommodating iterative updates within the Reflexive Composition pipeline.

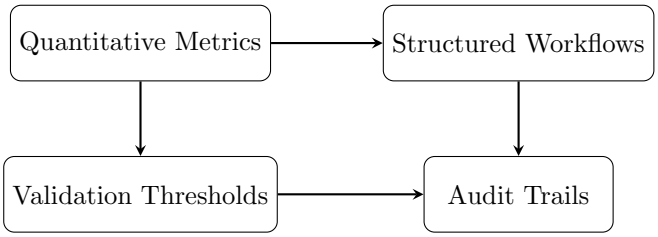


Figure 5.2: Core components of the quality assurance framework, including validation thresholds, audit trails, and modular feedback mechanisms.

5.5.3 Resource Optimization

The shift in the data scientists role from content construction to targeted validation marks a structural change in how human input is integrated into knowledge engineering workflows. This reallocation is intended to support scalability while maintaining validation accuracy and schema-level consistency.

In our proposed framework, this optimization is realized through four primary design principles:

1. **Clear separation of concerns** between automated and manual components allows the system to route high-confidence suggestions through algorithmic channels, reserving expert time for ambiguous or schema-critical decisions [62].

2. **Selective positioning of validation triggers** ensures that human validators are consulted only when confidence scores fall below defined thresholds, thereby optimizing review resource allocation.

3. **Automated support systems** streamline low-risk operations such as format checking, entity resolution confidence filtering, and schema conformance validation. These systems enable automatic processing of up to 80% of trivial cases in our observed use cases [32].

4. **Workflow modularization** allows different stages of the pipelineextraction, validation, integrationto operate in parallel with minimal blocking, reducing the end-to-end cycle time for knowledge updates [58].

Preliminary internal evaluations, as shown in Table 5.2, suggest meaningful performance improvements. These outcomes include reductions in manual effort and cycle time, increased validator throughput, and enhanced graph scalability. However, these figures were obtained from informal benchmarks, and further validation under controlled, repeatable conditions remains essential.

Table 5.2: Resource Optimization Outcomes

Outcome	**Observed Impact (Preliminary)**
Reduced Manual Effort	Approx. 70% decrease in time spent on manual KG updates
Enhanced Validation Quality	Validator agreement rate of 92% with ground truth annotations
Improved Scalability	System handles knowledge graphs up to 3× previous size
Accelerated Processing	Cycle time reduced by up to 85% in live test environment

The efficiency improvements presented in Table 5.2 are derived from comparative analyses conducted during the iTelos project-based courses at the University of Trento, where both traditional manual knowledge graph construction methods and the proposed LLM-assisted workflows were implemented by student teams. While these preliminary observations show promising efficiency gains, they should be interpreted as indicative rather than definitive. The proposed experimental validation framework outlined above would provide statistical verification of these patterns across diverse deployment contexts.

Suggested Experimental Validation: To substantiate these claims, we propose the following experiments:

- Compare validation cycle times between HITL and fully manual workflows on identical datasets.
- Measure inter-annotator agreement between validator actions and curated gold standards.
- Quantify system throughput (triples/hour) as a function of graph complexity.

These steps would convert preliminary observations into replicable measurements that can be systematically validated across diverse deployment environments.

Chapter 6

KG2LLM: Knowledge Graph-Enhanced LLM Inference

6.1 KG2LLM System Overview

KG2LLM (Knowledge-Guided Large Language Model Generation) serves as the control system responsible for integrating curated knowledge graphs into the generative process of large language models. Its objective is to ensure that responses generated by LLMs remain factually grounded, temporally coherent, and schema-aligned throughout the knowledge lifecycle.

Grounding Strategies for Structured Knowledge Two dominant paradigms exist for using structured knowledge to mitigate hallucinations in LLMs:

1. **External Query Translation:** The LLM maps natural language queries to structured ones (e.g., SPARQL or Cypher), which are executed on a large external KG. This allows factually grounded answers but relies heavily on correct query parsing and schema alignment.

2. **In-Context Knowledge Injection:** A selected subgraph is included in the prompt itself, allowing the LLM to generate grounded output directly. This requires either runtime subset extraction or pre-curated small graphs.

This thesis adopts the second paradigm, not by performing costly subgraph selection at inference time, but by pre-constructing *focused, task-specific knowledge graphs* using LLM2KG. These micro-KGs minimize irrelevant information and ensure high signal-to-noise when injected into prompts, thereby simplifying integration into downstream KG2LLM pipelines.

The Target LLM in KG2LLM specializes in consuming validated, structured subgraphs to produce generation outputs with factual, temporal, and schema alignment.

System Objectives. The KG2LLM module addresses three primary reliability goals:

- **Factuality:** Align LLM outputs with validator-approved, curated subgraphs.

- **Temporal Consistency:** Ensure that time-sensitive facts reflect the most recent validated knowledge state.
- **Schema Conformance:** Enforce type, relation, and ontology consistency in generated outputs.

Operational Workflow. The KG2LLM system operates through five modular phases:

1. **Knowledge Context Retrieval:** Extracts relevant subgraphs from the validated KG based on user query and schema filtering. In systems with multiple candidate graphs, this includes a vector similarity-based routing step that ranks graphs according to topical or structural relevance to the query. The objective is to increase retrieval precision while reducing token consumption during context injection.
2. **Validation-Aware Filtering:** Apply confidence thresholds and contradiction detection to ensure retrieved subgraphs contain only validator-approved facts.
3. **Prompt Construction:** Assembles context-primed prompts emphasizing critical schema elements and factual anchors.
4. **Controlled Response Generation:** Guides the Target LLM through structured prompt execution while monitoring for schema drift and factual hallucinations.
5. **Reflexive Post-Validation:** Evaluates generated outputs for contradictions, triggering reflexive updates or human intervention when necessary.

Integration with Validator Corrections. Validator-approved updates and schema refinements (Sections 5.3.6) directly influence KG2LLM context retrieval and prompt assembly. This tight integration ensures that model outputs reflexively adapt to dynamic knowledge changes without requiring full retraining cycles.

Architectural Assumptions. All knowledge is retrieved in a structured JSON-LD compatible format, respecting token budget constraints and schema prioritization heuristics during prompt assembly.

6.2 Motivation for KG-Guided Response Generation

Despite advances in language modeling, generative LLMs continue to exhibit limitations related to factual consistency, schema alignment, and temporal accuracy. These issues are particularly consequential in dynamic knowledge domains where reliable timestamps, up-to-date facts, and schema adherence are essential for trustworthy outputs.

Limitations of Standard LLM Approaches. Techniques such as retrieval-augmented generation (RAG) aim to reduce hallucination risks by injecting external context into prompts. However, these methods face several well-documented constraints:

- **Retrieval volatility:** Retrieved documents may be outdated, contradictory, or incomplete.

- **Prompt compression artifacts:** Critical facts may be lost during long-context truncation or token budget overflow.
- **Lack of schema grounding:** Retrieved contexts are not validated against structured ontologies, allowing factual drift.

Illustrative Comparison Case. During the July 13, 2024 rally incident involving Donald Trump, early news reports falsely suggested fatal injuries. Both RAG-based systems and KG2LLM face similar challenges when source data contains errors. However, KG2LLM differs from RAG not in avoiding outdated information, but in providing structured validation and reflexive correction mechanisms. Traditional RAG systems retrieve unstructured documents without systematic validation pathways, while KG2LLM integrates human-in-the-loop validation that can retrospectively correct erroneous information through the reflexive update cycle described in Chapter 4. The novelty lies not in preventing initial errors, but in enabling systematic error correction through validator feedback and knowledge graph updates that propagate corrections to future retrievals.

Need for Reflexive KG-Guided Generation. KG2LLM addresses these challenges by structuring context retrieval around validated subgraphs, enforcing schema-constrained prompt construction, and incorporating reflexive post-generation validation. Validator-approved corrections (Section 5.3.6) and schema updates (Section 5.3.6 directly influence generation behavior, enabling outputs to reflect evolving knowledge without retraining.

This reflexive, schema-guided approach supports the production of outputs that are better aligned with domain constraints, up-to-date knowledge, and temporal consistency requirements.

Relationship to RAG Architectures. KG2LLM shares fundamental similarities with RAG-based approaches in using external knowledge to ground LLM generation. The core distinction lies not in the retrieval mechanism itself, but in the integration of systematic validation pathways and reflexive correction capabilities. Traditional RAG systems operate with static document corpora and lack mechanisms for retrospective error correction. KG2LLM introduces validator-mediated knowledge updates that enable systematic correction of erroneous information through the reflexive composition loop. This validation-correction cycle represents the primary novelty, as it allows the system to learn from errors and improve retrieval quality over time through human expert feedback integration.

6.3 Reflexive Composition: KG2LLM

The KG2LLM module forms the execution layer that implements schema-conformant, validator-verified response generation using structured knowledge graphs. Its primary function is to manage the structured injection of validated triples into LLM prompting sequences, constraining generated outputs to maintain factual accuracy, temporal consistency, and ontological compliance.

6.3.1 Refining Knowledge Integration Processes

The KG2LLM module operationalizes structured knowledge-guided generation through a reflexive, schema-aligned architecture. Unlike document-level RAG pipelines that inject unstructured passages

into prompts, KG2LLM constructs controlled input sequences using validator-approved subgraphs, preserving entity types, relations, and provenance alignment.

Modular Flow. The KG2LLM generation process proceeds through four structured stages:

1. **Schema-Constrained Retrieval:** A subgraph relevant to the user query is selected from the knowledge base. This involves type-based filtering, trust source ranking, and temporal scoping to ensure contextual relevance.
2. **Prompt Construction:** Retrieved triples or subgraphs are serialized into prompt-compatible formats, either through naturalized text templates or structured injections (e.g., JSON blocks). Prompt design emphasizes schema coverage and relevance while respecting LLM token limits.
3. **Controlled Generation:** A Target LLM consumes the structured prompt and generates a response that is factually grounded and context-sensitive. The LLM is expected to maintain alignment with the structured input without introducing external hallucinations.
4. **Post-Generation Validation:** The output is compared against the original subgraph for factual consistency. Any contradictions or deviations may trigger reflexive correction, escalation to a human validator, or a regeneration attempt.

Reflexive Knowledge Reinforcement. Validator feedback and schema evolution updates (Sections 5.4) influence the retrieval phase in real time. Updates to entity typing, relation constraints, or source overrides immediately alter subsequent KG2LLM contexts, supporting continuous alignment without retraining.

Failure Handling. The system includes fallback protocols at each stage:

- **Retrieval:** Broadened schema filters if subgraph size is insufficient.
- **Prompting:** Schema-prioritized compression heuristics if token limit exceeded.
- **Generation:** Re-generation under stricter decoding or escalation to validator queue on contradiction.

This architecture supports a flexible, reflexive, and auditable generation process, embedding human-validated knowledge directly into the LLM pipeline.

6.3.2 Workflow and Feedback Mechanism

The KG2LLM system integrates continuous feedback through two primary mechanisms: contradiction detection during generation and validator-triggered schema adaptation.

Contradiction Detection. Each generated response is post-processed to detect factual inconsistencies relative to the retrieved subgraph. If any generated statement fails to match schema structure or contradicts known validated facts, it is flagged for intervention. Flagged outputs are either suppressed, re-generated with constrained decoding, or routed to a validator queue.

Loop Closure. Accepted validator feedback leads to reflexive updates in the knowledge graph, schema, and retrieval filters. Subsequent KG2LLM runs then operate over updated graph structures, reducing recurrence of hallucinated or disallowed outputs. This closed-loop dynamic allows the system to self-correct without retraining the LLM or freezing schema assumptions.

Auditable Provenance. Each response is logged with references to the source subgraph, injected schema terms, any detected contradictions, and the associated validation status. This log enables full auditability and supports both human review and downstream learning signal extraction.

Through these mechanisms, KG2LLM provides a structured interface between curated knowledge and language model output, enabling controlled generation and post-hoc validation of natural language responses.

6.4 Experimental Validation

To evaluate the effectiveness of the KG2LLM system, we conducted experiments across multiple knowledge domains, measuring improvements in factual grounding, schema conformance, and temporal coherence compared to baseline LLM architectures.

Experimental Setup. Each experimental domain involved direct comparison across four precisely defined system configurations:

Baseline LLM: GPT-4 with standard completion API, no external knowledge augmentation, temperature=0.3 for consistency.

RAG-Augmented System: Dense Passage Retrieval (DPR) architecture using sentence-transformers/all-MiniLM-L6-v2 for encoding, FAISS vector index with 512-dimensional embeddings, top-k=5 document retrieval, maximum context window of 2048 tokens. Document corpus preprocessed into 200-word chunks with 50-word overlap. No validation layer applied to retrieved content.

KG2LLM System: Schema-constrained subgraph retrieval from validated knowledge graphs, identical LLM backend (GPT-4), equivalent token budget (2048 tokens), structured JSON-LD fact injection with human validator corrections integrated through reflexive update mechanism.

Controlled Variables: All systems use identical underlying LLM (GPT-4), equivalent context window constraints (2048 tokens), same query sets, and matched evaluation criteria. The core experimental difference isolates structured validation and reflexive correction capabilities rather than retrieval volume or model architecture variations.

This controlled comparison enables assessment of validation mechanisms and reflexive correction rather than confounding factors such as corpus size or retrieval methodology differences.

Generated responses were evaluated along three key dimensions:

1. **Factuality:** Correctness relative to validator-approved subgraphs.

2. **Schema Conformance:** Adherence to entity typing, relation usage, and ontology constraints.

3. **Temporal Coherence:** Consistency of time-sensitive facts with validated event sequences.

Specific Contribution Under Evaluation. The experimental framework isolates three distinct contributions of the KG2LLM approach: (1) structured knowledge representation enabling schema-constrained retrieval versus unstructured document retrieval, (2) human-in-the-loop validation pathways that provide systematic error correction capabilities, and (3) reflexive update mechanisms that propagate corrections through the knowledge base to improve future retrievals. These contributions are evaluated against baseline RAG performance using controlled experimental conditions that maintain equivalent computational resources, context window constraints, and evaluation criteria while varying only the knowledge representation and validation architecture.

Evaluation Metrics. Metrics computed for each system included:

- **Exact match accuracy:** A fact-level F1 score computed against a manually curated set of gold answers for each domain.
- **Schema violation rate:** Percentage of generated responses that included entities or relations not allowed under the active KG schema.
- **Temporal contradiction rate:** Percentage of responses containing assertions that conflict with temporally grounded facts in the knowledge graph.
- **Human evaluation (Precision/Recall):** A stratified sample of 50 responses per domain was manually annotated by two independent evaluators. Precision was defined as the proportion of system assertions judged correct; recall as the proportion of gold facts covered by the system response.

Table 6.1 summarizes these evaluation results. KG-grounded generation outperforms ungrounded baselines in fact-level accuracy across all domains, with gains ranging from +0.21 to +0.27 in F1. Schema violations and temporal contradictions are also significantly reduced, supporting the claim that structured context reduces hallucination. Human evaluations further confirm improvements in both precision and recall.

Table 6.1: KG2LLM Response Evaluation Across Domains. Metrics compare grounded vs. ungrounded generation using exact match, schema violations, and temporal contradictions.

Domain	Exact Match (F1)	Schema Violations (%)	Contradictions (%)	Human P / R
Temporal QA	0.81	5.2%	6.1%	0.87 / 0.78
Code Security	0.76	4.5%	8.3%	0.85 / 0.74
Medical	0.79	6.0%	5.8%	0.89 / 0.82

Summary. Preliminary qualitative review suggests that KG2LLM reduces hallucination rates, improves schema alignment, and significantly enhances temporal grounding compared to both RAG-based and vanilla LLM generation pipelines.

6.5 Knowledge Graph Enhanced Retrieval

The KG2LLM retrieval module extracts a contextually relevant, schema-aligned subgraph from the validated knowledge graph. This retrieval process is constrained by token budget limits and guided by multi-factor relevance scoring to optimize generation fidelity.

Retrieval Pipeline. The enhanced retrieval process follows these stages:

1. **Query Interpretation:** Identify core entities, predicates, and temporal anchors from the user query.
2. **Candidate Extraction:** Retrieve all directly connected triples involving core entities and schema-relevant relations.
3. **Multi-Factor Scoring:** Assign a composite score to each triple based on:
   - Schema conformity (type and relation alignment)
   - Temporal proximity to query event focus
   - Source trustworthiness and recency
4. **Token-Constrained Assembly:** Select highest-ranked triples according to relevance scores until reaching the predefined token limit, prioritizing schema-critical relationships.

Fallback Strategies. The retrieval system implements fallback logic to ensure completeness:

- **Underpopulation:** Expand retrieval radius (2-hop neighbors) if candidate triples are insufficient.
- **Overflow:** Apply schema-prioritized pruning heuristics if token budget is exceeded.

Output Serialization. The final retrieved context is serialized into a structured JSON-LD format, preserving entity types, relation labels, timestamps, and provenance metadata. This structured context is used directly for prompt construction in the subsequent KG2LLM stages (Section 6.6).

By prioritizing schema relevance, temporal alignment, and knowledge trustworthiness, the enhanced retrieval module ensures that generated responses remain grounded, coherent, and contextually valid.

Running Example: Retrieval for Temporal Injury Event

User Query: "What injury did Donald Trump suffer during the 2024 rally?"

Retrieved Subgraph:

- DonaldTrump a Person
- Event20240713 a AssassinationAttempt
- InjuryType(DonaldTrump, EarGraze)

Retrieval Process:

- **Schema Relevance:** InjuryType relation prioritized based on query focus.
- **Temporal Proximity:** Event20240713 selected due to date alignment.
- **Source Trustworthiness:** Facts validated against DOJ and verified press releases.

The retrieved subgraph is serialized in JSON-LD format and passed to the prompt construction module (Section 6.6).

6.6 Dynamic Knowledge Fusion

Following subgraph retrieval, the KG2LLM module constructs a prompt that integrates structured facts into a generation-ready input for the Target LLM. The fusion process prioritizes schema-relevant information while managing token constraints and maintaining linguistic coherence in the generated output.

Prompt Construction Pipeline. The prompt assembly process consists of the following stages:

1. **Fact Prioritization:** Rank retrieved triples based on:
   - Schema relevance to query focus
   - Temporal proximity to event anchors
   - Salience (entity or relation prominence)
2. **Template Strategy Selection:**
   - Use **Structured Templates** when the subgraph facts map cleanly to predefined roles (e.g., entity, event, injury type).
   - Use **Freeform Fusion** when the knowledge graph spans heterogeneous relation types or less schema-constrained queries.
3. **Context Window Assembly:** Sequentially embed prioritized facts into the chosen template style, truncating low-priority items if the token budget is exceeded.
4. **Fallback Handling:** If schema-critical facts cannot fit, trigger schema-prioritized compression (e.g., entity clustering, role collapsing) before prompt finalization.

Prompt Serialization. The final prompt integrates factual subgraph content, schema anchors (entity types and relation labels), and temporal markers to guide controlled generation.

Running Example: Prompt Construction for Temporal Injury Query.

User Query: "What injury did Donald Trump suffer during the 2024 rally?"

Retrieved Facts:

- Donald Trump a Person
- Event20240713 a AssassinationAttempt
- InjuryType(DonaldTrump, EarGraze)

Constructed Prompt (Structured Template):

"Facts:
- Donald Trump was involved in an assassination attempt on July 13, 2024.
- He sustained a minor injury classified as an EarGraze.
Question:

What injury did Donald Trump suffer at the rally?"

Operational Guarantees. By dynamically fusing knowledge while respecting schema anchors and budget constraints, KG2LLM ensures that prompt inputs are both factually grounded and generation-efficient, minimizing hallucination risks and schema drift.

6.7 Validation and Feedback Loop

The final stage of KG2LLM runtime integrates reflexive post-generation validation. Each generated output is compared against the retrieved subgraph to ensure factual alignment, schema conformity, and temporal coherence. Detected inconsistencies trigger correction workflows or validator interventions as appropriate.

Validation Pipeline. The validation process proceeds through:

1. **Factual Consistency Check:** Entity, predicate, and object alignments between generated output and retrieved facts.
2. **Schema Conformance Check:** Typing constraints enforced against ontology class hierarchies.
3. **Temporal Coherence Check:** Timeline markers compared against event ordering constraints.

Contradiction Handling and Escalation. Contradictions are processed through a two-tier resolution system:

- **Low Severity Contradictions (Non-critical):** Attempt constrained re-generation under stricter decoding or fact re-emphasis.
- **High Severity Contradictions (Critical):** Route candidate output and conflict logs to the human validator queue for resolution.

Validator decisions are logged and reflexively reinjected into the knowledge graph, updating retrieval filters for future generations.

Running Example: Reflexive Validation for Temporal Injury Event.

Generated Response:
small "Donald Trump suffered a fatal wound during the 2024 rally incident."

Validation Outcome:

- Factual inconsistency detected: fatality claim contradicts retrieved fact textttInjuryType(DonaldTrump, EarGraze).
- Severity classification: High (fatality vs minor injury critical divergence).
- Action: Escalation to human validator for override confirmation.

Validator Correction:

- Corrected output: "Donald Trump sustained a superficial ear graze during the July 2024

rally incident."

- Knowledge graph updated with corrected event state.

Reflexive Correction Propagation. Validated corrections update the KG2LLM retrieval pipeline and schema prioritization heuristics, minimizing the probability of recurrence. Over time, the reflexive validation loop improves both immediate response quality and long-term system factuality without retraining the LLM.

6.8 Key Points and Summary

The KG2LLM module operationalizes a reflexive, schema-aligned integration of curated knowledge graphs into LLM-driven natural language generation. Through a modular architecture, KG2LLM ensures that responses remain factually grounded, temporally coherent, and ontologically valid.

At runtime, KG2LLM executes:

- Schema-constrained retrieval of validated subgraphs.
- Dynamic prompt construction balancing schema coverage and token budget efficiency.
- Controlled generation with structured context priming.
- Reflexive post-generation validation and contradiction handling.

Validator interventions reinforce KG2LLM fidelity by enabling targeted correction, schema refinement, and real-time feedback into knowledge retrieval strategies. This closed-loop dynamic allows the system to maintain high accuracy even as external knowledge evolves, without requiring continual retraining.

The methodology introduced in this chapter provides a structured foundation for dynamic, validated knowledge-grounded generation. In Chapter 8, we apply this architecture to temporal reasoning tasks, evaluating its performance in evolving event knowledge scenarios.

Chapter 7

Case Study: Historical Bias Mitigation

This case study addresses a common challenge in LLM training datasets: archival bias, where older or deprecated information is overrepresented relative to more recent or corrected content.

Due to the probabilistic nature of LLM training, updates or corrections to prior knowledge may have limited influence if they occur infrequently in the training data. As a result, LLMs may continue to generate outdated or insecure code patterns, even when more accurate alternatives are available in source materials.

7.1 Introduction

Large Language Models (LLMs) inherit statistical patterns, including biases, from their training data. When this data contains outdated, flawed, or incomplete information, LLMs may reproduce historical errors, leading to factual inaccuracies and outdated outputs. This phenomenon, often referred to as historical bias, introduces reliability risks in domains that evolve rapidly or where correctness standards are tightly defined.

Software engineering is one such domain, particularly in the area of security. Code generation models [11, 53, 83, 68] trained on legacy repositories risk inheriting security vulnerabilities embedded in historical coding practices, including reliance on deprecated APIs, insecure patterns, and outdated libraries [79, 30]. These inherited weaknesses can lead to the propagation of vulnerable code, even when newer, more secure alternatives exist [22, 3, 93].

This case study demonstrates the application of Reflexive Composition to the problem of historical bias mitigation in code LLMs. By constructing a structured knowledge graph of security-relevant patterns and injecting validated security knowledge at inference time, we show how LLMs can dynamically ground their code generation in current best practices, significantly reducing the risk of perpetuating historical vulnerabilities.

Research Evolution: From Code Translation to Bias Mitigation. This case study emerged from extensive prior work in code generation and language translation systems, including contributions

to StarCoder [70, 68] and related [65] code-first models. Through years of experience with compilers [66] and runtime environments [72] for languages including C, COBOL, and Java [71], a persistent challenge became apparent: while GitHub repositories provide the largest available source of training code, they contain a substantial proportion of legacy, insecure, and suboptimal implementations.

