IPA: Inference Pipeline Adaptation to Achieve High Accuracy and Cost-Efficiency

Anonymous authors
Paper under double-blind review

Abstract

Efficiently optimizing multi-model inference pipelines for fast, accurate, and cost-effective inference is a crucial challenge in ML production systems, given their tight end-to-end latency requirements. To simplify the exploration of the vast and intricate trade-off space of accuracy and cost in inference pipelines, providers frequently opt to consider one of them. However, the challenge lies in reconciling accuracy and cost trade-offs.

To address this challenge and propose a solution to efficiently manage model variants in inference pipelines, we present IPA, an online deep learning **I**nference **P**ipeline **A**daptation system that efficiently leverages model variants for each deep learning task. Model variants are different versions of pre-trained models for the same deep learning task with variations in resource requirements, latency, and accuracy. IPA dynamically configures batch size, replication, and model variants to optimize accuracy, minimize costs, and meet user-defined latency SLAs using Integer Programming. It supports multi-objective settings for achieving different trade-offs between accuracy and cost objectives while remaining adaptable to varying workloads and dynamic traffic patterns. Extensive experiments on a Kubernetes implementation with five real-world inference pipelines demonstrate that IPA improves normalized accuracy by up to 35% with a minimal cost increase of less than 5%.

1 Introduction

Noaways, companies run some or all of their ML pipelines on cloud computing platforms [70]. The efficient deployment of machine learning models is crucial in contemporary systems where ML inference services consume more than 90% of datacenter resources dedicated to machine learning workloads [12, 16]. In various critical applications, such as healthcare systems [29], recommendation systems [56], question-answering, and chatbots [20], a range of machine learning models, including computer vision models [29] and speech models [45], play an essential role. It is imperative to deploy these models cost-efficiently while maintaining system performance and scalability.

Automatic resource allocation is a complex problem that requires careful consideration and has been extensively studied in various domains, including stream processing [27, 31, 46], serverless computing [52, 66], and microservices [30, 33, 76, 77]. Static auto-configuration of hardware resources [68], dynamic rightsizing of resources through autoscaling [73], and maximizing utilization with batching [13] are some of the techniques that have been used for resource management of ML models. In addition to efficient resource allocation, *accurate prediction* is another essential factor influencing machine learning model deployment. In many real-world scenarios, the predictions of these models have significant implications for business [9], industry [24], or human lives [1], and inaccuracies can lead to serious consequences [38, 49]. Hence, ensuring that machine learning models make accurate predictions is critical to producing reliable and trustworthy outcomes.

Table 1: Comparison with previous works

System	Pipeline	Cost	Accuracy	Adaptive
Rim [42]	✓	✗	✓	✗
INFaaS [60]	✗	✓	✓	✗
Inferline [25]	✓	✓	✗	✗
GPULet [22]	✓	✓	✗	✗
Llama [61]	✓	✓	✗	✗
FA2 [59]	✓	✓	✗	✗
Model Switch [75]	✗	✗	✓	✗
Scrooge [41]	✓	✓	✗	✗
Nexus [64]	✓	✓	✗	✗
Cocktail [36]	✗	✓	✓	✗
InfAdapter [63]	✗	✓	✓	✓
IPA	✓	✓	✓	✓

ML inference pipelines, as a chain of ML models, will raise several challenges for performance optimization. In contrast to individually optimizing each stage, optimizing ML inference pipelines end-to-end will capture the correlation between configuration change across multiple pipeline steps. Previous works [25, 41, 47, 61] have proposed solutions to address the challenges of efficient autoscaling, batching, and pipeline scheduling to not only consider the above challenges but also consider the dynamic nature of ML workloads. Nevertheless, none of the above approaches consider the combined optimization of accuracy and resource allocation across pipelines.

In ML inference pipelines, two adaptation techniques are commonly used: ***Autoscaling*** which adjusts resources based

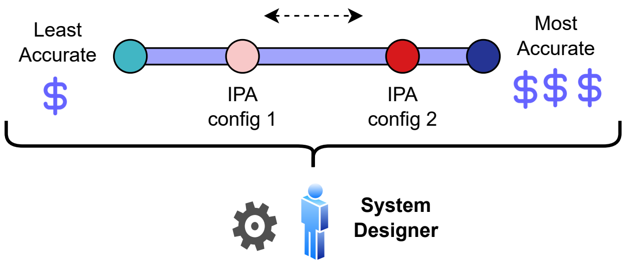


Figure 1: IPA provides a tunable framework for adjusting the system based on two contradictory cost and accuracy rank objectives.

on workload, and ***Model-switching***, which employs different model variants with different accuracies/latencies to vary workloads and tasks, enabling finer control over resource allocation and accuracy. The combination of these two techniques has been advocated to achieve more precise adjustments in accuracy and cost trade-offs. Previous autoscaling [62] and model-switching [63, 75] works have argued that using both techniques in conjunction with each other is beneficial by providing more precise adjustments in terms of accuracy and cost trade-offs, providing greater flexibility and efficiency in ML model resource allocation. However, none of the above works have considered optimizing accuracy and cost jointly in a multi-stage pipeline setting. Table 1 presents an overview of related inference serving works. Systems with inference *pipeline* serving have often overlooked the presence of multiple model variants for each inference task [25, 41, 47, 59, 61]. The heterogeneity of these model variants presents an opportunity not only to configure the pipeline to meet latency objectives but also to opportunistically select the most suitable model variant to enhance the accuracy of the pipeline output.

In this paper, we propose IPA, a system for jointly optimizing accuracy and cost objectives. It can achieve multiple trade-offs between these two contradictory objectives based on the pipeline designer's preference. Figure 1 shows the premise of IPA that provides a tunable framework for adapting the inference pipeline to contradictory accuracy and cost objectives. The choice of models in previous works was limited to using just one pre-selected model variant. IPA can broaden this search space by considering all model variants and dynamically adapting to a suitable choice of models based on the pipeline designer's preference.

The main contributions of this paper are as follows:

- We revisit the resource management design in inference pipelines by incorporating model switching, autoscaling, and pipeline reconfiguration (stage batch size). IPA proposes a new optimization formulation to allow for a more granular trade-off between the accuracy and cost objectives. It is also adaptable based on the inference pipeline designer's preference for each accuracy and cost objective.
- We propose a new optimization formulation based on the interaction between model switching, replication, and batch size. It can find (1) exact resources to allocate to each stage of the pipeline, (2) variants to decide for each stage, and (3) dependency between stages that enable accurate estimation of demand while guaranteeing end-to-end latency.
- The full implementation of IPA is built on top of Kubernetes. IPA integrates open-source technologies in its stack to facilitate seamless integration into production clusters.
- Experimental results show that IPA can achieve more granular trade-offs between the two contradictory objectives of cost and accuracy. In some scenarios, it was able to provide an improvement of up to 35% in normalized accuracy with a negligible increase in cost.

2 Background and motivation

This section provides the background of the inference pipeline and discusses the challenges of reconciling cost and accuracy trade-offs.

2.1 Inference Pipeline

Traditional ML applications revolve around utilizing a singular deep neural network (DNN) for executing inference tasks, such as identifying objects or comprehending natural language. Conversely, modern ML systems (ML inference pipelines) are more intricate scenarios, such as digital assistant services like Amazon Alexa, where a series of interconnected/chained DNNs (in the form of DAG structures) is employed to undertake various inference tasks, spanning speech recognition, question interpretation, question answering, and text-to-speech conversion, all of which contribute to fulfilling user queries and requirements [59, 61]. As these systems frequently interact with users, it becomes essential to have a stringent service level agreement (SLA), which in our case is end-to-end latency.

Figure 2 illustrates the latency and throughput differences across different versions of image classification models within the ResNet family. Notably, there exists an inverse relationship between ***latency***, ***throughput*** and ***accuracy*** for each model variant, considering the same number of ***CPU cores*** and fixed ***batch sizes***. This variability adds another dimension to the trade-off space for selecting model variants within inference pipelines.

Choosing the best configuration among the highlighted parameters is non-trivial and subjective to multiple objectives and constraints, e.g., cost efficiency and SLA requirement between cloud users and providers. Table 2 demonstrates

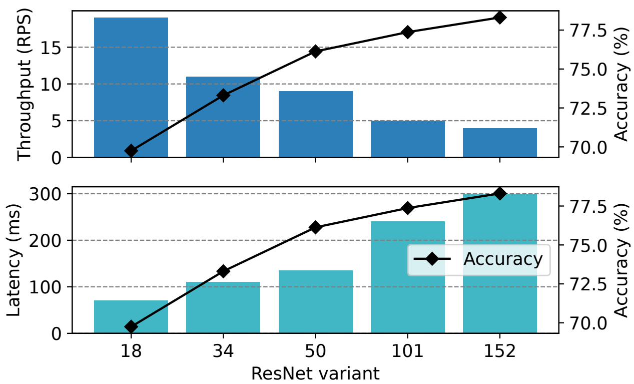


Figure 2: Performance difference across ResNet Family models for a batch size of one and one CPU core allocation

Table 2: Performance difference across ResNet family models under different CPU allocations for a batch size of one, both blue and red core/model configuration can respond to 20 RPS and 75 ms throughput and latency requirements but with different accuracy and costs.

CPU Cores	ResNet18		ResNet50	
	Latency (ms)	Throughput (RPS)	Latency (ms)	Throughput (RPS)
1	75	20	135	9
4	23	37	57	21
8	14	62	32	29

that under the same incoming throughput of 20 RPS and mutual SLA agreement of 75 ms between the user and service provider, a user with high accuracy goals will choose a suitable configuration of four core assignments under ResNet50 (highlighted in red). But a user with lower accuracy demands will choose ResNet18 with one core, which can respond to latency and throughput requirements under lower core allocations (highlighted in blue).

