Towards Workload-aware Optimization and Reconfiguration of ML Serving Pipelines [Vision]

Matthias Boehm
Arnab Phani
TU Berlin & BIFOLD

Khuzaima Daudjee
Runsheng Benson Guo
University of Waterloo

Pamela Delgado
HES-SO

Thaleia Dimitra Doudali
IMDEA Software Institute

Ana Klimovic
Xiaozhe Yao
ETH Zurich

James Kwok
HKUST

Manisha Luthra
TU Darmstadt & DFKI

Tilmann Rabl
Ricardo Salazar-Díaz
HPI & University of Potsdam

Pınar Tözün
Ehsan Yousefzadeh-Asl-Miandoab
IT University of Copenhagen

ABSTRACT

Serving machine learning (ML) models at scale in cloud environments is a well-studied and important, yet very resource-intensive problem. Existing approaches like data batching, model pruning, and quantization, as well as specialized models, show great promise but are typically applied heuristically or in a static offline step before model deployment. Multiple trends motivate a *new paradigm* for serving ML models: personalization for different domains and user groups, new challenges regarding serving Large Language Models (LLMs), more practical sparsity exploitation, and new hardware features for reconfiguration and tuning. In order to provide better accuracy and efficiency trade-offs and inspired by adaptive query processing, we envision a dynamic, workload-aware reconfiguration of ML model serving configurations. We deploy multiple variants of sparsified and specialized models, profile samples of user serving requests for accuracy and resource consumption, and dynamically reconfigure serving configurations (variant selection, placement, and parameter tuning) in order to maximize throughput subject to accuracy, energy, and latency constraints. Our workload-aware reconfiguration provides a great opportunity for adapting the configurations to actual workload characteristics of complex (multi-model) deployed ML pipelines.

PVLDB Reference Format:
Matthias Boehm, Arnab Phani, Khuzaima Daudjee, Runsheng Benson Guo, Pamela Delgado, Thaleia Dimitra Doudali, Ana Klimovic, Xiaozhe Yao, James Kwok, Manisha Luthra, Tilmann Rabl, Ricardo Salazar-Díaz, Pınar Tözün, and Ehsan Yousefzadeh-Asl-Miandoab. Towards Workload-aware Optimization and Reconfiguration of ML Serving Pipelines [Vision]. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

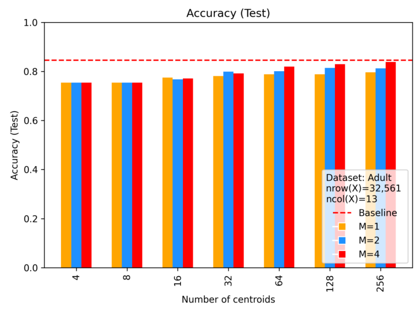
PVLDB Artifact Availability:
The source code and other artifacts have been made available at https://github.com/damslab/reproducibility/tree/master/tmp-mlserving-p1720.

1 INTRODUCTION

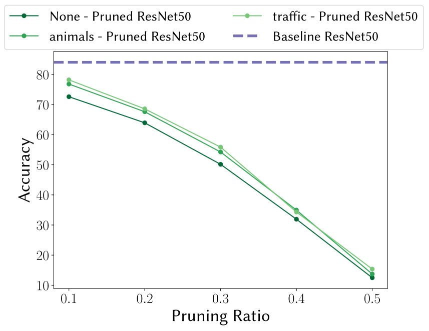
As machine learning (ML) models are more and more widely deployed in end-user applications, data analysis pipelines, and various optimization toolkits, ensuring both high-quality and resource-efficient serving of these models at scale becomes crucially important in practice. Besides embedded ML serving (with a focus on mobile/edge devices), the predominant deployment form—especially for modern Large Language Models (LLMs)—are ML serving services (with a focus on cloud environments).

Resource-intensive Serving: In the context of cloud-based ML serving services, we observe major trends which render these serving workloads increasingly resource-intensive, but also new capability that may mitigate these challenges. First, there is an increasing scale in terms of number of client requests and model sizes. Increasing context-lengths in conversational LLM interactions as well as emerging reasoning (chain-of-thought) models also cause large state per client. Second, we see increasing model personalization for different domains and user groups (through fine tuning), model specialization for different tasks (through custom models), as well as multi-modal and multi-component models. Third, mixture-of-expert models (with very few active experts per serving request; e.g., 8 out of 128 for DeepSeek-V3) [12], weight pruning, and other forms of sparsity exploitation as well as new hardware features for reconfiguration and tuning also become more practical.

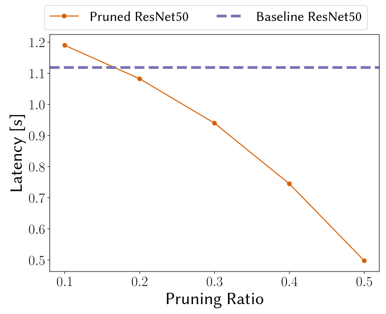
Limitations of Existing Work: Throughput optimization in ML serving services—such as TensorFlow Serving [35], TorchServe, Ray Serve, NVIDIA Triton, Clipper [11], Pretzel [29], Rafiki [52]), ORCA [60], Shepherd [62], Llumnix [46], and ServerlessLLM [16]—is a well-studied but challenging problem. Existing runtime techniques for exploiting different throughput-latency-accuracy trade-offs include data/request batching, model pruning and quantization, result caching, dedicated scheduling approaches, multi-model optimizations, and specialized models. Despite promising performance impact, these strategies are mostly applied in a heuristic manner,



(a) Quantization Accuracy of LogReg on Adult



(b) Sparsification Accuracy of ResNet50 on ImageNet



(c) Sparsification Latency of ResNet50 on ImageNet

Figure 1: Quantization and Sparsification of Traditional ML/DNN Models.

in isolation, and/or statically before deployment. Similarly, recent work on optimizing memory management and throughput for conversational LLMs (with large state) [31, 42] select model alternatives but focus on a single model type, no multi-modal/-component models, and assume that alternatives pre-exist upfront.

