

# A Deployment-Aware Multi-Objective Hyper-Parameter Optimization Framework for Machine Learning

1<sup>st</sup> Matteo Zanutto  
University of Trento\*  
Trento, Italy  
matteo.zanutto@unitn.it

2<sup>nd</sup> Romeo Kienzler  
IBM Research  
Zurich, Switzerland  
romeo.kienzler1@ibm.com

3<sup>rd</sup> Gabriele Padovani  
University of Trento  
Trento, Italy  
gabriele.padovani@unitn.it

4<sup>th</sup> Sandro Fiore  
University of Trento  
Trento, Italy  
sandro.fiore@unitn.it

**Abstract**—State-of-the-art ML models deliver unprecedented performance across computer vision, natural language processing, and multi-modal tasks. These improvements have been driven by significantly growing computational costs, memory footprint, and consequently, energy consumption. Several model compression techniques and hyperparameter optimization strategies have been proposed to address these increases. Most of these approaches, however, target the training phase and often overlook the deployment context and the model usage. We propose a general-purpose framework for ML model optimization, focused on deployment and inference, to assist practitioners by quantifying trade-offs between model accuracy, power consumption, and other deployment-relevant metrics such as latency and memory usage. The proposed system is validated in three use cases: deploying a computer vision model on a resource-constrained edge device, optimizing LLM serving on data center-grade hardware, and running a large-scale benchmark on the Frontier HPC system. The system successfully identifies optimal configurations, demonstrating its versatility and applicability to practical scenarios.

**Index Terms**—Machine Learning, Hyperparameter optimization, Pareto optimization, Energy efficiency

## I. INTRODUCTION

Recent advances in Artificial Intelligence (AI) have been achieved through the development of larger and larger Machine Learning (ML) models, with transformers and LLMs emerging as one of the most impactful breakthroughs of our generation [26], [33]. Current state-of-the-art architectures achieve unprecedented performance across a variety of tasks, including computer vision, natural language processing, and their multi-modal variations. These improvements, however, have come at the cost of significantly increased computational complexity [11], as models now feature from millions to billions parameters, and require more compute and memory capability during both training and inference [9].

A direct consequence of this growth is the sharp rise in energy requirements [21], which is compounded in resource-constrained deployments, such as edge and embedded devices. In these cases, conditions such as processing speed, energy and temperature are subject to tight constraints and force tradeoffs

often impractical for current state of art models [20]. Moreover, as large-scale AI deployments in data centers are rapidly increasing energy demands and raising concerns regarding sustainability and operational costs [17], research goals have shifted from pure performance to finding the optimal trade-offs between resources and the best achievable performance [12]. Consequently, the development of AI systems that are jointly accurate, computationally efficient, and energy-aware has recently emerged as a pressing need.

Addressing these challenges, a wide range of model optimization techniques have been proposed from both research and industry communities. Compression methods, such as model pruning [23], knowledge distillation [16] and quantization [15] aim at reducing the computational footprint of models while attempting to preserve their accuracy [10]. Execution runtimes and accelerators optimize deployment and inference, often targeting specific hardware configurations or vendors [14], [29]. In parallel, Hyperparameter Optimization (HPO) [39] techniques automate the search for optimal architectures and training configurations, often leveraging Bayesian Optimization (BO) as the underlying search approach [19], [37].

However, despite the extensive research, most model development efforts and optimization research mainly target training phases, even though inference costs typically dominate the footprint of a model’s life-cycle once deployed at scale [36]. Moreover, although a variety of model optimization techniques and efficient serving tools exist, the type of supported optimization functionality, compatibility, and resulting trade-offs often differ substantially. As such, MLOps [30] practitioners are required to manually select, integrate, and fine-tune different techniques into coherent deployment pipelines, increasing the complexity of AI applications and demanding specialized expertise that spans both ML and hardware characteristics [31].

To address these limitations, this study proposes a general-purpose framework for AI model optimization, with a specific focus on the model deployment phase. More specifically, the proposed system integrates post-training optimizations and inference runtime configurations as part of the HPO process. This end-to-end approach explores the trade-offs

\*Work done while at IBM Research.

between model accuracy, power consumption, latency, as well as other deployment-relevant metrics such as memory usage. This enables practitioners to evaluate and select optimization techniques for a given model and target hardware, streamlining the otherwise manual and ad hoc pipeline tuning process while quantifying the impact of each choice on both performance and sustainability. The proposed system is evaluated in three diverse scenarios: (i) the optimization of a computer vision model in a resource-constrained edge-computing deployment, (ii) the optimization of an LLM served in a data center environment, and (iii) a large-scale benchmark experiment on the Frontier HPC system [1].

The remainder of this paper is organized as follows. Section II reviews related work on model compression, hardware-aware optimization, and multi-objective search. Section III details the design and implementation of the proposed system. Section IV presents our experimental setup and trade-off results. Finally, Section V concludes the paper, presenting the limitations of our study and discussing directions for future work.

## II. RELATED WORK

Extensive research is still being conducted in the area of ML model optimization, as the advancements in performance of recent models are paid for with exponential increases in processing complexity and memory requirements, which consequently raise the electricity costs [11]. To ease the computational footprint of ML models, various information compression techniques have been proposed. Pruning methods remove redundant components of a model, such as weights [23] or training samples [41], and can be performed in either an unstructured (increasing the sparsity of the model by removing individual weights) or in a structured fashion (removing entire architectural layers) [8]. In addition, a common practice nowadays is to release pruned variants of models with varying numbers of parameters on community repositories, such as HuggingFace<sup>1</sup>, to enable various deployment use cases [25]. Quantization lowers the numerical precision of the weights and activations to reduce the memory footprint and to accelerate the arithmetic operations [15]. It can be integrated during the training phase, a process known as Quantization-Aware Training (QAT) [18], or applied only at deployment time, as Post-Training Quantization (PTQ) [13]. Knowledge distillation techniques transfer knowledge from a large teacher to a smaller student model, by training the student to replicate the outputs of the teacher [16].