2.2 Configuration Space

Batch Size Neural network structure provides parallel computation capability. Batching multiple requests will leverage the parallelization capability of neural networks and increase the utilization of the assigned resources while increasing the total latency. Previous works [13, 25, 41] have shown that there is a relationship between the system utilization and the request latency resulting in a non-trivial trade-off between maximizing the resource utilization without violating the latency SLA.

Replication There are mainly two resource provisioning techniques, vertical scaling and horizontal scaling. In vertical scaling, the assigned resources are modified, while in horizontal scaling, the number of replicas with the same amount

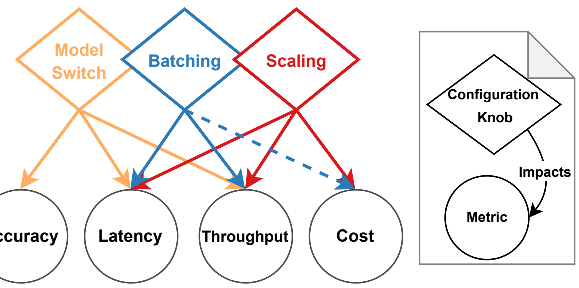


Figure 3: Impact of configuration knobs, batching indirectly affects the cost, e.g., decreasing the throughput will affect the IPA to more scaling and increase in the cost

of resources is adjusted to keep a balance between performance and cost. Horizontal scaling allows predictable performance using a similar environment [35]. while vertical scaling enables a more fine-grained resource allocation of the ML model. We use horizontal scaling for the current work similar to [25, 59].

Variant Selection Previous works [60, 75] have shown that there is an abundance of ML models for a single ML task. This brings the opportunity to abstract away the ML task from the underlying model and opportunistically switch the model based on the performance needs of the system.

Figure 3 shows the complex relationship between changing each configuration knob and performance objectives. Changing the batch size will affect the throughput and latency of each stage of the pipeline, while changes in the replication factor will directly impact the pipeline deployment cost. Model switching will result in changes both in accuracy and cost as different models have different resource requirements.

2.3 Challenges

The nonlinear dependency between the three configuration knobs (batch size, replication, and variant selection) and the pipeline variables introduces a complex decision space between multiple conflicting goals. Figure 6(a) depicts one of the evaluated pipelines consisting of two stages, an object detection stage and an object classifier. A subset of the configuration space is presented in Table 3 with different batch size, accuracy, and cost. We denote cost as the number of replicas $\times$ allocated CPU cores per replica. The chosen configuration at each stage first should support the incoming workload into the pipeline while guaranteeing SLA, e.g., sum of the latency of both stages should be less than the SLA requirement. Under the arrival rate of 20 RPS (Request Per Second) and SLA requirement of 600 milliseconds, both combinations of (A1, B1) and (A2, B2) can be responsive to the latency and throughput requirements. However, the first combination can sustain these requirements under the cost of $2+2=4$ CPU cores with a lower accuracy combination while the latter can sustain the same load with the cost of $10+3=13$ CPU cores

Table 3: Two stage pipeline tasks options.

Variant	Scale	Batch	Latency	Cost	Accuracy
A1: YOLOv5n	2	1	80	2×1	45.7
A2: YOLOv5m	5	1	347	5×2	64.1
A3: YOLOv5n	2	8	481	2×1	45.7
A4: YOLOv5m	5	8	1654	5×2	64.1
B1: ResNet18	2	1	73	2×1	69.75
B2: ResNet50	3	1	136	3×1	76.13
B3: ResNet18	2	8	383	2×1	69.75
B4: ResNet50	3	8	833	3×1	76.13

and a higher possible accuracy combination.

Challenge 1 Multiple configurations can satisfy the latency constraints of the inference pipeline. The "optimal" configuration depends on the accuracy and cost goals.

The next challenge is that in inference pipelines, the model selection at an earlier stage of the pipeline will affect the optimal model selection at downstream models as the latency of a model at an earlier stage will affect the end-to-end latency. Consequently, the available options for the downstream models are more limited. In the similar example of pipeline Figure 6(a) and Table 3, and under an SLA of 500 ms choosing a high latency and high accuracy configuration of A2 at the first stage will eliminate the variant B2 from the second stage's option.

Challenge 2 The choice of replication factor, batch size, and model variant is a joint decision across multiple stages of the inference pipeline.

3 System Design

In this section, we provide a high-level overview of the main system components in IPA as illustrated in Figure 4.

Model Loader Users submit the models they intend to use for each stage of a pipeline. These models should provide a reasonable span of accuracy, latency, and resource footprint trade-offs. Model variants can also be generated using model optimizers such as TensorRT [67], and ONNX graph optimization by using different quantization level of neural networks [32] and Neural Architectural Search methods [21]. After the model submission, the profile (discussed in Section 4.2) will be executed for each model variant and store their latency under multiple batch sizes and resource assignments. Furthermore, the optimizer, as discussed in Section 4, uses the offline profiling to find the optimal solution during the runtime. Finally, the models are stored in object storage

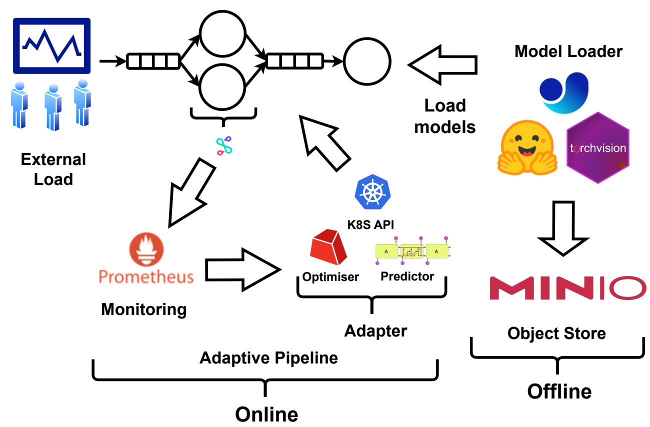


Figure 4: IPA system design

to reduce container creation times. In this work, we have used MinIO object store [5].

Monitoring The monitoring daemon uses the highly-available time-series database Prometheus [8] underneath to observe incoming load to the system. It will periodically monitor the load destined for the pipeline.

Predictor In our load forecasting process, we employ an LSTM (Long Short-Term Memory), which is a type of recurrent neural network [40]. Our LSTM model is designed to predict the maximum workload for the next 20 seconds based on a time series of loads per second collected from the monitoring component over the past 2 minutes. To train the LSTM model, we utilized the initial two weeks of the Twitter-trace dataset [15]. The architecture of our LSTM neural network consists of a 25-unit LSTM layer followed by a one-unit dense layer serving as the output layer.

Pipeline System A centralized load balancer distributes the inference requests between multiple stages of the inference pipeline. A centralized queue is behind each stage of the pipeline. Centralized queues help to have a deterministic queuing behavior and to model its latency efficiently, as described in Section 4. A comparison of using distributed and central queues is provided in Appendix 8. The queues of each stage then distribute the batched requests between model replicas. It uses a round-robin policy for load balancing the batched requests between model replicas. Communications between multiple stages of the pipeline are implemented using gRPC [2]. Load balancing between multiple containers of the same stage is achieved by using Istio [3] sidecar containers. Each model container is deployed using Docker containers built from a forked version of MLServer [6] and Seldon Core [10] for implementing the gRPC web servers and deployment on Kubernetes.

Pipeline Simulated Model The profiling data gathered by the profiler provides the information about latency and throughput of each model variant under different batch sizes. For runtime decision-making, a discrete event simulator uses this profiling

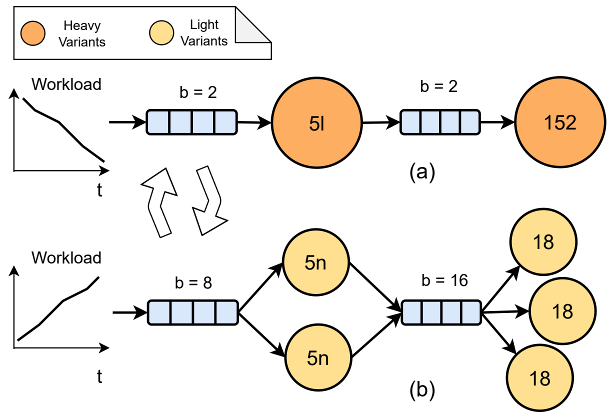


Figure 5: Switching between different configurations under (a) low and (b) high loads.

data to estimate the end-to-end latency and throughput of the pipeline based on the number of replicas, model variants used, and batch sizes at each stage. The predicted latencies and throughputs of the pipeline are then used by the optimizer in the adapter to the optimal configuration in terms of accuracy and cost objectives.

Adapter The Adapter is the auto-configuration model that periodically (1) Fetch incoming load from the monitoring daemon, (2) Predict the next reference load based on observed historical received load using the LSTM module, (3) Obtain optimal configuration in terms of used variant, batch size and number of replicas for the next timestep, and (4) Finally, the new configuration is applied to the pipeline through Kubernetes Python API [4]. Figure 5 shows a snapshot of the system with two available options per each model on a video analysis pipeline, stage one options are $\{Yolo5l, Yolo5n\}$ and stage 2 are $\{ResNet18, ResNet152\}$. In low loads (a), choosing more accurate models $Yolo5l$ and $ResNet152$ with small batch sizes is preferable to ensure low latency. However, in higher loads, it is preferable to choose lightweight models like $Yolo5n$ and $ResNet18$ with more replication and larger batch sizes to ensure high throughput for the system.

4 Problem Formulation

We now present the details of the optimizer discussed in Section 3. The problem formulation needs to have a robust definition of inference pipeline accuracy and also offline latency profiles of the model variants. Subsections 4.1 and 4.2 discuss details of the inference pipeline accuracy definition and profiling methodology respectively.