Vision and Contributions: Inspired by two decades of research on adaptive query processing [14, 15, 23], we make a case for adaptive, workload-aware optimization and reconfiguration of ML serving pipelines. To this end, we envision deploying multiple variants of sparsified and specialized models as well as policies of parallelization strategies, periodically profiling samples of real client requests (with several variants) for accuracy and resource consumption, and dynamically reconfiguring serving configurations to maximize throughput subject to accuracy, energy, and latency constraints. Our technical contributions of this vision paper are:

- *Potential Analysis:* We summarize a workload characterization, and opportunities for adaptation in Section 2.
- *Reconfiguration Framework:* We describe the design of a holistic reconfiguration framework—including objectives, tuning knobs, and optimization algorithm—in Section 3.

2 WORKLOAD CHARACTERIZATION

In order to motivate the impact of model variants, we explore different techniques for resource-efficient model serving for both traditional ML and DNN models as well as LLMs.

2.1 Traditional ML/DNN Models

We start with traditional linear and tree-based ML models as well as a spectrum of DNNs. After a summary of typical workload characteristics, we share selected experiments with a variety of techniques for improving the resource efficiency.

Existing Traces: There are a number of existing characterizations of ML workloads. First, Xin et al. summarized a Google trace [57] of 3,000 ML pipelines and 450,000 trained models. Key findings are the frequency of different feature transformations (e.g., top-K vocabulary encoding of tokens), diversity of model types (65% DNNs and 30% linear and tree-based models), and distribution of compute costs (only 25% spent in training, the rest is spent in data pre-processing and validation). Second, a team from Microsoft analyzes 8M public and 2M internal data science notebooks [40]. This analysis compares the use of libraries and models over time. The top-5 ML models comprise the classical Logistic Regression, Naïve Bayes, SVM, Linear Regression, and Random Forest, while the top feature transformations are Z-scoring, Bag-of-Words, TF-IDF, and polynomial features. These characterizations show the persistent relevance of such traditional ML models.

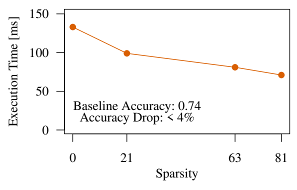
Table 1: Train/Test Accuracy w/ Clustered Models (up to 16).

Dataset	Baseline	Clustered Models
Adult	85.1% / 85.0%	85.3% / 85.1%
Covtype	70.8% / 70.8%	**73.8% / 73.8%**
KDD 98	94.9% / 95.0%	98.2% / 31.7%

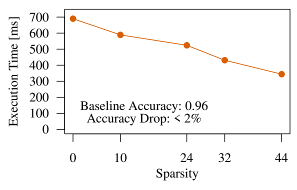
Resource-efficient Variants: Common strategies to improve the accuracy-runtime trade-off of such models are

- *Sparsity:* Examples of enforcing sparsity are lasso regression as a form of feature selection, as well as weight pruning toolkits [17]. Such a pruning can yield 4× to 10× reductions in non-zeros (and thus, reduce data transfer and floating point operations) with less than 2% accuracy degradation.
- *Clustering:* In order to learn simple models per sub-space of the data distribution, one can cluster the training data and learn individual models per cluster [20, 48].
- *Quantization:* Especially for DNNs, it is customary to run inference on quantized representations (e.g., `UINT8` instead of `FP32`). Typically, simple approaches like min-max quantization [18] (equi-width) are applied but there are also optimized quantization schemes [61] (equi-height bins) and learned quantization schemes (number of bins).
- *Cluster-based Quantization:* Several techniques for compressed vector similarity search emerged in the context of vector databases [51]. One of such techniques splits input vectors into M subvectors, performs clustering, and then represents every subvector as the cluster ID.

Quantization Experiments: Figure 1(a) shows the results of applying cluster-based quantization to the Adult dataset and training a basic logistic regression model. We observe that even with 2 or 4 subvectors and 128/256 centroids (so only four 8bit values per data point), we reach the baseline accuracy. Furthermore, Table 1 shows the results of clustered linear models (again with a basic logistic regression) for three datasets, where on Covtype this strategy indeed improves accuracy by more than 3% while on other datasets this strategy is either ineffective or even degrades test accuracy.



(a) LogReg on Covtype



(b) LogReg on Criteo

Figure 2: Runtime and Accuracy Impact of Sparsification.

Sparsification Experiments: In addition, Figures 1(b) and 1(c) show the accuracy of ResNet50 on subsets of ImageNet. We vary the pruning ratio (fraction of pruned weights) without any retraining. We observe significant reductions in accuracy, but latency improvements that are proportional to the pruning ratio. Figures 2 further shows the runtime impact of sparsification for logistic regression on Covtype and Criteo, where we see substantial improvements at very small accuracy degradation of 4% and 2%, respectively.

2.2 Foundation Models

In addition to traditional ML models, foundation models [7]—which are high-capacity models trained on massive datasets—have been widely used in various applications. We summarize existing traces, model diversity, resource-efficient variants, and potential optimizations for LLMs as a particular type of foundation models.

Existing Traces: LLMs exhibit different characteristics than traditional ML models: the execution time of a single request is usually not constant, but depends on the input and output length. To facilitate research on LLM serving, several traces have been released. For example, the BurstGPT [54] and AzureLLM [38] traces contain the request arrival times and the input and output lengths of the requests. Furthermore, the Mooncake trace [41] has a `hash_id` that can be used to study caching strategies, and LMSys-chat-1M [64] includes the real user requests and the corresponding responses.

Models & Applications: LLMs are often deployed together with other ML/DNNs models and vector DBs to serve applications. For example, in RAG, a smaller model typically reranks relevant documents, while a larger LLM processes the prompt with those documents as context [34, 39]. Other platforms deploy multiple LLMs and use a routing model to select the most suitable one for generating a response [21, 36]. Accuracy, freshness, and latency requirements also largely differ. In some applications, such as chat bots, low latency for fast responses is the top priority, while offline batch processing and text summarization tolerate higher latencies and are optimized for throughput and resource utilization [47].