Another set of optimization techniques and tools has been developed to directly target the deployment phase, by operating at a downstream stage on trained models through compilers and runtimes that optimize inference execution over hardware-specific backends. Solutions such as PyTorch compile [4] and JAX<sup>2</sup> perform model optimization applying Just-In-Time (JIT) compilation of a model’s source code tracing its runtime

execution graph into optimized kernels that minimize Python runtime overheads. Examples of backends for exported models instead include NVIDIA TensorRT<sup>3</sup> for GPU-based deployments, Intel OpenVINO<sup>4</sup> for CPU-based systems, and ONNX<sup>5</sup> as a standardized model export representation for interoperable distribution and execution<sup>6</sup>. Optimization techniques targeting bottlenecks of specific models architectures have also been developed, with examples including vLLM<sup>7</sup> or llama.cpp<sup>8</sup> for LLMs deployments, and DistriFusion [22] for diffusion models. The usage of these systems becomes particularly required in resource-constrained environments, such as edge devices, where models trained in data center infrastructures cannot be deployed without further optimizations to size and computational efficiency. Studies have also provided a comparison of various runtimes, showing that no system dominates across all metrics simultaneously (e.g., accuracy, latency, throughput, memory, and power), and that the best choice is both model- and hardware-dependent [29]. Additionally, evaluation studies of neural network graph compilers demonstrated the effectiveness of their optimizations by benchmarking various configurations across architectures and hardware configurations [14]. Hyper-Parameter Optimization (HPO) automates the search for optimal training and model configurations, and is a standard phase of ML models life-cycle [39], [42]. Bayesian Optimization (BO) is commonly used for HPO as a sample-efficient technique to explore the search space by modeling the complex black-box function to optimize as a probabilistic surrogate, balancing accuracy and computation costs [19]. Practical implementations of HPO are largely enabled through frameworks such as Optuna [2] and BoTorch [5]. A limited but growing number of studies investigate HPO and trade-off evaluations for sustainable ML development, advocating for considering electricity consumption as a first-class metric alongside accuracy for deep learning model design [6], [40]. In recent years, additionally, some production-ready HPO frameworks for identifying Pareto-optimal configurations in terms of accuracy and electricity consumption have also been released, such as ECOpt [12] and e<sup>2</sup>HPO [7].

Despite the benefits in terms of accuracy and computational efficiency of these contributions, the majority of existing research focuses primarily on model development and parameter tuning during the training phase, without considering inference-time optimization [6], [7], [12], [40]. Consequently, deployment-specific metrics, such as inference-time power consumption, latency, and memory consumption, are often treated as a secondary consideration. Furthermore, although various optimization techniques and deployment frameworks are available, they differ substantially in the supported optimization, hardware compatibility and performance characteristics [14], [29]. Practitioners are then required to man-

<sup>3</sup><https://developer.nvidia.com/tensorrt>

<sup>4</sup><https://github.com/openvinotoolkit/openvino>

<sup>5</sup><https://onnx.ai/>

<sup>6</sup><https://onnxruntime.ai/>

<sup>7</sup><https://vllm.ai/>

<sup>8</sup>[urlhttps://github.com/ggml-org/llama.cpp](https://github.com/ggml-org/llama.cpp)

<sup>1</sup><https://huggingface.co/>

<sup>2</sup>[jax.dev](https://jax.dev)

ually coordinate complex optimization workflows for model deployment, hindering the systematic identification of optimal trade-offs in model accuracy, inference latency, and energy consumption [31].

### III. METHODOLOGY

The proposed framework formalizes an HPO pipeline as a multi-objective search over an arbitrary set of parameters and output metrics. The system optimization includes the end-to-end MLOps pipeline, starting from the training phase, selecting parameters to maximize accuracy, to the application of model optimization techniques, such as pruning and quantization, and also the configuration of the final downstream runtime, such as PyTorch compile or TensorRT. The user is enabled to choose the metrics to monitor, with power consumption being considered as a first-class metric alongside accuracy. This approach enables the system to capture the interactions between choices at different stages of the pipeline for the user to explore the resulting trade-offs across the objectives. The HPO search itself is conducted through an iterative and parallelized BO process, as the evaluation pipeline can include complex training, optimization, and deployment profiling workflows, thus requiring efficient exploration. This allows the framework to converge to optimal solutions with fewer trials than exhaustive or random search strategies and, being a multi-objective formulation, a Pareto front of optimal trade-off solutions is discovered. Within the proposed system, the multi-objective BO process is implemented using Optuna [2] with the Tree-structured Parzen Estimator [34] as the underlying algorithm, with Expected Improvement as acquisition function [38]. This method models the likelihood  $P(x|y)$  of hyperparameter configurations  $x$  yielding a score  $y$  and scales more efficiently compared to standard Gaussian Processes [27].

The Optuna search process is orchestrated through Iterate<sup>9</sup>, an open-source tool for parallel optimization and benchmarking. It optimizes code with respect to arbitrary metrics through (i) a configuration file in YAML format defining the variables to optimize, and (ii) an executable containing the model optimization and evaluation pipeline.

Based on these components, the evaluation pipeline spans the entire life-cycle of the model, from training and optimization to inference profiling. As such, each HPO trial corresponds to the execution of the pipeline under a specific configuration sampled by the Bayesian search controller. This process consists of the following core steps, illustrated also in Figure 1:

- 1) Instantiating the AI model through HuggingFace model ID, raw Python source code, or from serialized model artifacts such as ONNX files. A range of options are offered to work with any stage of the model life-cycle (e.g., importing pre-trained models), as well as to handle both publicly available models and custom architectures.

- 2) Optimizing the parameters according to the configuration file. This step encompasses any approach, from classical training-time hyper-parameter tuning to configuring compression techniques such as pruning and quantization.
- 3) Deployment to a target runtime and hardware configuration. Treating this phase as part of the search space allows the framework to explore different runtimes (such as OpenVINO or ONNX) and inference time settings (such as different PyTorch compile modes).
- 4) Profiling performance at inference time with respect to the metrics defined in the configuration file. Arbitrary evaluators can be plugged in to observe joined behaviors such as prediction accuracy, power consumption, and execution latency.
- 5) Once the previous steps are completed, the metrics observed during the profiling phase are fed back to Optuna to update the Bayesian search estimator.