4.1 Accuracy Definition over Pipeline

To the best of our knowledge, there is no direct way to compute the end-to-end accuracy of a pipeline unless one trains the labeled data on the pipeline. In this work, we used a heuristic for ranking the inference pipelines. In this work, we have only considered linear inference pipelines with one input and one output stage where a set of consecutive models are connected. The accuracy of each model is computed offline and is part of the property of the model. Statically computed accuracy is sufficient since in this work we do not consider model drifts [53]. For models with a qualitative performance measure other than accuracy (e.g., mAP for object detection. WER for Speech recognition tasks, and ROUGE for NLP tasks), as long as we have the *higher (or lower) means better* specification in the measure, we can substitute the accuracy with that measure. For defining a metric over the accuracy in a pipeline, we first sort the accuracy of each stage's model variants from lowest to highest. Then, we assign a zero scale to the least accurate model and one to the most accurate model (normalizing the accuracy). Intermediate model variants are assigned scaled accuracy values between zero and one, proportionally aligned with their rankings in the ordered list. For example, if three model variants exist, the models' scaled accuracy is assigned 0, 0.5, and 1. The overall accuracy over the pipeline is considered as the sum of each stage's model variants' scaled accuracy value. For example, a two-stage pipeline with three model variants per stage and the second most accurate model chosen in each pipeline will have an end-to-end accuracy rank of 0.5 + 0.5 = 1.

4.2 Profiler

The joint cost-accuracy problem formulation needs information about the latency, accuracy, and throughput of each model variant. Previous works have discovered that [35, 41, 59, 60] the latency of a model under a specific resource allocation is predictable based on the incoming batch sizes. The profiler will record the latency on the target hardware for different batch sizes under specific allocations of resources. We found out in our experiments that assigning memory beyond a certain value to the models does not have an impact on the performance; therefore, for memory, we only need to find just enough memory allocation. This will be the memory requirement for running the largest batch size. It is necessary to find a minimum allocation to containers since we have chosen horizontal scaling for workload adaptation in this work. Previous works on inference graphs with CPU evaluation [47, 59] have assigned one core to each container. Adapting a similar approach is not practical in our case as more resource-intensive models cannot execute inference under given latency requirements with one core per container. Therefore, We find a base configuration for each model variant in terms of the number of cores that can provide a reasonable base performance. The solver selects model variants and horizontally scales them with the chosen base configurations.

Table 4 provides some of the base CPU allocations under thresholds 5 RPS, 10 RPS, and 15 RPS for two model variants

Table 4: Sample CPU cores base allocation for different Yolo variants under different RPS thresholds (Capped on maximum 32 cores)

load	Yolov5n	Yolov5s	Yolo5m	Yolov5l	Yolo5x
5	1	1	4	8	16
10	1	2	8	16	×
15	1	8	16	32	×

of the Object Detector task. Values in the allocation columns show the minimum number of CPU allocations per container needed for being responsive to a certain load under a certain *SLA*. We refer to these values as the base resource allocation to a model variant. The base resource allocation for all of the stages ($\forall s \in S$) and their corresponding model variants ($\forall m \in M_s$) is the minimum number of resources in terms of CPU cores (Eq. 1a) that can respond to a certain threshold (Eq. 1b) and be responsive to a predefined per-model base latency *SLA* for the largest batch size in our system (Eq. 1c). Following previous works [34, 59], we define the per-task latency *SLA* as the average latency of all available variants for the task for serving batch size one under the base resource allocation multiplied by 5 as suggested by Swayam [34]. Therefore, the minimum number of resources per model variant can be formulated as:

$$\min R_m \tag{1a}$$

$$\text{subject to } th \leq h(m, R_m) \tag{1b}$$

$$l_m(\max(b_s)) \leq SLA_s \tag{1c}$$

Where threshold th is fixed, and the base resource allocation for all models can be found statistically. In the same Object Detector task with different model variants, as shown in Table 4, we choose the first configuration row as it supports the highest RPS with the minimum resource allocation per container. As a consequence, the base resource allocation is fixed during the runtime, and the throughput of each stage of the pipeline $h(m, R_m)$ will be a function of the used model variant $h(m)$ and the number of replicas n_s.

We follow the practice of the profiler in [60] and record latency and throughput on the power of two increments of 1 to 64 batch sizes. Profiling per all batch sizes is costly, therefore following [59] for each model variant, we fit the observed results on the profiled batch sizes under the base resource allocation to a quadratic polynomial function $l_m(b_s) = \alpha b_s^2 + \beta b_s + \gamma$ that can infer the latency for unmeasured batch sizes. Multiple model variants are available per stage of the inference pipelines, and the mentioned approach can decrease the profiling cost by an order of magnitude.

Table 5: Notations

Symbol	Description
P	Inference pipeline
$s \in P$	Inference pipeline stage
SLA_P	Latency service-level agreement for pipeline P
λ_P	Request arrival rate of pipeline P
M_s	Sets of available model variants for stage s
m	A model variant
b_s	Batch size of stage s
R_m	Resource allocation of variant m
a_m	Accuracy rank of variant m
n_s	Number of replicas of stage s
$I_{s,m}$	Indicator of activeness of variant m in stage s
$q_s(b_s)$	Queuing time of stage s under batch size b
$l_{s,m}(b_s)$	Latency of variant m under batch size b_s
$h_{s,m}(b_s)$	Throughput of variant m under batch size b_s
A_P	End to End accuracy of pipeline P
th	Threshold RPS of base allocation of R_m
α	Accuracy objective weight
β	Resource allocation objective weight
δ	Penalty term for batching

4.3 Optimization Formulation

The goal of IPA is to minimize the accuracy and maximize the cost while guaranteeing SLAs.

$$\text{Objectives} = \begin{cases} \text{Maximizing the Accuracy} \\ \text{Minimizing the Cost} \end{cases} \tag{2}$$

There are $|M_s|$ model variants for each inference stage $s \in P$ in the pipeline P. Each $m \in M_s$ is a different model variant for doing the same inference tasks (e.g., image classification) that exhibit different resource requirements, latency, throughput, and accuracy. Resource requirements of the models are estimated offline in the profiling step (section 4.2). The profiler provides us with the resource requirements of the model variants per each task and their latencies under different batch sizes. The two main goals of IPA are to maximize the accuracy over pipelines and minimize the resource cost by using lighter models. We define $I_{s,m}$ as an indicator of whether a selected model variant is currently active for task s or not:

$$I_{s,m} = \begin{cases} 1 & \text{if } m \text{ is active in stage } s \\ 0 & \text{Otherwise} \end{cases} \tag{3}$$

At each point in time, only one model variant m can be active for each inference stage; therefore, the resource requirement of each replica of models is equal to the active variant resource requirement.

$$R_s = \sum_{m \in M_s} R_{s,m}.I_{s,m} \tag{4}$$

Similarly, the latency and throughput of each pipeline stage are calculated based on the active model variant latency and throughput for that stage.

$$l_s = \sum_{m \in M_s} l_{s,m}(b_s).I_{s,m} \tag{5}$$

$$h_s = \sum_{m \in M_s} h_{s,m}(b_s).I_{s,m} \tag{6}$$

Another contributing factor to the pipeline's end-to-end latency is the time spent on the queue of each inference stage. For queue modeling, we have used the theoretical upper bound formulation introduced in [59]:

$$q_s(b_s) = \frac{b_s - 1}{\lambda} \tag{7}$$

Equation 7 illustrates the worst-case queuing delay based on the arrival rate and the batch size. The first arrived request in a batch should wait for $b_s - 1$ additional request before being sent to the models.

The multi-objective goal (8) is to maximize the pipeline's end-to-end accuracy and minimize the cost. Based on the definition described in section 4.1, the pipeline's end-to-end accuracy is achieved by summing each stage's active model normalized accuracy. The cost objective is achieved in two ways, using smaller models (models with fewer resource requirements) and using the least number of replicas for them. The batch size for each model should be chosen carefully as larger batch sizes will increase utilization and throughput of the entire load but also increase the per batch latency. We should find a batch that increases the utilization at a reasonable scale. Following [59], we have added a small penalty term for batch size in the objective function δ that tries to keep the batch sizes at a reasonable amount.

$$\begin{aligned} f(n,s,I) = \alpha \sum_{s \in P} \Big(\sum_{m \in M_s} a_{s,m}.I_{s,m} \Big) \\ - \beta \sum_{s \in P} n_s.R_s \\ - \delta \sum_{s \in P} b_s \end{aligned} \tag{8}$$

The two α and β variables adjust the preference level given to each objective. We now can describe the auto-configuration of the three online configuration knobs explained in section 2.2 as an IP problem:

$$\begin{aligned}
\max \quad & f(n,s,I) && \text{(9a)} \\
\text{subject to} \quad & \sum_{s \in P} l_s(b_s) + q_s(b_s) \leq SLA_P, && \text{(9b)} \\
& \text{if } I_{s,m} = 1, \text{ then} \\
& \quad n_s \cdot h_s(b_s) \geq \lambda_p, \quad \forall s \in P && \text{(9c)} \\
& \sum_{m \in M_s} I_{s,m} = 1, \quad \forall s \in P && \text{(9d)} \\
& n_s, b_s \in \mathbb{Z}^+, \quad I_{s,m} \in \{0,1\}, \quad \forall s \in S, \forall m \in M_s && \text{(9e)}
\end{aligned}$$

The chosen combination of models should be able to meet the latency (9b) constraint of the pipeline. The pipeline *SLA* of the pipeline is the aggregation of per stage *SLA* as described in Section 4.2. End-to-end latency of the pipeline is obtained by summing the inference latency of the chosen model variant $l_s(b_s)$ and queuing time of each model server $q_s(b_s)$ in the inference path. Also, the sum of the throughput of all replicas of an active model should be higher than the incoming arrival rate 9c into the pipeline.