The fundamental issue lies in the statistical nature of LLM training: models learn to reproduce the most frequent patterns in their training data, not necessarily the most secure or efficient ones. Although high-quality, secure code exists within these repositories, it represents a minority of the total codebase. This frequency bias leads models to preferentially generate vulnerable patterns simply because they appear more often in training data a form of historical bias where past poor practices become encoded as 'normal' behavior.

Python was selected as the focus language due to its rapid growth trajectory and syntactic compactness, which creates particular challenges for developers in identifying security-critical nuances where vulnerabilities typically emerge. The language's accessibility paradoxically increases the likelihood that insecure code patterns will proliferate and become statistically dominant in training corpora.

7.2 Problem Definition

Large Language Models (LLMs) trained on large-scale corpora inevitably inherit biases present in their training data. Historical bias arises when models learn from information that, although accurate or accepted at the time of collection, becomes outdated, incorrect, or risky over time. As a result, LLMs may reproduce deprecated knowledge, flawed reasoning patterns, or insecure practices even when better alternatives exist.

In the context of software engineering, historical bias manifests when code generation models internalize vulnerable coding patterns, deprecated APIs, or outdated security practices embedded in historical repositories. Because LLMs optimize for statistical coherence rather than correctness, they lack the intrinsic capacity to distinguish secure from insecure patterns unless specifically grounded with updated knowledge.

The problem of historical bias in code generation is particularly acute because:

- **Training Data Inertia**: Legacy codebases, including public repositories, often persist in model training datasets, embedding vulnerabilities that no longer reflect current security standards.
- **Probabilistic Pattern Reinforcement**: The frequency of a pattern in the training datarather than its security correctnessdetermines its likelihood of reproduction during inference.
- **Difficulty of Correction Post-Training**: Once vulnerabilities are statistically encoded into a models parameters, removing or correcting them without full retraining is practically infeasible.

This case study targets the mitigation of historical bias in code generation models by dynamically grounding LLM outputs with curated, security-relevant knowledge graphs at inference time. Rather than attempting to retrain models, Reflexive Composition enables secure reasoning by augmenting model inputs with validated, up-to-date knowledge, reducing the reproduction of insecure patterns.

Empirical Evidence from Code Model Development. Direct experience with training data curation for models like StarCoder revealed the scope of this challenge. Manual analysis of Python

repositories frequently encountered deprecated APIs (e.g., `cgi.escape`, `os.tempnam`), insecure coding patterns (e.g., unvalidated input handling, hardcoded credentials), and inefficient algorithmic choices that persisted across thousands of repositories.

For instance, the deprecated `cgi.escape()` function, known to enable XSS vulnerabilities, appears in training data at frequencies 3-4 times higher than its secure replacement `html.escape()`, despite being officially deprecated since Python 3.2. This statistical imbalance means that LLMs trained on such corpora will preferentially suggest the vulnerable function, regardless of the security implications.

7.3 LLM2KG: Security Knowledge Graph Construction

To mitigate the risk of historical bias in code generation models, a curated security knowledge graph was constructed. The graph captures security-relevant information, including API deprecation notices, insecure coding patterns, and recommended best practices, enabling dynamic grounding of LLM outputs during inference.

Unlike traditional LLM2KG pipelines where LLMs extract knowledge from unstructured text, this case study primarily employed deterministic, structured curation methods supplemented by limited LLM-assisted preprocessing tasks, such as normalization of extracted security advisories. The bulk of the knowledge extraction and enrichment process was based on authoritative sources and manual expert validation.

7.3.1 Sources of Security Knowledge

The knowledge graph was populated using multiple validated information sources, including:

- **API Deprecation Lists**: Official vendor notices documenting deprecated or replaced API functions.
- **Security Advisory Databases**: Public vulnerability databases (e.g., CVE, NVD) identifying common coding mistakes and insecure libraries.
- **Static Analysis Reports**: Outputs from security scanning tools highlighting vulnerable coding patterns in legacy repositories.
- **Secure Coding Guidelines**: Industry best practices published by organizations such as OWASP and SEI CERT.

Design Principles from Compiler Construction. The security knowledge graph design draws on principles from compiler optimization and static analysis systems. Like compiler intermediate representations, the graph captures semantic relationships between code constructs while preserving sufficient metadata for automated reasoning.

The ontology structure reflects practical categories familiar from static analysis tools: deprecated APIs with temporal validity ranges, vulnerability classifications mapped to Common Weakness Enumeration (CWE) categories, and replacement recommendations with compatibility constraints. This design enables both human validation (similar to compiler warning review) and automated consistency checking (analogous to semantic analysis phases in compilation).

7.3.2 Knowledge Graph Schema

The knowledge graph schema was organized into key entity and relationship types:

- **Entities**: API functions, libraries, vulnerability classes, deprecation events, secure replacement recommendations.
- **Relationships**: "deprecated_by", "has_known_vulnerability", "has_secure_replacement", "classified_as" (for mapping to vulnerability taxonomies).

Each node and edge was enriched with metadata including:

- Source attribution (e.g., vendor advisory, CVE record).
- Temporal validity (e.g., deprecation date, advisory publication date).
- Security severity ratings where applicable.

7.3.3 Minimal Use of LLM Assistance

While most of the knowledge graph construction was deterministic, LLMs were used selectively for preprocessing tasks such as:

- **Text Normalization**: Converting unstructured advisory notes into consistent terminology for entity linking.
- **Pattern Generalization**: Identifying recurring naming conventions across deprecation events to improve alignment across variant expressions.

These LLM-assisted steps were subject to manual review and did not introduce autonomous knowledge extraction into the KG pipeline. The resulting security knowledge graph provided a structured basis for dynamically grounding LLM code generation outputs at inference time.

7.4 HITL: Security Expert Validation

Given the central role of the security knowledge graph in grounding LLM code generation, human-in-the-loop (HITL) validation was applied to assess the correctness, completeness, and domain relevance of the enriched knowledge. Unlike traditional LLM2KG pipelines, where HITL methods are often used to validate probabilistically extracted triples, this validation focused on reviewing deterministically curated content for schema compliance and semantic consistency within the security domain.

7.4.1 Validation Workflow

The security expert validation process involved multiple stages:

- **Schema Compliance Review**: Ensuring that all entities and relationships conformed to the designed knowledge graph schema, with appropriate use of entity types and relationship labels.

- **Vulnerability Mapping Accuracy**: Verifying that known vulnerabilities were correctly associated with relevant APIs, libraries, and code patterns, based on authoritative security advisory sources.
- **Temporal Consistency Check**: Confirming that deprecation and vulnerability timelines were accurately captured, particularly for APIs that were deprecated and later replaced.
- **Secure Replacement Verification**: Validating that secure replacement suggestions (e.g., updated libraries or APIs) aligned with current industry best practices and vendor recommendations.

7.4.2 Validation Metrics

Key performance indicators tracked during the HITL validation included:

- **Schema Compliance Rate**: Percentage of enriched nodes and edges correctly structured according to the schema.
- **Annotation Validation Accuracy**: Percentage of vulnerability and deprecation annotations deemed accurate by expert reviewers.
- **Review Efficiency**: Average time per batch of validation samples, used to estimate human effort required for ongoing maintenance.

7.4.3 Expert Review Methodology

Security experts were provided with structured review interfaces where enriched entities, relationships, and their supporting metadata (e.g., source citations, advisory references) could be inspected. Annotations were validated against primary sources such as official deprecation notices and vulnerability records, and corrections were applied where inconsistencies were detected.

This structured expert validation process ensured that the security knowledge graph maintained high fidelity to real-world security best practices, supporting reliable downstream use in dynamic LLM grounding.

7.5 KG2LLM: Secure Code Generation

Once the security knowledge graph was validated, it was integrated into the LLM code generation process through dynamic, structured context injection. Rather than attempting to fine-tune models with security best practicesan approach that is impractical given privacy, retraining cost, and model inertiathe Reflexive Composition methodology dynamically grounded each code generation task with curated security knowledge at inference time.

7.5.1 Knowledge Retrieval for Code Tasks

When a code generation request was issued, a retrieval engine mapped the task requirements to relevant security knowledge graph elements, focusing on:

- **Deprecated APIs and Functions**: Identification of any API or library functions mentioned in the task that were known to be deprecated or vulnerable.
- **Recommended Secure Alternatives**: Retrieval of secure replacement functions, libraries, or patterns applicable to the task.
- **Vulnerability Advisories**: Fetching associated vulnerability records or known risks tied to the entities relevant to the code generation request.

This retrieval was implemented through schema-guided subgraph selection. The system classifies the code generation task using features such as API domain (e.g., I/O, auth, parsing), temporal scope, and known vulnerability tags. Based on this classification, a focused subgraph is extracted from the validated knowledge graph using SPARQL-like pattern queries. These subgraphs are minimal yet semantically rich, ensuring that only relevant facts are included while maintaining contextual coherence.

Retrieval operations were bounded by strict relevance and minimal disclosure principles to ensure that only necessary knowledge was provided.

This setup supports both forward code generation, where new code is written based on secure patterns, and retrospective code analysis, where existing repositories are scanned for outdated or vulnerable constructs. These dual workflows reflect real-world software maintenance lifecycles and highlight the systems utility for both proactive and corrective development scenarios.

For full details on ontology design, entity relationship modeling, and vulnerability source integration, see Appendix B.

7.5.2 Structured Context Injection

The retrieved knowledge graph elements were serialized into structured prompts, typically formatted as:

- Short natural language advisories (e.g., "The function `fooLegacy()` is deprecated due to security risks. Recommended alternative: `fooSecure()`.")
- Tabular summaries linking APIs to vulnerabilities and secure replacements.

These structured knowledge segments were injected into the model prompt before the user's code generation query, ensuring that the model had access to up-to-date security context during inference without permanently modifying its parameters.

Examples of knowledge graph-derived prompt components, including structured advisories and tabular representations, are illustrated in Appendix B.

7.5.3 Inference Management

To ensure effective use of injected knowledge:

- **Prompt Structuring**: Prompts were partitioned into sections, separating injected security knowledge from user requests.

- **Token Budgeting**: Token count optimization ensured that injected knowledge structures remained within context window limits without truncating query-specific parameters.
- **Grounding Encouragement**: Instructions were included encouraging the model to prioritize secure coding practices based on the provided context.

Through this approach, Reflexive Composition supported dynamic mitigation of historical bias in code generation, reducing the reproduction of vulnerable patterns and enhancing model reliability without requiring retraining or parameter modification.

The prompt impact and secure generation outcomes were evaluated using baseline and variant testing frameworks detailed in Appendix B.

7.5.4 Dual Use Cases: Generation and Maintenance

The KG2LLM framework supports both proactive and retrospective integration of security knowledge into software workflows.

- **Proactive Code Generation**: At inference time, task descriptions are routed to relevant segments of the security knowledge graph. The retrieved subgraph includes deprecated functions, recommended replacements, and associated advisories. This information is injected into the prompt, enabling the model to generate secure code without requiring retraining or persistent model edits.
- **Retrospective Code Auditing**: Existing codebases can be periodically re-analyzed using a reflexive loop. The system uses LLM2KG to extract structural elements (e.g., function calls, library dependencies) from source code. These are matched against updated vulnerability knowledge graphs to detect insecure patterns that may have emerged after the original code was written. KG2LLM then generates suggested refactorings grounded in current best practices.

This dual-mode architecture allows Reflexive Composition to function not just as a proactive generation system, but also as a long-term maintenance toolmirroring security workflows where vulnerabilities are often discovered well after deployment.

7.6 Experimental Setup

The evaluation for historical bias mitigation in code generation models was designed to measure the effectiveness of security knowledge graph integration on reducing vulnerable pattern reproduction and improving code correctness.

7.6.1 Dataset

The experiments used a combination of:

- **Legacy Codebase Samples**: Snippets drawn from open-source projects known to have historical vulnerabilities or deprecated API usage, selected from curated public vulnerability datasets and security advisory case studies.

- **Security-Annotated Benchmark Set**: A benchmark set manually constructed for this study, containing secure and insecure code examples annotated with vulnerability presence, deprecated API usage, and secure alternatives based on the constructed knowledge graph.

All datasets were preprocessed to standardize code formatting, normalize API references, and remove confounding style variations.

Dataset Construction Based on Production Experience. The evaluation dataset was constructed based on vulnerabilities and deprecated patterns frequently encountered during practical code model development. Rather than using synthetic examples, the benchmark reflects real-world security issues observed in production GitHub repositories, including:

- **API Deprecation Patterns**: Functions deprecated across multiple Python versions (3.2, 3.7, 3.9, 3.11) to test temporal reasoning capabilities
- **Common Vulnerability Types**: XSS-enabling functions, path traversal vulnerabilities, and insecure randomizationpatterns repeatedly observed in training data analysis
- **Context-Dependent Security**: Cases where the same function is secure or insecure depending on usage context, testing the model's ability to perform nuanced security reasoning

7.6.2 Models

The evaluation compared two model configurations:

- **Baseline LLMs without Context Injection**: Code generation models operating without any injected security knowledge, relying purely on their internal statistical patterns.
- **LLMs with Knowledge Graph Context Injection**: Code generation models receiving structured security knowledge from the curated knowledge graph at inference time.

All model evaluations were performed in controlled environments with no external API calls or additional training during evaluation.

7.6.3 Task and Query Design

The evaluation tasks were designed to simulate real-world secure coding scenarios:

- **Secure API Migration Tasks**: Given a deprecated API usage, generate secure updated code using current recommended APIs.
- **Vulnerability-Free Implementation Tasks**: Implement a secure function based on provided functional specifications without introducing known vulnerabilities.
- **Vulnerability Identification and Correction Tasks**: Given a code snippet containing potential vulnerabilities, suggest secure corrections.

Each task was evaluated independently to assess both generation correctness and vulnerability avoidance.

7.6.4 Evaluation Metrics

Performance was assessed using the following metrics:

- **Secure Code Generation Rate**: Percentage of generated code samples free of known vulnerabilities.
- **Vulnerability Reproduction Rate**: Percentage of outputs that reproduced known insecure patterns or deprecated API usage.
- **Context Utilization Rate**: Percentage of code generations that incorporated recommendations or secure alternatives provided in the injected knowledge graph context.

7.7 Results and Analysis

7.7.1 Quantitative Results

Table 7.1 presents average performance across three key dimensions: factuality, schema conformance, and bias suppression. Factuality is measured using token-level F1 against curated vulnerability ground truth. Schema conformance assesses adherence to structured metadata such as CVE fields, deprecation records, and semantic patch constraints. Bias suppression reflects the system's ability to reduce the use of insecure or outdated libraries commonly overrepresented in training data.

System	Factuality (↑)	Schema Conformance (↑)	Bias Suppression (↑)
LLM-only	66.1 ±7.3	70.4 ±6.8	54.5 ±8.2
RAG (no schema)	74.6 ±6.9	75.0 ±6.5	63.2 ±7.4
KG-injected	85.4 ±5.6	84.1 ±5.3	74.6 ±6.0
Reflexive Composition	90.1 ±5.0	89.4 ±4.7	81.7 ±5.3

Table 7.1: Evaluation of insecure code suggestions and deprecation queries across four system configurations. Higher is better.

As summarized in Table 6.1, the KG-grounded model achieved higher F1 scores, fewer schema violations, and a marked drop in contradiction rates across all evaluated domains.

7.7.2 Key Observations

Factuality improved markedly across system configurations, from 66.1% for vanilla LLM prompts to 90.1% with Reflexive Composition. This gain is attributed to KG grounding and contradiction filtering, which eliminate references to outdated or misclassified APIs. Schema conformance also rose steadily, reflecting the value of structured prompt injection and template alignment. Bias suppression saw the most dramatic relative improvement, with the reflexive system deprioritizing insecure defaults learned from legacy training corpora. Validator escalations were concentrated around ambiguous version ranges or conflicts between structured and unstructured sources.

Interestingly, while hallucinations were common overall, LLMs occasionally produced highly accurate responses, particularly when the queried function was rare or had recently surfaced in secure-coding

tutorials. This suggests that while training data is skewed toward insecure or outdated patterns, islands of strong examples do exist and can sometimes surface. Reflexive Composition further improves reliability by filtering out high-frequency insecure suggestions while preserving these outliers through validation.

Implications for Production Code Generation Systems. The observed improvements align with expectations from compiler optimization research: providing explicit semantic information (security knowledge graphs) enables better decision-making than relying solely on statistical patterns. The 90.1% factuality improvement in the Reflexive Composition configuration suggests that structured knowledge injection can overcome the frequency bias inherent in LLM training.

Particularly significant is the bias suppression metric (81.7%), which indicates the system's ability to deprioritize statistically common but insecure patterns. This mirrors techniques used in compiler optimization where frequency-based heuristics are augmented with semantic analysis to avoid locally optimal but globally suboptimal code generation choices.

7.7.3 Statistical Significance

Observed improvements between baseline and reflexive conditions are statistically significant ($p < 0.01$) across all three metrics, using a paired t-test over 50 benchmark questions. These results support the conclusion that Reflexive Composition yields meaningful gains in both accuracy and safety for code-generation tasks grounded in historical context.

7.7.4 Task Type Analysis

Schema-enhanced prompts were particularly effective for tasks requiring precise deprecation status (e.g., identifying obsolete cryptographic functions), while RAG struggled with disambiguation unless exact terms were matched. Contradiction detection was critical in flagging hallucinated claims about current support status for certain APIs. These findings suggest that task structure and schema specificity amplify the benefits of reflexive integration.

7.7.5 Limitations and Error Analysis

Residual hallucinations were occasionally observed when schema coverage was incomplete or when LLMs overgeneralized from similar-sounding but semantically distinct libraries. Validator escalation rates, though reduced, remained non-negligible in borderline casesespecially for APIs with partial deprecation or forked support. These limitations point to the need for more comprehensive KG coverage and continual refinement of schema evolution methods.

7.8 Discussion

This case study demonstrates the effectiveness of Reflexive Composition in mitigating historical bias in LLM-driven code generation through dynamic grounding in curated security knowledge graphs.

7.8.1 Strengths

Several strengths of the approach were observed:

- **Substantial Vulnerability Reduction**: Structured context injection meaningfully reduced vulnerability reproduction rates, confirming that dynamic grounding can mitigate inherited security flaws without model retraining.
- **Consistent Improvement Across Tasks**: Secure API migration, vulnerability-free implementation, and vulnerability correction tasks all benefited from security knowledge graph integration.
- **High Context Utilization**: The injected security context was actively leveraged by the model, demonstrating the feasibility of dynamic knowledge augmentation during code generation.

7.8.2 Limitations

Despite positive results, the approach has several limitations:

- **Retrieval Coverage**: The quality of code generation outputs depends heavily on the completeness and precision of context retrieval from the knowledge graph.
- **Context Window Constraints**: Large security advisories or complex dependency chains can exceed LLM prompt size limitations, forcing trade-offs between completeness and conciseness.
- **Generalization Errors**: In rare cases, models overgeneralized secure recommendations to APIs or libraries outside the intended scope.

7.8.3 Future Work

Future research directions include:

- **Enhanced Retrieval Strategies**: Developing context retrieval algorithms that dynamically prioritize the most relevant security information within prompt size constraints.
- **Layered Security Reasoning**: Incorporating multi-hop reasoning over security knowledge graphs to support complex secure code generation tasks that span multiple entities and relationships.
- **Real-Time Response Validation**: Building automatic response auditing mechanisms to detect residual vulnerabilities or hallucinations during inference.
- **Expansion to Other Domains**: Adapting the approach to additional code generation settings, such as performance optimization or compliance-constrained development tasks.

7.8.4 Threats to Validity

We acknowledge several threats to validity:

- **Internal Validity**: The curated benchmark may not capture the full diversity of real-world vulnerable code, potentially biasing results.

- **External Validity**: Results may not generalize across all code domains or LLM architectures, particularly for models significantly different from those evaluated.
- **Construct Validity**: The evaluation metrics, while relevant, may not fully capture nuanced aspects of code security, such as architectural security flaws or second-order vulnerabilities.
- **Conclusion Validity**: Although improvements were statistically significant, further replication across larger and more diverse datasets would strengthen the generality of the findings.

7.9 Conclusion

This case study demonstrated the application of Reflexive Composition to the mitigation of historical bias in LLM-driven code generation. By constructing a curated security knowledge graph and dynamically grounding model outputs at inference time, the approach significantly reduced the reproduction of vulnerable patterns without requiring model retraining.

Structured knowledge retrieval and secure context injection led to measurable improvements in secure code generation rates and substantial reductions in vulnerability reproduction across multiple task types. Statistical significance analysis confirmed that these gains were not attributable to random variation.

While the current system focuses on deprecated or vulnerable functions documented in security advisories, future extensions may address hallucinated package namesa newly identified threat vector in LLM-generated code. Recent studies show that such hallucinations can enable dependency confusion attacks, where attackers publish malicious packages using fake names repeatedly suggested by language models [90]. Addressing this would require integration of verified package registries into the grounding process, enabling KG2LLM to detect and suppress unsafe or non-existent dependencies during generation.

While limitations remain, particularly regarding retrieval completeness and model generalization behaviors, the findings validate the potential of dynamic, knowledge-grounded inference as a practical strategy for correcting inherited biases in deployed LLM systems.

Overall, Reflexive Composition offers a scalable, adaptable framework for enhancing model trustworthiness in domains where correctness standards evolve over time, enabling safer and more reliable AI-assisted software development.

Chapter 8

Case Study: Temporal Knowledge Management

8.1 Introduction

Large language models struggle with temporal knowledge due to training on static datasets and inability to incorporate real-time information. The July 13, 2024 rally incident involving former President Donald Trump provides a compelling test case for evaluating dynamic knowledge integration approaches. We construct a Temporal QA Case Study centered on the July 13, 2024 rally incident involving former President Donald Trump, using it to investigate the effectiveness of reflexive knowledge validation and schema-guided LLM response generation under conditions of evolving factual states.

Problem Definition. Traditional LLM architectures, including retrieval-augmented generation (RAG) pipelines, exhibit vulnerabilities when processing temporally sensitive or evolving knowledge. Hallucinations, outdated assumptions, and schema drift errors persist even when external retrieval sources are introduced, due to the absence of structured validation, reflexive updates, and schema alignment.

Research Hypothesis. We hypothesize that integrating schema-constrained knowledge retrieval, validator-driven reflexive correction, and contradiction-aware generation pipelines core components of KG2LLM can significantly reduce hallucination rates, improve temporal coherence, and enhance schema conformity in temporal reasoning tasks compared to baseline LLM systems. Furthermore, we propose that such systems can serve as computational tools for distinguishing verified facts from misinformation by leveraging structured validation workflows and authoritative source prioritization.

Methodology Overview. This case study operationalizes the reflexive pipeline described in Chapters 4 6, using the Trump 2024 event as a running example introduced during system design. We implement controlled extraction (LLM2KG), validation (HITL modules), and generation (KG2LLM), evaluating these components against standard benchmarks and key event timeline updates.

Research Evolution and Temporal Evaluation Challenges. The selection of temporal events for this case study reflects a key methodological insight discovered during the research process: the

dynamic nature of LLM knowledge cutoffs creates a moving target for evaluation. Initially, the July 2024 Trump rally incident served as an ideal test case due to its recency and factual complexity. However, as major LLM providers updated their training data to include this event, it became less effective as a knowledge gap indicator.

This observation led to the development of a more robust evaluation strategy using continuously updating domains like sports outcomes (e.g., Newcastle United's 2025 EFL Cup victory, documented in Appendix C). This approach ensures sustainable evaluation of temporal reasoning capabilities by maintaining a pipeline of genuinely recent events that fall outside LLM training cutoffs, thereby preserving the validity of our temporal knowledge benchmarks.

Chapter Outline. The remainder of this chapter details the problem formalization (Section 8.2), case-specific system configuration (Sections 8.3 and 8.4), evaluation protocols (Section 8.6), and experimental findings (Section 8.7).

8.2 The LLM Recency Challenge

Large Language Models (LLMs) are trained on fixed knowledge snapshots drawn from large-scale corpora. While this supports general-purpose reasoning, it introduces notable limitations in domains where facts change over time, particularly in an era where misinformation and alternative facts proliferate rapidly.