This framework is designed to be modular and extensible to support different model architectures, optimization techniques, deployment targets, and metric evaluators. This choice enables the system to adapt to rapidly emerging technologies and varying use cases while maintaining the core infrastructure.

The iterative search process is repeated until convergence or until a maximum number of search trials are performed. Through Iterate, this search can also be parallelized and scaled to multi-process clusters with Ray<sup>10</sup>, distributing trials across jobs on multiple nodes. Currently, Slurm and LSF clusters are supported by the system, with Kubernetes and PBS extensions planned for the future.

Upon completion of the search process, the framework returns to the user the Pareto front of optimal configurations that were discovered, informing MLOps practitioners about model optimizations and deployment strategy by quantifying the trade-offs across multiple dimensions. The complete HPO experiment history is also made available to the user for further analysis, recording which parameters and outcome each experiment featured.

### IV. EVALUATION

The system was validated by addressing model optimization in three distinct use cases: a resource-constrained deployment of a computer vision ML model on an edge device, the serving of an LLM on data center GPUs, and a large-scale benchmark experiment in an HPC environment. The diversity of these applications aims at underscoring the versatility of the framework and its applicability to scenarios of practical importance.

#### A. Edge computing optimization

The first evaluation scenario involved the optimization of a YOLOv8 [32] model for object detection, deployed on an NVIDIA Jetson Nano device. The evaluation was conducted considering increasingly pruned versions of the YOLOv8

<sup>9</sup><https://github.com/claimed-framework/claimed>

<sup>10</sup><https://github.com/ray-project/ray>

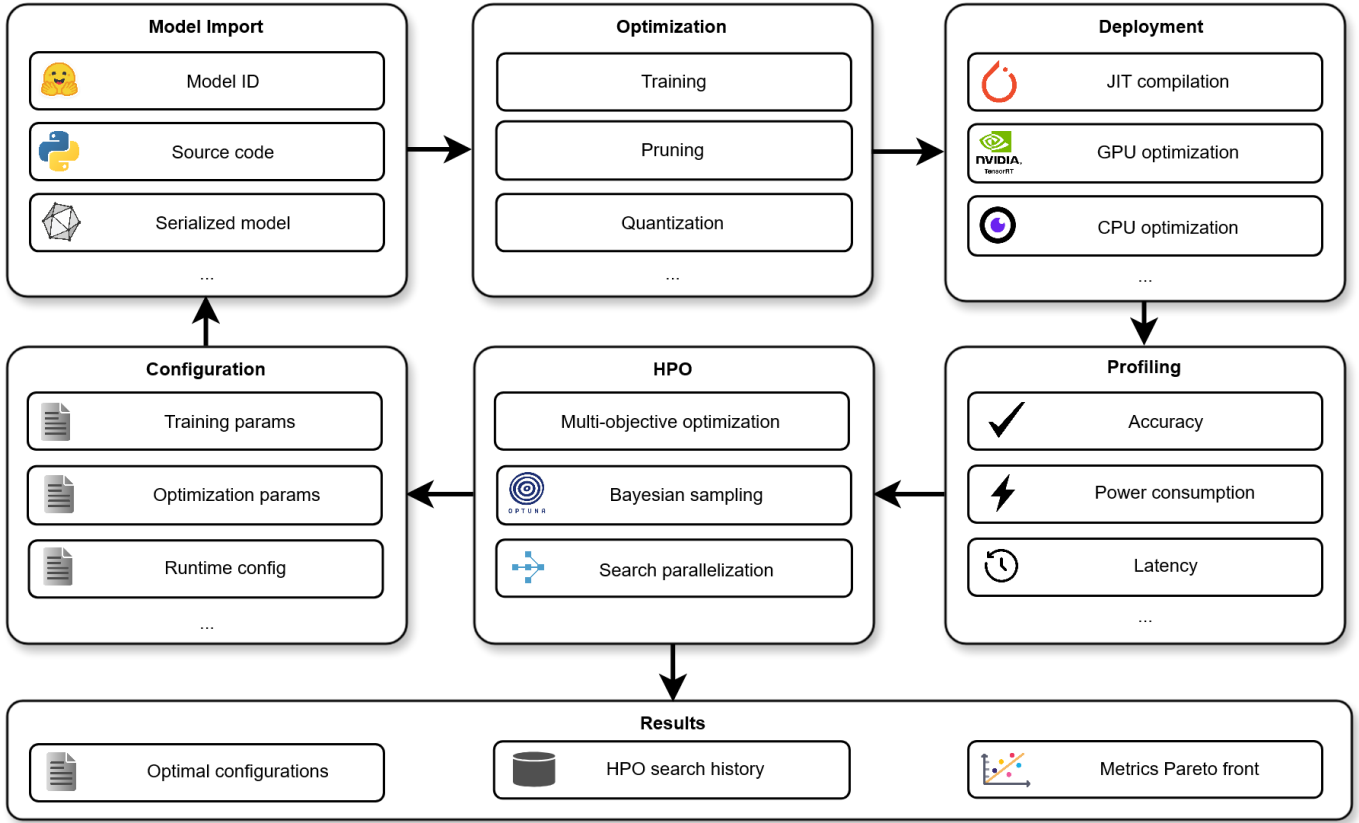


Fig. 1. Overview of the proposed methodology. For each phase, some implementative examples are reported, such as the usage of the model ID or its source code for import, the deployment as JIT compiled code or through a GPU optimizer, and the profiling of accuracy, power consumption, and latency, among other possible options.

model<sup>11</sup> (medium at 25.9M params, small at 11.2M params, and nano at 3.2M params), each one was trained for 100 epochs on the coco128 [24] dataset. The resulting ONNX-exported models were then compiled on-device using TensorRT (version 802, with CUDA 10.2) to perform PTQ at fp32, fp16, and int8 precision, exploring different network optimization tactics (cudnn, cublas, and cublas\_l1t, with ranging values of available workspace memory). Automated multi-objective HPO was performed leveraging the underlying Iterate layer to optimize across throughput (Queries-Per-Second), latency (ms), accuracy (mAP@0.5), and energy per inference request (joule, measured through the tegrastats utility<sup>12</sup>) as objectives. Table I recaps the metrics used in this use case, along with the direction of optimization, while Table II summarizes the hyper-parameters search space of this evaluation.