Resource-efficient Variants: Common strategies to improve the accuracy-throughput-latency trade-off of such models are

- *Quantization:* Different quantization strategies offer different trade-offs of accuracy and runtime on different tasks.
- *Model Routing:* In multi-model serving, a model router can be deployed to select the most resource- and cost-efficient model or their variants for each request.
- *Parallelization Strategies:* Data [43], tensor [44], and pipeline paralellism [22] are common parallelization strategies that offer different tradeoffs in throughput (measured as *Tokens Per Second: TPS*) and latency (measured as *Time to First Token: TTFT*) at varying levels of computational load.

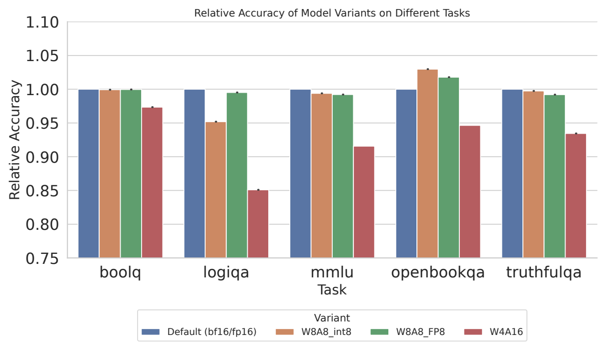


Figure 3: Impact of Quantization on the Model Accuracy.

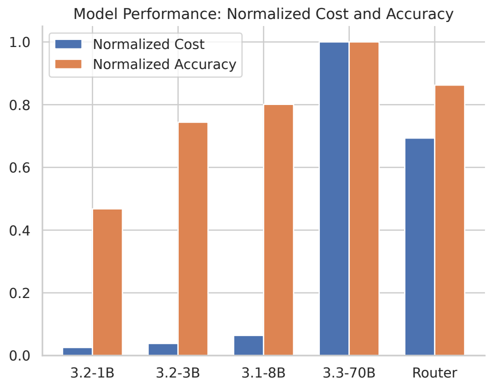


Figure 4: Impact of Router on Model Accuracy and Cost.

Quantization Experiments: We evaluate the impact of quantization on the accuracy of the model across various downstream tasks. Figure 3 shows the relative accuracy of the *Llama-3.1-8B-Instruct* model after quantization to three different precision levels: `FP16` (baseline) `FP8,Int8` (quantization of both weights and activations), and `Int4` (weight-only quantization). We find that on different tasks, different precisions yield different accuracy. For example, `Int4` weight-only quantization reached baseline accuracy on `boolq` but only 85% accuracy on the `logiqa` task.

Routing Experiments: We evaluate a simple router that uses a learned model selection policy on the `mmlu` dataset with 80/20 split of training and test data. The model router is essentially a binary classifier that predicts whether the request should be served by each model, with a penalty for costs, aiming to minimize both latency and resource usage. Figure 4 shows the impact of using different models on the accuracy and cost. While using the largest, 70B parameter model yields the highest accuracy, it also incurs the highest cost. The router can select a smaller model for some requests, achieving a good trade-off between accuracy and cost.

Parallelization Experiments: Generally, tensor parallelism can provide the lowest latency at low load by distributing the computation for each request across multiple GPUs. In contrast, data parallelism achieves higher throughput at elevated loads, as it avoids the need for inter-GPU communication, allowing each GPU to process different batches independently. Figure 5 compares the latency and throughput at different request rates for using data (DP), tensor (TP), or pipeline (PP) parallelism on 4×A100 GPUs. These parallelism strategies can also be combined, and Figure 6

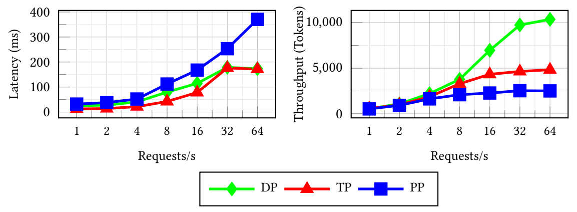
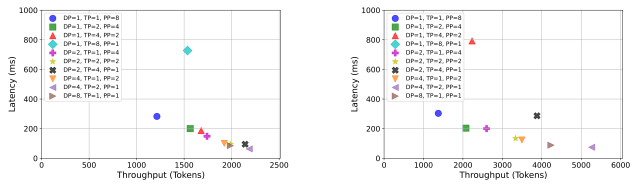


Figure 5: Latency and throughput for different parallelization strategies on LLama 2-13B on 4×A100 GPUs.



(a) Request Rate: 8 requests/s (b) Request Rate: 32 Requests/s

Figure 6: Latency vs throughput for hybrid-parallel strategies on `Int8`-quantized Llama 3.3-70B on 8×H100 GPUs.

shows their latency-throughput trade-offs on a larger cluster of 8×H100 GPUs at different request rates.

Summary: Finally, the behavior of resource-efficient model variants and runtime strategies is strongly workload-dependent and thus, would benefit from adaptive reconfigurability.

3 DYNAMIC RECONFIGURATION FRAMEWORK

We envision a holistic ML serving framework for both traditional ML models and LLMs that is capable of monitoring the serving workload and performance of model variants in order to dynamically reconfigure serving configurations. In this section, we first outline key design principles, before describing the system architecture, service level objectives, monitoring framework, individual tuning knobs, and the overall tuning algorithm in detail.

3.1 Design Principles

Our vision is based on the following four key design principles, which set the scope and influence the overall system architecture.

- **P1 – ML Pipeline Serving:** We aim to serve end-to-end ML pipelines, including pre-processing steps as well as both traditional ML and DNN models and LLMs with their specific runtime strategies. There is a trend towards composite multi-model pipelines (e.g., object detectors, classifiers, and LLMs) such that joint serving is crucial. An awareness of data dependencies is further important for understanding error propagation and latency implications.
- **P2 White-box Model Variants:** To understand the characteristics and lineage of model variants, we do not treat models as black boxes with static accuracy/runtime profiles like prior work [1, 31, 42, 63]. Instead, we explicitly describe how the model variants were created (e.g., through sparsification or quantization) and the expected impact on efficiency. Understanding the model variants then even allows deriving new model variants during serving runtime.