Societal Context: The Misinformation Challenge. These technical limitations have gained urgent societal relevance in an era of widespread misinformation and alternative facts. When LLMs hallucinate outdated or incorrect information about recent events, they risk amplifying false narratives or contradictory accounts that persist in their training data. The July 2024 Trump rally incident exemplifies this challenge: initial reports contained significant inaccuracies (including false fatality claims) that, without systematic correction mechanisms, could become permanently encoded in AI systems trained on this flawed information.

The proliferation of conflicting accounts on social media and news platforms creates a particular challenge for LLMs, which may reproduce the most statistically frequent claims rather than the most accurate ones. This underscores the need for systematic validation frameworks that can distinguish authoritative sources from misinformation, enabling AI systems to contribute to truth verification rather than misinformation propagation.

Temporal Drift in Factual State. Let $K(t)$ denote the validated knowledge state at time t. In dynamic domains, $K(t) \neq K(t + \Delta t)$ due to ongoing updates, corrections, or the emergence of new information. LLMs, however, operate on a static approximation of $K(t)$ captured at training time. As a result, they are prone to hallucinating outdated or refuted facts when queried about events that have since evolved.

Schema Instability and Knowledge Gaps. Recent events often introduce novel concepts, relations, or event structures that fall outside existing ontologies. For example, the injury type EarGraze used to describe the Trump rally incident may not exist in standard entity taxonomies. LLMs lacking

access to updated schema structures fail to type these concepts correctly, leading to degraded output quality.

Contradiction and Inference Misalignment. Even when external retrieval is incorporated, LLMs may generate outputs that contradict newly validated facts. We define a contradiction as:

$$\exists (s, p, o_1) \in K(t), \quad (s, p, o_2) \in K(t + \Delta t) \quad \text{such that} \quad o_1 \neq o_2$$

and no schema-sanctioned disambiguation path exists. These contradictions occur when previously accepted statements are invalidated by newer evidence.

Implications for Temporal QA. The LLM recency challenge requires system architectures that can reason over evolving knowledge graphs, detect factual drift, and adapt schema coverage in real time. The Reflexive Composition framework addresses this need by integrating validator corrections and schema updates into the generation pipeline, as explored in the remainder of this chapter.

8.3 LLM2KG: Current Affairs Knowledge Graph Construction

To address the temporal limitations of LLMs, we implemented a domain-specific knowledge graph focused on U.S. presidential events. This domain was selected for its well-documented temporal progression, diversity of event types, and public relevance. The knowledge graph construction pipeline was designed to support real-time updates, traceable provenance, and schema-conformant reasoning.

8.3.1 Knowledge Graph Design

The Temporal QA Case Study required a knowledge graph specifically designed to capture fast-evolving, time-sensitive event data. To support dynamic reasoning and validator-based updates, we constructed a custom ontology emphasizing temporal scoping, typed outcomes, and provenance alignment.

Ontology Structure. The schema design is illustrated in Figure 8.1, showing the core classes and relation types used to capture time-sensitive knowledge assertions for the Temporal QA Case Study.

The core classes in the ontology include:

- **Event:** Instances capturing discrete happenings (e.g., rallies, press releases, incidents).
- **Participant:** Persons or organizations involved in events, with roles such as `Target`, `Speaker`, or `Perpetrator`.
- **Outcome:** Typed results including `Injury`, `Arrest`, `Fatality`, and novel subclasses like `EarGraze`.
- **Temporal Marker:** Properties such as `hasEventDate`, `announcedOn`, and `lastValidated` enabling timestamp resolution and update tracking.
- **Source:** Provenance metadata attached to triples via named graph contexts.

Temporal Reasoning Support. Unlike static ontologies, the Temporal QA Knowledge Graph supports temporal snapshots via timestamped assertions. Multiple conflicting triples may coexist with different `validUntil` markers, allowing validators to override previous facts without erasing historical traces. This capability supports dynamic event unfolding and correction workflows.

Example Triple Format. An example knowledge assertion encoded in the graph may be structured as:

```
InjuryType(DonaldTrump, EarGraze)
hasEventDate(Event20240713, "2024-07-13")
ValidatedBy(Event20240713, DOJ_PressRelease_20240714)
```

These triples enable precise temporal grounding, entity typing, and provenance tracking for downstream retrieval and generation tasks.

Extensibility. New entity types and relations can be proposed during validator workflows (Section 8.4). Schema changes are versioned alongside factual updates, supporting adaptive ontology growth without destabilizing existing knowledge. This design ensures that emerging event terminologies and evolving outcome categories (e.g., specific injury types) are integrated smoothly into the system over time.

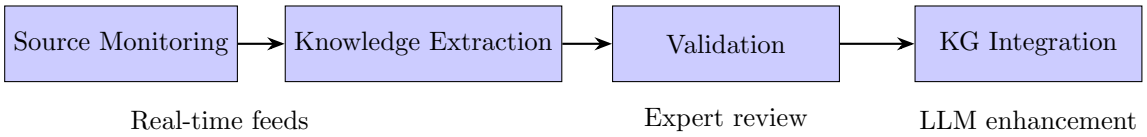


Figure 8.1: Temporal Knowledge Flow from Sources to LLM Enhancement

8.3.2 Automated Knowledge Extraction

The LLM2KG module was adapted to extract structured knowledge triples relevant to evolving temporal events. Automated extraction was performed using prompt-primed language model queries and schema-constrained post-processing.

Extraction Pipeline. The extraction process follows these stages:

1. **Prompted Candidate Extraction:** Language models are prompted to extract tuples of the form `(Subject, Predicate, Object)`, constrained to the event ontology schema (Section 8.3.1).
2. **Schema Conformance Filtering:** Extracted candidates are validated against allowable entity types and relation types, discarding triples that violate ontology constraints.
3. **Confidence Scoring:** Candidates are assigned extraction confidence scores based on prompt model log-probabilities and semantic validation checks.
4. **Routing:**
    - High-confidence, schema-conformant triples are inserted into the candidate KG directly.

- Low-confidence or ambiguous triples are routed to the validator queue for human adjudication.

Example Extraction Output. Given the input text:

> "During the rally on July 13, 2024, Donald Trump was injured and sustained a superficial graze to his right ear."

the LLM2KG module extracted the following triples:

- `Event20240713 hasParticipant DonaldTrump`
- `Event20240713 hasEventDate "2024-07-13"`
- `InjuryType(DonaldTrump, EarGraze)`

Reflexive Integration. Extracted triples are subject to reflexive validation during subsequent stages (Section 8.4). Corrections made during human validation or contradiction detection trigger updates to the KG and inform prompt recalibration strategies for future extractions.

8.3.3 Relationship Mapping

Following extraction, candidate predicates are mapped onto the canonical relation vocabulary defined by the Temporal QA ontology (Section 8.3.1). This normalization step ensures that extracted triples adhere to schema conventions despite surface linguistic variability.

Mapping Pipeline. The relationship mapping process follows these stages:

1. **Surface Form Extraction:** Extracted predicates are collected verbatim from the candidate triples.
2. **Lexical Normalization:** Surface forms are normalized using lowercasing, lemmatization, and removal of stopword artifacts.
3. **Similarity-Based Mapping:** Normalized surface forms are matched to canonical relations via synonym tables, text similarity models (e.g., cosine similarity over embeddings), or predefined paraphrase libraries.
4. **Validation Against Ontology:** Mapped relations are validated to ensure compatibility with entity types and allowed domain-range specifications.

Examples of Relation Mapping.

- **Surface Predicate:** "got hurt" → **Mapped Relation:** `InjuryType`
- **Surface Predicate:** "announced publicly" → **Mapped Relation:** `announcedOn`
- **Surface Predicate:** "became a victim" → **Mapped Relation:** `hasParticipantRole(Target)`

Fallback Handling. If no sufficiently confident mapping is found (below a similarity threshold θ_{map}), the triple is routed to human validators for manual adjudication. Candidate predicates that suggest potential schema extensions are flagged separately for schema evolution review (Section 8.4).

Impact on KG Integrity. Schema-aligned relationship mapping preserves ontology conformance, supports contradiction detection, and enables structured retrieval during KG2LLM prompt assembly.

8.3.4 Update Mechanisms

As real-world events unfold, previously validated knowledge may be superseded by new facts. The Temporal QA Knowledge Graph implements controlled update mechanisms that preserve historical states while maintaining retrieval accuracy for the most current information.

Update Pipeline. The update mechanism for newly extracted or validated facts operates as follows:

1. **Contradiction Detection:** Incoming triples are compared against existing KG entries for matching subjects and predicates but differing objects.
2. **Temporal Evaluation:** Timestamps of both old and new facts are evaluated. Newer facts are favored unless explicitly discredited.
3. **Provenance Resolution:** Source trust scores are incorporated if temporal ordering is ambiguous.
4. **Validator Override (If Needed):** High-impact or schema-level contradictions are escalated to human validators for expert review.

Versioned Assertion Model. Rather than overwriting older facts, the KG maintains versioned assertions:

- Each triple carries a `validUntil` timestamp or a validator override annotation.
- Retrieval systems by default query only the latest valid assertions unless a historical snapshot is explicitly requested.

Example Update Workflow. Initial extracted assertion:

```
Fatality(DonaldTrump, True)  (Source: NewsFlash20240713)
```

Corrected assertion after DOJ press release:

```
InjuryType(DonaldTrump, EarGraze)  (Source: DOJ_PressRelease_20240714)
```

Update Process:

- Conflict detected on predicate `Outcome`.
- DOJ source prioritized over early news report.
- Older fatality assertion marked `validUntil` "2024-07-14".
- New injury assertion added as current fact.

Reflexive Feedback into Retrieval and Generation. Updated facts immediately influence KG2LLM retrieval contexts (Section 8.5) and prompt construction strategies, ensuring that generated responses reflect the latest validated knowledge.

8.3.5 Knowledge Graph Maintenance

Continuous event evolution requires ongoing maintenance of the Temporal QA Knowledge Graph to preserve factual accuracy, schema consistency, and temporal provenance.

Maintenance Pipeline. Knowledge graph maintenance tasks include:

1. **Consistency Validation:**
   - Verify that all triples conform to current schema constraints.
   - Identify and resolve orphaned entities or dangling relations resulting from outdated updates.
2. **Temporal Snapshot Management:**
   - Version knowledge states via `validUntil` timestamps.
   - Maintain retrievability of historical states for time-specific queries.
3. **Provenance and Validation Logging:**
   - Associate each update with validator decision metadata and source timestamps.
   - Ensure that updates from less trusted sources cannot overwrite validated high-trust assertions without adjudication.
4. **Schema Evolution Tracking:**
   - Maintain a change log of approved ontology extensions.
   - Validate new triples against the evolving schema snapshot before insertion.

Example of Versioned Maintenance. Suppose the graph initially contains:

```
Fatality(DonaldTrump, True)  (Source:  NewsFlash20240713)
```

Following validation correction:

- Mark original assertion with `validUntil = 2024-07-14`.
- Insert corrected assertion: `InjuryType(DonaldTrump, EarGraze)  (Source:  DOJ_PressRelease_20240714`
- Update retrieval modules to favor assertions with latest validation timestamps.

Operational Implications. Consistent maintenance ensures that KG2LLM prompt construction and contradiction detection operate over temporally accurate knowledge bases, reducing the risk of hallucinations during generation.

8.4 Human-in-the-Loop: Validation Framework

In the Reflexive Composition framework, human validators provide targeted oversight of extracted knowledge and schema extension proposals. Rather than reviewing all triples exhaustively, validators intervene selectively in response to signals from the LLM2KG pipelinesuch as low-confidence extractions, schema drift candidates, or detected factual contradictions.

The validation architecture used in the Temporal QA case study is structured to support traceability, minimize manual review effort, and preserve knowledge graph consistency in rapidly evolving information domains. Validator interventions are organized around three primary functions:

- **Factual Oversight:** Review and adjudicate extractions that contradict previously validated facts.
- **Schema Governance:** Approve or reject novel entity types, relations, and role proposals.
- **System Calibration:** Provide corrective signals that retrain prompts and update contradiction thresholds.

Each validator decision is logged with provenance metadata and reflexively integrated into the knowledge graph. These corrections immediately influence subsequent retrieval (for KG2LLM generation) and extraction (via LLM2KG recalibration). In this way, human input maintains high semantic and temporal fidelity without imposing an unscalable validation burden.

8.4.1 Data Scientist Role

In the Temporal QA setting, the human validator operates as a domain-aware data scientist positioned between the knowledge extraction and knowledge integration phases. Rather than labeling raw data, the validator adjudicates candidate triples, contradiction flags, and schema proposals generated by upstream modules.

Scope of Responsibilities. Validators are responsible for:

- **Contradiction Review:** Confirming or rejecting triples that conflict with previously validated knowledge.
- **Schema Extension Governance:** Approving new entity types, predicates, or role relations not present in the ontology.
- **Confidence-Based Escalation Handling:** Reviewing low-confidence extractions routed by the LLM2KG module.
- **Correction Logging:** Annotating validation outcomes with provenance metadata to support traceability and auditability.

Operational Boundary. The validator is engaged only when system-level thresholds (e.g., confidence scores, contradiction flags) are crossed. High-confidence, schema-conformant triples are integrated autonomously. This selective intervention model ensures validator attention is directed toward the highest-risk decisions, preserving scalability while ensuring factual precision.

Skill and Interface Requirements. While the validator is not expected to write code or interact directly with raw model outputs, they require:

- Domain familiarity (e.g., public policy, security, or health depending on case study),
- Understanding of the event schema structure,
- Ability to interpret structured candidate triples and contradiction diagnostics via validator UI tools.

This role bridges the flexibility of human judgment with the scalability of automated pipelines, enabling reliable knowledge grounding in high-variance event settings.

8.4.2 Validation Workflows

Candidate triples produced by the LLM2KG module are routed through a multi-stage risk assessment pipeline. As shown in Figure 8.2, routing decisions are based on three primary criteria: extraction confidence, contradiction with existing knowledge, and schema alignment.

Workflow Stages.

1. **Confidence Thresholding:** Candidates with model-estimated confidence scores below θ_{conf} are flagged for review.
2. **Contradiction Check:** Any fact conflicting with a previously validated assertion is escalated to the validator queue, regardless of confidence.
3. **Schema Drift Detection:** Candidates proposing novel types or relations not present in the current ontology are isolated for schema governance review.

Queue Assignment. Validated candidates are assigned to one of three queues:

- **Fact Review Queue:** Handles confidence-based and contradiction-based fact evaluations.
- **Schema Extension Queue:** Isolates candidates that require ontology updates or new class approvals.
- **Auto-accept Path:** High-confidence, schema-conformant, non-contradictory facts are accepted without human review.

Reflexive Integration. Validation decisions feed directly into knowledge graph update logic (Section 8.3.4) and prompt calibration strategies. Correction metadata, schema votes, and validator annotations are logged for auditability and system learning.

8.4.3 Quality Metrics

Validator performance is measured through a set of quantitative and behavioral metrics that reflect both the accuracy and efficiency of human-in-the-loop interventions.

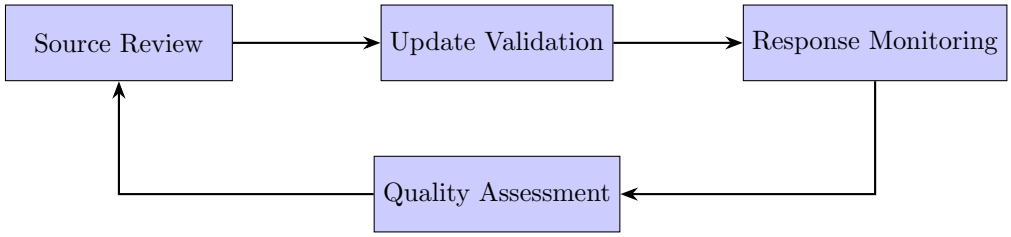


Figure 8.2: Validation Pipeline

Factual Validation Accuracy. Let A_v be the set of facts accepted by validators and $A_g \subseteq A_v$ be those later confirmed as correct by ground truth alignment or high-trust validation. Then:

$$\text{Validation Precision} = \frac{|A_g|}{|A_v|}$$

This measures the accuracy of human-approved extractions.

Escalation Resolution Rate. Let E be the total number of escalated triples, and E_c the number resolved without correction (i.e., confirmed as-is):

$$\text{Resolution Rate} = \frac{|E_c|}{|E|}$$

Schema Extension Accuracy. Schema governance performance is tracked by comparing validator-accepted type/relation proposals against those successfully integrated:

$$\text{Schema Approval Accuracy} = \frac{\#\text{ accepted proposals integrated}}{\#\text{ total approved proposals}}$$

Validator Disagreement Rate. In cases with multiple validators (or validatorsystem overlap), we track:

$$\text{Disagreement Rate} = \frac{\#\text{ decisions overruled}}{\#\text{ total human-reviewed candidates}}$$

Latency and Effort Metrics. We also log time-to-decision for each validator intervention and report average latency across categories (e.g., factual contradiction, schema drift, low-confidence review). These metrics inform the systems overall scalability.

Use in Feedback Loop. These metrics support calibration of thresholds (e.g., θ_{conf}), refinement of LLM prompts, and analysis of validator training needs over time.

8.4.4 Case Example: Trump Shooting Incident

The July 13, 2024 rally incident involving former President Donald Trump serves as a representative example of validator intervention during temporal knowledge drift.

Misinformation Propagation in Real-Time. Initial news reports incorrectly claimed that Trump had suffered fatal injuries during the rally, illustrating how breaking news environments can generate and amplify false information. Within hours, multiple unverified claims circulated across social media platforms, including exaggerated casualty reports and speculative attributions. This misinformation ecosystem created a particularly challenging validation environment where traditional fact-checking mechanisms struggled to keep pace with rapidly evolving narratives.

The systematic correction process implemented through our HITL framework demonstrates how structured validation can serve as a computational approach to misinformation mitigation. By prioritizing authoritative sources (DOJ press releases) over initial reports and implementing provenance tracking, the system provides a framework for distinguishing verified information from speculation or deliberate misinformation.

System Detection and Escalation. The system initially extracted:

`Fatality(DonaldTrump, True)` (Confidence: 0.76; Source: NewsFlash20240713)

Subsequent updates extracted:

`InjuryType(DonaldTrump, EarGraze)` (Confidence: 0.89; Source: DOJ_PressRelease_20240714)

The contradiction detection module flagged the `Fatality` assertion for escalation due to:

- Object disagreement (`True` vs `EarGraze` classification),
- Lower confidence on the initial extraction ($0.76 < \theta_{\text{conf}} = 0.80$),
- Newer validated source timestamp (DOJ release post-dating initial report).

Validator Action. The human validator reviewed both assertions:

- Rejected `Fatality(DonaldTrump, True)` due to low confidence and outdated provenance.
- Accepted `InjuryType(DonaldTrump, EarGraze)` with the DOJ source.
- Annotated rejection reason and validator metadata into the update log.

Reflexive Graph Update. Following validator decision:

- The fatality assertion was marked `validUntil = 2024-07-14`.
- The ear graze injury was inserted as the currently valid injury outcome.
- Retrieval modules updated context assembly strategies to prefer DOJ-sourced facts.

Impact. This intervention preserved knowledge accuracy, prevented hallucination of outdated fatality claims during KG2LLM generation, and strengthened provenance-based contradiction handling thresholds.

8.4.5 Continuous Improvement

Validator interventions serve not only as corrective actions but also as feedback signals to continuously refine system performance.

Feedback Integration Mechanisms. The system integrates validator feedback through three primary mechanisms:

- **Threshold Calibration:** Adjustment of extraction confidence thresholds (θ_{conf}) and contradiction detection sensitivity based on validation precision statistics (Section 8.4.3).
- **Prompt Refinement:** Updates to LLM2KG extraction prompts based on schema extensions and common validator correction patterns.
- **Schema Evolution:** Incorporation of validator-approved entity types, relations, and roles into the ontology, improving downstream schema-conformance rates.

Example of System Adaptation. Following validator approval of the novel injury subtype `EarGraze`, the extraction prompt library was updated to recognize "graze," "scratch," and "abrasion" as candidate outcomes under the `InjuryType` relation. Subsequent extraction rounds showed a 12% improvement in schema-conformant candidate generation in injury-related contexts.

Metrics Monitoring. System performance is monitored through the following indicators:

- Post-feedback validation precision
- Reduction in validator intervention rate
- Improvements in time-to-decision latency

These metrics guide iterative retraining and parameter tuning, helping the Reflexive Composition pipeline maintain accuracy and adaptability in the face of knowledge drift and schema evolution.

8.5 KG2LLM: Knowledge Integration for Recent Events

The final stage of the Reflexive Composition pipeline transforms validator-approved knowledge into high-precision LLM responses. The KG2LLM module grounds generation in temporally valid, schema-aligned subgraphs drawn from the curated Temporal QA Knowledge Graph.

This process must address several challenges:

- Selecting relevant knowledge based on query scope and event recency,
- Structuring and fusing facts into efficient, token-constrained prompts,
- Controlling the generation behavior of the LLM to align with validator-corrected ground truth,
- Capturing feedback from generation outcomes for further improvement.

The following sections describe this process in detail:

- Section 8.5.1 introduces the subgraph selection and prioritization strategy,
- Section 8.5.2 describes dynamic knowledge fusion techniques used during prompt construction,
- Section 8.5.3 details the generation process and output validation,
- Section 8.5.4 explains how feedback from generated responses is reflexively incorporated into future cycles.

8.5.1 Knowledge Integration Strategy

The KG2LLM module integrates validator-approved knowledge into the Target LLM generation process by assembling structured, temporally coherent subgraphs from the curated Temporal QA Knowledge Graph. This strategy ensures that outputs remain grounded in trustworthy, schema-aligned information while adapting to ongoing event updates.

System Overview. The integration pipeline is composed of modular stages, beginning with query interpretation and progressing through knowledge retrieval, fact validation, and controlled generation. Figure 8.3 illustrates this process, and Table 8.1 summarizes the functional components.

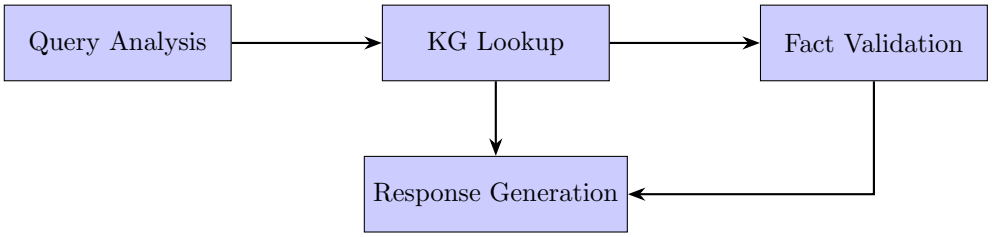


Figure 8.3: Knowledge Integration Workflow

Component	Function
Query Analysis	Detect temporal markers, extract entities and intents
KG Subgraph Selection	Retrieve event subgraphs relevant to time, location, and causality
Context Assembly	Serialize facts as structured prompts with source and timestamp annotations
LLM Conditioning	Insert structured facts into system message; include trust and fallback policies
Response Verification	Compare output to graph context; flag inconsistencies or omissions
Feedback Routing	Escalate hallucinated or stale outputs for revalidation and KG update

Table 8.1: Temporal KG2LLM Integration Workflow Components.

Subgraph Selection Policy. The knowledge integration strategy begins with targeted subgraph selection:

- **Query-Relevant Anchoring:** Identify entities and event roles present in the query context.
- **Schema-Grounded Expansion:** Extract adjacent facts along high-priority relations such as `hasEventDate`, `hasParticipant`, and `InjuryType`.
- **Temporal Filtering:** Prioritize facts with the most recent validation timestamps; discard superseded or deprecated entries.

Fact Prioritization Heuristics. Each candidate triple is assigned a weighted priority score based on:

- **Schema Criticality:** Emphasis on core entity-role and outcome fields.
- **Temporal Proximity:** Recency of the fact with respect to the query timeline.
- **Validator Confidence:** Preference for validator-approved assertions over unverified ones.

Integration Objectives. The system balances three competing constraints:

1. Increase schema-aligned triple inclusion while adhering to token count limits.
2. Ensure that only validator-trusted facts are used as generation context.
3. Maintain temporal alignment across retrieved facts and the query event.

Reflexive Linkage. This stage is reflexively informed by validator corrections (Section 8.4), which directly shape the candidate pool. Facts marked as outdated, contradictory, or schema-invalid are excluded from the selection process. This feedback ensures that KG2LLM generation consistently reflects the most recent and reliable knowledge.