Across 40 search trials, 14 optimal model configurations were identified on the Pareto fronts on the observed metrics. Notably, among these optimal configurations, a reduction in energy consumption of 89% can be achieved within a 23% loss of accuracy. Moreover, an increase in accuracy of nearly 18% can be achieved at a comparable latency, but with a staggering

TABLE I  
OBJECTIVES OF THE HYPERPARAMETER OPTIMIZATION OF THE EDGE DEPLOYMENT USE CASE, AND DIRECTION OF OPTIMIZATION IN THE BAYESIAN PROCESS.

| Metric           | Unit               | Direction |
|------------------|--------------------|-----------|
| Throughput       | Queries-Per-Second | Maximize  |
| Latency          | ms                 | Minimize  |
| Accuracy         | mAP@0.5            | Maximize  |
| Energy/inference | Joule              | Minimize  |

TABLE II  
HYPERPARAMETER SEARCH SPACE OF THE EDGE DEPLOYMENT USE CASE.

| Parameter        | Search Space                   |
|------------------|--------------------------------|
| Number of params | 25.9M, 11.2M, 3.2M             |
| Quantization     | fp32, fp16, int8               |
| Batch size       | 1, 4, 8                        |
| Tactic           | cublas, cudnn, cublas_l1t, all |
| Workspace (MB)   | 256, 512, 1024, 2048           |

eight times increase in energy.

As expected, configurations that minimize computation time with a higher throughput and lower latency result in lower energy consumption per inference request, at the cost of model

<sup>11</sup><https://yolov8.com>

<sup>12</sup><https://docs.nvidia.com/jetson/archives/r36.5/DeveloperGuide/AT/JetsonLinuxDevelopmentTools/TegrastatsUtility.html>

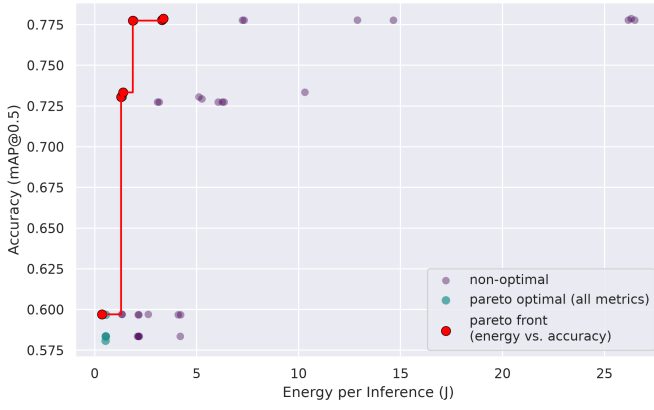


Fig. 2. Energy consumption per inference request vs. Accuracy trade-off in the edge deployment evaluation.

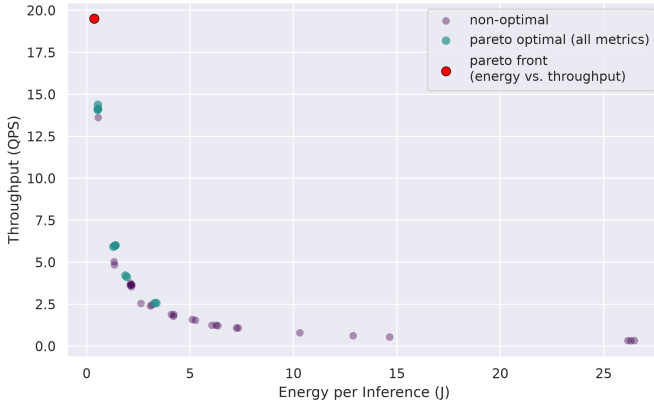


Fig. 3. Energy consumption per inference request vs. Throughput trade-off in the edge deployment evaluation.

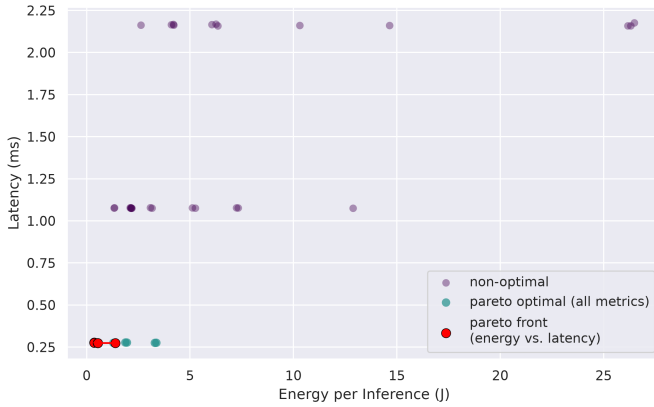


Fig. 4. Energy consumption per inference request vs. Throughput trade-off in the edge deployment evaluation.

accuracy, as can be seen in Figures 2, 3, and 4. Evaluation results indicate that model pruning (number of parameters) and batch size are the most influential aspects across all observed metrics, with quantization influencing the optimization much less than initially expected – as int8 precision not natively

supported on the tested Jetson platform and therefore incurring in higher latency due to casting operations by TensorRT. The evaluation of this quantization mode was still included as a worst case for testing the Bayesian search efficacy. Finally, our observations highlight the need to benchmark different optimization tactics and varying workspace memory, instead of relying on "all-on" full memory usage approaches.

The resulting parameter importance is shown in Figure 5.



