SneakPeek: Data-Aware Model Selection and Scheduling for Inference Serving on the Edge

Joel Wolfrath
University of Minnesota
Minneapolis, MN, USA
wolfr046@umn.edu

Daniel Frink
University of Minnesota
Minneapolis, MN, USA
frink021@umn.edu

Abhishek Chandra
University of Minnesota
Minneapolis, MN, USA
chandra@umn.edu

ABSTRACT

Modern applications increasingly rely on inference serving systems to provide low-latency insights with a diverse set of machine learning models. Existing systems often utilize resource elasticity to scale with demand. However, many applications cannot rely on hardware scaling when deployed at the edge or other resource-constrained environments. In this work, we propose a model selection and scheduling algorithm that implements accuracy scaling to increase efficiency for these more constrained deployments. We show that existing schedulers that make decisions using profiled model accuracy are biased toward the label distribution present in the test dataset. To address this problem, we propose using ML models– which we call SneakPeek models– to dynamically adjust estimates of model accuracy, based on the underlying data. Furthermore, we greedily incorporate inference batching into scheduling decisions to improve throughput and avoid the overhead of swapping models in and out of GPU memory. Our approach employs a new notion of request priority, which navigates the trade-off between attaining high accuracy and satisfying deadlines. Using data and models from three real-world applications, we show that our proposed approaches result in higher-utility schedules and higher accuracy inferences in these hardware-constrained environments.

CCS CONCEPTS

• **Mathematics of computing** → Bayesian computation; • **Information systems** → **Data analytics**; • **Computing methodologies** → *Supervised learning*.

KEYWORDS

Inference Serving, Edge Computing, Model Selection, Scheduling

ACM Reference Format:
Joel Wolfrath, Daniel Frink, and Abhishek Chandra. 2025. SneakPeek: Data-Aware Model Selection and Scheduling for Inference Serving on the Edge. In *ACM Symposium on Cloud Computing (SoCC '25), November 19–21, 2025, Online, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3772052.3772217

SoCC '25, November 19–21, 2025, Online, USA

ACM ISBN 979-8-4007-2276-9/25/11
https://doi.org/10.1145/3772052.3772217

1 INTRODUCTION

The widespread adoption of machine learning across diverse application domains has created a pressing need for efficient and scalable inference serving systems. Inference serving systems expose APIs that allow client applications—such as mobile apps, web services, or enterprise platforms—to submit inference requests to pre-trained models, often at massive scale, with some deployments handling billions of requests daily [26]. These systems support critical applications such as healthcare monitoring [27], language translation [26], recommendation systems [32], and mobile services [15], where timely and accurate responses are essential. To alleviate the burden on developers and end-users of manually navigating complex accuracy-latency trade-offs, *model-less inference serving* has emerged as a paradigm in which the system itself dynamically selects the most suitable model for each request, based on contextual factors such as latency constraints, accuracy requirements, and system load. This model selection paradigm is common in modern inference serving frameworks [1, 40, 46, 54], which often host multiple models with varying accuracy-latency profiles. A key challenge in these systems is selecting the most appropriate model variant to serve each request—a decision that must be made in conjunction with scheduling policies to ensure efficient resource utilization and responsiveness.

While existing inference serving systems are often deployed in the public cloud, inference serving is also critical for applications that run in more constrained environments, such as edge clusters or private clouds. Local deployments, such as those found in healthcare facilities or smart cities, are often necessary to avoid the increased cost, latency, and privacy concerns associated with streaming data over wide-area networks to a cloud setting. However, there are several challenges associated with efficient inference serving in these local deployments. First, such inference serving systems must service requests for *multiple applications*, each with their own set of data streams which may require real-time processing. These applications also have unique service level objectives (SLOs), including low-latency deadlines which are critical to satisfy, especially for real-time applications [54, 56]. Next, local deployments often have resource constraints (e.g. limited to a single GPU) and lack the resource elasticity of the cloud, which prevents systems from using autoscaling to meet demand. Finally, deep neural networks (DNNs) are a common model choice due to their high accuracy, but they can be computationally intensive. An inference serving system may offer a collection of multiple DNN models for each application to choose from, with each model presenting a different latency-accuracy tradeoff (e.g., some models may be lightweight but less accurate, while others may have higher accuracy but take longer to execute) [54]. Thus, selecting and scheduling the right

model for each request in a multi-application, constrained resource environment is challenging.

Existing work focuses mainly on inference serving in public clouds or other settings that support hardware scaling [4, 16, 23, 40, 45, 46, 55]. A few works have explored *accuracy scaling* as a strategy for inference serving systems operating under fixed resource constraints [1, 2, 41, 54]. When resource elasticity is not available, systems must find alternative ways to manage fluctuating query loads. Accuracy scaling addresses this by profiling the accuracy of available models and dynamically adjusting the trade-off between accuracy and throughput. This allows the system to remain responsive by selectively deploying faster, less accurate models during periods of high demand. This approach works in practice, but existing designs are suboptimal for two reasons: (1) they are data-oblivious: profiled model accuracy is biased toward the test dataset and may not reflect the distribution of the out-of-sample data, thus reducing the quality of the associated scheduling decisions; and (2) they perform model selection for each inference request in isolation. As a result, inference batching—a key optimization for improving throughput—is only applied opportunistically when requests happen to align in time and model usage. Without explicit coordination in the scheduling process to group compatible requests, batching opportunities are missed, leading to increased model loading overhead and reduced scheduling efficiency.

Existing systems that rely on profiled model accuracy make suboptimal model selection decisions when properties of the data are ignored. We propose leveraging *data-awareness* to understand how each model will perform over the actual data being processed in real-time. In this framework, we treat class frequencies as parameters, which are dynamically estimated based on the data. These parameters are then used to inform model selection decisions by providing sharper estimates of model accuracy. We propose a joint model selection and scheduling algorithm for hardware-constrained, multi-application inference serving. Our algorithm uses a new notion of request priority based on model accuracy and request deadlines, which attempts to maximize inference accuracy while minimizing deadline violations. Furthermore, our approach greedily incorporates inference batching (or request grouping) into the resulting schedules. This grouping provides the scheduler with a more global view of request dependencies, while reducing overheads and model swap latency in and out of GPU memory.

Contributions. We make the following research contributions:

- We formulate an optimization framework for joint model selection and request scheduling which implements accuracy scaling for hardware-constrained environments.
- We present our data-awareness mechanism, *SneakPeek*, which sharpens accuracy estimates in real-time based on the underlying data and is easily incorporated into existing schedulers.
- We propose heuristics for efficiently performing model selection and scheduling in practice. These heuristics employ a new notion of priority as well as inference batching (grouping).
- We show that our proposed data-awareness techniques can also be incorporated into other state-of-the-art schedulers.
- We thoroughly evaluate our proposed approach using three, real-world applications and a variety of model types. Our methods achieve up to a 2x increase in *utility*, a metric that combines model accuracy with a penalty for any missed deadlines.