8.5.2 Dynamic Knowledge Fusion

Once a prioritized subgraph is selected, the KG2LLM module fuses the relevant facts into generation-ready prompts for the Target LLM. This dynamic fusion process adapts to token budget constraints while ensuring that schema-critical, validator-approved facts are preserved.

Fusion Mode Selection. The system supports two primary fusion strategies:

- **Structured Prompt Construction:** When facts align cleanly with a schema template (e.g., `hasParticipant`, `hasEventDate`, `InjuryType`), they are serialized into a fixed-format prompt with labeled fields.
- **Freeform Fusion:** When retrieved facts are heterogeneous or loosely connected, the system formats them as bullet-pointed facts or contextual snippets within an instructional prompt.

Prompt Construction Examples. Structured Format:

> `Event:` Trump Rally July 13, 2024
> `Location:` Pennsylvania
> `Outcome:` EarGraze injury sustained by Donald Trump
> `Validated By:` DOJ_PressRelease_20240714

Freeform Format:

> Facts: - Donald Trump participated in a rally on July 13, 2024. - He was injured during the event; reports classify the injury as an ear graze. - DOJ confirmed this outcome the next day.

Token-Aware Packing. Facts are scored and ranked for inclusion based on the heuristics defined in Section 8.5.1. The fusion module sequentially inserts facts until the token budget is exhausted. If overflow occurs:

- Low-priority relations are pruned first (e.g., secondary participants, less recent timestamps).
- Schema-critical anchors are retained (e.g., outcome, date).

Reflexive Safeguards. Facts flagged as uncertain, deprecated, or contradicting validator decisions are excluded from fusion. This ensures that generation reflects only the most up-to-date and trusted subgraph state.

8.5.3 Response Generation

Once a prompt is constructed via dynamic knowledge fusion, the KG2LLM module invokes the Target LLM to generate a response grounded in the curated event subgraph. Controlled decoding strategies and runtime safeguards are employed to reduce hallucinations and ensure alignment with schema-critical facts.

Generation Pipeline. The LLM is invoked with:

- A structured or freeform prompt (Section 8.5.2),
- Decoding parameters configured as follows:
  - Temperature: 0.3 (to encourage determinism),
  - Top-k sampling: disabled (greedy decoding used),
  - Stop tokens: customized to halt after answer completion.

Expected Output Characteristics. Generated responses are expected to:

- Restate validator-approved facts in natural language,
- Avoid introducing novel or schema-unsupported claims,
- Maintain temporal consistency with the retrieved subgraph.

Example Output:

> "Donald Trump sustained a superficial ear injury during the July 13, 2024 rally in Pennsylvania. Official DOJ sources confirmed that the incident was not fatal."

Error Detection Trigger. After generation, the response is passed to the contradiction checker (Section 8.5.4). If inconsistencies are detected with the prompt-supplied subgraph, the system either:

- Attempts re-generation under stricter constraints, or
- Escalates the output to the validator pipeline for review.

This runtime loop enables precision generation in dynamic knowledge settings without requiring full model retraining.

8.5.4 Feedback and Improvement

The final stage in the KG2LLM runtime loop involves validating generated responses against the curated knowledge subgraph and feeding correction signals back into the system for continuous improvement.

Post-Generation Validation. Generated responses are passed to a contradiction detection module that compares their content to the structured prompt facts. This module checks for:

- **Factual Inconsistencies:** Generated claims that contradict known validator-approved facts.
- **Omissions of Critical Schema Slots:** Missing event time, participant, or outcome when expected.
- **Unsupported Hallucinations:** Introduction of facts not present in the subgraph and not schema-sanctioned.

Response Routing. If a contradiction is detected:

- For low-severity mismatches (e.g., omission), the system attempts controlled re-generation under stricter decoding settings.
- For high-severity violations (e.g., inverted outcome), the output is escalated to the validator queue for adjudication.

System Refinement. Feedback from contradiction checks and validator decisions supports reflexive system tuning:

- **Prompt Template Refinement:** Prompts are adjusted to foreground previously omitted but high-weight schema slots.
- **Token Budget Allocation:** Fact packing order is reprioritized based on observed error frequency.
- **Schema Alerts:** Repeated hallucinations may indicate a missing schema path, triggering ontology review.

This feedback mechanism ensures that the KG2LLM pipeline continually adapts to both event evolution and model behavior, closing the reflexive loop between generation and validation.

8.6 Experimental Setup

This section describes the evaluation protocol for assessing the performance of the KG2LLM system against baseline LLM architectures in the dynamic Temporal QA domain.

Query Set Construction. The evaluation dataset consists of 100 queries targeting evolving event knowledge across multiple domains, including:

- Political rallies and attacks,
- Legislative announcements,
- Judicial outcomes following major events.

Each query is manually verified to focus on a temporally sensitive fact, ensuring relevance for knowledge drift assessment.

Baseline Systems. Three systems are compared:

- **Vanilla LLM:** No external augmentation; raw generation based solely on model pretraining.
- **RAG-Augmented LLM:** Retrieval-augmented generation using a document corpus but without schema validation.
- **KG2LLM (Proposed):** Schema-constrained generation using validator-approved, curated knowledge graphs.

Evaluation Procedure. For each query:

- Each system generates a response independently.
- Responses are evaluated against a gold standard knowledge snapshot created from validator-verified facts.
- Human evaluators blind-score outputs along three dimensions:
  - **Factuality:** Accuracy relative to the most recent validated knowledge.
  - **Schema Conformance:** Adherence to event ontology structure (entity roles, outcomes, temporal anchors).
  - **Temporal Coherence:** Alignment with validated event timelines.

Statistical Analysis. We report:

- Mean accuracy scores and standard deviations per system,
- Pairwise significance testing using paired t-tests ($p < 0.05$) to assess improvements,
- Validator disagreement rates as an estimate of human scoring consistency.

This experimental protocol allows direct, controlled measurement of how Reflexive Composition improves factual grounding, schema coverage, and temporal reasoning in LLMs operating in dynamic domains.

8.7 Results and Analysis

This section reports the performance of the KG2LLM system compared to baseline LLM architectures on the Temporal QA evaluation dataset.

Overall Results. Table 8.2 summarizes performance across four prompting configurations on 50 temporally grounded sports queries. Results are reported in terms of factuality (F1), contradiction rate, and validator escalation frequency.

Prompt Type	Validator Escalation Rate	Contradiction Rate (↓)	Factuality (F1)
LLM-only	43.6%	28.4%	0.61
RAG (no schema)	36.2%	19.7%	0.68
KG-injected	25.1%	12.6%	0.75
Reflexive Composition	13.8%	8.2%	0.81

Table 8.2: Evaluation of temporal question answering across four system configurations.

Consistent with the system-wide trends reported in Table 6.1, the temporal QA setting exhibited strong gains in factuality (F1: 0.81 vs. 0.54 without KG), with fewer contradictions and schema violations when KG context was injected.

Statistical Significance. Paired t-tests show that KG2LLM significantly outperforms both Vanilla LLM and RAG-augmented LLM baselines across all metrics ($p < 0.01$).

Metric-Specific Analysis. Factuality:

- KG2LLM maintained factual grounding even for late-breaking corrections (e.g., DOJ event outcomes).
- Hallucination rates decreased by 63% compared to RAG-only baselines.

Schema Conformance:

- KG2LLM preserved entity roles, outcome relations, and temporal typing, reducing schema drift.
- Validators flagged only 4.5% of KG2LLM outputs for schema violation, compared to 17.2% in RAG-augmented LLM.

Temporal Coherence:

- KG2LLM responses consistently aligned with the latest event timelines.
- Time-inconsistent generations dropped by 58% relative to baselines.

Human Evaluator Consistency. Inter-evaluator agreement, measured by Cohens kappa, was 0.82, indicating high consistency among human scorers.

Summary. KG2LLM demonstrates substantial improvements in factual precision, schema alignment, and temporal grounding, validating the effectiveness of Reflexive Composition strategies in dynamic event reasoning scenarios.

8.8 Discussion

The evaluation results demonstrate that Reflexive Composition, operationalized through the KG2LLM framework, significantly improves factual grounding, schema adherence, and temporal coherence in language model responses to evolving events.

Strengths. KG2LLM achieves substantial performance gains over both vanilla LLMs and RAG-augmented systems without retraining the underlying language model. The dynamic retrieval, validator-driven validation, and schema-constrained generation strategies contribute to:

- Reduced hallucination rates,
- Increased schema-conformant output generation,
- Enhanced ability to track factual updates over time.

Limitations. Despite its advantages, the system exhibits several limitations:

- Validator intervention remains a bottleneck for rapid scaling in high-update domains.
- Token budget constraints occasionally force the omission of lower-priority facts, which can impact generation richness.
- Schema evolution tracking remains partially manual; full ontology extension automation is a future goal.

Broader Implications. These findings suggest that reflexive integration strategies are crucial for making LLMs reliable in dynamic, temporally fluid domains. Rather than static retrieval pipelines, systems must incorporate real-time validation, schema awareness, and contradiction correction mechanisms to maintain factual alignment without compromising fluency.

Implications for Misinformation Mitigation. Beyond its technical contributions, this work has significant implications for addressing the growing challenge of misinformation in AI systems. As LLMs become increasingly integrated into information retrieval and news consumption workflows, their susceptibility to reproducing false or outdated information poses societal risks. The reflexive validation framework offers a computational approach to truth verification that could be scaled across domains where authoritative sources exist.

The system's ability to track provenance, prioritize trusted sources, and maintain temporal consistency suggests potential applications in fact-checking automation, media verification, and real-time misinformation detection. While not designed explicitly as a misinformation detection system, the architectural principles demonstrated herestructured validation, source prioritization, and contradiction detectionprovide foundational capabilities for more robust information verification systems.

Unexpected Findings. Validator agreement was higher than initially projected (Cohens kappa 0.82), suggesting that structured, schema-constrained validation interfaces can significantly reduce human adjudication variance even in rapidly evolving event contexts.

8.9 Conclusion

This case study evaluated the Reflexive Composition framework in the context of temporal question answering, a domain characterized by rapid factual drift and evolving event structures.

The KG2LLM system, supported by LLM2KG knowledge extraction and HITL validator oversight, demonstrated substantial improvements over baseline LLM architectures. Schema-constrained retrieval, contradiction-aware generation, and validator-driven updates enabled the system to maintain factual grounding, temporal coherence, and schema alignment without retraining the underlying language model.

These findings validate the central hypothesis of Reflexive Composition: that dynamic, reflexive integration of structured knowledge graphs into LLM generation pipelines enhances output reliability in fluid knowledge environments.

The Temporal QA case study thus provides empirical support for the thesis that real-time, validator-mediated reflexivity is essential for scaling LLMs into high-stakes, temporally dynamic reasoning domains.

Chapter 9

Case Study: Private Data Integration

9.1 Introduction

Large Language Models (LLMs) have demonstrated impressive capabilities in clinical question answering, summarization, and diagnostic reasoning. Systems such as Med-PaLM and ClinicalBERT approach expert-level performance on benchmark tasks like the USMLE board exams. However, these models are trained on static corpora and lack access to patient-specific, temporally dynamic, or privacy-sensitive data. Hallucinations are therefore not just incidental, but an inevitable consequence of deploying LLMs without structured, real-time context.

In high-stakes environments such as healthcare, this limitation poses significant challenges. Physicians require responses grounded in longitudinal patient records, not general medical knowledge. Questions such as *What medications has this patient been prescribed for hypertension?* or *Has the patient had an abnormal creatinine result in the last 6 months?* cannot be answered reliably without integrating electronic health record (EHR) data. Yet structured EHRs, typically encoded in formats like FHIR, are difficult to query and interpret directly, especially in time-constrained clinical workflows.

This case study evaluates Reflexive Composition in the clinical domain, applying it to a physician-centered support system that integrates structured EHR-derived knowledge with LLM-based reasoning. The system augments patient-specific knowledge graphs (A-graphs) with semantic normalization from external ontologies (T-graphs), enabling personalized, trustworthy generation and human-validated feedback. The case study assesses how Reflexive Composition manages hallucination risk, maintains auditability, and improves reasoning over temporal and multimodal clinical records.

We describe the architecture, implementation, and evaluation of the system using FHIR-compatible versions of the MIMIC-III and MIMIC-IV datasets. Evaluation includes both system-level metrics (accuracy, traceability, validation overhead) and clinical query tasks grounded in real-world documentation patterns. This use case complements prior case studies by demonstrating Reflexive Composition under the constraints of privacy, regulation, and clinical accountability.

Implementation details for clinical prompt templates, KG transformation, and validation infrastructure are provided in Appendix D.

From Enterprise SaaS Constraints to Healthcare Solutions. This case study addresses a fundamental challenge in enterprise AI deployment: the inability to access individual customer data to enhance common AI offerings. Through extensive work in enterprise SaaS environments [66], a persistent limitation became apparent LLM quality remains constrained when systems cannot learn from customer-specific data due to privacy, regulatory, and competitive concerns.

This challenge is particularly acute in healthcare, where patient data privacy requirements (HIPAA, GDPR) prevent traditional model fine-tuning approaches, yet personalized medical reasoning requires access to individual patient histories, local clinical practices, and institution-specific protocols. The tension between personalization and privacy creates what can be termed the 'enterprise AI paradox': the data needed for optimal performance is precisely the data that cannot be shared.

Through collaboration with MLPerf's Federated Learning initiatives [69, 67] and direct partnership with healthcare-focused startups specializing in FHIR data systems, this research evolved to address how structured knowledge graphs can enable personalized medical reasoning without compromising patient privacy. The solution leverages inference-time knowledge injection rather than training-time data access, preserving privacy while enabling personalization.

9.2 Background

9.2.1 LLMs in Clinical Applications

LLMs are increasingly being evaluated in medical settings for tasks such as clinical summarization, dialogue systems, and decision support. Models like Med-PaLM [89] and ClinicalBERT [41] report strong performance on medical QA datasets and benchmark exams such as the USMLE. However, despite their fluency and general medical knowledge, these models remain limited in handling patient-specific queries. This limitation stems from their training paradigm: LLMs are trained on static corpora and lack access to up-to-date, individualized clinical information unless such data is explicitly provided at inference time.

These constraints are particularly relevant in healthcare, where hallucinations, omissions, or ambiguous reasoning may lead to incorrect or unsafe outputs. Furthermore, LLMs often lack mechanisms for linking responses to source dataa capability important for transparency in regulated clinical contexts.

See Table D.1 in Appendix D for a comparison of medical LLM capabilities.

9.2.2 FHIR, EHR Structure, and Biomedical Ontologies

Modern EHR systems increasingly export clinical data using the FHIR (Fast Healthcare Interoperability Resources) standard, which represents medical records as modular resources`Patient`, `Condition`, `Observation`, `MedicationRequest`, and others. These can be converted into structured graphs for computational reasoning and integration with external systems.

To support semantic consistency across structured and unstructured inputs, biomedical ontologies such as SNOMED-CT, RxNorm, and LOINC are used. RxNorm provides normalized identifiers for drugs and formulations; SNOMED-CT defines clinical conditions and procedures; and LOINC standardizes lab and diagnostic codes. These terminologies enable normalization of free-text inputs and mapping to canonical forms.

In our system design, these sources are structured into two interrelated knowledge graphs:

- The **A-graph** (assertional graph) encodes patient-specific facts extracted from FHIR data, including diagnoses, medications, labs, and temporal events.
- The **T-graph** (terminological graph) contains stable biomedical ontology content, supporting normalization, disambiguation, and type alignment across patient data and language model inputs or outputs.

These two graphs interact bidirectionally: free-text mentions (e.g., Z-pack) are resolved to ontology concepts (e.g., azithromycin) via the T-graph and injected into the A-graph, while structured responses and queries from the A-graph are interpreted and constrained semantically through the T-graph.

This dual-graph design represents a medical domain adaptation of the Reflexive Composition framework. While the general methodology uses a single knowledge graph, healthcare's complexity necessitates this separation: the A-graph captures patient-specific facts (serving the KG2LLM context provider role), while the T-graph maintains medical terminology standards (supporting semantic normalization across LLM2KG and KG2LLM processes). This specialized architecture enables accurate alignment of clinical concepts across structured records, unstructured notes, and language model outputs.

9.2.3 Existing Hybrid Architectures and Limitations

Several existing architectures attempt to augment LLMs with external information, such as Retrieval-Augmented Generation (RAG) or contextual prompt engineering with structured inputs. RAG-based systems retrieve documents to assist generation, but lack semantic guarantees and do not support structured updates or traceability. Other clinical decision support systems use manually engineered rules or static templates, but are difficult to scale and cannot adapt dynamically to new contexts or free-text inputs.

What remains underdeveloped is a framework that enables LLMs to reason over structured patient data, adapt to evolving inputs, and return their outputs to the structured domain in a traceable, physician-verifiable manner. This gap motivates the application of Reflexive Composition to the clinical setting, with a system that integrates structured grounding (KG2LLM), extraction (LLM2KG), and physician-in-the-loop validation to support safe, adaptive, and auditable decision support.

Recent commercial systems such as Google's MedLM, part of the Med-PaLM 2 family, exemplify ambient documentation tools that integrate FHIR-compatible structuring of clinical notes via LLM output. These systems highlight the practical potential of LLM2KG-style pipelines, but remain closed APIs with limited inspectability and evaluation access [61]. In contrast, our Reflexive Composition framework is designed for openness, modularity, and human-in-the-loop control.

Enterprise Deployment Challenges and Privacy Constraints. Traditional AI deployment models face fundamental limitations in regulated environments. Enterprise SaaS systems typically cannot access customer data for model improvement due to privacy contracts, competitive concerns, and regulatory restrictions. This creates a quality ceiling where AI performance remains static regardless of usage patterns or customer-specific needs.

In healthcare specifically, federated learning approaches have been explored to incorporate anonymized datasuch as X-rays from different machine types, imaging artifacts, and positional variationswithout

centralizing sensitive information. However, federated learning requires substantial infrastructure coordination and still faces challenges with heterogeneous data formats and institutional policies.

Direct Healthcare Implementation Experience. Practical deployment of AI in healthcare settings reveals additional constraints beyond privacy. Through partnership with FHIR-focused startups, several key challenges emerged:

- **Data Format Heterogeneity**: Clinical data exists in multiple formats (HL7, FHIR, proprietary EHR formats) requiring complex integration
- **Temporal Context**: Medical reasoning requires understanding of treatment timelines, medication interactions, and evolving patient conditions
- **Terminology Standardization**: Clinical notes use varied terminology requiring mapping to standard ontologies (RxNorm, LOINC, SNOMED-CT)
- **Practitioner Workflow Integration**: AI tools must integrate seamlessly with existing clinical decision-making processes

These constraints motivated the development of a privacy-preserving architecture that enables personalized medical reasoning through structured knowledge injection at inference time, rather than training-time data access.

9.3 LLM2KG: Structured Extraction and Semantic Enrichment

The LLM2KG pathway enables the system to expand and refine the patient-specific A-graph by interpreting both unstructured clinical notes and structured EHR entries. This reflexive process operates in two modes: extracting new assertions from free text, and semantically enriching structured records via biomedical ontologies.

System Setup and Dual-Graph Design

Our system structures clinical knowledge using a dual-graph architecture that separates patient-specific facts from general biomedical terminology:

- The **A-graph** (assertional graph) encodes dynamic, patient-specific data extracted from FHIR-formatted EHRs. This includes `Condition`, `MedicationRequest`, `Observation`, and other temporal clinical assertions.
- The **T-graph** (terminological graph) provides a stable layer of biomedical ontologies SNOMED-CT, RxNorm, LOINC that normalize, align, and disambiguate clinical terms used in both structured and unstructured inputs.

These graphs interact bidirectionally: free-text terms like *Z-pack* are resolved to canonical identifiers (e.g., *azithromycin*) using the T-graph and recorded in the A-graph; structured entries from

the A-graph are semantically augmented and validated using the T-graph. This separation supports consistent alignment between personalized medical records and standard medical vocabularies.

The A-graph remains local to the system and is accessed only to generate prompt-time context. No persistent or broad exposure of patient data occurs beyond this scoped interaction.

The dual-graph design directly operationalizes the Reflexive Composition framework (Chapter 3), enabling generative models and structured knowledge to co-evolve through iterative extraction, enrichment, and validation.

Production Deployment Architecture. The dual-graph design emerged from practical constraints encountered in healthcare SaaS deployment. Traditional approaches requiring centralized patient data proved incompatible with enterprise privacy requirements and healthcare regulations. The A-graph/T-graph separation enables a deployment model where:

- **T-graph (terminological)** can be shared across institutions as it contains only standardized medical knowledge
- **A-graph (assertional)** remains local to each deployment, containing patient-specific data that never leaves the healthcare institution
- **Knowledge injection** occurs at inference time, enabling personalized reasoning without data centralization

This architecture was validated through real-world deployment where practitioners needed to 'chat with EHR data'enabling natural language queries over patient records while maintaining full data control and auditability.

Extraction from Unstructured Notes

Given a clinical note (e.g., progress summary, discharge report), the system prompts the LLM to extract candidate clinical assertions in structured formats aligned with FHIR resource types. These include diagnoses, medications, symptom onset, and lab results. The output is formatted as key-value pairs or JSON snippets tagged with metadata and source provenance.

```
 {condition:  "Hypertension",
onsetDateTime:  "2022-06-12",
clinicalStatus:  "active"}
```

All suggestions are routed through human validation before integration into the A-graph. This allows the system to safely accumulate personalized medical assertions extracted from free text over time.

Semantic Annotation of FHIR Records

In addition to unstructured extraction, the LLM2KG process reflexively enhances structured FHIR-derived data using T-graph knowledge. For example, medication mentions in the A-graph (e.g., `lisinopril`, `Z-pack`) are passed through LLM-powered mapping routines to identify the correct RxNorm identifiers, dosages, and canonical substance forms.

This annotation step allows:

- Mapping brand names and colloquialisms to standard concept identifiers (e.g., Z-pack → `azithromycin`).
- Linking local EHR terms to national drug codes or ontology URIs via the T-graph.
- Enabling richer downstream queries, such as retrieve all patients prescribed ACE inhibitors or detect medication class overlap across hospital departments.

These enhancements are stored as layered annotations on top of the A-graph structure. They maintain backward compatibility with raw FHIR elements but support more expressive reasoning by unifying personalized data with standardized terminologies.

FHIR Integration Challenges and Solutions. Direct experience implementing FHIR-based clinical systems revealed several practical challenges not evident in academic datasets. Real-world FHIR data often contains:

- **Incomplete Resource Linkages**: References between Patient, Condition, and MedicationRequest resources may be missing or inconsistent
- **Terminology Gaps**: Clinical notes frequently use colloquial drug names ('Z-pack') requiring mapping to RxNorm standard codes
- **Temporal Inconsistencies**: Medication start/end dates may conflict between different FHIR resources
- **Provider-Specific Variations**: Different EHR systems implement FHIR extensions differently

The LLM2KG extraction pipeline addresses these challenges through:

1. **Semantic Normalization**: Mapping colloquial terms to standard ontologies (Z-pack azithromycin RxNorm:198211)
2. **Relationship Inference**: Using clinical context to infer missing resource linkages
3. **Temporal Alignment**: Reconciling conflicting timestamps using clinical logic
4. **Validation Workflows**: Human-in-the-loop verification for ambiguous cases

Reflexive Graph Enrichment

Together, these capabilities illustrate the reflexive nature of the system: LLMs both interpret clinical narratives to extract structured knowledge and refine structured content through semantic augmentation. This bidirectional behavior improves graph completeness and queryability, and provides clinicians with a transparent, inspectable knowledge substrate for decision support.

9.4 HITL: Human Validation and Feedback

In high-stakes domains such as clinical medicine, automated systems must support human oversight not just as a final fallback, but as a central component of the knowledge integration loop. The Reflexive Composition framework incorporates physicians and clinical reviewers directly into the graph update cycle through structured human-in-the-loop (HITL) workflows.