Fig. 5. Parameter importance measured in the edge deployment evaluation

### B. LLM serving optimization

The second evaluation scenario involved the optimization of a general-purpose conversational agent deployment on a cloud data center equipped with Nvidia A100 SXM 80GB GPUs. Leveraging the model import interface of the proposed framework, two LLM models were evaluated: the smaller SmoLM2 model [3] was briefly used for the first development phase, while the bigger Qwen2.5 model [28] was then used for the full evaluation at scale. Similar results were observed for the two models, but the full evaluation is reported only for the Qwen2.5 model for brevity. The evaluation process focused on the inference stage, using the pretrained Qwen2.5 model imported from HuggingFace at 500M, 1.5B, and 3B parameters, with 512, 1024, 2048 and 4096 tokens of context length. Quantization was also applied at fp32, fp16, and int8 precision levels. For inference execution, different inference runtimes were tested, namely vLLM<sup>13</sup>, ONNX<sup>14</sup>, PyTorch compile<sup>15</sup> in default, max-autotune, and reduce-overhead modes, and native Python code execution as a baseline. The model deployment was optimized across these configuration in terms of latency (ms), accuracy (computed by selecting, for each example, the ending with the lowest perplexity and matching to the ground truth using the HellaSwag dataset [43]), energy per inference request (in Joule, measured through the Python bindings for the NVIDIA Management Library (NVML)<sup>16</sup>), and VRAM usage (MBs). The multi-objective HPO was again automated

<sup>13</sup><https://vllm.ai/>

<sup>14</sup><https://huggingface.co/docs/optimum-onnx>

<sup>15</sup><https://docs.pytorch.org/docs/2.13/generated/torch.compile.html>

<sup>16</sup><https://developer.nvidia.com/management-library-nvml>

through the Iterate layer for efficient Bayesian search space exploration. Tables III and IV recap the objectives and the hyper-parameters search space of this evaluation. Figures 6, 7, 8 and 9 recap the experiments results, showing the trade-offs achieved across the configurations.

TABLE III

OBJECTIVES OF THE HYPERPARAMETER OPTIMIZATION OF THE LLM SERVING USE CASE, AND DIRECTION OF OPTIMIZATION IN THE BAYESIAN PROCESS.

| Metric           | Unit  | Direction |
|------------------|-------|-----------|
| Latency          | ms    | Minimize  |
| Accuracy         | -     | Maximize  |
| Energy/inference | Joule | Minimize  |
| VRAM usage       | MB    | Minimize  |

TABLE IV

HYPERPARAMETER OPTIMIZATION SEARCH SPACE OF LLM SERVING SCENARIO.

| Parameter               | Search Space                                                                                         |
|-------------------------|------------------------------------------------------------------------------------------------------|
| Number of params        | 0.5B, 1.5B, 3B                                                                                       |
| Precision               | fp32, fp16, int8                                                                                     |
| Context-length (tokens) | 512, 1024, 2048, 4096                                                                                |
| Runtime                 | vLLM, ONNX, Torch Compile default, Torch compile max-autotune, Torch compile reduce-overhead, Python |

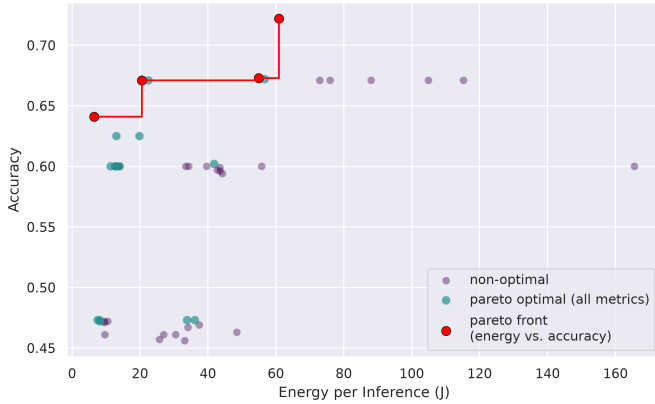


Fig. 6. Energy consumption per inference request vs. Accuracy trade-off in the LLM serving use case.

Across 50 search trials, 19 optimal configurations were observed on the Pareto front of the measured metrics. Between these options, the most energy-efficient deployment achieves nearly 90% energy savings at the cost of an accuracy loss of 11%. The most accurate configuration instead increases accuracy by 8%, but at the important costs of 2 times more latency and 8 times more energy.

From the evaluation results, the choice of model size is the only critical factor for its accuracy, with the biggest Qwen model (at 3B params) as a clear winner, suffering no loss

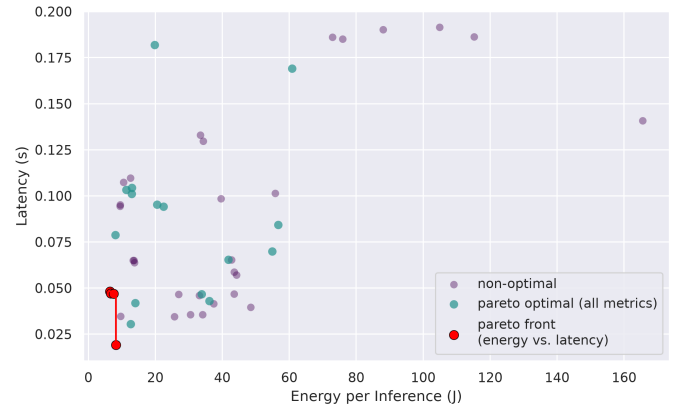


Fig. 7. Energy consumption per inference request vs. Latency trade-off in the LLM serving use case.

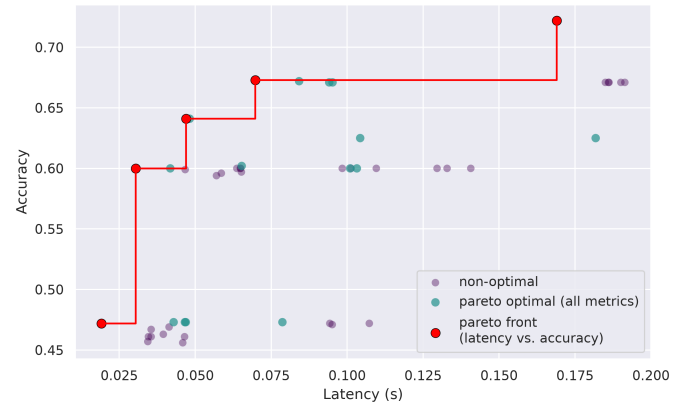


Fig. 8. Latency vs. Accuracy trade-off in the LLM serving use case.