2 PRELIMINARIES

2.1 Motivating Applications

Edge analytics is critical for smart city deployments [3], which process data from a wide array of sensors and cameras to optimize traffic flow, manage waste, enhance public safety, and support urban planning. In the realm of surveillance, video analytics improves security by automatically identifying unusual behaviors or specific actions. This capability enables systems to alert security teams about potential threats, ensuring rapid, proactive responses [43]. Furthermore, modern surveillance extends beyond video alone, and requires capturing data and making real-time decisions in various sectors such as retail, transportation, and service industries.

Hospitals, assisted living facilities, and in-home care systems also collect and process real-time data from a variety of sources (such as wearable sensors, monitors, video cameras, etc.), which can be used to improve patient care and assist medical staff [12, 25]. Many of these tasks require low-latency inference, since the results directly affect patient well-being. For example, care facilities may monitor patient movement or gait, which could alert medical staff if an elderly patient is wavering or falling over. There are several other applications in this domain which leverage inference serving, including arrhythmia detection [6], seizure forecasting [44, 49], and respiratory compromise [8, 19]. Since offloading data and tasks to a remote cloud is not an option due to latency, cost, and privacy concerns, many of these care facilities are equipped with edge clusters that have limited resource capacity and elasticity. Real-time inferences are generated by streaming video frames or data from wearable sensors to the edge cluster and issuing recurring queries to monitor for critical events. We use healthcare informatics as a running example in this work.

2.2 System Model

In our framework, the system is tasked with selecting a model variant for each request and scheduling it for execution. Users wishing to access the inference-serving APIs first register their application with the system. This registration process includes providing metadata for the application, including which data streams it should process and any service level objectives (SLOs) which specify deadlines for each query type. The application owner also uploads pre-trained ML models to be used for inference along with a testing or validation dataset for the application. The system then profiles the specified models on the target hardware to obtain estimates of inference latency, model accuracy, GPU memory usage, and latency for loading the model into GPU memory.

3 PROBLEM STATEMENT AND SYSTEM ARCHITECTURE

3.1 Problem Statement

Our system is tasked with providing inference serving in a hardware-constrained, multi-application setting. In this paper, we focus on execution on a single GPU worker, but our techniques can be extended to multi-worker settings, as discussed in section 8. Each

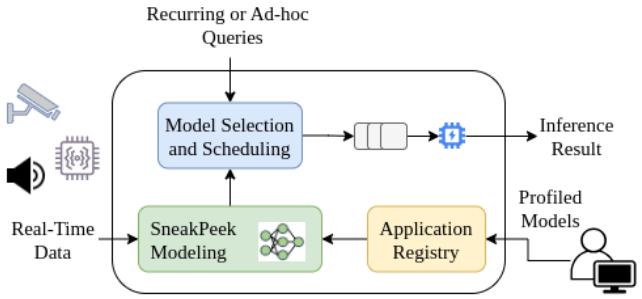


Figure 1: Our system model for inference serving includes a data-aware component – SneakPeek – which provides input to the model selection and scheduling algorithms.

application registers one or more classification models[1], which the system can use to satisfy inference requests. For a given set of requests, our task is to *assign a model* to each request and *schedule* the associated inference tasks for execution. More formally, let $\mathcal{R}$ be our set of requests, where each request $r_i \in \mathcal{R}$ has an associated deadline of d_i. Each request also belongs to a target application, which we denote a_i. We define $\mathcal{M}_a$ to be the set of registered model variants for application a.

Our objective is to find a schedule $\mathcal{S}$ that maximizes the average *utility* for all requests. The utility of a request (defined below) represents the expected inference accuracy, which degrades if a deadline is missed. The schedule can be represented by a set of nonnegative integers, denoted s_{ij}, mapping requests to models. Here, a nonzero value of s_{ij} indicates that request r_i is assigned to model $m_j \in \mathcal{M}_{a_i}$ (for $i \in 1..|\mathcal{R}|$ and $j \in 1..|\mathcal{M}_{a_i}|$). If s_{ij} is positive, the integer value represents the execution order, relative to the other requests (with lower values of s_{ij} being executed first, the first request being assigned an s_{ij} value of 1). We also define the execution start time t_i for a given request r_i (omitting the dependence on $\mathcal{S}$ for brevity):

$$t_i = \sum_{j=1}^{|\mathcal{M}_{a_i}|} \sum_{h=1}^{|\mathcal{R}|} \sum_{l=1}^{|\mathcal{M}_{a_h}|} \mathbb{1}_{s_{hl}<s_{ij}} \, \ell(m_l) \tag{1}$$

where $\ell(m_l)$ is the (profiled) inference latency for model m_l, *including any context switch time required to swap the model variant into GPU memory*. This works by finding all requests that are ordered ahead of request r_i and summing their expected model latencies.

We now define the utility for a given request r_i:

$$u_{a_i}(m_j, d_i, t_i) = Accuracy(m_j)[1 - \gamma_{a_i}(d_i, t_i + \ell(m_j))] \tag{2}$$

where $\gamma_{a_i}(d_i, e_i) \geq 0$ is a monotonically increasing penalty function, which generates positive values when a deadline is expected to miss (i.e. when the deadline d_i is less than the expected request completion time e_i). Some examples include a sigmoid function, linear penalty, or step function, e.g. $\gamma_{a_i}(d_i, e_i) = \mathbb{1}_{d_i<e_i}$. If all the deadlines are met, the utility is just the profiled accuracy[2] of the selected model. These utility (and penalty) functions are defined at the application level. Even applications which require high accuracy (e.g. in a healthcare setting), can define a utility function in a way which satisfies their requirements. For example, if we define our penalty function to be a constant, the optimization will strictly maximize accuracy. We now define our optimization problem for scheduling and model selection, which seeks to maximize the average utility across all requests:

$$\max_{\mathcal{S}} \quad \frac{1}{|\mathcal{R}|} \sum_{i=1}^{|\mathcal{R}|} \sum_{j=1}^{|\mathcal{M}_{a_i}|} \mathbb{1}_{s_{ij}>0} \, u_{a_i}(m_j, d_i, t_i) \tag{3}$$

$$\text{s.t.} \quad s_{ij} \in \mathbb{Z}_0^+ \tag{4}$$

$$\sum_{j=1}^{|\mathcal{M}_{a_i}|} \mathbb{1}_{s_{ij}>0} = 1, \quad \forall i \tag{5}$$

$$s_{hl} \neq s_{ij}, \quad \forall h, l \quad \text{s.t.} \quad h \neq i, \; s_{hl} > 0, \; s_{ij} > 0 \tag{6}$$

Constraint (4) forces the s_{ij} terms to be non-negative integers. A non-zero value indicates request r_i is assigned model $m_j \in \mathcal{M}_{a_i}$. The integer value of s_{ij} indicates its order in the execution sequence. Constraint (5) ensures that each request gets assigned exactly one model. Constraint (6) ensures that requests have distinct integers for ordering.