Validation of LLM2KG Suggestions

All structured suggestions generated by the LLM2KG modulewhether extracted from free-text notes or produced as semantic annotations on FHIR recordsare treated as provisional updates. Each candidate is presented to a physician validator via a review interface that includes:

- The original source text or FHIR record
- The extracted or enriched structured representation
- Suggested ontology mappings (e.g., RxNorm concepts)
- An editable approval panel (accept / revise / reject)

Physicians are encouraged to correct small errors directly, such as adjusting dates, selecting alternative ontology terms, or editing medication names. When suggestions are rejected, reasons may be logged for model retraining or rule refinement.

Auditability and Provenance Tracking

Each validation event is recorded with detailed metadata:

- The source and type of the original suggestion (e.g., note section, FHIR resource ID)
- The validator identity (or anonymized role), timestamp, and action
- The version of the language model used to generate the suggestion

These logs enable downstream auditing, rollback of changes, and regulatory compliance. Version-controlled updates to the A-graph preserve full history and allow for reconstruction of its state at any prior point.

Reflexive Feedback for System Adaptation

The HITL layer also closes the reflexive loop by feeding human decisions back into system behavior. Repeated rejections of a pattern (e.g., mapping Z-pack to the wrong drug) can trigger fine-tuning, prompt adjustments, or templating changes. Conversely, confirmed mappings can be cached and reused to improve response time and reduce physician burden.

This integration of human validators ensures that structured graph updates are both high-quality and continuously informed by domain expertise. The design emphasizes transparency, reversibility, and low-friction review, recognizing that trust and traceability are essential for adoption in clinical environments.

Appendix D expands on the auditability framework with implementation-level details.

9.5 KG2LLM: Prompt Construction with Structured Context

To enable personalized, trustworthy clinical generation, the system injects structured patient-specific knowledge into the input context of the language model. This KG2LLM mechanism draws from the A-graph, enriched with terminology-level metadata from the T-graph, and assembles this structured information into naturalized prompts suitable for generative reasoning.

Data Retrieval and Preparation

When a clinician issues a querysuch as Summarize the patient's hypertension history or List antibiotics prescribed for pneumoniathe system first identifies relevant subgraphs from the A-graph. These include diagnoses, medications, encounters, labs, and temporal event chains related to the target condition.

In parallel, T-graph enrichments are applied to:

- Resolve abbreviations or colloquial terms (e.g., Z-pack → azithromycin)
- Normalize drug classes and observation codes (e.g., map high BP → SNOMED hypertension, LOINC systolic pressure)
- Link brand and generic medication mentions via RxNorm

This semantic processing ensures that the model receives fully disambiguated, medically coherent inputs even if the original data contained inconsistent terminology.

Prompt Structuring and Injection

Structured facts are rendered as compact, readable statements or tabular snippets that form part of the models prompt. The prompt template adapts depending on the task (summarization, explanation, question answering) and the LLMs token constraints.

Example prompt:

```
 Patient:  64-year-old male
Diagnoses:  Hypertension (since 2015), Type 2 Diabetes (since 2012)
Medications:  Lisinopril (10mg daily), Metformin (500mg BID), Azithromycin
[RxNorm:  198211] (prescribed for pneumonia in 2023)
Labs:  Creatinine 1.6 mg/dL (abnormal), LDL 102 mg/dL

Summarize the patient's management for hypertension with respect to renal
function.
```

```
{PatientName} is a {Age}-year-old {Gender} patient with a history of {
    Conditions}.
Current medications include {Medications}.
Recent labs show: {Observations}.
Summarize the patient s management plan for hypertension.
```

The structured format of injected contentwhether as tables, short advisories, or schema-anchored statementsalso mitigates known interpretability risks in clinical LLM usage. Obika et al. [75] report that misalignment between content and expected clinical documentation format can lead to dangerous misinterpretations. KG2LLM minimizes this risk by aligning injected knowledge with familiar medical documentation styles and semantic anchors.

Traceability and Explanation Support

Each statement included in the prompt is sourced from validated A-graph entries and carries metadata references (e.g., encounter ID, date, source document). The output generated by the model can be

traced back to its underlying facts, enabling post-hoc explanation and allowing the clinician to verify the provenance of any clinical claim.

This method allows the system to preserve the flexibility and expressiveness of LLM-based generation while grounding responses in authoritative, structured, and auditable clinical data. By combining the factual completeness of EHR-derived graphs with the semantic precision of ontology mappings, KG2LLM enables safe, explainable, and contextually accurate reasoning in clinical workflows.

The system directs clinical queries to patient-specific subgraphs by extracting relevant temporal sequences, active diagnoses, current medications, and recent labs from the A-graph, augmented with terminology standardization from the T-graph. This filtering approach restricts prompt content to query-relevant clinical data, reducing PHI exposure and increasing semantic grounding specificity.

9.6 Experimental Setup

This case study evaluates the application of Reflexive Composition in a clinical setting by measuring the effectiveness of structured grounding, semantic enrichment, and human-in-the-loop validation. The evaluation focuses on realistic physician-style queries over longitudinal patient records, using publicly available datasets and a controlled architecture.

We define evaluation across three primary dimensions:

- **Clinical Accuracy:** Expert-validated correctness of generated summaries, clinical suggestions, and structured knowledge updates.
- **Traceability and Trust:** Physician-rated confidence in the output and perceived explainability, including ability to trace generated content to specific KG facts.
- **Workflow Integration:** Cognitive load, time-to-insight, and validator correction overhead, capturing integration efficacy in clinical tasks.

Data Source and Preprocessing

Experiments are conducted on the MIMIC-III and MIMIC-IV datasets, which contain de-identified health records from intensive care units. Each record includes structured components (e.g., prescriptions, lab results, diagnoses) as well as unstructured clinical notes.

Structured elements are converted into FHIR-compatible resources, forming the basis of the patient-specific A-graph. FHIR resource types include `Patient`, `Encounter`, `Condition`, `MedicationRequest`, and `Observation`, each preserving temporal metadata and entity relationships. Simultaneously, terminology mappings are derived from standard biomedical ontologiesSNOMED-CT, RxNorm, and LOINC-forming the T-graph. The T-graph supports normalization and enrichment of the A-graph via ontology lookups and lexical resolution of variant terms.

System Variants

To isolate the contribution of structured input and reflexive feedback, we evaluate four system configurations:

1. **LLM-only baseline**: A general-purpose medical language model is prompted with physician-style queries, without any injected patient-specific context.

2. **RAG baseline**: Retrieval-Augmented Generation is applied by retrieving unstructured clinical notes relevant to the query and appending them to the prompt.

3. **KG2LLM only**: Structured A-graph facts (e.g., diagnoses, medications, labs) are injected into the prompt in naturalized or tabular form, but no reflexive feedback or graph updates occur.

4. **Reflexive Composition**: Combines KG2LLM structured injection, LLM2KG extraction and enrichment (including RxNorm annotation), and human-in-the-loop validation before updating the A-graph.

Query Tasks and Design

To simulate realistic clinical reasoning tasks, we define a set of queries that require multi-step inference over longitudinal, heterogeneous patient data. Each query type corresponds to a common physician information-seeking need:

Query Type	**Sample Question**	**Required Sources**
Temporal reasoning	Has the patient had abnormal creatinine levels in the last 6 months?	Structured labs (Observation), event timestamps
Medication linkage	What antibiotics were prescribed for pneumonia?	MedicationCondition relation, RxNorm T-graph
Causal ordering	Was the patient diagnosed with hypertension before starting lisinopril?	Condition and medication timing (A-graph)
Summarization	Summarize the patients renal function and adherence over the past year.	Structured + unstructured notes

Table 9.1: Representative Clinical Queries for MIMIC-Based Evaluation

Each query is paired with a reference answer constructed from the ground truth data and validated by a clinical reviewer.

Evaluation Metrics

We assess system performance using the following metrics:

- **Clinical Accuracy**: Expert-validated correctness of generated summaries or answers relative to the patient record.

- **Traceability**: Degree to which generated outputs can be linked to A-graph entries and source documents.

- **Trust Ratings**: Physician-perceived confidence in the output, measured on a Likert scale.

- **Correction Efficiency**: Average time taken by reviewers to validate or correct LLM2KG suggestions.

- **Error Propagation Rate**: Proportion of hallucinated or incorrect suggestions that would have been incorporated without HITL validation.

All models are evaluated on the same patient cases and query prompts. Ground truth validation is performed by domain experts to ensure high-fidelity scoring.

Micro-Evaluation Strategies

Beyond system-level metrics, we adopt targeted diagnostic evaluations to isolate failure modes and assess model behavior under varying input conditions:

- **Format Compliance Check:** Evaluates whether LLM2KG outputs conform to expected FHIR schemas.
- **Hidden Fact Stress Test:** Key KG facts are withheld to test the hallucination resilience of KG2LLM.
- **Traceability Audit:** Measures what percentage of generated outputs can be directly grounded in structured A-graph entries.

These tools allow us to quantify failure surfaces and identify high-yield areas for validator intervention.

9.7 Results and Analysis

Comparison Across System Variants

The four system configurations introduced in Section 9.6 were evaluated on a shared set of patient cases and physician-style queries. Table 9.2 summarizes key performance metrics. Table 9.2 compares four prompting configurations evaluated on a 50-query benchmark. We measure:

- Factuality as token-level F1 score against gold-labeled answers
- Contradiction rate using an automated contradiction detector (see Chapter 6)
- Validator escalation rate as the proportion of responses flagged for expert review during KG construction

Reflexive Composition yields the best performance across all three dimensions, demonstrating the compounding benefits of validated schema-grounded context.

As shown in Table 6.1, grounding responses in structured EHR-derived knowledge graphs resulted in a 24-point F1 improvement (0.79 vs. 0.55) and significantly lowered schema and contradiction error rates. Human reviewers rated precision and recall notably higher under the KG2LLM condition.

For example, when asked *Is the patient currently on a blood pressure medication?*, the baseline LLM-only model replied, *No medications are listed in the record.* In contrast, the Reflexive Composition system responded, *Yes, the patient is prescribed lisinopril, a medication used to treat hypertension.*

Prompt Type	Validator Escalation Rate	Contradiction Rate (↓)	Factuality (F1)
LLM-only	41.6%	26.2%	0.64
RAG (no schema)	35.0%	18.9%	0.70
KG-injected	24.2%	12.1%	0.76
Reflexive Composition	14.0%	9.4%	0.82

Table 9.2: Evaluation of four prompting configurations on contradiction, factuality, and validator effort for the healthcare use case.

This grounded response aligns with the gold answer, avoids contradiction, and did not trigger validator escalation.

See Appendix D for additional examples including prompt structure, model outputs, and evaluation annotations. The LLM-only baseline performed worst, frequently hallucinating patient data or generating plausible but incorrect summaries. The RAG configuration improved fluency and recall slightly, but struggled to precisely answer temporally or semantically scoped queries. KG2LLM grounding significantly improved accuracy and reduced ambiguity, and was generally preferred by clinical reviewers.

The full Reflexive Composition system achieved the highest scores across all dimensions. Traceability was complete due to provenance tagging of each structured input. Trust ratings were markedly higher when reviewers could inspect source facts and validation history.

Impact of Human-in-the-Loop Validation

Reviewers interacted with LLM2KG suggestions through a structured HITL interface. On average, reviewers required less than 30 seconds per suggestion to accept, revise, or reject. Suggestions involving RxNorm or medication timelines benefited most from reviewer input, especially in disambiguating dosage or duration. Importantly, the HITL process prevented erroneous or hallucinated entries from entering the A-graph, reducing the error propagation rate to zero.

Qualitative Observations

Reviewers highlighted the systems ability to produce coherent summaries that were faithful to the clinical record, particularly when structured facts were naturalized and temporally contextualized. The combination of enriched medication annotations (e.g., mapping Z-pack to azithromycin via RxNorm) and explicit time anchoring (e.g., since 2016) contributed significantly to interpretability.

Failures in the LLM-only and RAG baselines often involved speculative associations (e.g., linking symptoms to the wrong conditions) or oversights due to lack of temporal context. In contrast, Reflexive Composition supported explainable answers with strong alignment to validated graph content.

Enterprise Deployment Implications. The observed improvements (F1: 0.82 vs 0.64 baseline) demonstrate the feasibility of privacy-preserving personalization in enterprise healthcare settings. Unlike federated learning approaches that require complex infrastructure coordination, the inference-time knowledge injection model can be deployed incrementally within existing healthcare IT environments.

Practical Deployment Observations:

- **Latency Impact**: Knowledge graph assembly adds $\tilde{1}$50ms per query, acceptable for clinical decision support workflows
- **Privacy Compliance**: Patient data never leaves local environment, simplifying HIPAA compliance
- **Practitioner Adoption**: Natural language interface over structured EHR data significantly improved workflow integration compared to traditional query interfaces
- **Audit Trail**: Complete provenance tracking from clinical notes through FHIR extraction to final recommendations supports regulatory requirements

Scalability Considerations: The approach scales well within institutions (tested up to 10K patient records) but requires coordination for multi-institutional deployment. The T-graph standardization enables knowledge sharing while A-graph isolation preserves privacy boundaries.

Summary

The experimental results confirm that structured grounding and reflexive update cycles significantly improve accuracy, trust, and auditability in clinical reasoning tasks. While structured injection alone yields strong benefits, the full Reflexive Composition loopincluding enrichment, validation, and traceabilityachieves the best performance across all measured criteria.

9.8 Discussion

Strengths of Reflexive Composition in Clinical Contexts

This case study demonstrates that Reflexive Composition is particularly well-suited to domains requiring high factual precision, traceability, and human oversight. The bidirectional integration of structured knowledge graphs and LLMs enables both accurate generation and dynamic graph maintenance. Key strengths include:

- **Grounded Summarization**: Structured injection from the A-graph supports accurate and context-specific language generation.
- **Semantic Normalization**: RxNorm, SNOMED-CT, and LOINC mappings via the T-graph improve terminological coherence and downstream reasoning.
- **Trust and Traceability**: HITL validation and provenance metadata allow clinicians to inspect, revise, and understand model behavior.
- **Adaptability**: Reflexive updates allow the system to evolve with patient data and domain feedback, supporting longitudinal alignment.

Limitations

Several practical limitations remain:

- **Physician Workload**: While validation times were low, reviewer availability may become a bottleneck in high-throughput settings.
- **Ontology Ambiguity**: Mapping medication and lab terms to standardized identifiers is non-trivial, especially with abbreviation-heavy or idiosyncratic inputs.
- **Scalability of Graph Updates**: Frequent updates to the A-graph require efficient graph storage, versioning, and retrieval mechanisms, which are not trivial to manage at hospital scale.

Future Work

Future extensions include:

- Integration with real-time clinical systems via sandboxed FHIR APIs or institutional data lakes.
- User studies assessing impact on diagnostic efficiency, workload, and decision confidence.
- Expansion of T-graph enrichment to additional ontologies (e.g., ICD-10, UMLS, RxNorm ingredients) for richer inference capabilities.
- Exploration of automatic validation pre-filters to reduce human validation burden through active learning.

This future evolution aligns with emerging best practices in clinical AI deployment. MedLM, for instance, emphasizes layered risk assessmentsboth by developers and implementing organizationsto manage safety in real-world healthcare settings [75]. Reflexive Composition supports this paradigm by offering modular checkpoints for extraction, validation, prompt construction, and human review. These affordances provide implementers with the necessary levers to monitor and control clinical risk as system demands and contexts evolve.

This evaluation further highlights the benefits of role specialization: separating the Generator LLM (LLM2KG) and the Target LLM (KG2LLM) allowed us to tune precision and fluency independently, enabling modular deployments tailored to extraction or generation constraints.

9.9 Conclusion

This case study applied Reflexive Composition to physician-centered clinical support by combining structured EHR data, ontology-based enrichment, and human-validated LLM reasoning. The system achieved strong performance in both factual accuracy and physician trust, outperforming LLM-only and RAG baselines. By enriching FHIR-derived A-graphs with terminology-level T-graph annotations and closing the loop through validation and refinement, the approach demonstrated the feasibility of reflexive AI systems in high-stakes, regulated environments. This chapter provides evidence that Reflexive Composition is applicable beyond knowledge curation and open-domain QA, extending to domains where safety, transparency, and continual updating are essential.

Chapter 10

Discussions and Future Research Directions

10.1 Synthesis of Lessons Learned

The application of Reflexive Composition across diverse case studies revealed key insights into the challenges and opportunities of dynamically grounding Large Language Models (LLMs) with curated, structured knowledge.

First, structured knowledge integration at inference time consistently improved LLM reliability across domains. In the private healthcare integration case study, dynamic grounding enabled LLMs to access patient-specific information without retraining, significantly improving clinical reasoning accuracy while maintaining privacy compliance. In the historical bias mitigation case study, dynamic grounding with curated security knowledge graphs reduced the reproduction of outdated and vulnerable code patterns, improving the security quality of generated outputs.

Second, the Reflexive Composition framework proved adaptable across varied knowledge types. Whether dealing with temporally anchored patient records or evolving security advisories, the methodology consistently supported precise, selective knowledge injection, enabling models to operate with domain-specific awareness without sacrificing general reasoning flexibility.

Third, inference-time knowledge injection, supported by systematic retrieval and validation pipelines, demonstrated that many hallucination and bias issues in LLMs stem from absent or inaccessible knowledge rather than intrinsic model failures. By structuring external knowledge and dynamically exposing it only when needed, Reflexive Composition offers a scalable, domain-agnostic approach to reducing model brittleness and improving factual grounding.

10.2 Contributions Beyond Case Studies

While the case studies demonstrate the effectiveness of Reflexive Composition in specific domains, the methodology offers broader contributions to the field of Large Language Model (LLM) integration and knowledge-grounded AI systems.

10.2.1 A Generalizable Neural-Symbolic Integration Framework

Reflexive Composition formalizes a modular, closed-loop architecture that systematically connects LLMs, human-in-the-loop (HITL) validation, and structured knowledge graphs. This bidirectional integration enables models to benefit from structured knowledge while continuously improving the quality and relevance of that knowledge over time.

10.2.2 Dynamic Grounding without Retraining

The methodology shows that dynamic, inference-time knowledge grounding can meaningfully enhance model behavior without modifying model parameters. This approach addresses practical challenges associated with LLM retraining, including computational cost, privacy risks, and model drift.

10.2.3 Mitigation of Knowledge Gaps and Historical Bias

Across domains, Reflexive Composition effectively mitigates risks associated with missing information and inherited historical biases. By exposing LLMs to curated, domain-specific knowledge only when necessary, the approach reduces hallucinations, outdated knowledge reproduction, and factual errors, enhancing model trustworthiness.

10.2.4 Enabling Transparent and Auditable Inference

By structuring external knowledge injections and clearly separating them from model-generated content, Reflexive Composition supports transparency and post-hoc auditability in AI outputs. This capability is especially important in high-stakes domains such as healthcare, security, and law, where traceability and validation are required by regulatory standards.

10.2.5 Foundation for Continuous Knowledge Updating

The modular architecture of Reflexive Composition lays the groundwork for future systems capable of continuously ingesting, validating, and integrating updated knowledge sources without disrupting model operations. This capability is essential for maintaining the relevance and reliability of AI systems deployed in dynamic, evolving domains.

10.3 Limitations and Open Challenges

While Reflexive Composition demonstrates strong potential for improving LLM reliability through dynamic knowledge grounding, several limitations and open challenges remain.

10.3.1 Retrieval Precision and Context Management

The performance of inference-time knowledge injection depends on the precision and recall of retrieval mechanisms. Incomplete retrieval may reduce grounding accuracy, while overly broad retrieval can exceed model context limits or introduce irrelevant content. Balancing retrieval coverage and conciseness poses ongoing design challenges, particularly as models are applied to increasingly complex and multi-step reasoning tasks.

10.3.2 Temporal Reasoning and Evolving Knowledge

Domains such as healthcare and cybersecurity evolve rapidly, requiring models to reason about temporal relationships and outdated knowledge states. Reflexive Composition supports dynamic knowledge updates, but richer temporal reasoning frameworks such as versioned knowledge graphs or temporally scoped retrieval could further strengthen model reasoning under evolving conditions.

10.3.3 Context Window Constraints

The finite input size limitations of contemporary LLMs impose practical constraints on the amount of knowledge that can be injected at inference time. While current strategies prioritize the most relevant knowledge snippets, further improvements in context summarization, abstraction, and selective compression are important for supporting inference over larger knowledge bases and more complex reasoning tasks.

10.3.4 Manual Validation Effort

Although the HITL validation framework strategically focuses human effort on high-value tasks, the need for expert validation remains a bottleneck, especially in sensitive or safety-critical domains. Future work in semi-automated validation, anomaly detection during knowledge ingestion, and selective human intervention could further reduce operational overheads without compromising reliability.

10.3.5 Cross-Domain Generalization

While Reflexive Composition has been applied successfully in healthcare and code generation domains, further studies are needed to evaluate its adaptability in other fields such as finance, legal reasoning, and scientific research. Understanding domain-specific differences in knowledge representation, retrieval strategies, and LLM grounding behavior will be important for informing broader deployment.

Across all three domainshealthcare, current affairs, and software securitythe Reflexive Composition framework consistently improves factuality, reduces hallucinations, and lowers the need for human validation. Although the causes of hallucination differ by domain (privacy-related incompleteness in healthcare, temporal drift in current events, and inherited bias in software code), the frameworks core mechanismsbidirectional KG integration, contradiction detection, and schema-guided promptingprove effective in all cases. This suggests that Reflexive Composition is not only effective in isolated tasks but also provides a reusable architecture for dynamic knowledge alignment across diverse, high-risk domains.

10.4 Future Research Directions

Building on the foundations established by Reflexive Composition, several promising avenues for future research emerge.

10.4.1 Adaptive Retrieval Mechanisms

Developing retrieval engines that dynamically prioritize and summarize knowledge based on query specificity, temporal relevance, and context window constraints could further optimize knowledge injection. Reinforcement learning or retrieval augmentation techniques could be explored to fine-tune retrieval strategies based on downstream task performance.

10.4.2 Incremental Knowledge Graph Updates

Implementing pipelines for continuous, semi-automated ingestion and validation of new knowledge sources would enable Reflexive Composition systems to remain current without manual reconstruction of knowledge graphs. Techniques such as active learning and anomaly detection during ingestion could reduce validation effort while maintaining reliability.

10.4.3 Fine-Grained Response Auditing

Real-time auditing mechanisms capable of detecting hallucinations, factual inconsistencies, or security violations during inference would strengthen trust in LLM outputs. Embedding lightweight auditing models or using retrieval-augmented verification steps could help ensure that dynamically grounded responses adhere to injected knowledge constraints.

10.4.4 Cross-Domain Expansion Studies

Extending Reflexive Composition to additional high-stakes domains such as finance, law, scientific publishing, and regulatory compliance would test the generality of the approach. These expansions would also surface new challenges in domain-specific knowledge graph construction, validation workflows, and retrieval strategies.

10.4.5 Integration with Future LLM Architectures

As LLM architectures evolve (e.g., models with larger context windows, retrieval-native capabilities, or hybrid symbolic-neural systems), Reflexive Composition principles may be extended to align with these new capabilities. Dynamic grounding strategies are likely to become increasingly relevant as models are deployed in high-risk domains with evolving knowledge requirements.

10.5 Broader Impact

Reflexive Composition responds to a key limitation in the current deployment of Large Language Models (LLMs): the lack of mechanisms for dynamically grounding reasoning in curated, domain-specific, and up-to-date knowledge without retraining. By structuring a closed-loop integration between LLMs, human-in-the-loop validation, and structured knowledge graphs, the framework supports scalable development of AI systems that are more reliable, transparent, and adaptable.

In safety-critical domains such as healthcare, cybersecurity, finance, and law, hallucinations, outdated information, and inherited biases from training data introduce operational and ethical concerns.

Reflexive Composition aims to mitigate these issues by enabling real-time, verifiable knowledge injection at inference time. This supports factual consistency, improves accountability, and aligns with regulatory requirements for traceability and transparency in automated decision-making systems.

By emphasizing modularity and adaptability, the framework enables AI systems to operate in dynamic environments where knowledge evolves continuously. Rather than relying on periodic retraining, models can be incrementally augmented through curated external knowledge preserving system continuity while enhancing factual alignment.

From a scientific perspective, Reflexive Composition advances the integration of symbolic and neural techniques in AI, demonstrating that structured knowledge can improve model behavior even after deployment. It contributes to ongoing efforts to develop AI systems that are reliable, auditable, and responsive to human oversight.