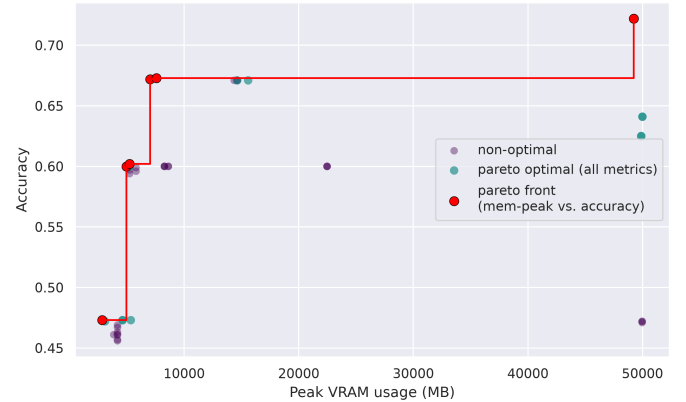


Fig. 9. VRAM usage vs. Accuracy trade-off in the LLM serving use case.

in accuracy due to quantization or runtime-specific optimization. Numerical precision instead affects the computational complexity and memory footprint of the model, affecting its latency. All these factors combine for the energy costs, where all hyperparameters contribute to its electricity consumption. The choice of the runtime emerges as the determining factor for minimizing its memory requirements, with Python and

PyTorch runtimes as clear winners in this regard. As can be seen in Figure 9, configurations with a higher memory usage yield more accurate model responses. Similar to the edge use case, configurations that minimize compute time, lowering the latency, also contribute to lowering the electricity consumption of the model at the cost of accuracy, as can be seen in Figures 7, 6 and 8. Finally, the choice of the context length has little to no effect across all the metrics, probably due to the HellaSwag dataset comprising short phrases. Figure 10 illustrates the captured hyper-parameters’ importance across the observed metrics.



Fig. 10. Parameter importance measured in the LLM serving use case

### C. HPC benchmark experiment

The third evaluation scenario involved a large-scale experiment in an HPC environment. Frontier [1], a Slurm-based cluster with nodes equipped with AMD Instinct MI250X GPUs, each having 64 GB of high-bandwidth memory, was used to benchmark a variable-size CNN-LSTM model [44] for pixel segmentation in the context of ClimateBench [35]. More specifically, the parallelization capabilities of the proposed system were used to run two additional benchmarks to understand, first, whether in large use cases, the additional training necessary for HPO could be outweighed by the inference benefit; and secondly, if given a common and constant energy budget, a user should prioritize HPO parallelization or training parallelization.

TABLE V  
HYPERPARAMETER OPTIMIZATION SEARCH SPACE OF HPC SCENARIO.

| Parameter     | Search Space     |
|---------------|------------------|
| Conv. Filters | [8 .. 64]        |
| LSTM Layers   | [8 .. 64]        |
| Learning Rate | [1e-4 .. 1e-2]   |
| Batch Size    | 8, 16, 32        |
| L2 Layers     | [0 .. 1e-2]      |
| Precision     | fp32, fp16, int8 |

When tracking the consumption of inference calls, the benefits of using this HPO approach can be quantified, with more

efficient models created through the optimization process. The additional training steps which are run to identify this final architecture, however, are a large cumulative overhead. Assuming a better model is found, there will therefore exist a break-even point, after which all the energy utilized during the training phase will be paid for by the lighter inferences.

Table VI contains results for this additional training experiment, in which the model structure was optimized, with the HPO process deciding how many layers, epochs, batch size, and precision to use in order to achieve the best trade-off between inference cost, training time, and accuracy. The HPO process minimizes the model loss, in addition to the training and inference energy consumption, both treated as separate dimensions in the objective function. In this use case, the optimal model was found with a usage of around 2000J of energy, but taking into account the lighter inference phase, the model repaid the extra training cost after 5000 to 7000 passthroughs.

On the other hand, to benchmark the best energy budget use, an hour allocation with 8 GPUs was created on the aforementioned cluster. In this setting, two runs were executed, with one HPO process running distributed training on 8 GPUs, and with HPO parallelized on 8 processes, respectively, with Iterate starting an HPO run as soon as the previous finishes. In particular, in the former setting, Iterate chooses a candidate hyperparameter configuration and orchestrates a single training process; while for the latter setting, Iterate samples eight different candidates and orchestrates one training for each, distributing them across each available GPU.

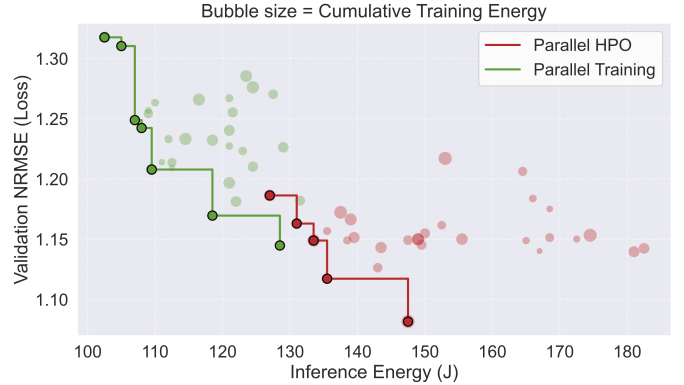


Fig. 11. Validation NRMSE and inference energy trade-off, the line shows each of the two experiments’ Pareto frontier, with bubble size representing cumulative training energy.

Results for these campaigns are shown in Figure 11, with additional statistics presented in Figure 12. When using the common budget, the HPO-parallelized process finds overall better-performing solutions, converging quickly to a lower NRMSE value. However, the parallelized training approach identifies a set of models that are more energy-efficient during the inference phase, while still using a comparable amount of energy during training. Interestingly, the training time for the 8 GPU run, shown in Figure 12, is often longer than the sequential version. This is believed to be caused by

TABLE VI  
ENERGY CONSUMPTION METRICS FOR TRAINING, INFERENCE, SAVINGS AND BREAK-EVEN ANALYSIS ACROSS RELEVANT HPO ITERATIONS.

| Iteration | Cumulate Training Energy (J) | Best Inference (J/sample) | Inference Savings (J/sample) | Best NRMSE | Break-Even Inferences |
|-----------|------------------------------|---------------------------|------------------------------|------------|-----------------------|
| 0         | 120.5                        | 1.0437                    | 0.0000                       | 1.1402     | 0                     |
| 1         | 227.0                        | 0.8187                    | 0.2250                       | 1.1402     | 474                   |
| 10        | 1239.5                       | 0.8187                    | 0.2250                       | 1.1402     | 4,974                 |
| 15        | 1790.5                       | 0.8187                    | 0.2250                       | 1.1172     | 7,423                 |
| 21        | 2465.5                       | 0.7937                    | 0.2500                       | 1.1172     | 9,380                 |
| 29        | 3348.5                       | 0.7937                    | 0.2500                       | 1.0816     | 12,912                |