We can show that specific instantiations of the optimization problem in equation 3 are NP hard, e.g. by excluding model selection from the optimization. More specifically; let $|\mathcal{M}_{a_i}| = 1$ for all applications and define $Accuracy(m_j) = 1$ for all $m_j \in \bigcup_a \mathcal{M}_a$. In addition, define the penalty function to be a step function, i.e. $\gamma_{a_i}(d_i, e_i) = \mathbb{1}_{d_i<e_i}$. Then we are left with an optimization problem that minimizes a unit penalty for jobs with *sequence independent startup times* over an arbitrary number of application families. This is a known NP-hard problem [7, 9], even without considering more complicated choices for $\mathcal{M}$. Evaluating all candidate solutions is extremely expensive, with the number of solutions for n requests (for a single application) totaling $n!|\mathcal{M}_{a_i}|^n$. Computing an exact solution may be feasible when the number of requests and/or models is very small. However, in many practical settings, we require an approximate solution that can be found quickly.

3.2 System Architecture

In our proposed system (figure 1) application owners register their applications with the system and upload model variants to service their requests. We assume model profiles are specified, which include accuracy measurements *for every possible target label*, along with profiled latency measurements for the GPU.

After an application is registered, real-time data is then streamed to our SneakPeek module (section 4) which is a distinct process for asynchronous data staging, preprocessing, and sharpening accuracy estimates via data-awareness. Inbound requests are enqueued during a scheduling window, then scheduled for inference. The scheduler obtains metadata from the SneakPeek module, then assigns models and produces a schedule for the provided requests (section 5). These requests are then dispatched to the worker queue. The worker loads the data and model variant required to service the request and generates the inference result.

[1]All of our approaches can be adapted to support other modeling types (e.g. regression) but we focus on classification to simplify the presentation.
[2]Note: Some implementations may prefer to normalize the accuracy values across applications, to ensure fairness.

4 SNEAKPEEK: INCORPORATING DATA-AWARENESS

4.1 Dependence on Model Accuracy

Hardware-constrained scheduling algorithms in the literature often utilize accuracy scaling, *which requires high-quality estimates of model accuracy for correctness.* If the accuracy estimates for each model variant are poor, the resulting schedules will be suboptimal. Existing schedulers are often *data-oblivious* and rely on a single, profiled estimate of model accuracy for making decisions. Relying on a single summary statistic to make decisions can cause schedulers (including exact solvers) to produce suboptimal solutions.

Model accuracy is not a static quantity; it often varies across target classes in a dataset [54]. For example, consider the task of performing human action recognition over video frames. Some actions, such as walking and sitting, are easier for a model to distinguish than others, such as loitering or talking on the phone. This makes the profiled model accuracy very sensitive to the frequency of each class in the test data set. Therefore, it is an (often unstated) requirement that the profiled test set match the distribution of classes that will be observed out-of-sample. However, even if a practitioner can match the distribution exactly, using a static accuracy value ignores the heterogeneity present in the data, which can be used to obtain better accuracy values for each inference request.

We can also observe this phenomenon by examining model accuracy analytically. In the context of multi-class classification with a set of class labels c, the profiled accuracy of a model m is computed as [21]:

$$\text{Accuracy}(m) = \frac{\text{tr}(Z)}{\sum_{i=1}^{|c|}\sum_{j=1}^{|c|} z_{ij}} \tag{7}$$

where $Z = [z_{ij}]_{1\le i,j\le |c|}$ is the confusion matrix generated by evaluating the model on a test data set. We can rewrite this expression as:

$$\text{Accuracy}(m) = \sum_{i=1}^{|c|}\left(\frac{\sum_{j=1}^{|c|} z_{ij}}{\sum_{j=1}^{|c|}\sum_{k=1}^{|c|} z_{jk}}\right)\frac{z_{ii}}{\sum_{j=1}^{|c|} z_{ij}} \tag{8}$$

$$= \sum_{i=1}^{|c|} \theta_i \frac{z_{ii}}{\sum_{j=1}^{|c|} z_{ij}} \tag{9}$$

where θ_i is the frequency of class c_i in the test set and the remaining term is the recall for c_i. While the recall for a given class depends on the trained model, *the frequency of each class (θ) depends exclusively on the test data set.* This implies that the profiled accuracy can be a poor estimator of out-of-sample accuracy if the class frequencies do not match the frequency in the test data set.

To address this issue, we propose dynamically computing model accuracy by treating θ as a parameter in equation 9 and estimating it using the data rather than implicitly assigning it the frequencies in the test set. While we use the accuracy metric to illustrate the approach, the same principle applies to other scoring rules. For example, the weighted F_1 score uses $\boldsymbol{\theta}$ directly when averaging. The quadratic score can also be rewritten in terms of $\boldsymbol{\theta}$.

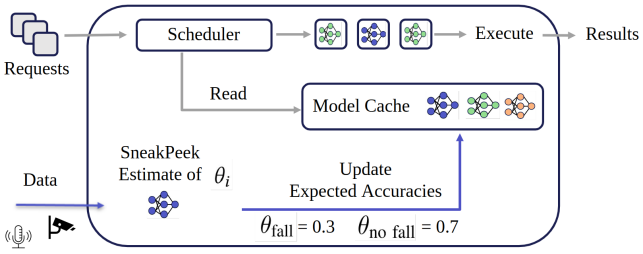


Figure 2: Data-awareness workflow for a fall detection application. A SneakPeek model updates expected model accuracies in real-time to improve scheduling decisions.

4.2 SneakPeek Estimation Scheme

We require a mechanism for estimating the accuracy of each model, given the data it will operate on. Our task is to dynamically estimate $\boldsymbol{\theta} = \langle\theta_1, ..., \theta_{|c|}\rangle$ based on the data, which will produce a better estimate of the accuracy of a given model.

Definition 4.1.1. A *SneakPeek model* is a model that uses real-time data to estimate $\boldsymbol{\theta}$.