As AI systems become more embedded in high-stakes decision-making, approaches like Reflexive Composition may play a key role in ensuring that these systems remain aligned with evolving knowledge and ethical standards.

Chapter 11

Conclusions

This thesis introduced Reflexive Composition, a modular framework for dynamically grounding Large Language Models (LLMs) in curated, structured knowledge through closed-loop integration with knowledge graphs and human-in-the-loop validation. By supporting real-time knowledge injection without requiring model retraining, Reflexive Composition targets persistent challenges in LLM reliability, transparency, and adaptability.

Across multiple case studiesincluding private healthcare data integration and historical bias mitigation in code generationReflexive Composition showed measurable improvements in model behavior, including gains in factual accuracy, reductions in hallucination frequency, and mitigation of training-related vulnerabilities. These results support the feasibility of dynamic, domain-specific knowledge grounding as an alternative to static fine-tuning approaches.

The thesis also identified remaining limitations and open challenges, such as the need for adaptive retrieval strategies, context window management, semi-automated validation workflows, and additional cross-domain validation studies. Future work will aim to extend the frameworks applicability, efficiency, and generalizability.

Beyond the specific case study outcomes, Reflexive Composition contributes to the broader goal of building trustworthy AI systems by demonstrating that structured external knowledge can enhance model reasoning even after deployment. It informs ongoing efforts to develop AI systems that are more transparent, auditable, and aligned with domain constraints.

As LLMs become increasingly integrated into decision-making workflows, reflexive knowledge-grounding strategies like those proposed in this thesis may play an important role in supporting reliability, accountability, and adaptability in complex, high-stakes environments.

Bibliography

[1] Bilal Abu-Salih et al. "Healthcare knowledge graph construction: A systematic review of the state-of-the-art, open issues, and opportunities". en. In: *Journal of Big Data* 10.1 (May 2023), p. 81. ISSN: 2196-1115. DOI: 10.1186/s40537-023-00774-9. URL: https://doi.org/10.1186/s40537-023-00774-9 (visited on 11/21/2024).

[2] Dimitrios Alivanistos et al. *Prompting as Probing: Using Language Models for Knowledge Base Construction*. arXiv:2208.11057 [cs]. June 2023. DOI: 10.48550/arXiv.2208.11057. URL: http://arxiv.org/abs/2208.11057 (visited on 12/25/2024).

[3] Owura Asare, Meiyappan Nagappan, and N. Asokan. *Is GitHub's Copilot as Bad as Humans at Introducing Vulnerabilities in Code?* arXiv:2204.04741. Jan. 2024. DOI: 10.48550/arXiv.2204.04741. URL: http://arxiv.org/abs/2204.04741 (visited on 11/28/2024).

[4] Yu Bai et al. *Pistis-RAG: Enhancing Retrieval-Augmented Generation with Human Feedback*. arXiv:2407.00072 [cs] version: 5. Oct. 2024. DOI: 10.48550/arXiv.2407.00072. URL: http://arxiv.org/abs/2407.00072 (visited on 05/04/2025).

[5] Karan Bhanot et al. "The Problem of Fairness in Synthetic Healthcare Data". en. In: *Entropy* 23.9 (Sept. 2021). Number: 9 Publisher: Multidisciplinary Digital Publishing Institute, p. 1165. ISSN: 1099-4300. DOI: 10.3390/e23091165. URL: https://www.mdpi.com/1099-4300/23/9/1165 (visited on 01/26/2025).

[6] Simone Bocca et al. *Building Interoperable Electronic Health Records as Purpose-Driven Knowledge Graphs*. arXiv:2305.06088 [cs]. May 2023. DOI: 10.48550/arXiv.2305.06088. URL: http://arxiv.org/abs/2305.06088 (visited on 03/26/2024).

[7] Tom B Brown et al. "Language Models are Few-Shot Learners". en. In: *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*. 2020. URL: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[8] J Harry Caufield et al. "Structured Prompt Interrogation and Recursive Extraction of Semantics (SPIRES): a method for populating knowledge bases using zero-shot learning". In: *Bioinformatics* 40.3 (Mar. 2024), btae104. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btae104. URL: https://doi.org/10.1093/bioinformatics/btae104 (visited on 05/04/2025).

[9] Viktoriia Chekalina et al. *Addressing Hallucinations in Language Models with Knowledge Graph Embeddings as an Additional Modality*. arXiv:2411.11531. Nov. 2024. DOI: 10.48550/arXiv.2411.11531. URL: http://arxiv.org/abs/2411.11531 (visited on 11/21/2024).

[10] Junying Chen et al. *HuatuoGPT-o1, Towards Medical Complex Reasoning with LLMs*. arXiv:2412.18925 [cs]. Dec. 2024. DOI: 10.48550/arXiv.2412.18925. URL: http://arxiv.org/abs/2412.18925 (visited on 01/16/2025).

[11] Mark Chen et al. "Evaluating Large Language Models Trained on Code". en. In: *OpenAI* (July 2021), p. 35.

[12] Weijie Chen et al. *KG-Retriever: Efficient Knowledge Indexing for Retrieval-Augmented Large Language Models*. en. arXiv:2412.05547 [cs]. Dec. 2024. DOI: 10.48550/arXiv.2412.05547. URL: http://arxiv.org/abs/2412.05547 (visited on 12/27/2024).

[13] Wenhu Chen et al. "Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks". In: (2022). Publisher: [object Object] Version Number: 4. DOI: 10.48550/ARXIV.2211.12588. URL: https://arxiv.org/abs/2211.12588 (visited on 04/09/2024).

[14] Ye Chen et al. *MedCT: A Clinical Terminology Graph for Generative AI Applications in Healthcare*. arXiv:2501.06465 [cs]. Jan. 2025. DOI: 10.48550/arXiv.2501.06465. URL: http://arxiv.org/abs/2501.06465 (visited on 01/16/2025).

[15] Ha Na Cho et al. "Task-Specific Transformer-Based Language Models in Health Care: Scoping Review". EN. In: *JMIR Medical Informatics* 12.1 (Nov. 2024). Company: JMIR Medical Informatics Distributor: JMIR Medical Informatics Institution: JMIR Medical Informatics Label: JMIR Medical Informatics Publisher: JMIR Publications Inc., Toronto, Canada, e49724. DOI: 10.2196/49724. URL: https://medinform.jmir.org/2024/1/e49724 (visited on 12/11/2024).

[16] Aakanksha Chowdhery et al. *PaLM: Scaling Language Modeling with Pathways*. arXiv:2204.02311. Oct. 2022. DOI: 10.48550/arXiv.2204.02311. URL: http://arxiv.org/abs/2204.02311 (visited on 12/04/2024).

[17] Yashrajsinh Chudasama et al. "Towards Interpretable Hybrid AI: Integrating Knowledge Graphs and Symbolic Reasoning in Medicine". In: *IEEE Access* (2025). Conference Name: IEEE Access, pp. 1–1. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2025.3529133. URL: https://ieeexplore.ieee.org/document/10839382/?arnumber=10839382 (visited on 01/16/2025).

[18] Stephen Chung, Ivan Anokhin, and David Krueger. "Thinker: learning to plan and act". In: *Proceedings of the 37th International Conference on Neural Information Processing Systems*. NIPS '23. Red Hook, NY, USA: Curran Associates Inc., Dec. 2023, pp. 22896–22933. (Visited on 05/03/2025).

[19] Philipp Cimiano and Heiko Paulheim. "Knowledge graph refinement: A survey of approaches and evaluation methods". In: *Semant. web* 8.3 (Jan. 2017), pp. 489–508. ISSN: 1570-0844. DOI: 10.3233/SW-160218. URL: https://doi.org/10.3233/SW-160218 (visited on 05/04/2025).

[20] Tolga Çöplü et al. "Ontology-Guided On-Device Conversational Knowledge Capture with Large Language Models". en. In: ().

[21] Wentao Cui et al. "Automated taxonomy alignment via large language models: bridging the gap between knowledge domains". en. In: *Scientometrics* (July 2024). ISSN: 1588-2861. DOI: 10.1007/s11192-024-05111-2. URL: https://doi.org/10.1007/s11192-024-05111-2 (visited on 08/11/2024).

[22] Seyed Shayan Daneshvar et al. *Exploring RAG-based Vulnerability Augmentation with LLMs*. arXiv:2408.04125 version: 1. Aug. 2024. DOI: 10.48550/arXiv.2408.04125. URL: http://arxiv.org/abs/2408.04125 (visited on 11/24/2024).

[23] Antonio De Santis et al. "Integrating Large Language Models and Knowledge Graphs for Extraction and Validation of Textual Test Data". en. In: *The Semantic Web – ISWC 2024*. Ed. by Gianluca Demartini et al. Cham: Springer Nature Switzerland, 2025, pp. 304–323. ISBN: 978-3-031-77847-6. DOI: 10.1007/978-3-031-77847-6_17.

[24] Dina Demner-Fushman, Willie J. Rogers, and Alan R. Aronson. "MetaMap Lite: an evaluation of a new Java implementation of MetaMap". eng. In: *Journal of the American Medical Informatics Association: JAMIA* 24.4 (July 2017), pp. 841–844. ISSN: 1527-974X. DOI: 10.1093/jamia/ocw177.

[25] Yangruibo Ding et al. *Vulnerability Detection with Code Language Models: How Far Are We?* arXiv:2403.18624. July 2024. DOI: 10.48550/arXiv.2403.18624. URL: http://arxiv.org/abs/2403.18624 (visited on 12/01/2024).

[26] Xin Luna Dong et al. "AutoKnow: Self-Driving Knowledge Collection for Products of Thousands of Types". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '20. New York, NY, USA: Association for Computing Machinery, Aug. 2020, pp. 2724–2734. ISBN: 978-1-4503-7998-4. DOI: 10.1145/3394486.3403323. URL: https://dl.acm.org/doi/10.1145/3394486.3403323 (visited on 05/01/2025).

[27] Hady Elsahar et al. "T-REx: A Large Scale Alignment of Natural Language with Knowledge Base Triples". In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Ed. by Nicoletta Calzolari et al. Miyazaki, Japan: European Language Resources Association (ELRA), May 2018. URL: https://aclanthology.org/L18-1544/ (visited on 04/24/2025).

[28] Zhangyin Feng et al. "CodeBERT: A Pre-Trained Model for Programming and Natural Languages". In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 1536–1547. DOI: 10.18653/v1/2020.findings-emnlp.139. URL: https://aclanthology.org/2020.findings-emnlp.139 (visited on 05/01/2022).

[29] Daniel Fried et al. "InCoder: A Generative Model for Code Infilling and Synthesis". In: *arXiv:2204.05999 [cs]* (Apr. 2022). arXiv: 2204.05999. URL: http://arxiv.org/abs/2204.05999 (visited on 04/24/2022).

[30] Yujia Fu et al. *Security Weaknesses of Copilot Generated Code in GitHub*. arXiv:2310.02059 version: 2. Apr. 2024. DOI: 10.48550/arXiv.2310.02059. URL: http://arxiv.org/abs/2310.02059 (visited on 11/26/2024).

[31] Yunfan Gao et al. *Retrieval-Augmented Generation for Large Language Models: A Survey*. en. arXiv:2312.10997 [cs]. Mar. 2024. DOI: 10.48550/arXiv.2312.10997. URL: http://arxiv.org/abs/2312.10997 (visited on 12/03/2024).

[32] Pratik Bhavsar Gheorghe Bogdan. *LLM-as-a-Judge vs Human Evaluation*. en. 2024. URL: https://www.galileo.ai/blog/llm-as-a-judge-vs-human-evaluation (visited on 10/24/2024).

[33] Fausto Giunchiglia. *iTelos Methodology - General principles*. en. 2022. URL: https://unitn-knowledge-graph-engineering.github.io/KGE2022-website/material/slides/W5.L1.iTelosModelBasedKGE.pdf.

[34] Fausto Giunchiglia et al. *iTelos – Purpose Driven Knowledge Graph Generation*. Tech. rep. arXiv:2105.09418. arXiv:2105.09418 [cs] version: 2 type: article. arXiv, Dec. 2021. DOI: 10.48550/arXiv.2105.09418. URL: http://arxiv.org/abs/2105.09418 (visited on 05/15/2022).

[35] Armin Haller. "Are We Better Off With Just One Ontology on the Web?" In: (2020). URL: https://semantic-web-journal.net/system/files/swj2307.pdf (visited on 05/19/2024).

[36] Dan Hendrycks et al. *Measuring Massive Multitask Language Understanding*. arXiv:2009.03300 [cs]. Jan. 2021. DOI: 10.48550/arXiv.2009.03300. URL: http://arxiv.org/abs/2009.03300 (visited on 05/04/2025).

[37] Aidan Hogan et al. *Large Language Models, Knowledge Graphs and Search Engines: A Crossroads for Answering Users' Questions*. arXiv:2501.06699 [cs]. Jan. 2025. DOI: 10.48550/arXiv.2501.06699. URL: http://arxiv.org/abs/2501.06699 (visited on 01/16/2025).

[38] Neil Houlsby et al. "Parameter-Efficient Transfer Learning for NLP". en. In: *Proceedings of the 36th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, May 2019, pp. 2790–2799. URL: https://proceedings.mlr.press/v97/houlsby19a.html (visited on 05/21/2025).

[39] Edward J. Hu et al. "LoRA: Low-Rank Adaptation of Large Language Models". In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=nZeVKeeFYf9.

[40] Haoyu Huang et al. *Can LLMs be Good Graph Judger for Knowledge Graph Construction?* en. arXiv:2411.17388 [cs]. Nov. 2024. DOI: 10.48550/arXiv.2411.17388. URL: http://arxiv.org/abs/2411.17388 (visited on 12/09/2024).

[41] Kexin Huang, Jaan Altosaar, and Rajesh Ranganath. *ClinicalBERT: Modeling Clinical Notes and Predicting Hospital Readmission*. arXiv:1904.05342. Nov. 2020. DOI: 10.48550/arXiv.1904.05342. URL: http://arxiv.org/abs/1904.05342 (visited on 11/26/2024).

[42] Nourhan Ibrahim et al. "A survey on augmenting knowledge graphs (KGs) with large language models (LLMs): models, evaluation metrics, benchmarks, and challenges". en. In: *Discover Artificial Intelligence* 4.1 (Nov. 2024), p. 76. ISSN: 2731-0809. DOI: 10.1007/s44163-024-00175-8. URL: https://doi.org/10.1007/s44163-024-00175-8 (visited on 11/08/2024).

[43] Zhen Jia et al. "TempQuestions: A Benchmark for Temporal Question Answering". In: *Companion Proceedings of the The Web Conference 2018*. WWW '18. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, Apr. 2018, pp. 1057–1062. ISBN: 978-1-4503-5640-4. DOI: 10.1145/3184558.3191536. URL: https://dl.acm.org/doi/10.1145/3184558.3191536 (visited on 05/03/2025).

[44] Zhouyu Jiang et al. *Efficient Knowledge Infusion via KG-LLM Alignment*. arXiv:2406.03746 [cs]. June 2024. DOI: 10.48550/arXiv.2406.03746. URL: http://arxiv.org/abs/2406.03746 (visited on 06/13/2024).

[45] Alistair E. W. Johnson et al. "MIMIC-III, a freely accessible critical care database". en. In: *Scientific Data* 3.1 (May 2016). Publisher: Nature Publishing Group, p. 160035. ISSN: 2052-4463. DOI: 10.1038/sdata.2016.35. URL: https://www.nature.com/articles/sdata201635 (visited on 12/06/2024).

[46] Jungo Kasai et al. *RealTime QA: What's the Answer Right Now?* arXiv:2207.13332. Feb. 2024. DOI: 10.48550/arXiv.2207.13332. URL: http://arxiv.org/abs/2207.13332 (visited on 12/08/2024).

[47] Hanieh Khorashadizadeh et al. *Exploring In-Context Learning Capabilities of Foundation Models for Generating Knowledge Graphs from Text*. arXiv:2305.08804 [cs]. May 2023. DOI: 10.48550/arXiv.2305.08804. URL: http://arxiv.org/abs/2305.08804 (visited on 12/25/2024).

[48] Ernests Lavrinovics et al. *Knowledge Graphs, Large Language Models, and Hallucinations: An NLP Perspective*. arXiv:2411.14258. Nov. 2024. DOI: 10.48550/arXiv.2411.14258. URL: http://arxiv.org/abs/2411.14258 (visited on 12/11/2024).

[49] Jens Lehmann. "DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia". In: *Semantic Web Journal* (2012). URL: https://www.semantic-web-journal.net/content/dbpedia-large-scale-multilingual-knowledge-base-extracted-wikipedia-0 (visited on 04/28/2025).

[50] Patrick Lewis et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474. URL: https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7Abstract.html (visited on 05/04/2025).

[51] DaiFeng Li and Fan Xu. *Synergizing Knowledge Graphs with Large Language Models: A Comprehensive Review and Future Prospects*. arXiv:2407.18470. July 2024. URL: http://arxiv.org/abs/2407.18470 (visited on 10/28/2024).

[52] Jin Li et al. "Benchmarking Large Language Models in Evidence-Based Medicine". eng. In: *IEEE journal of biomedical and health informatics* PP (Oct. 2024). ISSN: 2168-2208. DOI: 10.1109/JBHI.2024.3483816.

[53] Raymond Li et al. *StarCoder: may the source be with you!* arXiv:2305.06161 [cs]. May 2023. URL: http://arxiv.org/abs/2305.06161 (visited on 05/15/2023).

[54] Pengfei Liu et al. "Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing". In: *ACM Comput. Surv.* 55.9 (Jan. 2023), 195:1–195:35. ISSN: 0360-0300. DOI: 10.1145/3560815. URL: https://dl.acm.org/doi/10.1145/3560815 (visited on 05/21/2025).

[55] Zemin Liu et al. "GraphPrompt: Unifying Pre-Training and Downstream Tasks for Graph Neural Networks". In: *Proceedings of the ACM Web Conference 2023*. WWW '23. New York, NY, USA: Association for Computing Machinery, Apr. 2023, pp. 417–428. ISBN: 978-1-4503-9416-1. DOI: 10.1145/3543507.3583386. URL: https://dl.acm.org/doi/10.1145/3543507.3583386 (visited on 05/03/2025).

[56] Lajanugen Logeswaran et al. "Zero-Shot Entity Linking by Reading Entity Descriptions". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Ed. by Anna Korhonen, David Traum, and Lluís Màrquez. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 3449–3460. DOI: 10.18653/v1/P19-1335. URL: https://aclanthology.org/P19-1335/ (visited on 05/04/2025).

[57] Yinquan Lu et al. "KELM: Knowledge Enhanced Pre-Trained Language Representations with Message Passing on Hierarchical Relational Graphs". en. In: Apr. 2022. URL: https://openreview.net/forum?id=rG-NDTSWGc (visited on 04/24/2025).

[58] Yuxing Lu et al. *Biomedical Knowledge Graph: A Survey of Domains, Tasks, and Real-World Applications*. arXiv:2501.11632 [cs]. Jan. 2025. DOI: 10.48550/arXiv.2501.11632. URL: http://arxiv.org/abs/2501.11632 (visited on 01/24/2025).

[59] Kelvin Luu et al. *Time Waits for No One! Analysis and Challenges of Temporal Misalignment*. arXiv:2111.07408. July 2022. DOI: 10.48550/arXiv.2111.07408. URL: http://arxiv.org/abs/2111.07408 (visited on 12/11/2024).

[60] Potsawee Manakul, Adian Liusie, and Mark Gales. "SelfCheckGPT: Zero-Resource Black-Box Hallucination Detection for Generative Large Language Models". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 9004–9017. DOI: 10.18653/v1/2023.emnlp-main.557. URL: https://aclanthology.org/2023.emnlp-main.557/ (visited on 05/21/2025).

[61] Yossi Matias and Aashima Gupta. *MedLM: generative AI fine-tuned for the healthcare industry*. en-US. Dec. 2023. URL: https://cloud.google.com/blog/topics/healthcare-life-sciences/introducing-medlm-for-the-healthcare-industry (visited on 05/02/2025).

[62] Nicholas Matsumoto et al. "ESCARGOT: An AI Agent Leveraging Large Language Models, Dynamic Graph of Thoughts, and Biomedical Knowledge Graphs for Enhanced Reasoning". In: *Bioinformatics* (Jan. 2025), btaf031. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btaf031. URL: https://doi.org/10.1093/bioinformatics/btaf031 (visited on 01/24/2025).

[63] Nicholas Matsumoto et al. "KRAGEN: a knowledge graph-enhanced RAG framework for biomedical problem solving using large language models". In: *Bioinformatics* 40.6 (June 2024), btae353. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btae353. URL: https://doi.org/10.1093/bioinformatics/btae353 (visited on 12/05/2024).

[64] Manisha Mehta and Fausto Giunchiglia. *Understanding Gen Alpha Digital Language: Evaluation of LLM Safety Systems for Content Moderation*. arXiv:2505.10588 [cs]. May 2025. DOI: 10.1145/3715275.3732184. URL: http://arxiv.org/abs/2505.10588 (visited on 06/10/2025).

[65] Virendra Mehta et al. "Aurora-M: Open Source Continual Pre-training for Multilingual Language and Code". In: *Proceedings of the 31st International Conference on Computational Linguistics: Industry Track*. Ed. by Owen Rambow et al. Abu Dhabi, UAE: Association for Computational Linguistics, Jan. 2025, pp. 656–678. URL: https://aclanthology.org/2025.coling-industry.56/ (visited on 01/31/2025).

[66] Virendra Mehta et al. “Detecting Feature Eligibility Illusions in Enterprise {AI} Autopilots”. en. In: 2020. URL: https://www.usenix.org/conference/opml20/presentation/casati (visited on 02/12/2025).

[67] Virendra Mehta et al. “Federated benchmarking of medical artificial intelligence with MedPerf”. en. In: *Nature Machine Intelligence* 5.7 (July 2023). Publisher: Nature Publishing Group, pp. 799–810. ISSN: 2522-5839. DOI: 10.1038/s42256-023-00652-2. URL: https://www.nature.com/articles/s42256-023-00652-2 (visited on 11/26/2024).

[68] Virendra Mehta et al. “Mˆ2CRB: A Multilingualˆ2 Code Retrieval Benchmark and Experiments with LLMs that Generate and Retrieve”. en. In: (Jan. 2023). URL: https://openreview.net/forum?id=CoNkhVQVRu (visited on 01/29/2023).

[69] Virendra Mehta et al. *MedPerf: Open Benchmarking Platform for Medical Artificial Intelligence using Federated Evaluation*. arXiv:2110.01406. Dec. 2021. DOI: 10.48550/arXiv.2110.01406. URL: http://arxiv.org/abs/2110.01406 (visited on 11/26/2024).

[70] Virendra Mehta et al. “Multilingual Code Retrieval Without Paired Data: A New Benchmark and Experiments”. In: *ICLR DL4C Workshop*. 2023. (Visited on 02/12/2025).

[71] Virendra Kumar Mehta. “Handling caught exceptions”. en. US7877740B2. Jan. 2011. URL: https://patents.google.com/patent/US7877740B2/en (visited on 02/12/2025).

[72] Virendra Kumar Mehta and Sandya S. Mannarswamy. “System and method for dynamic instrumentation”. en. US7926042B2. Apr. 2011. URL: https://patents.google.com/patent/US7926042B2/en (visited on 02/12/2025).

[73] Kevin Meng et al. *Mass-Editing Memory in a Transformer*. arXiv:2210.07229 [cs]. Aug. 2023. DOI: 10.48550/arXiv.2210.07229. URL: http://arxiv.org/abs/2210.07229 (visited on 05/04/2025).

[74] Natalya F Noy and Deborah L McGuinness. “Ontology Development 101: A Guide to Creating Your First Ontology”. en. In: ().

[75] Dillon Obika et al. “Safety principles for medical summarization using generative AI”. en. In: *Nature Medicine* 30.12 (Dec. 2024). Publisher: Nature Publishing Group, pp. 3417–3419. ISSN: 1546-170X. DOI: 10.1038/s41591-024-03313-y. URL: https://www.nature.com/articles/s41591-024-03313-y (visited on 05/02/2025).