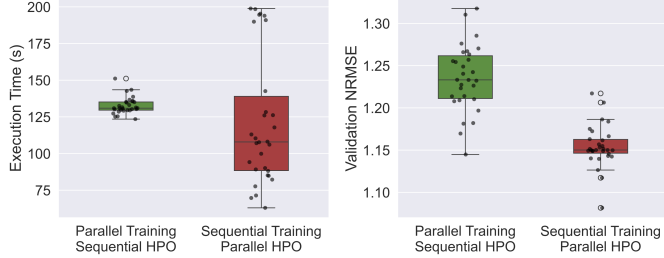


Fig. 12. Execution time and validation NRMSE distributions for parallel and sequential training/HPO configurations.

the relatively small model, especially when paired with the overhead introduced to set up the environment. On top of this finding, further HPO must still be conducted on the trained model to identify its optimal deployment configuration across inference runtimes and post-training optimizations, as demonstrated by the results of the use cases above.

## V. CONCLUSIONS

This work presented a general-purpose approach for ML model optimization with a particular focus on the deployment and inference phases. At its core, the framework consists of a parallelizable HPO controller that performs a sample-efficient, multi-objective Bayesian search, navigating the hyperparameter space to optimize arbitrary metrics, notably accuracy, power consumption, and other deployment-specific metrics such as execution latency. Through a configuration file, an end-to-end model optimization and deployment pipeline is instantiated, specifying steps such as the application of compression techniques, the configuration of deployment runtimes, and the performance profiling at inference time. The framework finally exposes the set of candidate optimal solutions on the Pareto front, quantifying the trade-offs between the observed metrics, to aid the ML practitioner in the practical usage and delivery of models.

The proposed framework was validated across three representative scenarios across heterogeneous deployment environments. The first considered the deployment of a computer vision model, YOLOv8, on the resource-constrained NVIDIA Jetson Nano edge platform. The second scenario involved the optimization of LLM serving on data center machines, specifically the Qwen2.5 model on nodes equipped with Nvidia A100

GPUs. In both cases, a Pareto front of optimal configurations was determined, with achievable energy per inference savings of up to 89% in the edge deployment and up to nearly 90% in the LLM serving, at the cost of 23% and 11% accuracy, respectively. A third large-scale experiment was conducted in the Frontier HPC system in the context of ClimateBench with a CNN-LSTM model, to quantify the break-even point where the inference benefits justify the HPO costs at 5000 to 7000 requests, and to determine the trade-offs between the parallelization of either the HPO trial or the training process for a given energy budget.

We acknowledge a set of limitations of this study. First of all, the iterative nature of the Bayesian search approach that was adopted incurs a computational cost that may seem to counter-intuitively result in further electricity consumption. However, this cost is incurred only once at optimization time, and the resulting optimal configuration can then be used repeatedly for the successive inference phase, which is typically the largest part of the ML operational life-cycle [36]. Additionally, we recognize the technical difficulty implied by the generic definition of the model import and optimization stages, as from a practical perspective it is inherently challenging to provide a unifying system that can cover all the possible technologies in the rapidly-changing ML ecosystem. Still, the plugin-based design approach enables the system to be adaptable and flexible, by the addition of support for new models and optimization techniques and the usage of standardized representations such as ONNX. Finally, we acknowledge the dependency of the proposed framework on third-party tools (such as tegrastats or NVML) for measuring electricity consumption, the accuracy of which is beyond the scope of this study. The proposed methodology remains applicable as the plugin-based design allows for adaptability to more accurate monitoring solutions if available.

Future work will focus on extending the implementation of the optimization pipeline integrating support at training time for further model compression and optimization techniques, such as Quantization-Aware Training (QAT) and Knowledge Distillation. Extensions to deployment-time optimizations are also planned, integrating support for additional inference runtimes such as llama.cpp and OpenVINO. In general, extensions to accommodate new models, optimizers, and hardware platforms profilers will be enabled by the plugin-based design

that we adopted, whose usability and ease of integration will be further improved.

## VI. ACKNOWLEDGMENT

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or Research Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. Grant Agreement n. 101086248 - CLOUDSTARS.

This work was partially funded by the European Union – Next Generation EU, Mission 4 Component 1 CUP E66E24000060008.

This work was partially funded under the National Recovery and Resilience Plan (NRRP), Mission 4 Component 2 Investment 1.4 - Call for tender No. 1031 of 17/06/2022 of Italian Ministry for University and Research funded by the European Union – NextGenerationEU (proj. nr. CN\_00000013), and the RI-SCALE project (Grant Agreement 101188168).