SneakPeek models use the underlying data to give us better estimates of model accuracy, which can be used to improve scheduling decisions. By using SneakPeek models to estimate $\boldsymbol{\theta}$ directly, we can perform a single inference and use the result to update model accuracies for every model in $\mathcal{M}_{a_i}$. At first glance, it appears we can simply assign $\boldsymbol{\theta}$ the resulting class probabilities from a SneakPeek model. However, most machine learning models do not generate true probabilities, in the sense that they do not represent long-run frequencies. They generate *scores*, then a decision rule is applied to generate a prediction. To overcome this limitation, we consider a Bayesian approach for obtaining a (posterior) estimate of $\boldsymbol{\theta}$ using the Dirichlet-Multinomial model.

Definition 4.1.2. *SneakPeek probabilities* are the posterior estimates $\boldsymbol{\theta} \mid \boldsymbol{y}$ computed after using a SneakPeek model to collect the evidence $\boldsymbol{y}$.

We now define our mechanism for obtaining SneakPeek probabilities, given a model. To obtain our posterior estimates of $\boldsymbol{\theta}$ we need to define our *prior distribution* and a mechanism for collecting *evidence*. To illustrate this process, we consider a fall detection application in a healthcare setting, where the candidate models must infer whether a subject has fallen over and requires immediate attention. We assume this is a binary classification task (i.e. $|c| = 2$). Figure 2 shows the inference process at a high-level.

Prior Distribution. Our prior distribution has the form:

$$\boldsymbol{\theta} \sim \text{Dirichlet}(\alpha_1, ..., \alpha_{|c|}) \tag{10}$$

which has hyperparameters α_i that provide a weight for each class. Assigning α_i to a constant for all i is a reasonable default and works well in practice (section 6.3.3). The application owner can optionally specify alternative prior distributions, based on expert experience (and not taken implicitly from a test set). For fall detection, a true

positive (a subject has fallen) is a relatively rare event, so we may prefer to encode that in our prior distribution.

Evidence. We require a low-latency mechanism for generating a multinomial observation for the dirichlet/multinomial model. For the fall detection application, we want to use the data (e.g. maybe a short video clip) to determine which classes are likely present in the data. We use k-nearest neighbors with the original training data to provide this evidence. For a given data point, we find the k nearest points in the training data and allow each neighboring point to count in our multinomial realization. For example, if we have $k = 5$ and two neighbors have the "no fall" label and three have the "fall" label, then our multinomial evidence would be $\boldsymbol{y} = \langle 2, 3 \rangle$. Note that k-nearest-neighbors may not be the fastest algorithm, so approximate nearest neighbors (e.g. locality sensitive hashing) may be required in practice. An alternative option would be to use a single model and decision rule to produce a unit vector with a single non-zero entry, e.g. $\langle 0, 1 \rangle$ if we believe a fall occurred. However, this is a low-information update and has the potential to introduce additional error if the predicted class is wrong.

Posterior Estimates. In our model, the Dirichlet distribution is a conjugate prior, so the posterior distribution of $\boldsymbol{\theta}$ is:

$$\boldsymbol{\theta} \mid \boldsymbol{y} \sim \text{Dirichlet}(\alpha_1 + y_1, ..., \alpha_{|c|} + y_{|c|}) \tag{11}$$

Therefore, the two hyperparameters in our system are the choice of prior distribution and the number of neighboring points (k) to include in the evidence. The value of k can be optimized via standard supervised learning procedures.

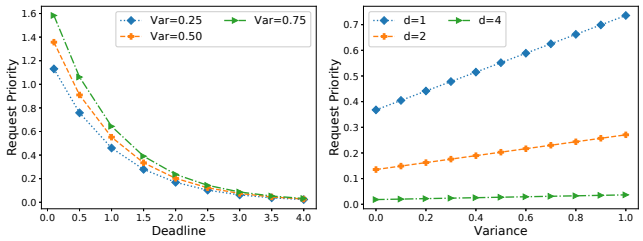
An important insight is that existing schedulers can directly incorporate SneakPeek modeling to improve accuracy estimation. The only change required is to include the per-class recall in model profiles. Furthermore, low-latency SneakPeek models can perform inference asynchronously and off the critical path, avoiding increased scheduling latency when estimating model accuracy.

5 MODEL SELECTION AND SCHEDULING

5.1 Proposed Approach

We require an efficient mechanism for solving the scheduling problem (eq. 3) since checking all candidate solutions is often infeasible. we propose splitting this problem into two sub-problems: (1) request ordering and (2) model selection. We first apply the ordering to our available requests, then select a locally-optimal model to satisfy each inference request in the order specified.

5.1.1 Priority-based Request Ordering. We require a strategy for ordering the requests prior to model selection. Common examples from the literature include first come first served (FCFS) [42, 54] and earliest deadline first (EDF) [40]. We propose a *priority-based ordering*, which attempts to dynamically navigate the trade-off between satisfying deadlines and maximizing inference accuracy. Our goal is to identify a priority function that gives higher values to requests that either (1) have tight deadlines or (2) have a high degree of variability across their choice of models (and thus, have a higher flexibility in model selection). We define the priority for a given request r_i to be:



(a) Priority vs. deadline **(b) Priority vs. variance**

Figure 3: Request priority as a function of deadline and variance in model accuracy.

$$\text{Priority}(r_i) = \left(1 + \text{Var}[Accuracy(\mathcal{M}_{a_i})]\right) e^{-d_i} \tag{12}$$

where $Accuracy(\mathcal{M}_{a_i}) = \{Accuracy(m_j) \mid m_j \in \mathcal{M}_{a_i}\}$ is the set of model accuracies for r_i's application models. Requests that have deadlines in the near future or have a high degree of variability[3] in model accuracy are prioritized. The intuition is that requests with upcoming deadlines should be prioritized with increasing urgency as we get closer to their deadlines. At the same time, whenever possible (given the deadlines), our scheduler should focus resources on higher model variance requests. If there is a low variance in model accuracy, then model selection is unlikely to have much effect on the average utility (which we are trying to maximize). Figure 3 illustrates how different deadlines and accuracy variances affect model priority. When requests are very close to their deadline, the priority increases rapidly. When deadlines are far in the future, the variance in model accuracy has a larger effect on priority.

5.1.2 Locally-Optimal Model Selection and Placement. Once requests have been ordered, then for each request r_i, we can obtain a locally-optimal solution by selecting the model with the highest utility, that is:

$$\underset{m_j \in \mathcal{M}_{a_i}}{\arg\max} \quad u_{a_i}(m_j, d_i, t_i) \tag{13}$$

where t_i is the current time (or the current time plus the expected wait time prior to execution). Finding the best model is linear in the number of available model variants.