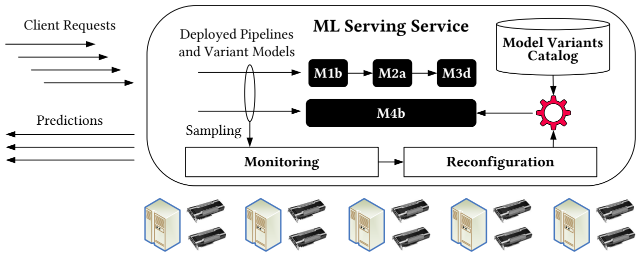


Figure 7: Reconfigurable ML Serving System Architecture.

- **P3 Continuous Monitoring:** A central piece of our dynamic reconfiguration framework, is the continuous monitoring of client requests. Sampling client requests allows bounding the overhead while comparing model variants (to their base variant with presumed highest accuracy) for runtime and accuracy on the actual client requests. This online monitoring further allows handling concept drift.
- **P4 Holistic Reconfigurability:** In contrast to the static configurations of ML/DNN and LLM serving services, we aim at a holistic dynamic reconfigurablity. Besides dynamic reconfiguration based on continuous monitored characteristics, this capability also entails the joint optimization of model variant selection as well as runtime optimizations like batching, reuse, and parallelization strategies.

3.2 System Architecture

Our overall system architecture—shown in Figure 7—follows the architecture of existing ML serving systems and uses open-source components from PyTorch and vLLM, but adds three components: a model variants catalog, as well as monitoring and reconfiguration components. In the following, we describe how these components interact before going into more detail in subsequent sections.

Model Variants Catalog: On deploying a new ML pipeline in the serving system, all models of this pipeline (e.g., M1, M2, M3) as well as their variants (e.g., M1a, M1b, M1c, M1d) are registered in the catalog. Besides the storage locations of the model weights, we store the artifacts for running these models, the dependencies, as well as lineage of the variants (e.g., M1a base model, M1b `UINT8`-quantization of M1a, M1c `UINT4`-quantization of M1a). The catalog further stores historic monitoring data of the variants (e.g., accuracy, latency, throughput, and energy) as well as workload characteristics (e.g., request rates per pipeline). With these statistics, we can even detect seasonal patterns and optimize pipelines proactively.

Monitoring: To enable adaptive, dynamic reconfiguration, we continuously monitor the performance of different model variants. A small fraction of client requests is sampled and run through all relevant variants of a model which enables comparing the accuracy and runtime to the base model (which predictions we assume as ground truth). Interestingly, this continuous monitoring can be performed off the critical path: we sample client requests and store them temporarily and schedule in idle times the evaluation of these variants and parallelization strategies. Evaluating batches of requests per variant also prevents too frequent swapping of variants, and allows measuring energy consumption without inference.

Reconfiguration: Our central reconfiguration component utilizes the collected statistics to deploy the most resource-efficient configuration (one variant per pipeline model, parallelization/batching/reuse strategies) that still preserves the defined service level objectives (SLOs). We trigger this reconfiguration periodically as a background task. Once the new optimal configuration is found, we deploy it in a delayed manner (during idle time with max waiting period) to avoid interfering with the main serving workload. The reconfiguration component can register monitoring policies for sampling different models and variants in a more directed search.

3.3 Monitoring

Continuous monitoring is the basis for workload-aware reconfiguration and relies on monitoring policies that we derive from pipeline SLOs. Here, we describe the registration of SLOs, monitoring policies, and measurement of service level indicators (SLIs).

SLOs and SLIs: Our overall optimization objective is to maximize throughput (number of user requests per second) subject to constraints of SLOs. When registering a new ML pipeline, users may specify different objectives for the ML pipeline and individual models of this pipeline. Example SLOs include accuracy (e.g., 95% of client requests need to match the base model), latency (e.g., 95% of client requests need to be answered in <100 ms), and energy consumption (e.g., average energy consumption or CO2 emissions per request). Additionally, we allow for other constraints like resource consumption and user/application-specific SLOs.

Monitoring Policies: Users and the reconfiguration component can register monitoring policies for individual ML pipelines and their models. Such a policy defines the sampling fraction of requests, a subset of variants to evaluate (with weights to allow directed exploration proportional to expected returns), and flags for evaluating different parallelization strategies, batching, and others. Disabling certain exploration allows avoiding unnecessary overhead for rather static effects (e.g., impact of batching on latency and throughput). Finally, if an ML pipeline has no model variants other than the base models and parallelization strategies are not reevaluated, monitoring for this pipeline can be disabled altogether.

Data Collection and Measurement: In order to satisfy the SLOs, we measure various SLIs such as latency, accuracy, and energy consumption of sampled client requests. We randomly sample requests, store them temporarily, and evaluate combinations of different variants and runtime strategies batch-wise. Inspired by lossy compression with guarantees [25], we can derive high-probability guarantees on the respective SLOs. Evaluating larger batches of requests for different model variants enables effective model-hopper parallelism [33], evaluating different batch sizes, and creates homogeneous requests which makes runtime and energy measurements more reliable. For energy consumption, we utilize performance counters of the CPU and different hardware accelerators [59]. Inspired by AutoML tools, we then perform a Grid Search or Bayesian Optimization for different combinations of model variants, parallelization strategies, reuse, and batching while utilizing the measured request rate as upper bound for batching.

3.4 Reconfiguration Tuning

Finally, we describe the tuning of ML pipeline configurations. We start by defining the scope of tuning knobs as well as pruning opportunities, and then sketch our initial heuristic tuning algorithm.