In summary, the objective function tries to find the most accurate combination of models in the inference pipeline while also trying to allocate the least number of physical resources based on the pipeline designer's preference. This trade-off between the two objectives is configurable by modifying the α and β weights for accuracy and resource allocation.

The inputs of the optimization formulation are, therefore, resource requirements R, latency l, accuracy of all models a, and throughputs h of all model variants m in all stages s under all possible batch sizes b alongside queuing latency model q under all batch sizes, the incoming load λ coming to the pipeline and pipeline's latency *SLA*. In the output, it returns the optimal number of replicas n, batch size b, and the chosen model variant I per each stage of the pipeline.

4.4 Gurobi Solver

IPA has been designed to meet production-level requirements, (1) guaranteeing an optimal solution, (2) no need for additional pre-training or re-training costs, and (3) negligible overhead. Most optimization techniques can be classified into heuristics, IP, or ML approaches. Heuristics are ad-hoc solutions that are hard to generalize to different systems. ML approaches do not guarantee the optimal solution and typically incur long training times (e.g. Reinforcement Learning). In contrast, IP formulation meets all the requirements for a production-ready system. In our case, we chose the Gurobi solver [19] that guarantees the optimal solution; the only downside of using them is that in case of very large search spaces, they might take a long time to find the desirable configuration. In our case, Gurobi was able to solve the problem formulation in Formula 9 in less than a second.

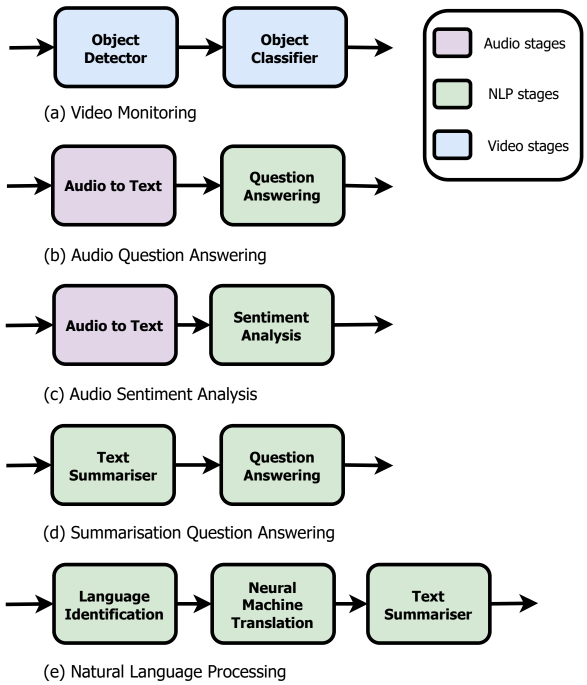


Figure 6: Representative pipelines used in this work

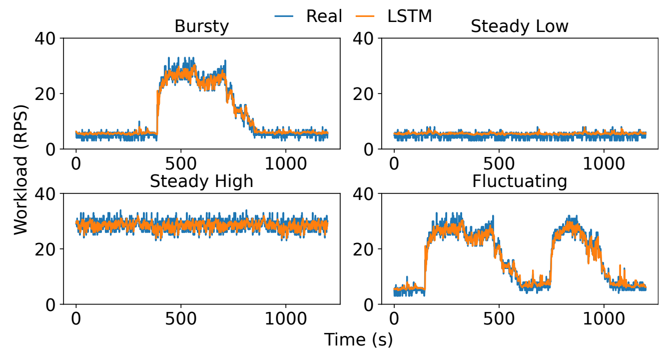


Figure 7: Representative tested load patterns from the Twitter trace [15], showing LSTM predictions

Table 6: Per stage and End to end service level agreements of inference pipelines (in seconds)

Pipelines	Stage 1	Stage 2	Stage 3	E2E
Video Monitoring	4.62	2.27	×	6.89
Audio QA	8.34	0.89	×	9.23
Audio Sentiment	8.34	1.08	×	9.42
Sum QA	2.52	1.32	×	3.84
NLP	0.97	12.76	3.87	17.61

4.5 Dropping

High workloads may cause heavy back pressure at the upstream queues. If a request has already passed its *SLA* at any stage in the inference pipeline, then there might be no point in continuing to serve it until the last stage and incurring high pressure on the system. One mechanism we have employed is to drop a request at any stage of the pipeline if it has already passed its *SLA* in the previous steps. We also consider that a request is dropped if its current latency has exceeded $2\times$ the SLA to avoid constant back pressure on the queues.

5 Evaluation

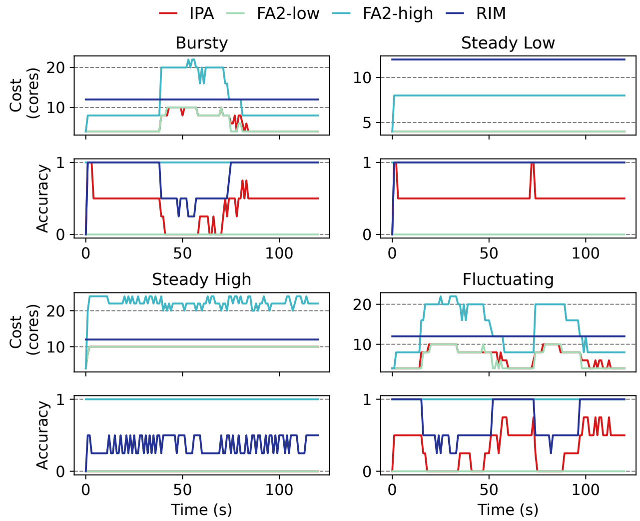
In this section, we conduct extensive experiments to showcase the practical efficacy of IPA in real-world scenarios using a diverse set of workloads. We evaluate IPA using six physical machines from Chameleon Cloud [48]. Each server is equipped with 96 Intel(R) Xeon(R) Gold 6240R CPU @ 2.40GHz cores and 188 Gb of RAM. The IPA is open-sourced at `Link-obscured-for-double-blind`.

5.1 Experimental Setup

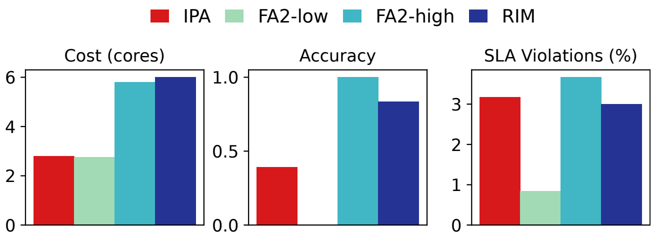
Most previous works on inference pipelines [18, 25, 58, 61] have implemented the entire pipeline on their self-made infrastructures. We have implemented our frameworks on top of Kubernetes, the de facto standard in the containerized world and widely used in industry. This will enable easier access to the framework for future use by developers. IPA is implemented in Python with over 8K lines of code, including the adapter, simulator, queuing, load balancer, and model container implementations.

Pipelines We use five descriptive pipelines with a wide variety of models for each stage as shown in Figure 6. The pipelines are adapted from previous works and also from industrial examples. Video monitoring pipeline (pipeline a) is a commonly used pipeline in previous works [25, 74] and industry [23] which an object detector sends the cropped images to a later model for doing classification tasks like license plate detection or human recognition. Audio and question answering/sentiment analysis pipelines (pipelines b and c) are adapted from use cases composing multiple ML model types [28]. NLP pipelines (pipeline d and e) are representative examples of emerging use cases of language models [71, 72]. For full specification of the used models in each stage of the pipelines, refer to Appendix 8.

Baselines We compare IPA against variations of two similar systems, namely FA2 [59] and RIM [42]. FA2 is a recent system that achieves cost efficiency using scaling and batching, however, compared to IPA it does not have model switching as an optimization angle. RIM, on the other hand, does not have scaling as a configuration knob but uses model switching for adapting to dynamic workloads. The original RIM does not include batching; To have a fair comparison, we also add batching to RIM. As RIM does not support scaling therefore we statically set the scaling of each stage of the inference

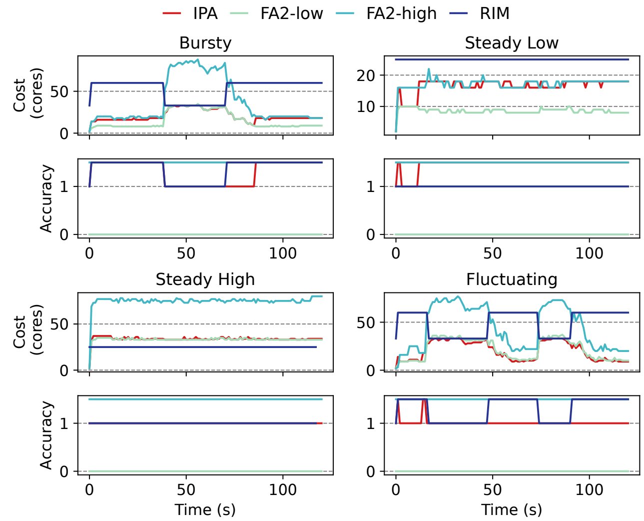


(a) Temporal analysis under different workloads

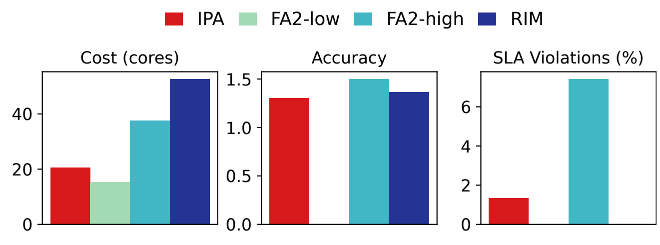


(b) Average analysis on bursty workload

Figure 8: Performance analysis of the Video pipeline



(a) Temporal analysis under different workloads



(b) Average analysis on bursty workload

Figure 9: Performance analysis of the Audio-qa pipeline

pipeline to a high value. Similarly, FA2 does not support model switching, therefore we use two versions of it, one FA2-low which sets the model variants to the lightest models, and FA2-high which sets the model variants to a heavy combination of models on each stage [1]. All three compared systems benefit from the LSTM predictor that was explained in Section 3.