5.2 Grouped Scheduling

Dividing the scheduling problem into request ordering and model selection provides a scalable way to schedule requests in practice. However, using locally-optimal decisions per request prevents schedulers from getting a global view of request dependencies and exploiting model reuse when making decisions. To get an intuition from optimal solutions, we brute forced solutions to the original scheduling problem in small dimensions (eq. 3). We observed that *the optimal solutions tend to group requests by application and typically assign the same model to all requests in the group.* Furthermore, this grouping pattern was present in 60-70% of scheduling windows when computing exact solutions (the remaining cases are discussed

[3]We use the population variance to compute a priority, so $|\mathcal{M}_{a_i}| = 1 \implies \text{Var}[Accuracy(\mathcal{M}_{a_i})] = 0$.

in section 5.3.2). These results suggest that inference batching (or request grouping) is a good strategy, which existing works can only exploit opportunistically if requests happen to be assigned the same model [1, 11, 45]. Using this intuition, we propose *grouping requests by application* and applying policies and selection strategies at the *group level*, rather than the request level. This also allows us to exploit model locality and avoid the additional latency associated with swapping models in and out of GPU memory.

We begin by defining the set of groups $\mathcal{G} = \{g_i \mid g_i \subseteq \mathcal{R}\}$ to be a partition of $\mathcal{R}$. Each group contains the subset of requests in $\mathcal{R}$ that belong to the same application (and therefore have the same candidate model variants). More precisely, $r_1, r_2 \in g_i \iff \mathcal{M}_{a_1} = \mathcal{M}_{a_2}$. Next, we define the priority of a group $g \in \mathcal{G}$ to be:

$$\text{Priority}(g) = \frac{1}{|g|} \sum_{r \in g} \text{Priority}(r) \tag{14}$$

that is, the group priority is simply the mean of the priority of each request in the group. This approach will attempt to greedily exploit inference batching, but it can also reduce the dimension of the original optimization problem. The full group-level scheduling algorithm is outlined in Algorithm 1. We first check to see if the number of groups is small enough to obtain an exact solution[4]. If not, we fall back to a locally-optimal solution (applied at the group level), which assigns priorities to each group and schedules all the requests within a group together. All of our proposed request-based approaches can be solved exactly for a very small number of requests; however, the grouping strategy allows many more scenarios where a solution can be brute forced, since it is a function of the number of applications $|A|$ rather than the number of requests $|\mathcal{R}|$, and $|A| << |\mathcal{R}|$ in practice.

Algorithm 1: Group-Level Scheduling

Input: $\mathcal{R}$, Model sets $\mathcal{M}_{a_i}$, Brute-Force Threshold τ

$\mathcal{S} \leftarrow$ Schedule initialized to all zeros
grouped_requests $\leftarrow$ HashMap < app : request list >

if *num_keys*(*grouped_requests*) $\leq \tau$ **then**
  return brute_force_solution(*grouped_requests*)
end

ordered_groups $\leftarrow$ sorted groups by avg priority (eq. 14)

for g ***in*** *ordered_groups* **do**
  $m_j =$ solution to eq. 13 using avg group utility
  ordered_requests $\leftarrow$ requests in g sorted by priority
  order $\leftarrow 1$
  for r_i ***in*** *ordered_requests* **do**
    $s_{ij} = order$
    $order = order + 1$
  end
end
return $\mathcal{S}$

[4]In practice, we found that 5 or fewer applications could be solved exactly in under 10ms (section 6.4.2).

5.3 Data-Aware Enhancements

In section 4, we proposed SneakPeek models within a Bayesian framework to obtain better estimates of model accuracy. We now discuss how our scheduling algorithms can be further improved using the results from our SneakPeek process.

5.3.1 Short-Circuit Inference. Since SneakPeek models estimate class probabilities before the main scheduling process begins, they can be used to provide an additional degree of freedom to schedulers. If request deadlines are tight or the SneakPeek model has very high accuracy, we can consider using the output of the SneakPeek model directly to satisfy the inference request. To enable this option within the scheduling framework, we treat the SneakPeek model as a candidate for scheduling, assigning it an effective inference latency of zero since its output has already been computed. This allows the scheduler to consider the SneakPeek model alongside other models and select it when doing so maximizes overall utility.

In figure 2, short-circuit inference places the SneakPeek model alongside the other models in the model cache, with an expected accuracy equal to the accuracy of the SneakPeek model and an inference latency of zero (returning the θ_i values if selected). However, assigning a latency of zero to the SneakPeek model does not guarantee it will be selected by the scheduler. Other models in the model cache will have higher accuracy than SneakPeek, which is designed to have low-latency. Therefore, defaulting to short-circuit inference with the SneakPeek model is only beneficial when deadlines are tight and waiting for a more accurate model results in a substantial loss in utility. Furthermore, we must rely on profiled accuracy when making scheduling decisions with SneakPeek models. We don't want to risk compounding errors by letting a SneakPeek model dynamically estimate its own accuracy based on the data. Existing scheduling algorithms and our proposed approaches can incorporate short-circuit inference directly, without requiring substantial changes to the system.

5.3.2 Enhancements for Grouped Scheduling. We can further improve our grouped scheduling algorithm by leveraging the fact that model accuracy varies, depending on the class label [54]. The group scheduler puts all requests for a given application into a single group, which then assigns a single model for all requests in the group. We propose going a step further and using data-awareness to split groups into subgroups based on the output from SneakPeek models. This approach is consistent with our results from computing exact solutions in small dimensions. While 60-70% of optimal solutions grouped requests based on target application (section 5.2), almost all of the remaining cases *grouped requests based on target class label within each application.* Using this insight and the information provided by SneakPeek models, we modify our approach to create a subgroup for each target label present in the data. We use the output from SneakPeek models to estimate class membership (indicated by $\theta_i > 0.5$). If $\theta_i < 0.5$ for all i, we do not split that request into a subgroup.