Algorithm 1 Reconfiguration Tuning

```
Input: History of measurements $\mathcal{M}$, deployed pipelines $\mathcal{P}$, state
    $\mathcal{S}$, hardware topology $\mathcal{T}$, and SLOs per pipeline $\mathcal{P}$, maxiter $L$
Output: Configurations of all pipelines $C$
 1: // a) variant selection
 2: for all i in 1 to $|\mathcal{P}|$ do
 3:     $C_i \leftarrow$ pick variant that satisfies the accuracy constraint
 4: // b) iterative refinements
 5: $i \leftarrow 0, C' \leftarrow \emptyset$
 6: while $C \neq C' \wedge i < L$ do
 7:     $C' \leftarrow C$
 8:     for all i in 1 to $|\mathcal{P}|$ do
 9:         $C'_i \leftarrow$ map $\mathcal{P}_i$ to share of hardware devices from $\mathcal{T}$
10:         $C'_i \leftarrow$ pick batching $\tau$ that satisfies latency constraint
11:         $C'_i \leftarrow$ pick parallelization, reuse, and eviction strategies
12:     $i \leftarrow i + 1$
13: return $C'$
```

Tuning Knobs: Our dynamic reconfiguration framework aims to provide holistic reconfigurability (**P4**). In this context, we simultaneously consider the following tunable parameters (knobs):

- **Model Variants:** Alternative models derived by sparsification, quantization, or specialization (e.g., fewer layers, or fine-tuned for specific task).
- **Data Batching:** Collect multiple client requests to the same model for a maximum duration or batch size and invoke the model once for entire batch of inputs.
- **Parallelization Strategies:** Different ways of parallelizing requests such as pipeline parallelism (pre-processing and sequence of models), task parallelism (different client requests, independent models or experts in MoE models), and data/tensor parallelism (operation parallelism across cores or devices), including the degree of parallelism.
- **Reuse Strategies:** Cache and reuse frequently appearing input-output pairs (e.g., in translation), as well as LLM KV cache reuse across prompts, and prompt history summarization, which can increase reuse.
- **Model/Operator Placement:** Mapping of models or operators with part of the model weights to different hardware devices which hold the model in memory.
- **Checkpointing and Eviction Strategies:** Different ways of swapping models in devices while preserving the state of conversational LLM interactions.

Pruning Opportunities: By explicitly modeling these different tuning knobs, we can exploit the known expected behavior for pruning the search space of evaluating model variants. Specifically, we assume *monotonicity* for batch sizes and differently sparsified/quantized model variants. If a batch size already fails the latency SLO, then larger batch sizes do not need to be evaluated, and similar for the accuracy degradation of model variants.

Tuning Algorithm: The optimization problem is to find the configuration that maximizes throughput subject to the SLOs. Although there are many ways for solving this problem (e.g., RL, MILP, or evolutionary algorithms), we start with a heuristic approach. Algorithm 1 takes the statistics, pipelines, state, hardware and SLOs as inputs and computes the configurations for all pipelines. First, we select the most resource-efficient model variant that still satisfies the accuracy constraint. Second, we iteratively refine the configurations by mapping the individual pipelines to a share of hardware devices (proportional to their total inference times in the last period) and optimize parallelization, reuse, and eviction accordingly.

4 RELATED WORK

Our vision is related to cloud-based ML serving services, optimizations for various (ML, DNN, LLM) serving workloads, adaptive query processing, as well as adaptive ML serving.

ML Serving Services: Cloud-based ML serving is very common, especially for complex models. Example systems, besides basic function-as-a-service (FaaS) auto-scaling [19], include TensorFlow Serving [35] and TorchServe (which are both also used in AWS model serving), Velox [10], Clipper [11], Pretzel [29], and Rafiki [52], which provide configurable serving with batching and different parallelization strategies. Clipper [11] further allowed for decoupled multi-framework scoring as well as result caching. Already in 2016 Google Translate [56] served 140B words a day running on 82,000 GPUs, which is a great use case for result caching [11]. Pretzel [29] and Rafiki [52] added dedicated multi-model optimizations, where Rafiki also reasons about accuracy in the form of ensembles of models. In contrast to these serving systems, our vision entails comparing resource-efficient model variants with their base models.

Optimizations for ML/DNN/LLM Workloads: Besides basic data and model batching, result caching, and common-subexpression elimination across models, there are additional runtime optimizations. First, both DNNs and LLMs can be quantized to low-precision floating-point or integer representations to improve instruction-level parallelism and reduce energy consumption [18, 61] at moderate accuracy degradation. Similarly, most LLMs are provided in different quantization levels [37]. Second, sparsifying the input data (i.e., feature selection) and model weights (i.e., pruning) can reduce the number of FLOPs with moderate accuracy degradation [17, 58]. Third, HummingBird vectorizes tree-based model inference for batches of client requests through alternative non-iterative and iterative tensor computations [32]. Fourth, systems like NoScope [27] utilized forms of model specialization for efficient video analytics. Instead of running every frame through a YOLO model, specialized few-layer DNNs were trained for specific queries. Similarly, IDK-Cascades [53] composes sequences of increasingly complex models to reduce costs for inputs where simple models yield high-confidence predictions. Fifth, with many specialized models, deduplication of model fragments (e.g., in lower layers) can effectively reduce the memory consumption of this multitude of models [28]. Many of these optimization techniques yield a spectrum of variants with different performance-accuracy tradeoffs. Our vision is to dynamically choose among such variants to satisfy constraints.

Adaptive Query Processing: There exist a spectrum AQP techniques [3, 14, 15, 23] which aim to address the challenges of unknown and changing data characteristics in query optimization and processing. Many of the existing techniques follow a continuous control loop of monitoring and analyzing statistics, replanning, and plan execution. Examples include inter-query optimization with learned cardinalities for expressions [8, 9, 45], late binding with re-optimization at pipeline breakers [14] or parameter binding [6], inter-operator re-optimization at checkpoints [26], progressive (reactive) and pro-active re-optimization with validity ranges of statistics [4, 30], intra-operator adaptivity with union stitch-up plans [24], intra-query adaptivity via reinforcement learning [49, 50, 55], as well as tuple routing policies in Eddies [2, 5, 13]. Inspired by AQP, we also periodically monitor different model variants, but then decide on ML pipeline and model configurations which affects serving throughput, latency, energy consumption, and accuracy.