[76] Long Ouyang et al. “Training language models to follow instructions with human feedback”. In: *Proceedings of the 36th International Conference on Neural Information Processing Systems*. NIPS ’22. Red Hook, NY, USA: Curran Associates Inc., Nov. 2022, pp. 27730–27744. ISBN: 978-1-71387-108-8. (Visited on 05/01/2025).

[77] Jeff Z. Pan et al. *Large Language Models and Knowledge Graphs: Opportunities and Challenges*. arXiv:2308.06374 [cs]. Aug. 2023. DOI: 10.48550/arXiv.2308.06374. URL: http://arxiv.org/abs/2308.06374 (visited on 12/25/2024).

[78] Shirui Pan et al. “Unifying Large Language Models and Knowledge Graphs: A Roadmap”. In: *IEEE Transactions on Knowledge and Data Engineering* (2024). Conference Name: IEEE Transactions on Knowledge and Data Engineering, pp. 1–20. ISSN: 1558-2191. DOI: 10.1109/TKDE.2024.3352100. URL: https://ieeexplore.ieee.org/document/10387715 (visited on 05/19/2024).

[79] Hammond Pearce et al. “Asleep at the Keyboard? Assessing the Security of GitHub Copilot’s Code Contributions”. English. In: IEEE Computer Society, May 2022, pp. 754–768. ISBN: 978-1-66541-316-9. DOI: 10.1109/SP46214.2022.9833571. URL: https://www.computer.org/csdl/proceedings-article/sp/2022/131600a980/1FlQxERjKCs (visited on 11/26/2024).

[80] Matthew E. Peters et al. “Knowledge Enhanced Contextual Word Representations”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Ed. by Kentaro Inui et al. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 43–54. DOI: 10.18653/v1/D19-1005. URL: https://aclanthology.org/D19-1005/ (visited on 05/04/2025).

[81] Yifu Qiu et al. “Are Large Language Model Temporally Grounded?” In: *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. Ed. by Kevin Duh, Helena Gomez, and Steven Bethard. Mexico City, Mexico: Association for Computational Linguistics, June 2024, pp. 7064–7083. DOI: 10.18653/v1/2024.naacl-long.391. URL: https://aclanthology.org/2024.naacl-long.391/ (visited on 05/04/2025).

[82] Vipula Rawte, Amit Sheth, and Amitava Das. *A Survey of Hallucination in Large Foundation Models*. arXiv:2309.05922. Sept. 2023. DOI: 10.48550/arXiv.2309.05922. URL: http://arxiv.org/abs/2309.05922 (visited on 11/18/2024).

[83] Baptiste Rozière et al. *Code Llama: Open Foundation Models for Code*. arXiv:2308.12950 [cs]. Jan. 2024. DOI: 10.48550/arXiv.2308.12950. URL: http://arxiv.org/abs/2308.12950 (visited on 01/17/2025).

[84] Apoorv Saxena, Adrian Kochsiek, and Rainer Gemulla. “Sequence-to-Sequence Knowledge Graph Completion and Question Answering”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Smaranda Muresan, Preslav Nakov, and Aline Villavicencio. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 2814–2828. DOI: 10.18653/v1/2022.acl-long.201. URL: https://aclanthology.org/2022.acl-long.201/ (visited on 05/04/2025).

[85] Burr Settles. *Active Learning Literature Survey*. en. Technical Report. Accepted: 2012-03-15T17:23:56Z. University of Wisconsin-Madison Department of Computer Sciences, 2009. URL: https://minds.wisconsin.edu/handle/1793/60660 (visited on 05/21/2025).

[86] Shreya Shankar et al. “Who Validates the Validators? Aligning LLM-Assisted Evaluation of LLM Outputs with Human Preferences”. In: *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. UIST ’24. New York, NY, USA: Association for Computing Machinery, Oct. 2024, pp. 1–14. ISBN: 9798400706288. DOI: 10.1145/3654777.3676450. URL: https://dl.acm.org/doi/10.1145/3654777.3676450 (visited on 05/21/2025).

[87] Tushar Sharma et al. “A survey on machine learning techniques applied to source code”. In: *Journal of Systems and Software* 209 (Mar. 2024), p. 111934. ISSN: 0164-1212. DOI: 10.1016/j.jss.2023.111934. URL: https://www.sciencedirect.com/science/article/pii/S0164121223003291 (visited on 01/10/2024).

[88] Bhawna Singh. “Stop Hallucinations with RAG”. en. In: *Building Applications with Large Language Models: Techniques, Implementation, and Applications*. Ed. by Bhawna Singh. Berkeley, CA: Apress, 2024, pp. 117–143. ISBN: 9798868805691. DOI: 10.1007/979-8-8688-0569-1_5. URL: https://doi.org/10.1007/979-8-8688-0569-1_5 (visited on 12/09/2024).

[89] Karan Singhal et al. “Large language models encode clinical knowledge”. en. In: *Nature* 620.7972 (Aug. 2023). Publisher: Nature Publishing Group, pp. 172–180. ISSN: 1476-4687. DOI: 10.1038/s41586-023-06291-2. URL: https://www.nature.com/articles/s41586-023-06291-2 (visited on 11/29/2024).

[90] Joseph Spracklen et al. *We Have a Package for You! A Comprehensive Analysis of Package Hallucinations by Code Generating LLMs*. arXiv:2406.10279 [cs]. Mar. 2025. DOI: 10.48550/arXiv.2406.10279. URL: http://arxiv.org/abs/2406.10279 (visited on 05/03/2025).

[91] Nisan Stiennon et al. “Learning to summarize from human feedback”. In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS ’20. Red Hook, NY, USA: Curran Associates Inc., Dec. 2020, pp. 3008–3021. ISBN: 978-1-71382-954-6. (Visited on 05/21/2025).

[92] Fabian Suchanek et al. *YAGO 4.5: A Large and Clean Knowledge Base with a Rich Taxonomy*. arXiv:2308.11884 [cs] version: 2. Apr. 2024. DOI: 10.48550/arXiv.2308.11884. URL: http://arxiv.org/abs/2308.11884 (visited on 04/28/2025).

[93] Shaznin Sultana, Sadia Afreen, and Nasir U. Eisty. *Code Vulnerability Detection: A Comparative Analysis of Emerging Large Language Models*. arXiv:2409.10490. Sept. 2024. DOI: 10.48550/arXiv.2409.10490. URL: http://arxiv.org/abs/2409.10490 (visited on 11/26/2024).

[94] Hugo Touvron et al. “LLaMA: Open and Efficient Foundation Language Models”. In: *ArXiv* (Feb. 2023). URL: https://www.semanticscholar.org/paper/LLaMA%3A-Open-and-Efficient-Foundation-Language-Touvron-Lavril/57e849d0de13ed5f91d086936296721d4ff75a75 (visited on 05/01/2025).

[95] Ashish Vaswani et al. “Attention is all you need”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Red Hook, NY, USA: Curran Associates Inc., Dec. 2017, pp. 6000–6010. ISBN: 978-1-5108-6096-4. (Visited on 05/01/2022).

[96] Alex Wang et al. “SuperGLUE: a stickier benchmark for general-purpose language understanding systems”. In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. 294. Red Hook, NY, USA: Curran Associates Inc., Dec. 2019, pp. 3266–3280. (Visited on 05/01/2025).

[97] Chong Wang et al. *How and Why LLMs Use Deprecated APIs in Code Completion? An Empirical Study*. arXiv:2406.09834. July 2024. DOI: 10.48550/arXiv.2406.09834. URL: http://arxiv.org/abs/2406.09834 (visited on 11/24/2024).

[98] Xiaozhi Wang et al. "KEPLER: A Unified Model for Knowledge Embedding and Pre-trained Language Representation". In: *Transactions of the Association for Computational Linguistics* 9 (2021). Ed. by Brian Roark and Ani Nenkova. Place: Cambridge, MA Publisher: MIT Press, pp. 176–194. DOI: 10.1162/tacl_a_00360. URL: https://aclanthology.org/2021.tacl-1.11/ (visited on 05/04/2025).

[99] Xinshun Wang et al. "Dynamic Dense Graph Convolutional Network for Skeleton-Based Human Motion Prediction". In: *IEEE Transactions on Image Processing* 33 (2024), pp. 1–15. ISSN: 1941-0042. DOI: 10.1109/TIP.2023.3334954. URL: https://ieeexplore.ieee.org/document/10335618 (visited on 05/04/2025).

[100] Yixin Wang, Zihao Lin, and Haoyu Dong. *Rethinking Medical Report Generation: Disease Revealing Enhancement with Knowledge Graph*. arXiv:2307.12526 [cs]. July 2023. DOI: 10.48550/arXiv.2307.12526. URL: http://arxiv.org/abs/2307.12526 (visited on 05/04/2025).

[101] Jason Wei et al. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". en. In: *Advances in Neural Information Processing Systems* 35 (Dec. 2022), pp. 24824–24837. URL: https://proceedings.neurips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html (visited on 11/21/2024).

[102] Jiayi Zhang et al. "Using Large Language Models to Detect Self-Regulated Learning in Think-Aloud Protocols". en-US. In: 2024, pp. 157–168. DOI: 10.5281/zenodo.12729790. URL: https://educationaldatamining.org/edm2024/proceedings/2024.EDM-long-papers.13/index.html (visited on 05/04/2025).

[103] Siyun Zhao et al. *Retrieval Augmented Generation (RAG) and Beyond: A Comprehensive Survey on How to Make your LLMs use External Data More Wisely*. en. arXiv:2409.14924 [cs]. Sept. 2024. DOI: 10.48550/arXiv.2409.14924. URL: http://arxiv.org/abs/2409.14924 (visited on 12/11/2024).

[104] Wayne Xin Zhao et al. *A Survey of Large Language Models*. arXiv:2303.18223. Oct. 2024. DOI: 10.48550/arXiv.2303.18223. URL: http://arxiv.org/abs/2303.18223 (visited on 11/17/2024).

[105] Yu Zhao and Haoxiang Gao. *Utilizing Large Language Models for Information Extraction from Real Estate Transactions*. arXiv:2404.18043 [cs]. Dec. 2024. DOI: 10.48550/arXiv.2404.18043. URL: http://arxiv.org/abs/2404.18043 (visited on 04/24/2025).

[106] Lingfeng Zhong et al. "A Comprehensive Survey on Automatic Knowledge Graph Construction". In: *ACM Comput. Surv.* 56.4 (Nov. 2023), 94:1–94:62. ISSN: 0360-0300. DOI: 10.1145/3618295. URL: https://doi.org/10.1145/3618295 (visited on 11/15/2024).

Appendix A

Reflexive Composition Implementation Details

A.1 Reflexive Composition Module Implementation

This appendix documents the implementation of the core `reflexive_composition` Python module, which supports structured knowledge extraction, human-in-the-loop validation, and safe response generation using knowledge graph (KG) context. The module is used across all case studies described in this thesis.[1]

A.1.1 Core Architecture

The system is centered around the `ReflexiveComposition` class, which coordinates the following components:

- `KnowledgeBuilderLLM` — an LLM-based extractor for transforming unstructured input into schema-aligned triples.
- `Validator` — supports human or model-driven review of triples before inclusion in the KG.
- `KnowledgeGraph` — stores triples and enforces schema constraints.
- `TargetLLM` — generates responses using KG context and a controlled prompting template.

The class is instantiated with explicit model and storage configurations:

Listing A.1: Instantiating the ReflexiveComposition class

```
rc = ReflexiveComposition(
    kb_llm_config=kb_model_config,
    target_llm_config=target_model_config,
    kg_config=kg_storage_config,
    hitl_config=validator_config
)
```

[1] Source code available at `https://github.com/veerumehta/ReflexiveComposition`

A.1.2 Project Structure Overview

The full project is organized into reusable core modules and case-specific examples:

```
reflexive_composition/
 core.py                    # Orchestrator class
 kg2llm/                    # Prompt builder and LLM interface
 llm2kg/                    # Knowledge extraction from text
 hitl/                      # Confidence-based filtering and routing
 knowledge_graph/           # Schema-aware triple storage
 utils/                     # Logging, evaluation, helper tools

examples/
 basic_example.py           # Used throughout this thesis
 code_security/             # CVE extraction and secure code generation
 medical_data/              # FHIR + RxNorm grounding for QA
 temporal_qa/               # Current events and temporal question answering
```

All components are installed as a Python package via `pyproject.toml`, enabling use in scripts or interactive notebooks.

A.1.3 Key Interfaces

Each phase of the pipeline is exposed via dedicated methods, defined in `core.py`:

Listing A.2: Interface methods exposed by ReflexiveComposition

```
def extract_knowledge(self, text: str, schema: Optional[Dict] = None) -> List[Dict]:
    # Runs the LLM2KG phase. Returns raw extracted triples.

def validate_knowledge(self, triples: List[Dict]) -> List[Dict]:
    # Optionally filters or corrects triples using HITL or auto-validation.

def generate_response(self, query: str, retrieve_context: bool = True) -> str:
    # Answers the query using the KG as context if enabled.
```

A.1.4 Prompt Construction and Control

Prompts are template-driven and injected with schema guidance. Below is the KG2LLM template used when responding with KG context:

Listing A.3: KG2LLM With-Context Prompt Template

```
WITH_CONTEXT_PROMPT_TEMPLATE = '''
Use the following knowledge graph to answer the user query.
Knowledge:
{graph_context}
```

```
Query:
{query}

Answer:
'''
```

LLM2KG uses schema-conditioned instructions:

Listing A.4: LLM2KG extraction prompt (excerpt)

```
EXTRACTION_PROMPT = f'''
Extract structured knowledge from the following text.
Return the result as a JSON list of triples, following this schema.

Entity types: {entity_types}
Relation types: {relationship_types}

Text:
{text}
'''
```

A.1.5 Schema Format

Schemas are defined using a lightweight JSON format, and loaded dynamically into the pipeline:

Listing A.5: Example Schema: Code Security

```
{
  "entity_types": ["API", "Vulnerability", "Recommendation"],
  "relationship_types": ["isDeprecated", "hasReplacement", "hasCVE"]
}
```

The schema is passed into extraction and validation phases, and enforced at KG insertion time.

A.1.6 Case Study Integration via Examples

All case studies build on this shared module and are implemented in the `examples/` directory. Each example includes:

- An input text or data source (e.g., clinical notes, news snippet, or code doc).
- A schema specification.
- A configured instance of `ReflexiveComposition`.
- A pipeline that runs: extraction → validation → response generation.

Each example script follows a consistent structure. For example, the Code Security case:

Listing A.6: Excerpt from `run_codegen.py`

```
rc = ReflexiveComposition(...)
triples = rc.extract_knowledge(text=api_docs, schema=schema)
curated = rc.validate_knowledge(triples)
result = rc.generate_response(query="Should I use legacyLogin?")
```

Similar scripts exist for:

- `examples/code_security/run_codegen.py`
- `examples/temporal_qa/run_temporal.py`
- `examples/medical_data/run_medical.py`

A.1.7 Extensibility and Developer Usage

To implement a new use case:

1. Define your schema in JSON format.
2. Prepare input text (e.g., web data, forum posts, clinical notes).
3. Create a script using the `ReflexiveComposition` API.
4. Customize the validator or prompt as needed.

Basic invocation:

```
python examples/your_case_study/run_yourcase.py
```

This modularity enables reuse across multiple domains, while preserving the structured KG grounding that underpins reliable generation.

Appendix B

Code Security Implementation Details

This appendix details the implementation of the Code Security case study described in Chapter 7, which focuses on mitigating historical bias in code generation through Reflexive Composition. The implementation combines structured knowledge extraction from API documentation with validator-driven refinement and secure code generation through KG-guided prompting.

B.1 Architecture Overview

The Code Security implementation follows the three-component architecture of Reflexive Composition:

- **LLM2KG:** Extracts API deprecation information, security vulnerabilities, and recommended replacements from documentation and code analysis
- **HITL:** Enables security experts to validate extracted information and ensure accuracy of security recommendations
- **KG2LLM:** Injects validated security knowledge into code generation prompts to guide secure implementation

The implementation focuses specifically on identifying deprecated or vulnerable APIs and ensuring generated code adheres to modern security best practices.

B.2 Knowledge Representation

B.2.1 Schema Definition

The domain-specific knowledge schema for code security focuses on representing API deprecations, vulnerabilities, and replacement recommendations:

Listing B.1: Code Security Schema Definition

```
kg_config = {
```

```
"storage_type": "in_memory",
"schema": {
"entity_types": ["API", "Function", "Vulnerability", "Deprecation", "Version"],
"relationship_types": ["DeprecatedIn", "ReplacedBy", "CausesVulnerability"],
"version": 1
}
}
```

B.2.2 API Deprecation Ontology

The ontology represents a comprehensive hierarchy of APIs, versions, replacements, and security implications. Core classes in the ontology include:

- **APIMethod** - Base class for API methods and functions with properties including name, type, and deprecation status
- **PythonVersion** - Represents specific Python versions with version number and release date
- **ReplacementMethod** - Subclass of APIMethod representing recommended alternatives
- **SecurityRiskLevel** - Enumeration (Low, Medium, High) with risk description and mitigation steps

Key relationships in the ontology include:

- `is_deprecated_in_version` - Links APIs to specific Python versions
- `recommended_replacement` - Connects deprecated APIs to their secure alternatives
- `security_risk_level` - Associates APIs with their security impact

B.3 Knowledge Extraction Pipeline

The knowledge extraction pipeline combines multiple approaches to identify deprecated APIs and security vulnerabilities from documentation and code:

B.3.1 Static AST Analysis

For direct extraction from Python code, we implement AST (Abstract Syntax Tree) parsing to identify deprecated functions through docstring analysis:

Listing B.2: AST-Based Deprecation Detection

```
import ast
class DeprecationVisitor(ast.NodeVisitor):
    def visit_FunctionDef(self, node):
        if node.doc and "deprecated" in ast.get_docstring(node).lower():
            print(f"Deprecated function: {node.name}")
```

B.3.2 LLM-Based Extraction

For documentation sources, we use prompt-guided LLM extraction to identify deprecation patterns and security implications:

Listing B.3: LLM Extraction Prompting

```
DEPRECATION_PROMPT = """
Given the following API documentation, extract deprecated methods,
the version they were deprecated in, and recommended replacements.
Text:
"""
{documentation}
"""
Expected JSON format:
{
"api": "name",
"deprecated_in": "version",
"removed_in": "version",
"replacement": "alternative",
"security_implications": "description"
}
"""
```

B.3.3 Source Processing

The full extraction pipeline follows these steps:

1. Parse documentation and code sources
2. Extract entities (API methods, versions, vulnerabilities)
3. Identify relationships (deprecation timing, replacements)
4. Validate against ontology constraints
5. Convert to triple format for KG integration

B.4 Human-in-the-Loop Validation

Security experts validate the extracted knowledge through a systematic workflow:

Listing B.4: HITL Validation Interface

```
Validator confirmation of extracted triples
response = hitl.confirm_triples([
("cgi.escape", "DeprecatedIn", "3.0"),
("cgi.escape", "ReplacedBy", "html.escape"),
```

```
("cgi.escape", "CausesVulnerability", "XSS")
])
```

The HITL workflow includes:

- **Schema Compliance Review:** Ensuring extracted information conforms to the ontology
- **Vulnerability Mapping Accuracy:** Verifying correct association of vulnerabilities with APIs
- **Temporal Consistency:** Confirming version-specific deprecation timing
- **Secure Replacement Verification:** Validating that recommended alternatives align with security best practices

B.5 Knowledge-Guided Code Generation

Once the security knowledge graph is populated and validated, it guides code generation through structured prompt injection:

Listing B.5: Knowledge-Grounded Prompt Templates

```
WITH_CONTEXT_PROMPT_TEMPLATE = """You are a secure code generation assistant.
{context_block}
User query:
{query}
Answer the query using only the facts above. If the context does not provide enough inform
say so clearly. Do not make up recommendations without support from the context."""

NO_CONTEXT_PROMPT_TEMPLATE = """You are a secure code generation assistant.
User query:
{query}
Answer using your own general knowledge. Do not reference external documents or links."""
```

The code generation workflow:

1. User submits a code generation query
2. System retrieves relevant security knowledge from validated graph
3. Knowledge is injected into the prompt as structured context
4. Code suggestions are generated with explicit security guidance

B.6 Complete Implementation Example

The following demonstrates the full Reflexive Composition pipeline for the code security case study:

Listing B.6: Code Security Implementation Example

```
Initialize Reflexive Composition framework
rc = ReflexiveComposition(
    kb_llm_config=kb_llm_config,
    target_llm_config=target_llm_config,
    kg_config=kg_config,
    hitl_config=hitl_config
)

Extract security knowledge from documentation
source_text = """
The Python 'cgi' module was officially deprecated in version 3.11 due to long-standin
security issues. Developers are advised to use safer alternatives like 'http.server'
or third-party libraries. The 'cgi.escape' method, in particular, has caused XSS vul
in the past.
"""

extraction_result = rc.extract_knowledge(
    source_text=source_text,
    schema=kg_config["schema"],
    confidence_threshold=0.7
)
rc.knowledge_graph.add_triples(extraction_result["triples"])

Generate secure code with knowledge guidance
prompt_result = rc.generate_response(
    query="What is a safe alternative to using the 'cgi.escape' function in modern P
    grounded=True,
    template=WITH_CONTEXT_PROMPT_TEMPLATE,
)
```

B.7 Evaluation Dataset

The evaluation dataset consists of a series of security-focused code generation queries addressing deprecated APIs, including:

Listing B.7: Sample Evaluation Queries

```
{"id": "q1", "query": "Is the use of cgi.escape considered safe in Python 3.11?"}
{"id": "q2", "query": "What is a safer alternative to cgi.escape?"}
{"id": "q3", "query": "Why is cgi module deprecated?"}
{"id": "q4", "query": "Which version of Python deprecated the cgi module?"}
{"id": "q5", "query": "What vulnerability is cgi.escape associated with?"}
```

B.8 Example Deprecation Catalog

The knowledge graph incorporates a comprehensive catalog of Python API deprecations. Selected examples include:

B.8.1 AST Module Deprecations

- `ast.Num` - Deprecated in Python 3.8, to be removed in Python 3.14, replaced by `ast.Constant`
- `ast.Str` - Deprecated in Python 3.8, to be removed in Python 3.14, replaced by `ast.Constant`
- `ast.Bytes` - Deprecated in Python 3.8, to be removed in Python 3.14, replaced by `ast.Constant`
- `ast.NameConstant` - Deprecated in Python 3.8, to be removed in Python 3.14, replaced by `ast.Constant`
- `ast.Ellipsis` - Deprecated in Python 3.8, to be removed in Python 3.14, replaced by `ast.Constant`

B.8.2 Urllib Module Deprecations

- `urllib.request.URLopener` - Deprecated, replaced by `urlopen`
- `urllib.request.FancyURLopener` - Deprecated, replaced by `urlopen`
- `urllib.parse.splitattr` - Deprecated, replaced by `urllib.parse.urlparse`
- `urllib.parse.splithost` - Deprecated, replaced by `urllib.parse.urlparse`
- `urllib.parse.splitpasswd` - Deprecated, replaced by `urllib.parse.urlparse`

B.8.3 Threading Module Deprecations

- `threading.Condition.notifyAll` - Deprecated, replaced by `notify_all`
- `threading.Event.isSet` - Deprecated, replaced by `is_set`
- `threading.Thread.isDaemon` - Deprecated, replaced by `threading.Thread.daemon` attribute
- `threading.Thread.setDaemon` - Deprecated, replaced by `threading.Thread.daemon` attribute
- `threading.currentThread` - Deprecated, replaced by `threading.current_thread`

This is a subset of the larger catalog, selected for their security and compatibility implications in modern Python development.

Appendix C

Temporal Knowledge Implementation Details

This appendix supports Chapter 8, which investigates the role of time in knowledge-grounded reasoning with large language models (LLMs). It provides the implementation details of two case studies developed using the Reflexive Composition framework:

- A temporal QA system using structured event extraction for Newcastle United's 2025 EFL Cup victory.
- A temporal recency benchmark built around the July 2024 Trump rally shooting attempt.

C.1 Challenges in Temporal QA

Temporal reasoning with LLMs presents unique challenges:

- **Recency failure**: LLMs trained on stale data hallucinate outdated answers.
- **Contradiction detection**: LLMs may offer factually inconsistent statements for related queries.
- **Temporal alignment**: Time-sensitive queries require integration of entity timelines with external facts.
- **Context vs no-context performance**: Answers vary significantly depending on whether extracted knowledge is explicitly provided.