Figure 4 shows how the group splitting works for two binary classification tasks. In the red group on the left, the SneakPeek probabilities indicate the same class, so no splitting is performed. For the white group on the right, the SneakPeek probabilities indicate different classes are present, so the requests are split into two

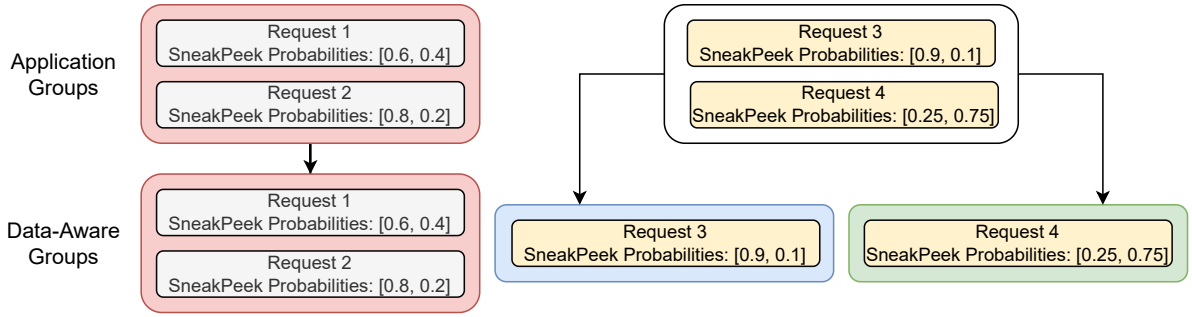


Figure 4: An illustration of the group splitting process using SneakPeek. The SneakPeek probabilities indicate that the first two requests have the same target class, and therefore require no splitting. We inferred that Requests 3 and 4 belong to different classes, so we split that application group.

subgroups. In the case of multi-class classification, the probabilities could also be inconclusive (i.e. $\theta_i < 0.5$ for all i) which would not result in any splitting.

6 EMPIRICAL EVALUATION

6.1 Methodology

Testbed. We conducted our experiments on a system with an Intel i5 CPU, 32 GiB of main memory, and an NVIDIA RTX 3060 graphics card, which has 12 GiB of memory.

Datasets and Models. We use systems and applications encountered in healthcare settings, including fall detection, speech commands, and heart monitoring (table 1). For fall detection, we have five model variants: 3 video models, one time series model (for accelerometer data), and one fusion model which uses both data modalities. Since falls are relatively rare events, we stream data such that 95% of the requests contain true negatives and 5% of the data contains true positives (a fall). For the voice command application, we have two model variants and we generate data uniformly across the target classes. The heart monitoring application also supports two models; we assume arrhythmias are somewhat infrequent and generate true negatives 80% of the time, with the other 20% distributed uniformly across the different arrhythmia types. Profiled model accuracy is averaged over the data in the test set. For generating a test set, we follow previously established methodology when available. If no methodology is published, we adopt the standard practice of randomly partitioning the available data points into training and testing sets.

Baselines and Metrics. We compare our proposed approach with several baselines. Each approach consists of an ordering policy– earliest deadline first (EDF) or our Priority ordering (eq. 12)– and a model selection strategy:

- *Max Accuracy*: Selects the highest accuracy model that can satisfy the request
- *Locally-Optimal*: A generalization of several existing works which are deadline-aware [42, 47, 54] and select the highest accuracy model that satisfies each deadline.

We consider the following combinations in our evaluation:

- *MaxAcc-EDF*: Max accuracy model selection + EDF ordering.
- *LO-EDF*: Locally-optimal model selection + EDF ordering.
- *LO-Priority*: Locally-optimal model selection + priority ordering.
- *Grouped*: Our group-level scheduling algorithm (Algorithm 1) which groups requests based on target application.
- *SneakPeek*: Our group-level scheduling algorithm with SneakPeek data-awareness and short-circuit inference.

All of these approaches can be combined with SneakPeek and our short-circuit (SC) inference optimization, which we evaluate in the next section. When we evaluate SneakPeek with the grouped approach, we also split groups based on target label (as discussed in section 5.3). We evaluate these approaches in terms of scheduling utility, accuracy, and deadline violations. For utility functions, we use the following penalties:

- *Step Function*: Requests that complete after the deadline receive a utility of zero, i.e. $\gamma_{a_i}(d_i, e_i) = \mathbb{1}_{d_i < e_i}$
- *Linear*: Penalty that increases linearly, i.e. $\gamma_{a_i}(d_i, e_i) = \mathbb{1}_{d_i < e_i} \max(1, \frac{e_i - d_i}{d_i})$
- *Sigmoid*: Penalty following a sigmoid curve: $\gamma_{g_i}(d_i, e_i) = \mathbb{1}_{d_i < e_i} \max\left(1, \frac{1}{1+(\frac{x}{1-x})^{-3}}\right)$ where $x = 1 - \frac{2d_i - e_i}{d_i}$.

By default, we use 12 requests (4 for each application) arriving uniformly over a scheduling window of 100ms. For approximate nearest neighbors, we use the Faiss library [14] and default to $k = 5$. Unless otherwise stated, we use an uninformative prior for SneakPeek estimation and default to the sigmoid penalty function.

6.2 Scheduling Performance

We first compare our proposed approaches in terms of the resulting utility, accuracy, and deadline violations. If a deadline is exceeded, the violation time for a given request is the completion time minus the deadline. For these experiments, we fix the average deadline per request at 150ms.

Figure 5 shows the resulting utility and model accuracy for this experiment, while deadline violations can be found in figure 6. We observe that SneakPeek achieves the highest utility compared to the baselines: a 2x increase in utility compared to LO-EDF. MaxAcc-EDF always selects the highest accuracy model, which results in high average accuracy, but lower utility due to the associated deadline

Table 1: Applications and Datasets

Application	Description	Dataset	Models
Fall Detection	Determine if a subject has fallen (2 classes)	MMAct [33]	X3D [17] for video data (small, medium, and large variants), MiniRocket [13] for time series, and a fusion model which leverages both modalities [10]
Voice Commands	Detect keywords spoken by staff members (6 classes)	Speech Commands [53]	Howl Framework [50] (with LSTM and MobileNet)
Heart Monitoring	Monitor ECG data for abnormal patterns (7 classes)	MIT-BIH Arrhythmia Database [20]	EcgResNet34, CNN [30]

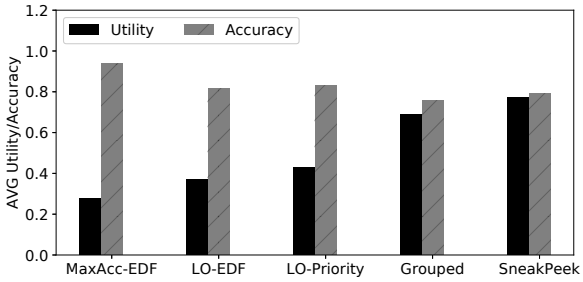


Figure 5: Comparison of average schedule utility and model accuracy across approaches.

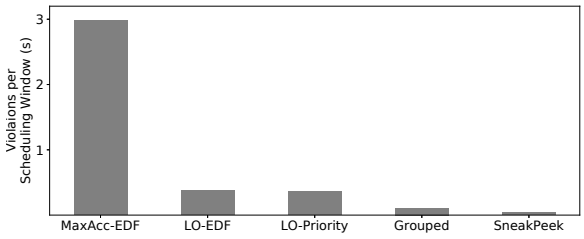


Figure 6: Deadlines violations across approaches.