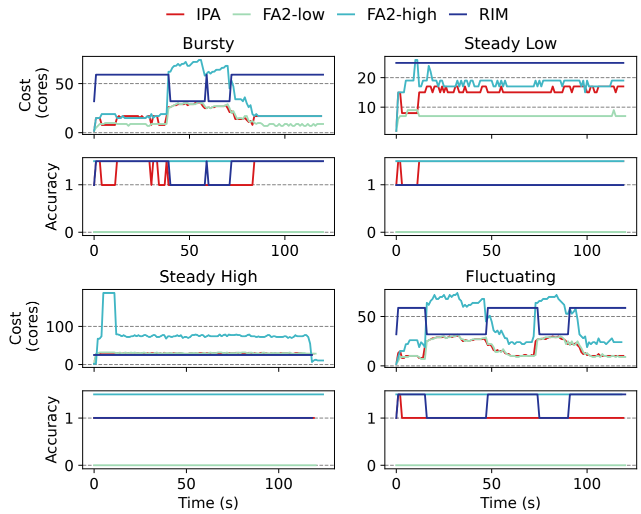
Workload Figure 7 shows excerpts from Twitter trace [15] that have been used for evaluating the performance of IPA against four parts of the dataset. It includes bursty, fluctuating, steady low, and steady high types of workloads. The LSTM predictor is able to predict the workload with a Symmetric Mean Absolute Percentage Error (SMAPE) [39] of 6.6% that is comparable to predictors used in systems with similar context [78]. Furthermore, an asynchronous load tester was implemented to emulate the behavior of users in real-world data-center.

SLA Table 6 shows the pipeline *SLA* of each inference pipeline that is calculated by summing the per stage *SLAs* heuristic that is explained in Section 4.2.

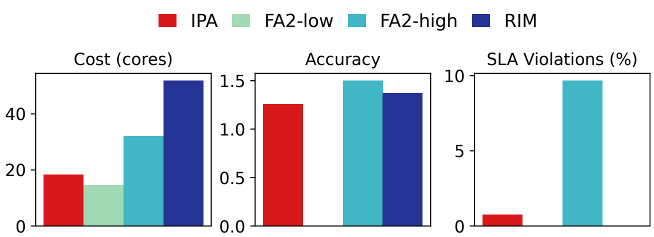
[1]Ideally we should have set the FA2-high to the heaviest models but due to resource limitations, we set it to models that on average give better accuracy compared to IPA

5.2 End-to-End Evaluation

Figure 8 shows the evaluation results for the video pipeline, on all four bursty, steady high, steady low and fluctuating workloads. Since FA2-high and FA2-low are always set to the lightest and heaviest variants they will always provide lowest and highest possible accuracies despite the load fluctuations. In the three bursty, steady low and fluctuating workloads IPA can always achieves a trade-off between the cost objective, in steady high workload IPA diverge to a configuration that uses the lowest cost model variants in order to adapt to the high resource demands of the steady high workload. The only available adaptation mechanism for RIM is changing the models, therefore under load variations in bursty and fluctuating workload it trades off accuracy for being responsive to the load bursts. As expected FA2-low and FA2-high have the highest and lowest *SLA* attainment. In video pipeline, IPA provides the same resource efficiency as FA2-low since the base allocation (explained in Section 4) of variants used for the first stage in video pipelines are similar in most cases and changing the model in favor of latency reduction does not result in higher computational costs. In total, IPA is able to show a better balance between the two cost and accuracy objectives. While both FA2-high and RIM provides the highest

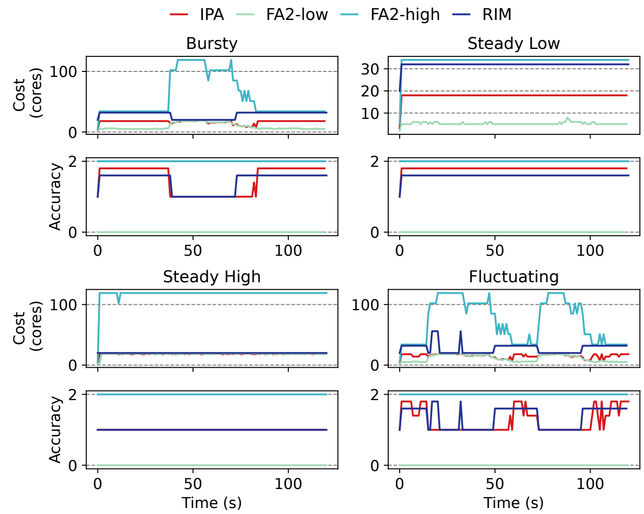


(a) Temporal analysis under different workloads

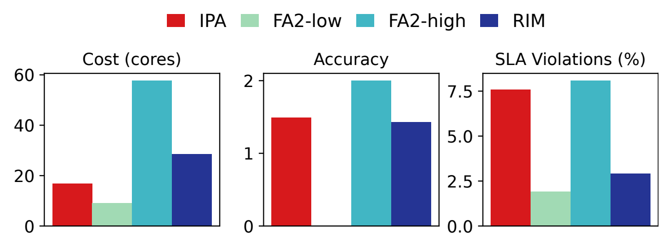


(b) Average analysis on bursty workload

Figure 10: Performance analysis of the Audio-sent pipeline



(a) Temporal analysis under different workloads



(b) Average analysis on bursty workload

Figure 11: Performance analysis of the Sum-qa pipeline

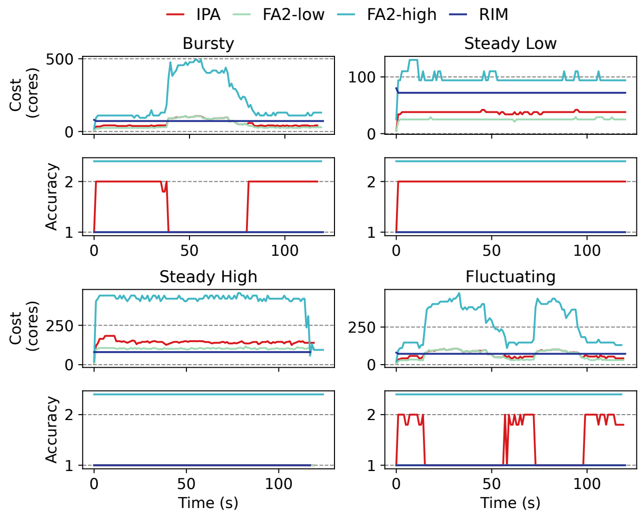
accuracies, their cost efficiency is compromised as a result of employing more accurate variants per stage and a high scaling factor. FA2-low is able to be responsive to *SLA* requirements and achieving the same cost efficiency as IPA but it is unable to improve the accuracy as it is fixed on the lightest variants. The higher violation rate in FA2-high, RIM, and IPA compared to FA2-low is due to the reason that the *SLAs* are defined using the processing latency of average models and the *SLAs* become tighter for more accurate models, resulting in a higher tail latency violation. Figure 9a and Figure10a show the same temporal and average results on the audio-qa and audio-sent inference pipelines. Due to lower number of used variants in these two pipelines ($5 \times 5 = 25$ for video and 5×2 and 5×3 on audio-qa and audio-sent pipelines) we observe less fluctuations in RIM in all the workloads. However, similar to video pipeline, IPA was able to achieve a trade-off between the two accuracy and cost objectives.

Compared to the stages used in the three mentioned pipelines, base allocations for the summarization stage used in the sum-qa and NLP pipelines provides a larger span of changes in terms of required CPU cores (see Appendix 8). For example, the resource difference between the heaviest and lightest model in the Object Detection stage of the video pipeline is $8 - 1 = 7$, while it is more than doubled in the summarization stage ($16 - 1 = 15$). Consequently, we observe a larger span of differences between FA2-low and FA2-high approaches in these two pipelines. In both of these approaches, IPA can adapt to the load by using the second least heavy models that result in 3x and 4x cost reduction with only 0.5 loss in the normalized accuracy measure.

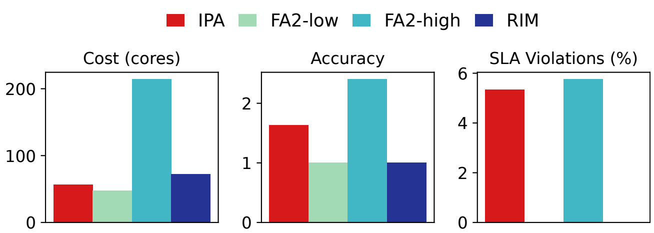
5.3 IPA Scalibility

System Scalibility To examine the effectiveness of the IPA in real-world systems we included the NLP pipeline in our evaluations which during bursts scales up to 500 cores (Figure 12a). Due to using production-grade best practices of ML deployment like using lightweight containers and Kubernetes as the backend with the benefit of distributed scheduling and cluster management, we believe IPA has the potential to scale to large clusters.

Optimizer Scalibility As mentioned in Section 4 we have used the Gurobi solver to solve the IP optimization problem. One critique of using Gurobi is its limitations in the solvable problem space in the time constraints of a real-world autoscaler. To guarantee fast autoscaler adaptation to workload fluctuations, the autoscaler should be able to find the next configuration in less than two seconds to leave enough



(a) Temporal analysis under different workloads



(b) Average analysis on bursty workload.