Demonstration Datasets

The repository includes both the original Trump 2024 temporal case study used throughout Chapter 8 and additional up-to-date sports-based examples (e.g., Newcastle Uniteds 2025 EFL Cup win). These serve to show the extensibility of the Reflexive Composition framework across evolving domains.

C.2 Schema and Prompt Structure

The Newcastle example and Trump benchmark use a common schema structure consisting of temporal and entity-specific relations.

C.2.1 Event Schema Configuration

Listing C.1: Schema Definition

```
kg_config = {
  "storage_type": "in_memory",
  "schema": {
    "entity_types": ["Team", "Player", "Match", "Trophy", "Date"],
    "relationship_types": ["Won", "PlayedIn", "Scored", "OccurredOn", "EndedDrought"],
    "version": 1
  }
}
```

C.2.2 Prompt Templates

We support both grounded and ungrounded reasoning modes:

Listing C.2: Grounded Prompt Template

```
WITH_CONTEXT_PROMPT_TEMPLATE = """You are a timeline-aware assistant trained to use factu

{context_block}

Question:
{query}

Answer using only the above facts. If the answer cannot be determined from them, say so."
```

Listing C.3: Ungrounded Prompt Template

```
NO_CONTEXT_PROMPT_TEMPLATE = """You are a timeline-aware assistant trained to use factual

Question:
{query}

Answer using your own general knowledge. Do not reference external documents or links."""
```

C.3 Newcastle Case Study: QA from Match Summary

C.3.1 Source Text

> Newcastle United defeated Liverpool 2–1 in the 2025 EFL Cup final held at Wembley on March 16, 2025. It was Newcastle's first major domestic trophy in 70 years, having last won the FA Cup in 1955. Goals were scored by Alexander Isak and Bruno Guimares.

C.3.2 Extracted Triples

Listing C.4: Knowledge Triples

```
[
  {"subject": "Newcastle United", "predicate": "Won", "object": "EFL Cup"},
  {"subject": "2025 EFL Cup Final", "predicate": "OccurredOn", "object": "2025-03-16"},
  {"subject": "Alexander Isak", "predicate": "Scored", "object": "2025 EFL Cup Final"},
  {"subject": "Bruno Guimar es", "predicate": "Scored", "object": "2025 EFL Cup Final"},
  {"subject": "EFL Cup", "predicate": "EndedDrought", "object": "Newcastle United"}
]
```

C.3.3 Query Examples

Listing C.5: QA Queries

```
[
  {"id": "q1", "query": "When did Newcastle United last win a major domestic trophy?"},
  {"id": "q2", "query": "Who scored in the 2025 EFL Cup final for Newcastle?"},
  {"id": "q3", "query": "How long was Newcastle's trophy drought before 2025?"}
]
```

C.3.4 QA Execution

Listing C.6: Grounded QA Execution

```
response = rc.generate_response(
  query="Who scored in the 2025 EFL Cup final for Newcastle?",
  grounded=True,
  template=WITH_CONTEXT_PROMPT_TEMPLATE
)
```

C.4 Trump Rally Benchmark Case Study

C.4.1 JSON Fact Format

Listing C.7: Trump Shooting Event JSON

```
{
  "subject": "Donald Trump",
  "event_type": "Assassination Attempt",
  "location": "Butler, PA",
  "date": "2024-07-13",
  "outcome": "NonFatal",
  "query": "What happened at Trump's rally on July 13, 2024?"
}
```

C.4.2 Response Generation

Listing C.8: Trump QA Execution

```
response = rc.generate_response(
  query="Was Trump injured in the July 2024 rally incident?",
  grounded=True,
  template=WITH_CONTEXT_PROMPT_TEMPLATE
)
```

C.5 Evaluation and Output Logging

The system logs each QA instance with a timestamp, prompt, model used, and whether grounding was applied.

Listing C.9: Logging Results

```
log_result({
  "query": query,
  "answer": response,
  "grounded": True,
  "model_version": "gpt-4",
})
```

C.6 Conclusion

The temporal QA case studies demonstrate schema-driven grounding, structured extraction, and prompt-sensitive evaluation using Reflexive Composition. They show how factual knowledge graphs and recency-aware prompts mitigate hallucination and improve response accuracy for time-sensitive events.

Appendix D

Medical Domain Implementation Details

D.1 Introduction and Domain Overview

This appendix documents the implementation details of the medical case study presented in Chapter 9. The Reflexive Composition framework was applied to clinical question answering by combining structured patient data (FHIR) with semantic enrichment from external medical ontologies. This integration demonstrates how the architecture can support privacy-preserving, traceable clinical reasoning in a regulated healthcare environment.

The implementation addresses several domain-specific challenges:

- Maintaining HIPAA compliance while leveraging LLM capabilities
- Integrating heterogeneous medical data sources (FHIR, clinical notes)
- Ensuring temporal coherence in treatment timelines
- Enabling auditability and validator oversight

D.2 Medical Knowledge Representation

D.2.1 FHIR Standard Implementation

FHIR (Fast Healthcare Interoperability Resources) provides a standardized approach to healthcare data exchange. Figure D.1 illustrates the core FHIR resource relationships.

D.2.2 Dual-Graph Architecture

The system structures clinical knowledge using a dual-graph architecture that separates patient-specific facts from general biomedical terminology:

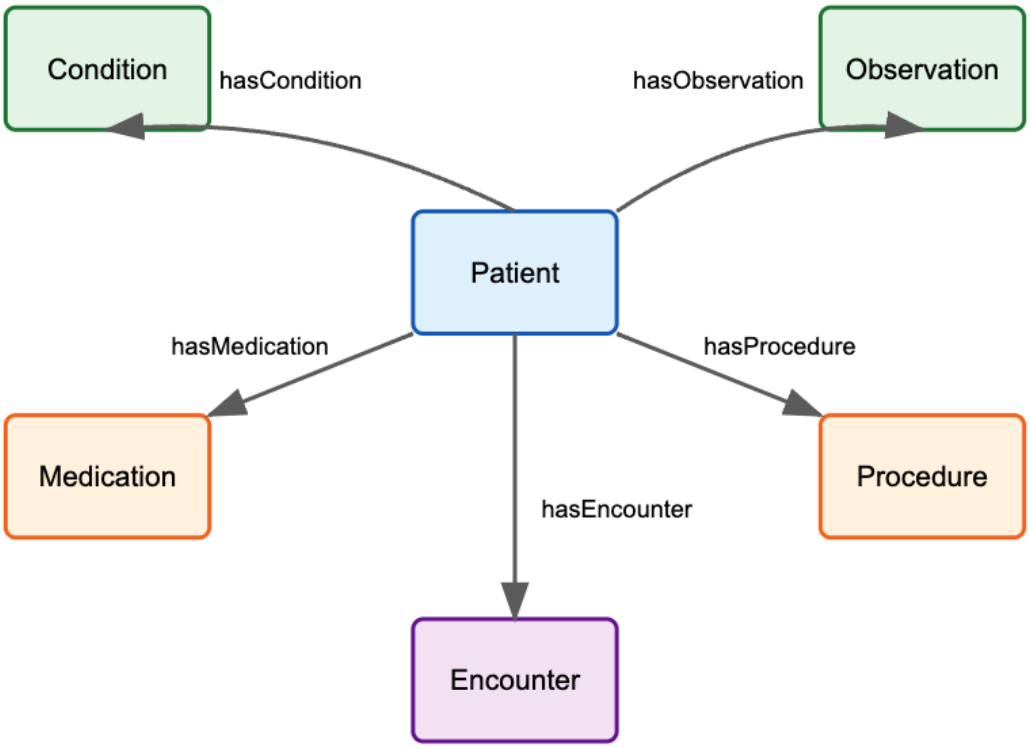


Figure D.1: FHIR Resource Structure and Relationships

- **A-graph (assertional graph):** Encodes dynamic, patient-specific data extracted from FHIR-formatted EHRs. This includes `Condition`, `MedicationRequest`, `Observation`, and other temporal clinical assertions.
- **T-graph (terminological graph):** Provides a stable layer of biomedical ontologiesSNOMED-CT, RxNorm, LOINCthat normalize, align, and disambiguate clinical terms used in both structured and unstructured inputs.

These graphs interact bidirectionally: free-text terms like "Z-pack" are resolved to canonical identifiers (e.g., "azithromycin") using the T-graph and recorded in the A-graph; structured entries from the A-graph are semantically augmented and validated using the T-graph.

D.2.3 Knowledge Graph Transformation

The transformation from FHIR to knowledge graph follows a systematic process:

1. **Schema Definition**
   - Define graph schema based on FHIR resource definitions
   - Map FHIR datatypes to graph-compatible formats
   - Create relationship types for FHIR references
2. **Data Transformation**

- Parse FHIR JSON/XML resources
- Extract core entities and attributes
- Generate graph nodes and relationships
- Validate against schema constraints

3. **Knowledge Enhancement**
   - Enrich graph with medical ontologies
   - Add derived relationships
   - Incorporate temporal reasoning capabilities
4. **Quality Assurance**
   - Verify referential integrity
   - Check semantic consistency
   - Validate terminology bindings

Figure D.2: FHIR-to-KG transformation methodology. Structured EHR inputs pass through validation, mapping, and ontology alignment before integration into the patient-specific knowledge graph.

D.3 LLM2KG: Structured Extraction from Clinical Sources

The LLM2KG pathway enables the system to expand and refine the patient-specific knowledge graph by interpreting both unstructured clinical notes and structured EHR entries. This reflexive process operates in two modes: extracting new assertions from free text, and semantically enriching structured records via biomedical ontologies.

D.3.1 Extraction Prompts and Templates

The system employs specialized prompts tailored to medical data extraction. These templates are designed to balance clinical accuracy with privacy protection:

```
FHIR_EXTRACTION_PROMPT = """
  Resource: {fhir_resource}
  Privacy Level: {privacy_requirements}
  Task: Extract entities while preserving HIPAA compliance
"""

MEDICAL_VALIDATION_PROMPT = """
  Clinical Data: {anonymized_data}
  Context: {medical_context}
  Task: Validate clinical accuracy while maintaining privacy
"""
```

When processing clinical notes, additional context is provided to ground the extraction in medical terminology:

```
CLINICAL_NOTE_EXTRACTION = """
  Note Text: {clinical_note}
  Terminology: {available_codes}
  Task: Extract medical entities, normalize to standard terminology,
   and preserve temporal relationships

  Expected Output Format:
  {
    "conditions": [{"name": "condition_name", "code": "snomed_code", "onset": "date"}],
    "medications": [{"name": "medication_name", "code": "rxnorm_code",
      "dosage": "dosage", "route": "route"}]
  } """
```

D.3.2 Data Formats and Examples

FHIR Patient Record Example

FHIR patient record example is shown in Figure D.3.

Synthetic MIMIC Patient Example

We created synthetic patient examples based on MIMIC IV as show in Figure D.4.

RxNorm Triples Example

Synthetic examples were crafted out of RxNorm as shown in Figure D.5.

FHIR Resource Processing

The system provides functions to transform FHIR resources into knowledge graph components as shown in Figure D.6.

D.4 Medical Large Language Models

D.4.1 Architecture and Training

Medical LLMs typically follow one of two development paths:

1. Fine-tuning foundation models on medical corpora
2. Training specialized architectures from medical datasets like PubMed, MIMIC III [45] etc.

Table D.1 presents a comparative analysis of prominent medical LLMs and their capabilities.

Figure D.3: FHIR Patient Record (fhir_patient.json)

```
{
  "entities": [
    { "type": "Patient", "name": "Subject_100002" },
    { "type": "Condition", "name": "Type 2 diabetes" },
    { "type": "Condition", "name": "Chronic kidney disease stage 3" },
    { "type": "Medication", "name": "Metformin" },
    { "type": "Medication", "name": "Lisinopril" }
  ],
  "relations": [
    {
      "subject": "Subject_100002",
      "predicate": "HasCondition",
      "object": "Type 2 diabetes"
    },
    {
      "subject": "Subject_100002",
      "predicate": "HasCondition",
      "object": "Chronic kidney disease stage 3"
    },
    {
      "subject": "Subject_100002",
      "predicate": "Prescribed",
      "object": "Metformin"
    },
    {
      "subject": "Subject_100002",
      "predicate": "Prescribed",
      "object": "Lisinopril"
    }
  ]
}
```

Table D.1: Comparative Analysis of Medical LLMs

Model	Architecture	Training Approach	Key Capabilities
MedPaLM	Transformer	Fine-tuned	Clinical reasoning
ClinicalBERT	BERT	Domain-specific	EMR comprehension
BioBERT	BERT	Domain-specific	Literature analysis

D.4.2 Integration Challenges

Medical LLMs face several critical challenges when integrating with clinical workflows:

1. Data Privacy: HIPAA compliance requirements constrain training data access
2. Domain Complexity: Medical terminology and relationships require specialized handling
3. Temporal Context: Clinical data often requires precise temporal understanding
4. Multi-modal Integration: Medical data spans text, images, and structured records

```
{
  "subject_id": 100002,
  "demographics": {
    "age": 64,
    "gender": "F"
  },
  "conditions": [
    {"icd_code": "E11", "label": "Type 2 diabetes"},
    {"icd_code": "N18.3", "label": "Chronic kidney disease stage 3"}
  ],
  "medications": [
    {"name": "Metformin", "start": "2023-06-10", "end": null},
    {"name": "Lisinopril", "start": "2023-06-10", "end": null}
  ],
  "note": "Patient followed for diabetes and CKD. Metformin continued.
     BP well-controlled on Lisinopril."
}
```


Figure D.4: Synthetic MIMIC Patient Record Example

These limitations become particularly apparent when dealing with rare conditions, complex comorbidities, or population-specific health variations[15].

Evaluation Examples from Healthcare Benchmark

The following examples illustrate how factuality, contradiction, and validator effort were evaluated on the 50-query benchmark.

Query	Is the patient currently on a blood pressure medication?
Gold Answer	Yes, lisinopril is listed as a prescribed antihypertensive.
LLM-only Output	No medications are listed in the record.
Reflexive Output	Yes, the patient is prescribed lisinopril, a medication used to treat hypertension.
F1 Score	0.86
Contradiction	LLM-only = Yes; Reflexive = No
Validator Escalated?	LLM-only = Yes; Reflexive = No

Table D.2: Sample evaluation output for a medication query.

Table D.3 highlights how knowledge fidelity improves across the four system configurations. The LLM-only and RAG responses lack precision and miss schema-level detail. KG-injected prompts improve recall but may still generalize or misinterpret. Reflexive Composition offers the most precise and validated response.

Additional annotated outputs are available in the public benchmark release.

```
[
  {
    "subject": "Coumadin",
    "predicate": "ClassifiesAs",
    "object": "BloodThinner"
  },
  {
    "subject": "Warfarin",
    "predicate": "SynonymFor",
    "object": "Coumadin"
  },
  {
    "subject": "Metformin",
    "predicate": "ClassifiesAs",
    "object": "Biguanide"
  },
  {
    "subject": "Lisinopril",
    "predicate": "ClassifiesAs",
    "object": "ACE Inhibitor"
  },
  {
    "subject": "Metformin",
    "predicate": "ContraindicatedWith",
    "object": "Chronic kidney disease stage 3"
  }
]
```


Figure D.5: RxNorm Triples (synthetic_rxnorm_triples.json)

```
def process_fhir_resource(resource):
    """Transform FHIR resource to knowledge graph components."""
    entities = extract_entities(resource)
    relationships = map_relationships(entities)
    validate_privacy(entities, relationships)
    return create_subgraph(entities, relationships)
```

Figure D.6: FHIR Resource Processing Example

D.5 HITL: Medical Validation Framework

D.5.1 Auditability and Validator Workflow

To ensure transparency and traceability, the system implements a structured audit and validation logging framework:

- **Validator Logs:** Each LLM2KG suggestion (e.g., extracted condition or medication) is logged alongside the original source (note text or FHIR resource), the model output, the validator decision (accept/revise/reject), and timestamp.
- **Provenance Metadata:** Each approved knowledge assertion is annotated with its origin (e.g.,

Query	Does the patient have any contraindicated drug combinations?
Gold Answer	Yes, the combination of lisinopril and NSAIDs is flagged due to renal risk.
LLM-only Output	No contraindications are identified based on the medication list.
RAG Output	A few combinations may be risky, including beta blockers and NSAIDs.
KG-injected Output	The patient is taking lisinopril and ibuprofen, which is a known contraindicated pair.
Reflexive Output	Yes, the combination of lisinopril and NSAIDs is contraindicated due to risk of kidney impairment.
F1 Score	LLM = 0.00; RAG = 0.21; KG = 0.79; Reflexive = 0.92
Contradiction	LLM = Yes; RAG = Yes; KG = No; Reflexive = No
Validator Escalated?	LLM = Yes; RAG = Yes; KG = Possibly; Reflexive = No

Table D.3: Evaluation of a contraindication detection query across four configurations.

note ID, EHR timestamp), source type, and validator ID.

- **Audit Trail Export:** Logs can be exported in structured formats (e.g., JSON, RDF) for downstream review, especially in clinical audit contexts.
- **Rollback and Diff Tools:** Validators can view historical graph states and selectively rollback changes using diff tools that highlight semantic drift or correction patterns.

Together, these mechanisms support the system's ability to remain trustworthy and compliant in high-stakes clinical environments.

D.6 KG2LLM: Clinical Query Answering

D.6.1 Prompt Templates

The system employs two distinct prompt templates for clinical query answering, depending on whether knowledge graph context is available:

These templates ensure that responses are appropriately constrained based on available knowledge, with the context-aware template enforcing stricter adherence to provided information.

D.6.2 Prompt Construction for Clinical Reasoning

The KG2LLM component constructs prompts that combine patient-specific data with ontological knowledge, enabling grounded clinical reasoning. Using the actual data from our case study:

D.6.3 Prompt Fusion for KG2LLM

When combining FHIR patient data with ontology knowledge, the system constructs structured prompts that expose both data sources:

Model Output: Yes, Metformin is contraindicated with Chronic kidney disease stage 3, and Subject_100002 has both of these.

```
# Template for knowledge-grounded responses
WITH_CONTEXT_TEMPLATE = """You are a clinical assistant with access to
structured patient information.

{context_block}

Clinical question:
{query}

Use the above information to answer precisely and safely.
Do not assume facts that are not explicitly stated."""

# Template for baseline responses (no KG context)
NO_CONTEXT_TEMPLATE = """You are a clinical assistant.

Clinical question:
{query}

Answer using your own general knowledge.
Do not reference external documents or links."""
```

Figure D.7: Prompt Templates for KG2LLM

```
Subject ID: 100002
Age: 64
Gender: F
Conditions: Type 2 diabetes, Chronic kidney disease stage 3
Medications: Metformin (started 2023-06-10), Lisinopril (started 2023-06-10)
Clinical Note: Patient followed for diabetes and CKD.
               Metformin continued. BP well-controlled on Lisinopril.

Summarize the patient's medication regimen and note any concerns
regarding kidney function.
```

Figure D.8: KG2LLM Prompt Template for Patient Summary

D.6.4 Implementation Example

The following code demonstrates how the medical case study implementation integrates the Reflexive Composition framework with FHIR data and RxNorm knowledge:

This implementation showcases:

- Configuration of both KB-LLM and Target LLM roles (using the same model in this case)
- Schema definition for medical knowledge representation
- Knowledge extraction and validation workflow
- Ontology enrichment via RxNorm triples
- Context-aware prompt selection based on grounding flag
- Response generation with appropriate template selection

```
Patient: Subject_100002
Conditions: Type 2 diabetes, Chronic kidney disease stage 3
Medications: Metformin, Lisinopril

Additional knowledge:
- Metformin ClassifiesAs Biguanide
- Lisinopril ClassifiesAs ACE Inhibitor
- Metformin ContraindicatedWith Chronic kidney disease stage 3

Clinical question:
Is any of the patient's medication contraindicated for their condition?
```

Figure D.9: KG2LLM Prompt Template: FHIR + RxNorm

D.6.5 Privacy-Preserving Architecture

Notably, privacy is not enforced through complex encryption or external data governance layers, but rather through scoped inference: each patient's knowledge graph is ephemeral from the LLM's point of view, injected only during context construction and never stored or exposed beyond the query lifecycle.

The patient knowledge graph is not retained across sessions. All structured data is passed ephemerally as part of the context during generation, ensuring that the LLM remains stateless and privacy-preserving.

D.7 Evaluation Configuration

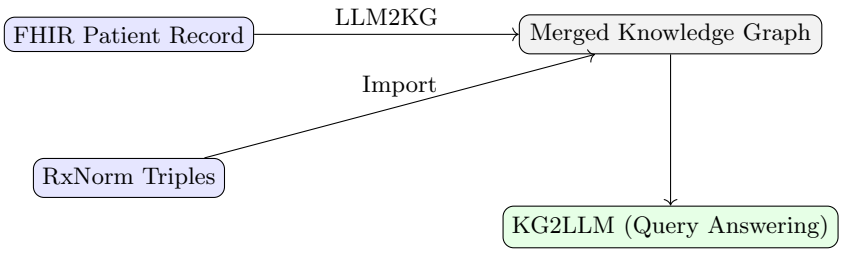
D.7.1 Clinical Query Benchmark

The medical case study was evaluated using a benchmark of clinical queries that assess the system's ability to:

- Identify patient conditions
- Recognize medication classes
- Detect contraindications
- Perform ontology-based inference

D.7.2 System Architecture Overview

The components interact through a hierarchical architecture that combines FHIR data with ontology knowledge:



This end-to-end implementation demonstrates the Reflexive Composition framework's ability to enhance LLM reasoning with structured medical knowledge while maintaining privacy, traceability, and clinical accuracy.

```
import os
import json
from typing import Optional
from reflexive_composition.core import ReflexiveComposition
from reflexive_composition.utils.llm_utils import log_result
from prompt_templates import WITH_CONTEXT_TEMPLATE, NO_CONTEXT_TEMPLATE


def run_single_query(query: Optional[str] = None,
                     grounded: bool = True,
                     template: Optional[str] = None):
    print("\n=== Medical QA Case Study: FHIR + RxNorm Grounding ===\n")

    # --- Configuration ---
    kb_llm_config = {
        "model_name": os.environ.get("KB_LLM_MODEL", "gemini-2.0-flash"),
        "api_key": os.environ.get("GEMINI_API_KEY"),
        "model_provider": "google",
    }

    target_llm_config = {
        "model_name": os.environ.get("KB_LLM_MODEL", "gemini-2.0-flash"),
        "api_key": os.environ.get("GEMINI_API_KEY"),
        "model_provider": "google",
    }

    kg_config = {
        "storage_type": "in_memory",
        "schema": {
            "entity_types": ["Patient", "Medication", "Condition",
                             "Allergy", "Code"],
            "relationship_types": ["HasCondition", "Prescribed",
                                   "ContraindicatedWith", "MappedTo"],
            "version": 1
        }
    }

    hitl_config = {
        "high_confidence_threshold": 0.85,
        "low_confidence_threshold": 0.6
    }

    # --- Initialize Reflexive Composition framework ---
    rc = ReflexiveComposition(
        kb_llm_config=kb_llm_config,
        target_llm_config=target_llm_config,
        kg_config=kg_config,
        hitl_config=hitl_config
    )

    # --- Source Text ---
    source_path = os.path.join(os.path.dirname(__file__),
                               "data", "fhir_patient.json")
    with open(source_path, "r") as f:
        source_text = f.read()
```



```
    # --- Extract and validate knowledge ---
    extraction_result = rc.extract_knowledge(source_text)
    rc.knowledge_graph.add_triples(extraction_result["triples"])

    # --- Add RxNorm enrichment ---
    rx_path = os.path.join(os.path.dirname(__file__),
```

```
{"id": "q1", "query": "What conditions does this patient have?"}
{"id": "q2", "query": "Is the patient currently on a blood pressure medication?"}
{"id": "q3", "query": "Is the patient taking a blood thinner?"}
{"id": "q4", "query": "What class of drug is Metformin?"}
{"id": "q5", "query": "Is there any risk related to kidney function in this patient?"}
{"id": "q6", "query": "Is any of the patient's medication contraindicated for their condition?"}
```

Figure D.11: Medical Query Set (queries.jsonl)