Adaptive ML Serving: Some of the existing ML serving systems also exhibit adaptive behavior and optimizations. For example, Clipper [11] collects varying batch sizes according to maximum latency constraints, and Rafiki [52] uses reinforcement learning to find the optimal batch size and model ensemble configuration. LLM inference comes with new challenges of load imbalance in mixture-of-experts parallelism (where requests are routed to a subset of experts) [12] as well as unknown state sizes in conversational and reasoning models. Llumnix [46] handles such unpredictable requests through runtime scheduling of multiple models and live migrations. Similarly, ServerlessLLM [16] provides fast checkpointing, restore, and live migrations for FaaS LLMs inference. ORCA [60] schedules at a fine granularity of individual generated tokens in order to avoid batching inefficiencies. Furthermore, Shepherd [62] uses statistical multiplexing for scheduling groups of requests with predictable performance. Finally, closely related to our vision are Model-Switching [63] (model variants with different numbers of layers), Loki [1] (model variants obtained through model compression), INFaaS [42] (compiler- and hardware-specific model variants), and RAMSIS [31] (arbitrary model variants), which provide hardware and accuracy scaling by switching to less-accurate models under heavy load or more-accurate models in idle times. In contrast to our holistic reconfiguration framework, these scheduling algorithms assume predetermined runtime and accuracy profiles of black-box model variants and treat clients homogeneously.

5 CONCLUSIONS

To summarize, we introduced our vision of a holistic reconfiguration framework—including objectives and optimization algorithm—for serving ML pipelines and models in a resource-efficient manner. A potential analysis with a variety of optimization techniques showed promising results. In conclusion, the dynamic reconfiguration of ML serving systems has the potential of automatically selecting the most resource-efficient serving configurations without danger of unintended degradation of model accuracy and/or latency. The periodic comparison of model variants ensures high-probability satisfaction of imposed constraints. Our vision opens up a wide spectrum of directions for future work: tuning different forms of sparsity exploitation, tuning model quantization (e.g., mixed precision), tuning conversational and reasoning LLMs, tuning batching and parallelization strategies, tuning deduplication strategies of specialized models and model variants, as well as special optimizations for different objectives such as latency and energy consumption.

ACKNOWLEDGMENTS

We thank all participants of the Dagstuhl Seminar 24311 for many valuable discussions on resource-efficient ML.

REFERENCES

[1] Sohaib Ahmad, Hui Guan, and Ramesh K. Sitaraman. 2024. Loki: A System for Serving ML Inference Pipelines with Hardware and Accuracy Scaling. In *HPDC*. 267–280. https://doi.org/10.1145/3625549.3658688

[2] Ron Avnur and Joseph M. Hellerstein. 2000. Eddies: Continuously Adaptive Query Processing. In *SIGMOD*. 261–272. https://doi.org/10.1145/342009.335420

[3] Shivnath Babu and Pedro Bizarro. 2005. Adaptive Query Processing in the Looking Glass. In *CIDR*. 238–249. http://cidrdb.org/cidr2005/papers/P20.pdf

[4] Shivnath Babu, Pedro Bizarro, and David J. DeWitt. 2005. Proactive Re-optimization. In *SIGMOD*. 107–118. https://doi.org/10.1145/1066157.1066171

[5] Pedro Bizarro, Shivnath Babu, David J. DeWitt, and Jennifer Widom. 2005. Content-Based Routing: Different Plans for Different Data. In *VLDB*. http://www.vldb.org/archives/website/2005/program/paper/thu/p757-bizarro.pdf

[6] Pedro Bizarro, Nicolas Bruno, and David J. DeWitt. 2009. Progressive Parametric Query Optimization. *IEEE Trans. Knowl. Data Eng.* 21, 4 (2009), 582–594. https://doi.org/10.1109/TKDE.2008.160

[7] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. 2022. On the Opportunities and Risks of Foundation Models. In *arXiv preprint*. https://doi.org/10.48550/arXiv.2108.07258

[8] Nicolas Bruno and Surajit Chaudhuri. 2002. Exploiting statistics on query expressions for optimization. In *SIGMOD*. 263–274. https://doi.org/10.1145/564691.564722

[9] Chung-Min Chen and Nick Roussopoulos. 1994. Adaptive Selectivity Estimation Using Query Feedback. In *SIGMOD*. https://doi.org/10.1145/191839.191874

[10] Daniel Crankshaw, Peter Bailis, Joseph E. Gonzalez, Haoyuan Li, Zhao Zhang, Michael J. Franklin, Ali Ghodsi, and Michael I. Jordan. 2015. The Missing Piece in Complex Analytics: Low Latency, Scalable Model Management and Serving with Velox. In *CIDR*. http://cidrdb.org/cidr2015/Papers/CIDR15_Paper19u.pdf

[11] Daniel Crankshaw, Xin Wang, Giulio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System. In *NSDI*. 613–627. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/crankshaw

[12] DeepSeek-AI. 2024. DeepSeek-V3 Technical Report. https://github.com/deepseek-ai/DeepSeek-V3/blob/main/DeepSeek_V3.pdf.

[13] Amol Deshpande. 2004. An initial study of overheads of eddies. *SIGMOD Rec.* 33, 1 (2004), 44–49. https://doi.org/10.1145/974121.974129

[14] Amol Deshpande, Joseph M. Hellerstein, and Vijayshankar Raman. 2006. Adaptive query processing: why, how, when, what next. In *SIGMOD*. 806–807. https://doi.org/10.1145/1142473.1142603

[15] Amol Deshpande, Zachary G. Ives, and Vijayshankar Raman. 2007. Adaptive Query Processing. *Found. Trends Databases* 1, 1 (2007), 1–140. https://doi.org/10.1561/1900000001

[16] Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. 2024. ServerlessLLM: Low-Latency Serverless Inference for Large Language Models. In *OSDI*. 135–153. https://www.usenix.org/conference/osdi24/presentation/fu

[17] Google. 2019. TensorFlow Model Optimization Toolkit — Pruning API. https://blog.tensorflow.org/2019/05/tf-model-optimization-toolkit-pruning-API.html.

[18] Google. 2020. Quantization Aware Training with TensorFlow Model Optimization Toolkit - Performance with Accuracy. https://blog.tensorflow.org/2020/04/quantization-aware-training-with-tensorflow-model-optimization-toolkit.html.