Figure 12: Performance analysis of the NLP pipeline.

room for the adaptation process itself which in our experiments were around the same number of 8 seconds sums up $8+2=10$ which we used as our adaptation monitoring interval. We ran a set of simulated experiments shown in Figure 13 to examine the decision-making time of the IPA growth with respect to changing the number of available model variants and the number of tasks in the inference pipelines (length of the inference graph). IPA is able to find the optimal configuration for inference pipelines with 10 stages each with 10 models in less than two seconds. Having an effective decision time for inference pipelines beyond these sizes demands faster optimization solutions. However, most of the existing inference pipelines [57] rarely go beyond 10 stages, therefore IPA will be effective for real-world use cases.

5.4 IPA Adaptability

The main premise of IPA is to provide an adaptable framework for achieving a trade-off between cost and accuracy objectives by leveraging the three configuration knobs of model switching, scaling, and batching. Instead of using a fixed value for α and β in previous experiments, we examined the effect of changing the given weights to each objective by modifying the α and β values for each of the inference

Figure 13: Decision time of Gurobi optimizer for IPA formulation with respect to the number of models and tasks on the inference graph.

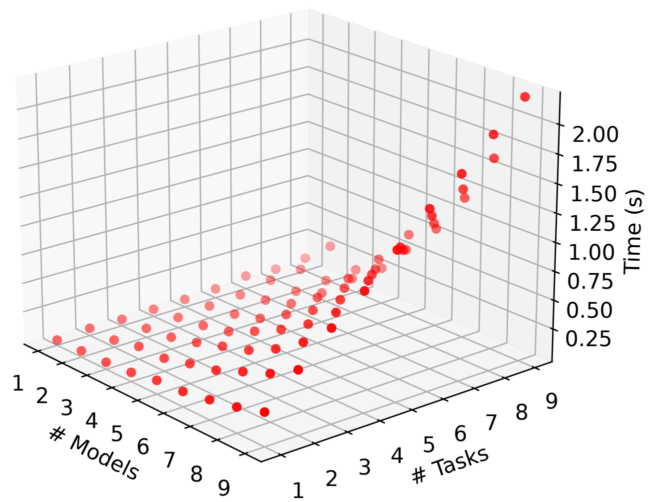
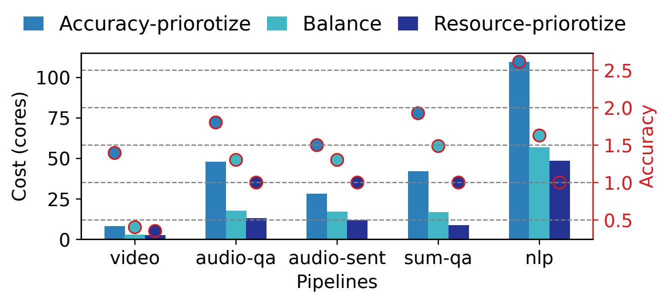


Figure 14: Comparison of IPA accuracy for different trade-offs between accuracy and cost objectives, IPA can navigate effectively between the two cost and accuracy objectives.



pipelines. Figure 14 shows a set of experiments conducted on all five pipelines where in one scenario cost optimization (resource prioritize) is set as the priority by setting the β to a larger value and in another scenario accuracy is set as the system priority by using a larger value for α. It is evident that IPA provides an adaptable approach to optimize different cost and accuracy preferences by the inference pipeline designer. For instance, one can choose a highly accuracy adaptation scenario for the NLP pipeline with 100 CPU cores and average accuracy of 2.5 or a lower accurate result of 1 with 46 CPU cores.

Figure 15 shows the latency CDF of end-to-end latencies over the five tested pipelines to further show the flexibility of IPA in dynamic workloads. IPA leverages its fast adaptation by using heavy models only when the load is low and achieves nearly the same latency efficiency as the FA2-low (with higher accuracy compared to FA2). Only RIM is able to provide

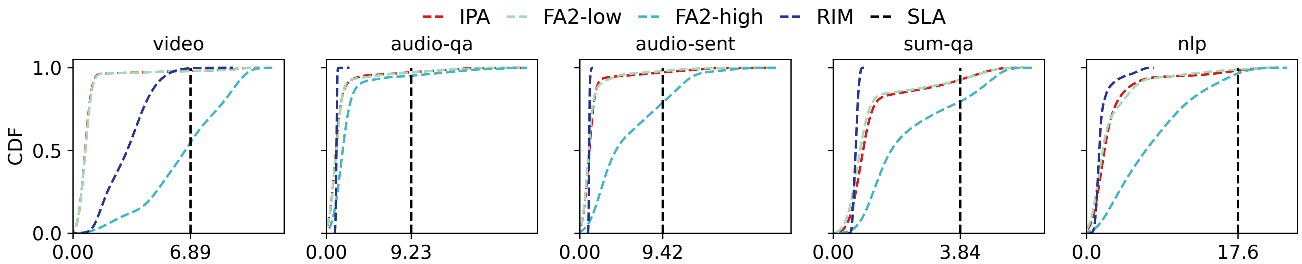
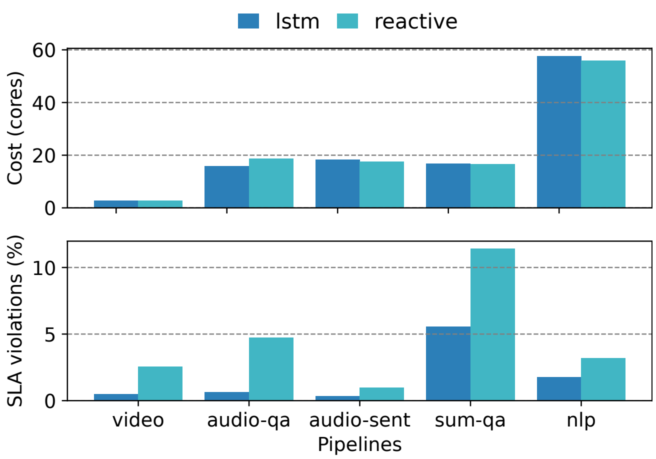


Figure 15: End-to-end latency distribution for the five tested inference pipelines under different approaches. IPA is able to achieve latency close to the FA2-low with light model variants and only RIM is achieving better latency at the expense of high resource over-provisioning.

Figure 16: Effect of using predictor on reducing SLA violations on bursty workload, IPA LSTM is able to reduce SLA violations up to 10x with the same resource usage



better latency compared to IPA but as shown in the previous examples (e.g., Figure 9b for Audio-qa pipeline) comes at the expense of high resource allocations (3x compared to IPA in the same pipeline).

5.5 IPA Predictor

Predictors are effective in reducing SLA violations. Most of the previous works on inference pipeline serving [25, 41, 42, 59] have done reactive auto-configuration. In reactive approaches, configuration changes happen with live monitoring of the load and in response to load changes. [73, 78] for predicting load prior to load changes. IPA uses a proactive approach by using an LSTM predictor that leverages historical data. The ablation analysis provided in Figures 16 shows that using the LSTM predictor is beneficial in reducing SLA violations in all the pipelines with negligible difference in resource consumption. The LSTM module is trained in less than ten minutes for the 14 days of Twitter traces, therefore using it is practical in real-world scenarios.

6 Related Works

Single stage inference serving: Several approaches in previous research have been proposed for improving the performance metrics without considering multiple stages inference models [13, 26, 34, 68, 73]. They intend to enhance a set of performance metrics, e.g., latency and throughput, and reduce resource utilization through adaptive batching, horizontal and vertical resource scaling, model switching, queue re-ordering, and efficient scheduling of models in heterogeneous clusters. A few works have considered the joint optimization of qualitative metrics like accuracy and performance for single-stage inference serving systems. Model Switching [75] proposes a quality adaptive framework for image processing applications. It uses switching between models trained for the same task configuration knob and switches from heavier models to lighter models in response to the load spikes. However, the proposed prototype does not consider the interaction of model switching between other resource configuration knobs like autoscaling and batching. INFaaS [60] abstracts away the selection of model variants in the single stage setting from the user and automatically selects the best-performing model within the user-defined SLOs. It also actively loads and unloads models based on their usage frequency. InfAdapter [63] and Cocktail [36] propose joint optimization formulations for maximizing accuracy and minimizing cost with predictive autoscaling in single stage inference scenarios.

Multi-stage inference serving: Several approaches in previous research have been proposed for improving the performance metrics for getting inference on multi-stage inference serving systems [7, 25, 37, 41, 42, 44, 47, 51, 58, 59, 61, 65, 69] since changing one model's configuration affects on the subsequent steps. InferLine [25] reduces the end-to-end latency of ML serving services by heuristically optimizing configurations such as batch sizes and horizontal scaling of each stage. Llama [61] is a use case-specific pipeline configuration system designed exclusively for video inference systems. It tries to reduce the end-to-end latency by interactively decreasing the latency of each stage in the pipeline. Stages of the pipeline can be either ML inference or non-ML video tasks like decoding. GrandSLAm [47] is a system designed to minimize latency and ensure compliance with service level agreement (SLA) requirements in the context of a chain of microservices dedicated to mainly machine learning (ML) tasks. The system achieves this by dynamically reordering incoming requests, prioritizing those with minimal computational overhead, and batching them to maximize each stage's throughput. FA2 [59] papooses a graph transformation and dynamic programming solution on inference pipelines with shared models. The graph transformation part breaks the execution graph to make it solvable in real-time, and the dynamic programming solution returns the optimal batch size and scaling factor per each DNN stage of the pipeline. Multi-model and Multi-task inference VR/AR/Metaverse pipelines [17,50] are other emerging use cases of inference pipelines. However, unlike IPA, none of the above approaches consider all three pillars of accuracy, cost, and end-to-end latency/throughput jointly for multi-stage inference serving systems.