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

## REFERENCES

- [1] Frontier (hpe cray ex) exascale supercomputer at the oak ridge leadership computing facility.
- [2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.
- [3] Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, et al. Smollm2: When smol goes big—data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*, 2025.
- [4] Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, et al. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM international conference on architectural support for programming languages and operating systems, volume 2*, pages 929–947, 2024.
- [5] Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems* 33, 2020.
- [6] Antonio Candelieri, Andrea Ponti, and Francesco Archetti. Fair and green hyperparameter optimization via multi-objective and multiple information source bayesian optimization. *Machine Learning*, 113(5):2701–2731, 2024.
- [7] Antonio Candelieri and Elena Signori. e2hpo: Energy efficient hyperparameter optimization via energy-aware multiple. In *Learning and Intelligent Optimization: 19th International Conference, LION 19, Prague, Czech Republic, June 15–19, 2025, Proceedings, Part I*, page 135. Springer Nature, 2026.
- [8] Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):10558–10578, 2024.
- [9] Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling vision transformers to 22 billion parameters. In *International conference on machine learning*, pages 7480–7512. PMLR, 2023.
- [10] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020.
- [11] Radosvet Desislavov, Fernando Martínez-Plumed, and José Hernández-Orallo. Trends in ai inference energy consumption: Beyond the performance-vs-parameter laws of deep learning. *Sustainable Computing: Informatics and Systems*, 38:100857, 2023.
- [12] Emile Dos Santos Ferreira, Andrei Paleyes, and Neil D Lawrence. Optimising for energy efficiency and performance in machine learning. *arXiv preprint arXiv:2601.08991*, 2026.
- [13] Elias Frantar and Dan Alistarh. Optimal brain compression: A framework for accurate post-training quantization and pruning. *Advances in Neural Information Processing Systems*, 35:4475–4488, 2022.
- [14] Alireza Furutanpey, Carmen Walser, Philipp Raith, Pantelis A Frangoudis, and Schahram Dustdar. Leveraging neural graph compilers in machine learning research for edge-cloud systems. *IEEE Transactions on Parallel and Distributed Systems*, 2026.
- [15] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-power computer vision*, pages 291–326. Chapman and Hall/CRC, 2022.
- [16] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International journal of computer vision*, 129(6):1789–1819, 2021.
- [17] International Energy Agency. Energy and AI. Technical report, International Energy Agency, April 2025.
- [18] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.
- [19] Florian Karl, Tobias Pielok, Julia Moosbauer, Florian Pfisterer, Stefan Coors, Martin Binder, Lennart Schneider, Janek Thomas, Jakob Richter, Michel Lang, et al. Multi-objective hyperparameter optimization in machine learning—an overview. *ACM Transactions on Evolutionary Learning and Optimization*, 3(4):1–50, 2023.
- [20] Mohit Kumar, Xingzhou Zhang, Liangkai Liu, Yifan Wang, and Weisong Shi. Energy-efficient machine learning on the edges. In *2020 IEEE international parallel and distributed processing symposium Workshops (IPDPSW)*, pages 912–921. IEEE, 2020.
- [21] Loïc Lannelongue, Jason Grealey, and Michael Inouye. Green algorithms: quantifying the carbon footprint of computation. *Advanced science*, 8(12):2100707, 2021.
- [22] Muyang Li, Tianle Cai, Jiaxin Cao, Qingsheng Zhang, Han Cai, Junjie Bai, Yangqing Jia, Kai Li, and Song Han. Distifusion: Distributed parallel inference for high-resolution diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7183–7193, 2024.
- [23] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021.
- [24] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.
- [25] Saurav Muralidharan, Sharath Turuvekere Sreenivas, Raviraj Joshi, Marcin Chochowski, Mostafa Patwary, Mohammad Shoeybi, Bryan Catanzaro, Jan Kautz, and Pavlo Molchanov. Compact language models via pruning and knowledge distillation. *Advances in Neural Information Processing Systems*, 37:41076–41102, 2024.
- [26] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A comprehensive overview of large language models. *ACM Transactions on Intelligent Systems and Technology*, 16(5):1–72, 2025.
- [27] Yoshihiko Ozaki, Yuki Tanigaki, Shuhei Watanabe, and Masaki Onishi. Multiobjective tree-structured parzen estimator for computationally expensive optimization problems. In *Proceedings of the 2020 genetic and evolutionary computation conference*, pages 533–541, 2020.
- [28] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang

- Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025.
- [29] Ishrak Jahan Ratul, Yuxiao Zhou, and Kecheng Yang. Accelerating deep learning inference: A comparative analysis of modern acceleration frameworks. *Electronics*, 14(15):2977, 2025.
  - [30] Georgios Symeonidis, Evangelos Nerantzis, Apostolos Kazakis, and George A Papakostas. Mlops-definitions, tools and challenges. In *2022 IEEE 12th annual computing and communication workshop and conference (CCWC)*, pages 0453–0460. IEEE, 2022.
  - [31] Damian A Tamburri. Sustainable mlops: Trends and challenges. In *2020 22nd international symposium on symbolic and numeric algorithms for scientific computing (SYNASC)*, pages 17–23. IEEE, 2020.
  - [32] Rejin Varghese and M Sambath. Yolov8: A novel object detection algorithm with enhanced performance and robustness. In *2024 International conference on advances in data engineering and intelligent computing systems (ADICS)*, pages 1–6. IEEE, 2024.
  - [33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
  - [34] Shuhei Watanabe. Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance. *arXiv preprint arXiv:2304.11127*, 2023.
  - [35] Duncan Watson-Parris, Yuhan Rao, Dirk Olivie, Øyvind Seland, Peer Nowack, Gustau Camps-Valls, Philip Stier, Shahine Bouabid, Maura Dewey, Emilie Fons, et al. Climatebench v1. 0: A benchmark for data-driven climate projections. *Journal of Advances in Modeling Earth Systems*, 14(10):e2021MS002954, 2022.
  - [36] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga, Jinshi Huang, Charles Bai, et al. Sustainable ai: Environmental implications, challenges and opportunities. *Proceedings of machine learning and systems*, 4:795–813, 2022.
  - [37] Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, and Si-Hao Deng. Hyperparameter optimization for machine learning models based on bayesian optimization. *Journal of Electronic Science and Technology*, 17(1):26–40, 2019.
  - [38] Zhaoyi Xu, Yanjie Guo, and Joseph H Saleh. Efficient hybrid bayesian optimization algorithm with adaptive expected improvement acquisition function. *Engineering Optimization*, 53(10):1786–1804, 2021.
  - [39] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.
  - [40] Tim Yarally, Luis Cruz, Daniel Feitosa, June Sallou, and Arie Van Deursen. Uncovering energy-efficient practices in deep learning training: Preliminary steps towards green ai. In *2023 IEEE/ACM 2nd International Conference on AI Engineering–Software Engineering for AI (CAIN)*, pages 25–36. IEEE, 2023.
  - [41] Ruonan Yu, Songhua Liu, and Xinchao Wang. Dataset distillation: A comprehensive review. *IEEE transactions on pattern analysis and machine intelligence*, 46(1):150–170, 2023.
  - [42] Tong Yu and Hong Zhu. Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint arXiv:2003.05689*, 2020.
  - [43] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 4791–4800, 2019.
  - [44] Wenshu Zha, Yuping Liu, Yujin Wan, Ruilan Luo, Daolun Li, Shan Yang, and Yanmei Xu. Forecasting monthly gas field production based on the cnn-lstm model. *Energy*, 260:124889, 2022.