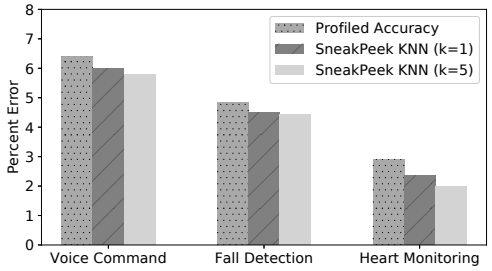


Figure 7: Average error when estimating the accuracy of each model.

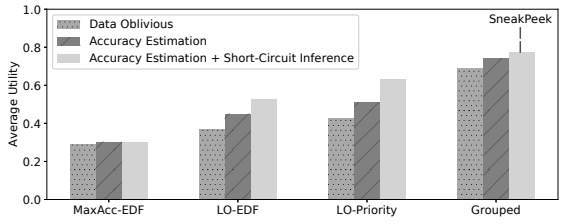


Figure 8: Comparison of schedule utility across approaches, including data-aware variants of each baseline approach.

violations. The other approaches have slightly lower accuracy but also substantially lower deadline violations. The grouped scheduler has the lowest accuracy, but relatively high utility, since it is able to greedily incorporate inference batching to meet more deadlines. SneakPeek has the fewest deadline violations (almost 0), since it has all the benefits of the grouped scheduler and is able to leverage short-circuit inference to meet deadlines when profiled models cannot.

6.3 SneakPeek Performance

We now evaluate the ability of SneakPeek models to estimate model accuracy and examine the effects of different prior distributions.

6.3.1 Dynamic Accuracy Estimation. We now evaluate the degree to which our SneakPeek probabilities reduce error when estimating model accuracy. We define the "true model accuracy" using the expression in equation 9, with $\theta_i = 1$ for the true class label and $\theta_i = 0$ for all other labels. We use approximate nearest neighbors as our SneakPeek model and evaluate the effect using $k = 1$ and $k = 5$. We measure the associated error using all three datasets and restrict ourselves to uninformative priors when generating SneakPeek probabilities.

Figure 7 shows the results for this experiment. We observe that in all cases, the SneakPeek probabilities are able to improve the estimates of model accuracy, with $k = 5$ performing the best, followed by $k = 1$ for approximate nearest neighbors. Improvements to these accuracy estimates is the main mechanism which allows the data-aware schedulers to increase average schedule utility.

We also evaluate how the various data-awareness mechanisms incrementally improve the existing approaches (figure 8). First, we observe that the data-oblivious grouped scheduler attains a higher utility than any of the data-aware baselines, showing the benefits of grouping. In all cases, we observe that incorporating data-awareness in the scheduling process improves schedule utility, showing the benefit of using SneakPeek with any scheduling policy.

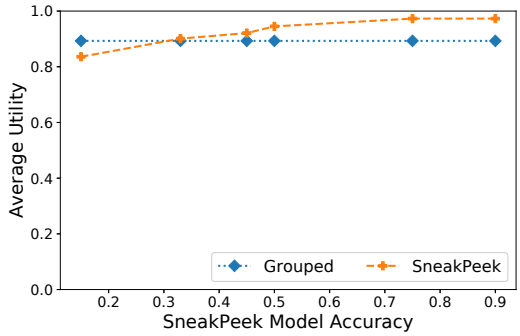


Figure 9: Schedule utility across different degrees of SneakPeek model accuracy.

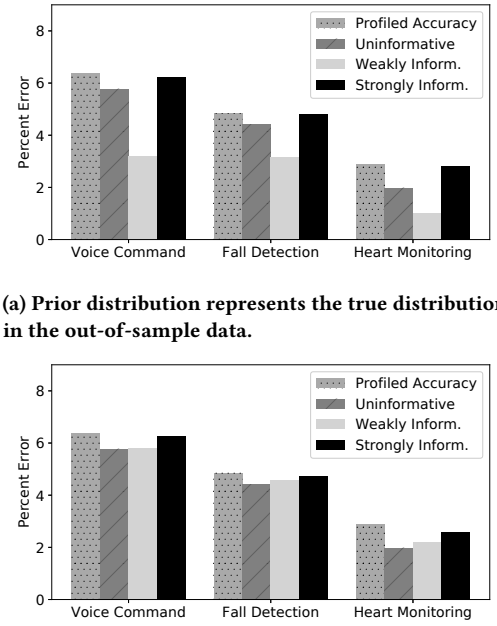
We also observe that in most cases, including SneakPeek models as a scheduling option for short-circuit inference improves average schedule utility, although the degree varies across application. Since most of our approaches involve locally-optimal decisions, the SneakPeek models improve the utility of requests that have been deemed low-priority. If we cannot satisfy the deadline for these requests, SneakPeek models provide a mechanism for salvaging utility. These improvements go beyond the benefits of data-awareness, which are provided by using the SneakPeek probabilities to sharpen model accuracy estimates. Note that MaxAcc-EDF does not benefit from short-circuit inference, since it always selects the model that maximizes accuracy, and SneakPeek is never the most accurate model available.

6.3.2 SneakPeek Accuracy Requirements. Using SneakPeek models to estimate accuracy raises another question: how accurate do SneakPeek models need to be? In order to answer this question, we create a special SneakPeek model for each application, which randomly returns class probabilities depending on a specified confusion matrix. We simply generate a confusion matrix with the specified accuracy (and uniformly distribute errors across the remaining classes). Then, given the data point, we randomly generate probabilities using the specified frequencies in the true label row.

Figure 9 shows that accuracy values above 30% provide *some* utility benefits. For lower accuracy values, we observe that the schedule utility degrades slightly. These results suggest that SneakPeek models do not require extremely high accuracy, but can begin improving scheduling decisions with moderate inference accuracy.

6.3.3 Choice of Prior. One of the SneakPeek hyperparameters is the choice of prior distribution for $\boldsymbol{\theta}$. We now evaluate our different priors and examine their effects on accuracy estimation. We consider the following prior distributions for SneakPeek estimation:

- *Uninformative*: Does not provide any information regarding the class frequencies. We use the Jeffreys prior, which assigns $\alpha_i = 0.5$ for all i.
- *Weakly Informative*: Incorporates expert knowledge, but does not weight it heavily, e.g. by simply assigning α_i the expected frequency of each label.
- *Strongly Informative*: Incorporates expert knowledge and weights it heavily, by assigning α_i the expected number of requests with label i in a scheduling window.