7 Conclusion and Future Works

In this work, we introduced IPA, an online auto-configuration system for jointly improving the resource cost and accuracy over inference pipelines. IPA uses a combination of offline profiling with online optimization to find the appropriate model variant, replication factor, and batch sizes for each step of the inference pipeline. Real-world implementation of IPA and experiments using real-world traces showed that it could preserve the same cost efficiency and SLA agreement while also having normalized accuracy improvement up to 35% over two compared approaches. The followings are some directions for future works:

Scalability IPA leverages Gurobi solver for finding the suitable configurations. This worked fine in our problem setting as a limited number of model variants were used in each step of the pipeline. However, one interesting future direction for IPA is to examine its performance where more model variants are available for each step of the pipeline and also cases where we have more complicated larger graphs [7]. The adapter needs to be able to respond to bursts in less than one second, which demands either designing new heuristic methods that can find a good enough but not necessarily optimal solution or data-driven solutions like some of the methods that have been used before in similar auto-configuration context like Bayesian Optimization [14], Reinforcement [68] Learning or Causal Methods [43].

Emerging machine learning deployment paradigms Multi model serving [12] enables more efficient usage of GPUs. This feature enables loading several models simultaneously on the same server instead of spinning up one microservice per each model like IPA. While the focus of IPA was on CPU serving, using it on GPUs and containerized platforms is not straightforward. To our knowledge, there isn't any built-in mechanism for sharing GPU on mainstream container orchestration frameworks like Kubernetes. Making IPA formulation consistent with GPU sharing and also considering interference between multiple models in the scheduler [55] as part of the IPA is a potential future extension.

References

[1] AI Techniques in Medical Imaging May Lead to False Positives and False Negatives. https://tinyurl.com/628z9tn4. Published on May 12, 2020.

[2] GRPC. URL: https://grpc.io/.

[3] Istio. URL: https://istio.io/.

[4] Kubernetes Python client. https://github.com/kubernetes-client/python.

[5] MinIO. URL: https://min.io/.

[6] MLServer. URL: https://github.com/SeldonIO/MLServer.

[7] Nvidia DeepStream. URL: https://developer.nvidia.com/deepstream-sdk.

[8] prometheus. URL: https://prometheus.io/.

[9] Reduce False Positives with Machine Learning. https://complyadvantage.com/insights/reduce-false-positives-with-machine-learning/. Accessed on July 27, 2023.

[10] Seldon core. URL: https://github.com/SeldonIO/seldon-core.

[11] Ultralytics yolov5. URL: https://github.com/ultralytics/yolov5.

[12] Sherif Akoush, Andrei Paleyes, Arnaud Van Looveren, and Clive Cox. Desiderata for next generation of ML model serving. *arXiv preprint arXiv:2210.14665*, 2022.

[13] Ahsan Ali, Riccardo Pinciroli, Feng Yan, and Evgenia Smirni. Batch: Machine learning inference serving on serverless platforms with adaptive batching. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2020.

[14] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 469–482, 2017.

[15] archiveteam. Archiveteam-twitter-stream-2021-08. https://archive.org/details/archiveteam-twitter-stream-2021-08, 2021.

[16] Jeff Bar. Amazon EC2 ML inference. https://tinyurl.com/5n8yb5ub, Dec 2019.

[17] Giovanni Bartolomeo, Simon Bäurle, Nitinder Mohan, and Jörg Ott. Oakestra: An orchestration framework for edge computing. In *Proceedings of the SIGCOMM'22 Poster and Demo Sessions*, pages 34–36. 2022.

[18] Vivek M Bhasi, Jashwant Raj Gunasekaran, Prashanth Thinakaran, Cyan Subhra Mishra, Mahmut Taylan Kandemir, and Chita Das. Kraken: Adaptive container provisioning for deploying dynamic DAGs in serverless platforms. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 153–167, 2021.

[19] Stephen P Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[21] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL: https://arxiv.org/pdf/1908.09791.pdf.

[22] Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh. Serving heterogeneous machine learning models on multi-GPU servers with spatio-temporal sharing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 199–216, 2022.

[23] clarifai. Clarifai. URL: https://clarifai.com/clarifai/main/workflows.

[24] CNBC. Tesla crash that killed two men. https://tinyurl.com/mr25a5mv. Published on April 18, 2021.

[25] Daniel Crankshaw, Gur-Eyal Sela, Xiangxi Mo, Corey Zumar, Ion Stoica, Joseph Gonzalez, and Alexey Tumanov. InferLine: Latency-aware provisioning and scaling for prediction serving pipelines. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, pages 477–491, 2020.

[26] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J Franklin, Joseph E Gonzalez, and Ion Stoica. Clipper: A low-latency online prediction serving system. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 613–627, 2017.

[27] Tathagata Das, Yuan Zhong, Ion Stoica, and Scott Shenker. Adaptive stream processing using dynamic batch sizing. In *ACM Symposium on Cloud Computing (SoCC)*, pages 1–13, 2014.

[28] Ruofei Du, Na Li, Jing Jin, Michelle Carney, Scott Miles, Maria Kleiner, Xiuxiu Yuan, Yinda Zhang, Anuva Kulkarni, Xingyu Liu, et al. Rapsai: Accelerating machine learning prototyping of multimedia applications through visual programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–23, 2023.

[29] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. A guide to deep learning in healthcare. *Nature medicine*, 25(1):24–29, 2019.

[30] Maria Fazio, Antonio Celesti, Rajiv Ranjan, Chang Liu, Lydia Chen, and Massimo Villari. Open issues in scheduling microservices in the cloud. *IEEE Cloud Computing*, 3(5):81–88, 2016.

[31] Avrilia Floratou, Ashvin Agrawal, Bill Graham, Sriram Rao, and Karthik Ramasamy. Dhalion: Self-regulating stream processing in Heron. *Very Large Data Bases (PVLDB)*, 10(12):1825–1836, 2017.

[32] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021.

[33] Alim Ul Gias, Giuliano Casale, and Murray Woodside. Atom: Model-driven autoscaling for microservices. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 1994–2004, 2019.

[34] Arpan Gujarati, Sameh Elnikety, Yuxiong He, Kathryn S McKinley, and Björn B Brandenburg. Swayam: Distributed autoscaling to meet SLAs of machine learning inference services with resource efficiency. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, pages 109–120, 2017.

[35] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. Serving DNNs like clockwork: Performance predictability from the bottom up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 443–462. USENIX Association, November 2020. URL: https://www.usenix.org/conference/osdi20/presentation/gujarati.

[36] Jashwant Raj Gunasekaran, Cyan Subhra Mishra, Prashanth Thinakaran, Bikash Sharma, Mahmut Taylan Kandemir, and Chita R Das. Cocktail: A multidimensional optimization for model serving in cloud. In *USENIX NSDI*, pages 1041–1057, 2022.

[37] Udit Gupta, Samuel Hsia, Jeff Zhang, Mark Wilkening, Javin Pombra, Hsien-Hsin Sean Lee, Gu-Yeon Wei, Carole-Jean Wu, and David Brooks. RecPipe: Co-designing models and hardware to jointly optimize recommendation quality and performance. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 870–884, 2021.

[38] Harvard Business Review. When machine learning goes off the rails. https://hbr.org/2021/01/when-machine-learning-goes-off-the-rails, January 2021.

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[40] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[41] Yitao Hu, Rajrup Ghosh, and Ramesh Govindan. Scrooge: A cost-effective deep learning inference system. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 624–638, 2021.

[42] Yitao Hu, Weiwu Pang, Xiaochen Liu, Rajrup Ghosh, Bongjun Ko, Wei-Han Lee, and Ramesh Govindan. Rim: Offloading inference to the edge. In *Proceedings of the International Conference on Internet-of-Things Design and Implementation*, pages 80–92, 2021.

[43] Md Shahriar Iqbal, Rahul Krishna, Mohammad Ali Javidian, Baishakhi Ray, and Pooyan Jamshidi. Unicorn: Reasoning about configurable system performance through the lens of causality. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 199–217, 2022.

[44] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. Chameleon: Scalable adaptation of video analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 253–266, 2018.

[45] Maree Johnson, Samuel Lapkin, Vanessa Long, Paula Sanchez, Hanna Suominen, Jim Basilakis, and Linda Dawson. A systematic review of speech recognition technology in health care. *BMC medical informatics and decision making*, 14(1):1–14, 2014.

[46] Vasiliki Kalavri, John Liagouris, Moritz Hoffmann, Desislava Dimitrova, Matthew Forshaw, and Timothy Roscoe. Three steps is all you need: Fast, accurate, automatic scaling decisions for distributed streaming dataflows. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 783–798, 2018.

[47] Ram Srivatsa Kannan, Lavanya Subramanian, Ashwin Raju, Jeongseob Ahn, Jason Mars, and Lingjia Tang. Grandslam: Guaranteeing SLAs for jobs in microservices execution frameworks. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pages 1–16, 2019.

[48] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbach, Alex Rocha, and Joe Stubbs. Lessons learned from the Chameleon testbed. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association, July 2020.

[49] Darpan Kulshreshtha. 10 Instances Where AI Went Wrong. https://tinyurl.com/2p8ywtpd. Published on LinkedIn.

[50] Hyoukjun Kwon, Krishnakumar Nair, Jamin Seo, Jason Yik, Debabrata Mohapatra, Dongyuan Zhan, Jinook Song, Peter Capak, Peizhao Zhang, Peter Vajda, et al. XRBench: An extended reality (XR) machine learning benchmark suite for the metaverse. *arXiv preprint arXiv:2211.08675*, 2022.

[51] Yunseong Lee, Alberto Scolari, Byung-Gon Chun, Marco Domenico Santambrogio, Markus Weimer, and Matteo Interlandi. PRETZEL: Opening the black box of machine learning prediction serving systems. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 611–626, 2018.

[52] Nima Mahmoudi and Hamzeh Khazaei. Performance modeling of serverless computing platforms. *IEEE Transactions on Cloud Computing*, pages 1–15, 2020.