[19] Joseph M. Hellerstein, Jose M. Faleiro, Joseph Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. 2019. Serverless Computing: One Step Forward, Two Steps Back. In *CIDR*. http://cidrdb.org/cidr2019/papers/p119-hellerstein-cidr19.pdf

[20] Vitali Hirsch, Peter Reimann, Dennis Treder-Tschechlov, Holger Schwarz, and Bernhard Mitschang. 2023. Exploiting domain knowledge to address class imbalance and a heterogeneous feature space in multi-class classification. *VLDB J.* 32, 5 (2023), 1037–1064. https://doi.org/10.1007/S00778-023-00780-6

[21] Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. 2024. RouterBench: A Benchmark for Multi-LLM Routing System. *ArXiv* abs/2403.12031 (2024). https://api.semanticscholar.org/CorpusID:268532126

[22] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. 2019. *GPipe: efficient training of giant neural networks using pipeline parallelism.* Curran Associates Inc., Red Hook, NY, USA.

[23] Zachary G. Ives, Amol Deshpande, and Vijayshankar Raman. 2007. Adaptive query processing: Why, How, When, and What Next?. In *VLDB*. 1426–1427. http://www.vldb.org/conf/2007/papers/tutorials/p1426-deshpande.pdf

[24] Zachary G. Ives, Alon Y. Halevy, and Daniel S. Weld. 2004. Adapting to Source Properties in Processing Data Integration Queries. In *SIGMOD*. 395–406. https://doi.org/10.1145/1007568.1007613

[25] Sian Jin, Chengming Zhang, Xintong Jiang, Yunhe Feng, Hui Guan, Guanpeng Li, Shuaiwen Song, and Dingwen Tao. 2021. COMET: A Novel Memory-Efficient Deep Learning Training Framework by Using Error-Bounded Lossy Compression. *Proc. VLDB Endow.* 15, 4 (2021), 886–899. https://doi.org/10.14778/3503585.3503597

[26] Navin Kabra and David J. DeWitt. 1998. Efficient Mid-Query Re-Optimization of Sub-Optimal Query Execution Plans. In *SIGMOD*. 106–117. https://doi.org/10.1145/276304.276315

[27] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. NoScope: Optimizing Deep CNN-Based Queries over Video Streams at Scale. *Proc. VLDB Endow.* 10, 11 (2017), 1586–1597. https://doi.org/10.14778/3137628.3137664

[28] Seulki Lee and Shahriar Nirjon. 2020. Fast and scalable in-memory deep multitask learning via neural weight virtualization. In *MobiSys*. 175–190. https://doi.org/10.1145/3386901.3388947

[29] Yunseong Lee, Alberto Scolari, Byung-Gon Chun, Marco Domenico Santambrogio, Markus Weimer, and Matteo Interlandi. 2018. PRETZEL: Opening the Black Box of Machine Learning Prediction Serving Systems. In *OSDI*. 611–626. https://www.usenix.org/conference/osdi18/presentation/lee

[30] Volker Markl, Vijayshankar Raman, David E. Simmen, Guy M. Lohman, and Hamid Pirahesh. 2004. Robust Query Processing through Progressive Optimization. In *SIGMOD*. 659–670. https://doi.org/10.1145/1007568.1007642

[31] Daniel Mendoza, Francisco Romero, and Caroline Trippel. 2024. Model Selection for Latency-Critical Inference Serving. In *Proceedings of the Nineteenth European Conference on Computer Systems* (Athens, Greece) *(EuroSys '24)*. Association for Computing Machinery, New York, NY, USA, 1016–1038. https://doi.org/10.1145/3627703.3629565

[32] Supun Nakandala, Karla Saur, Gyeong-In Yu, Konstantinos Karanasos, Carlo Curino, Markus Weimer, and Matteo Interlandi. 2020. A Tensor Compiler for Unified Machine Learning Prediction Serving. In *OSDI*. 899–917. https://www.usenix.org/conference/osdi20/presentation/nakandala

[33] Supun Nakandala, Yuhao Zhang, and Arun Kumar. 2020. Cerebro: A Data System for Optimized Deep Learning Model Selection. *Proc. VLDB Endow.* 13, 11 (2020), 2159–2173. http://www.vldb.org/pvldb/vol13/p2159-nakandala.pdf

[34] Rodrigo Nogueira, Wei Yang, Kyunghyun Cho, and Jimmy J. Lin. 2019. Multi-Stage Document Ranking with BERT. *ArXiv* abs/1910.14424 (2019). https://api.semanticscholar.org/CorpusID:207758365

[35] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. 2017. TensorFlow-Serving: Flexible, High-Performance ML Serving. In *ML-Systems@NeurIPS Workshop*. http://arxiv.org/abs/1712.06139

[36] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. 2024. RouteLLM: Learning to Route LLMs with Preference Data. arXiv:2406.18665 [cs.LG] https://arxiv.org/abs/2406.18665

[37] Yeonhong Park, Jake Hyun, SangLyul Cho, Bonggeun Sim, and Jae W. Lee. 2024. Any-Precision LLM: Low-Cost Deployment of Multiple, Different-Sized LLMs. In *ICML*. https://openreview.net/forum?id=u09gadH3BU

[38] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient generative LLM inference using phase splitting. arXiv:2311.18677 [cs.AR] https://arxiv.org/abs/2311.18677

[39] Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy J. Lin. 2023. RankZephyr: Effective and Robust Zero-Shot Listwise Reranking is a Breeze! *ArXiv* abs/2312.02724 (2023). https://api.semanticscholar.org/CorpusID:265659387

[40] Fotis Psallidas, Yiwen Zhu, Bojan Karlas, Jordan Henkel, Matteo Interlandi, Subru Krishnan, Brian Kroth, K. Venkatesh Emani, Wentao Wu, Ce Zhang, Markus Weimer, Avrilia Floratou, Carlo Curino, and Konstantinos Karanasos. 2022. Data Science Through the Looking Glass: Analysis of Millions of GitHub Notebooks and ML.NET Pipelines. *SIGMOD Rec.* 51, 2 (2022), 30–37. https://doi.org/10.1145/3552490.3552496

[41] Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. 2024. Mooncake: A KVCache-centric Disaggregated Architecture for LLM Serving. arXiv:2407.00079 [cs.DC] https://arxiv.org/abs/2407.00079

[42] Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. INFaaS: Automated Model-less Inference Serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 397–411. https://www.usenix.org/conference/atc21/presentation/romero

[43] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *ArXiv* abs/1802.05799 (2018). https://api.semanticscholar.org/CorpusID:3398835

[44] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053* (2019).

[45] Michael Stillger, Guy M. Lohman, Volker Markl, and Mokhtar Kandil. 2001. LEO - DB2's LEarning Optimizer. In *VLDB*. 19–28. http://www.vldb.org/conf/2001/P019.pdf

[46] Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. 2024. Llumnix: Dynamic Scheduling for Large Language Model Serving. In *OSDI*. 173–191. https://www.usenix.org/conference/osdi24/presentation/sun-biao

[47] Ting Sun, Penghan Wang, and Fan Lai. 2025. HyGen: Efficient LLM Serving via Elastic Online-Offline Request Co-location. https://api.semanticscholar.org/CorpusID:275921474

[48] Dennis Treder-Tschechlov, Peter Reimann, Holger Schwarz, and Bernhard Mitschang. 2023. Approach to Synthetic Data Generation for Imbalanced Multi-class Problems with Heterogeneous Groups. In *BTW*. 329–351. https://doi.org/10.18420/BTW2023-16

[49] Immanuel Trummer, Junxiong Wang, Deepak Maram, Samuel Moseley, Saehan Jo, and Joseph Antonakakis. 2019. SkinnerDB: Regret-Bounded Query Evaluation via Reinforcement Learning. In *SIGMOD*. 1153–1170. https://doi.org/10.1145/3299869.3300088

[50] Immanuel Trummer, Junxiong Wang, Ziyun Wei, Deepak Maram, Samuel Moseley, Saehan Jo, Joseph Antonakakis, and Ankush Rayabhari. 2021. SkinnerDB: Regret-bounded Query Evaluation via Reinforcement Learning. *ACM Trans. Database Syst.* 46, 3 (2021), 9:1–9:45. https://doi.org/10.1145/3464389

[51] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. 2021. Milvus: A Purpose-Built Vector Data Management System. In *SIGMOD*. 2614–2627. https://doi.org/10.1145/3448016.3457550

[52] Wei Wang, Jinyang Gao, Meihui Zhang, Sheng Wang, Gang Chen, Teck Khim Ng, Beng Chin Ooi, Jie Shao, and Moaz Reyad. 2018. Rafiki: Machine Learning as an Analytics Service System. *Proc. VLDB Endow.* 12, 2 (2018), 128–140. https://doi.org/10.14778/3282495.3282499

[53] Xin Wang, Yujia Luo, Daniel Crankshaw, Alexey Tumanov, and Joseph E. Gonzalez. 2017. IDK Cascades: Fast Deep Learning by Learning not to Overthink. *CoRR* abs/1706.00885 (2017). http://arxiv.org/abs/1706.00885

[54] Yuxin Wang, Yuhan Chen, Zeyu Li, Xueze Kang, Zhenheng Tang, Xin He, Rui Guo, Xin Wang, Qiang Wang, Amelie Chi Zhou, and Xiaowen Chu. 2024. BurstGPT: A Real-world Workload Dataset to Optimize LLM Serving Systems. arXiv:2401.17644

[55] Ziyun Wei and Immanuel Trummer. 2022. SkinnerMT: Parallelizing for Efficiency and Robustness in Adaptive Query Processing on Multicore Platforms. *PVLDB* 16, 4 (2022), 905–917. https://www.vldb.org/pvldb/vol16/p905-wei.pdf

[56] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR* abs/1609.08144 (2016). http://arxiv.org/abs/1609.08144

[57] Doris Xin, Hui Miao, Aditya G. Parameswaran, and Neoklis Polyzotis. 2021. Production Machine Learning Pipelines: Empirical Analysis and Optimization Opportunities. In *SIGMOD*. 2639–2652. https://doi.org/10.1145/3448016.3457566

[58] Huanrui Yang, Wei Wen, and Hai Li. 2020. DeepHoyer: Learning Sparser Neural Network with Differentiable Scale-Invariant Sparsity Measures. In *ICLR*. https://openreview.net/forum?id=rylBK34FDS

[59] Jie You, Jae-Won Chung, and Mosharaf Chowdhury. 2023. Zeus: Understanding and Optimizing GPU Energy Consumption of DNN Training. In *NSDI*. 119–139. https://www.usenix.org/conference/nsdi23/presentation/you

[60] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A distributed serving system for {Transformer-Based} generative models. In *OSDI 22*. 521–538.

[61] Hantian Zhang, Jerry Li, Kaan Kara, Dan Alistarh, Ji Liu, and Ce Zhang. 2017. ZipML: Training Linear Models with End-to-End Low Precision, and a Little Bit of Deep Learning. In *ICML*, Vol. 70. 4035–4043. http://proceedings.mlr.press/v70/zhang17e.html

[62] Hong Zhang, Yupeng Tang, Anurag Khandelwal, and Ion Stoica. 2023. SHEPHERD: Serving DNNs in the Wild. In *NSDI*. 787–808. https://www.usenix.org/conference/nsdi23/presentation/zhang-hong

[63] Jeff Zhang, Sameh Elnikety, Shuayb Zarar, Atul Gupta, and Siddharth Garg. 2020. Model-Switching: Dealing with Fluctuating Workloads in Machine-Learning-as-a-Service Systems. In *HotCloud*. https://www.usenix.org/conference/hotcloud20/presentation/zhang

[64] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric. P Xing, Joseph E. Gonzalez, Ion Stoica, and Hao Zhang. 2023. LMSYS-Chat-1M: A Large-Scale Real-World LLM Conversation Dataset. arXiv:2309.11998 [cs.CL]