[53] Ankur Mallick, Kevin Hsieh, Behnaz Arzani, and Gauri Joshi. Matchmaker: Data drift mitigation in machine learning for large-scale systems. In D. Marculescu, Y. Chi, and C. Wu, editors, *Proceedings of Machine Learning and Systems*, volume 4, pages 77–94, 2022. URL: https://proceedings.mlsys.org/paper_files/paper/2022/file/1c383cd30b7c298ab50293adfecb7b18-Paper.pdf.

[54] Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of torch. In *Proceedings of*

the 18th ACM international conference on Multimedia, pages 1485–1488, 2010.

[55] Daniel Mendoza, Francisco Romero, Qian Li, Neeraja J Yadwadkar, and Christos Kozyrakis. Interference-aware scheduling for inference serving. In *Proceedings of the 1st Workshop on Machine Learning and Systems*, pages 80–88, 2021.

[56] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091*, 2019.

[57] Nvidia. https://docs.nvidia.com/metropolis/deepstream/dev-guide/text/DS_Zero_Coding_Sample_Graphs.html#. Deepstream reference graphs.

[58] Alex Poms, Will Crichton, Pat Hanrahan, and Kayvon Fatahalian. Scanner: Efficient video analysis at scale. *ACM Trans. Graph.*, 37(4):138:1–138:13, July 2018. URL: http://doi.acm.org/10.1145/3197517.3201394, https://doi.org/10.1145/3197517.3201394.

[59] Kamran Razavi, Manisha Luthra, Boris Koldehofe, Max Mühlhäuser, and Lin Wang. FA2: Fast, accurate autoscaling for serving deep learning inference with SLA guarantees. In *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 146–159. IEEE, 2022.

[60] Francisco Romero, Qian Li, Neeraja J Yadwadkar, and Christos Kozyrakis. INFaaS: Automated model-less inference serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 397–411, 2021.

[61] Francisco Romero, Mark Zhao, Neeraja J Yadwadkar, and Christos Kozyrakis. Llama: A heterogeneous & serverless framework for auto-tuning video analytics pipelines. *arXiv preprint arXiv:2102.01887*, 2021.

[62] Krzysztof Rzadca, Pawel Findeisen, Jacek Swiderski, Przemyslaw Zych, Przemyslaw Broniek, Jarek Kusmierek, Pawel Nowak, Beata Strack, Piotr Witusowski, Steven Hand, et al. Autopilot: Workload autoscaling at Google. In *Proceedings of the Fifteenth European Conference on Computer Systems*, pages 1–16, 2020.

[63] Mehran Salmani, Saeid Ghafouri, Alireza Sanaee, Kamran Razavi, Max Mühlhäuser, Joseph Doyle, Pooyan Jamshidi, and Mohsen Sharifi. Reconciling high accuracy, cost-efficiency, and low latency of inference serving systems. In *Proceedings of the 3rd Workshop on Machine Learning and Systems*, pages 78–86, 2023.

[64] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. Nexus: A GPU cluster engine for accelerating DNN-based video analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 322–337, 2019.

[65] Vikram Sreekanti, Harikaran Subbaraj, Chenggang Wu, Joseph E Gonzalez, and Joseph M Hellerstein. Optimizing prediction serving on low-latency serverless dataflow. *arXiv preprint arXiv:2007.05832*, 2020.

[66] Vikram Sreekanti, Chenggang Wu, Xiayue Charles Lin, Johann Schleier-Smith, Joseph E. Gonzalez, Joseph M. Hellerstein, and Alexey Tumanov. Cloudburst: Stateful functions-as-a-service. *Very Large Data Bases (PVLDB)*, 13(12):2438–2452, July 2020. https://doi.org/10.14778/3407790.3407836.

[67] NVIDIA TensorRT. Programmable inference accelerator, 2018.

[68] Luping Wang, Lingyun Yang, Yinghao Yu, Wei Wang, Bo Li, Xianchao Sun, Jian He, and Liping Zhang. Morphling: Fast, near-optimal auto-configuration for cloud-native model serving. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 639–653, 2021.

[69] Wei Wang, Sheng Wang, Jinyang Gao, Meihui Zhang, Gang Chen, Teck Khim Ng, and Beng Chin Ooi. Rafiki: Machine learning as an analytics service system. *arXiv preprint arXiv:1804.06087*, 2018.

[70] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. MLaaS in the wild: Workload analysis and scheduling in large-scale heterogeneous GPU clusters. In *19th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 22)*, 2022.

[71] Tongshuang Wu, Ellen Jiang, Aaron Donsbach, Jeff Gray, Alejandra Molina, Michael Terry, and Carrie J Cai. Promptchainer: Chaining large language model prompts through visual programming. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, pages 1–10, 2022.

[72] Tongshuang Wu, Michael Terry, and Carrie Jun Cai. Ai chains: Transparent and controllable human-ai interaction by chaining large language model prompts. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, pages 1–22, 2022.

[73] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. Mark: Exploiting cloud services for cost-effective, SLO-aware machine learning inference serving. In *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, pages 1049–1062, 2019.

[74] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J Freedman. Live video analytics at scale with approximation and delay-tolerance. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 377–392, 2017.

[75] Jeff Zhang, Sameh Elnikety, Shuayb Zarar, Atul Gupta, and Siddharth Garg. Model-switching: Dealing with fluctuating workloads in machine-learning-as-a-service systems. In *12th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 20)*, 2020.

[76] Tianlei Zheng, Xi Zheng, Yuqun Zhang, Yao Deng, ErXi Dong, Rui Zhang, and Xiao Liu. SmartVM: a SLA-aware microservice deployment framework. *World Wide Web*, 22(1):275–293, 2019.

[77] Hao Zhou, Ming Chen, Qian Lin, Yong Wang, Xiaobin She, Sifan Liu, Rui Gu, Beng Chin Ooi, and Junfeng Yang. Overload control for scaling WeChat microservices. In *ACM Symposium on Cloud Computing (SoCC)*, pages 149–161, 2018.

[78] Zhuangzhuang Zhou, Yanqi Zhang, and Christina Delimitrou. Aquatope: Qos-and-uncertainty-aware resource management for multi-stage serverless workflows. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pages 1–14, 2022.

8 Appendix

8.1 Pipelines Stages Specifications

List of used models per each stage of the pipeline with their specification.

Object Detection

Performance Measure: Mean Average Precision (mAP)
Number of Variants: Five
Source: Ultralytics YoloV5 [11]
Threshold: 4 RPS

Table 7: Object Detection Task Models

Model	Params (M)	Base Allocation	mAP
YOLOv5n	1.9	1	45.7
YOLOv5s	7.2	1	56.8
YOLOv5m	21.2	2	64.1
YOLOv5l	46.5	4	67.3
YOLOv5x	86.7	8	68.9

Object Classification

Performance Measure: Accuracy
Number of Variants: 5
Source: Torchvision [54]
Threshold: 4 RPS

Table 8: Object Classification Task Models

Model	Params (M)	Base Allocation	Accuracy
ResNet18	11.7	1	69.75
ResNet34	21.8	1	73.31
ResNet50	25.5	1	76.13
ResNet101	44.54	1	77.37
ResNet52	60.2	2	78.31

Audio

Performance Measure: Word Error Rate (WER)
Number of Variants: Five
Source: HuggingFace
HuggingFace Source: facebook
Threshold: 1 RPS

Table 9: Audio Task Models

Model	Params (M)	Base Allocation	WER
s2t-small-librispeech	29.5	1	41.28
s2t-medium-librispeech	71.2	2	35.12
wav2vec2-base	94.4	2	33.85
s2t-large-librispeech	267.8	4	33.26
wav2vec2-large	315.5	8	27.65

Question Answering

Performance Measure: F1 Score

Number of Variants: Two
Source: HuggingFace
HuggingFace Source: depeest
Threshold: 1 RPS

Table 10: Question Answering Task Models

Model	Params (M)	Base Allocation	F1 Score
roberta-base	277.45	1	77.14
roberta-large	558.8	1	83.79

Summarisation
Performance Measure: Recall-Oriented Understudy for Gisting Evaluation (ROUGE-L)
Number of Variants: Six
Source: HuggingFace
HuggingFace Source: sshleifer
Threshold: 5 RPS

Table 11: Summarisation Task Models

Model	Params (M)	Base Allocation	ROUGE-L
distilbart-1-1	82.9	1	32.26
distilbart-12-1	221.5	2	33.37
distilbart-6-6	229.9	4	35.73
distilbart-12-3	255.1	8	36.39
distilbart-9-6	267.7	8	36.61
distilbart-12-6	305.5	16	36.99

Sentiment Analysis
Performance Measure: Accuracy
Number of Variants: Three
Source: HuggingFace
HuggingFace Source: Souvikcmsa
Threshold: 1 RPS

Table 12: Sentiment Analysis Task Models

Model	Params (M)	Base Allocation	Accuracy
DistillBerT	66.9	1	79.6
Bert	109.4	1	79.9
Roberta	355.3	1	83

Language Identification Task Models
Performance Measure: Accuracy
Number of Variants: One
Source: HuggingFace
HuggingFace Source: dinalzein
Threshold: 4 RPS

Neural Machine Translation
Performance Measure: Bilingual Evaluation Understudy (BELU)
Number of Variants: Two

Table 13: Language Identification Task Models

Model	Params (M)	Base Allocation	Accuracy
roberta-base-finetuned	278	1	79.62

Source: HuggingFace
HuggingFace Source: Helsinki-NLP
Threshold: 4 RPS

Table 14: Neural Machine Translation Task Models

Model	Params (M)	Base Allocation	Accuracy
opus-mt-fr-en	74.6	4	33.1
opus-mt-tc-big-fr-en	230.6	8	34.4