(a) Prior distribution represents the true distribution in the out-of-sample data.

(b) Prior distribution derived from test dataset, which is not aligned with out-of-sample data.

Figure 10: Accuracy estimation error with different prior distributions.

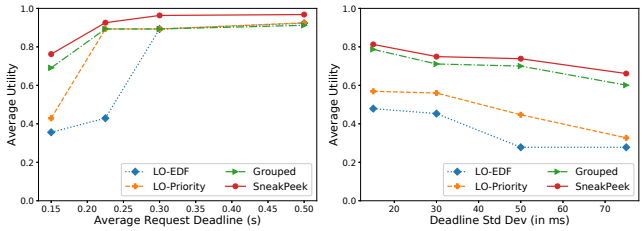
The "true distribution" is defined for each dataset as outlined in the section 5.1. The distribution of labels in the test set (for profiling) is formed by generating a uniform random sample from the entire dataset for each application.

Figure 10a shows the resulting effects on accuracy estimation when the prior captures the true distribution in the data. We observe that our accuracy estimation improves for both uninformative priors and weakly informative priors. However, the strongly informative priors incur higher error rates, even if the prior matches the true distribution. This occurs because a strong prior suppresses any signal from the data itself. By averaging over the heterogeneity in the data, strong priors give us essentially another profiled average. This average may be slightly better than a profiled accuracy with an arbitrary distribution, but it is still suboptimal.

Figure 10b shows the accuracy when the prior captures the distribution in the test set, rather than the true distribution. In this case, the uninformative prior provides the lowest error rates. The weakly informative and strongly informative priors are increasingly worse, since they incorporate information that does not represent the true distribution of the data.

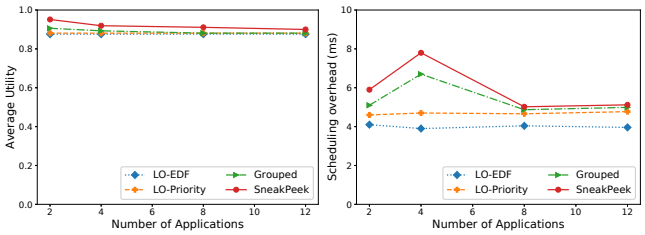
6.4 Sensitivity Analysis

For these experiments, we present results for LO-EDF (which is representative of existing approaches) and our proposed algorithms, omitting MaxAcc-EDF for clarity.



(a) Avg. utility w.r.t. deadlines (b) Increasing variance

Figure 11: Resulting schedule utility with increasing deadlines and deadline variance.



(a) Average Utility (b) Scheduling Overhead

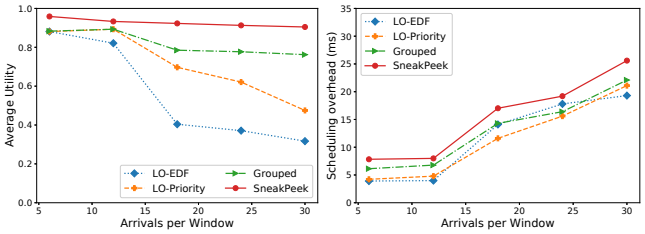
Figure 12: Scheduling utility and overhead as a function of the number of applications.

6.4.1 Deadlines. We now evaluate how different deadline patterns affect the proposed scheduling approaches.

Figure 11a examines how utility changes as request deadlines get increasingly larger. We observe that the grouped scheduler outperforms the proposed baselines for short-term deadlines and the data-aware grouped scheduler consistently outperforms the baselines. LO-EDF struggles with short deadlines, since EDF does not incorporate information regarding which requests have high accuracy variance. Furthermore, locally-optimal model selection does not exploit inference batching. After 300 milliseconds, most of the data-oblivious baselines converge to the same utility, since the larger deadlines allow for more flexibility.

We consider another deadline experiment where we increase the *variance of the deadlines* across requests. We generated deadlines following a normal distribution with a mean of 150ms. We then increase the variance of this distribution to observe the effect on utility. Figure 11b shows the results for this experiment. We observe that all approaches slowly degrade as the variance increases. When there are large differences between request deadlines, the schedulers have fewer degrees of freedom to optimize and must process more requests in the order specified by the deadlines, as the tighter deadline requests become comparatively more urgent.

6.4.2 Number of Applications. We now examine how our approaches behave as the number of applications increases. Note that increasing the number of applications is not the same as increasing the number of requests. We fix the number of requests at 24 for the scheduling window and set the average deadline to 200ms (after arrival). We selected a slightly longer deadline, since the number of requests is higher than in previous experiments.



(a) Utility w.r.t. arrival rate (b) Scheduling Overhead

Figure 13: Scheduling utility and overhead as a function of request arrival rate.

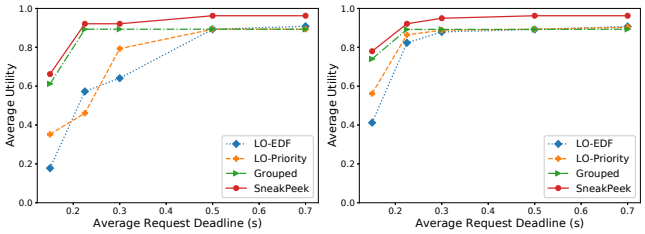
Figure 12a shows the results for this experiments. We observe that the grouped approaches slowly degrade toward the baseline approaches as the number of applications increases (for a fixed number of requests). This is expected, since increasing the number of applications increases the number of clusters. So our approach behaves more like LO-Priority as the number of applications increases. LO-EDF and LO-Priority have consistent performance across an increase in applications, since they make decisions at the request level and the underlying models are the same.

Figure 12b examines the computational overhead for scheduling as the number of applications increases. Each data point represents the mean scheduling latency across 5 invocations of the scheduler. We first note that all of the approaches are able to produce a schedule in under 10ms. We observe that for small number of applications, the grouped approach requires additional time, since it is able to solve the assignment problem via brute force. We also observe an additional increase for the data-aware group-level scheduler, as it creates additional groups based on the target label. Since the number of requests is fixed, the baseline approaches provide a more consistent overhead. For larger numbers of applications, the approaches roughly converge.

6.4.3 Request Arrival Rate. Our previous experiments assumed an arrival rate of 12 requests, uniformly distributed across a 100ms scheduling window. We now fix the deadline at 200ms and vary the number of requests per window to observe the effects on schedule utility. We can think of this as scaling the number of patients that must be monitored by the existing applications.

Figure 13a shows that all approaches slowly experience degraded utility as the number of arrivals increases. This is expected, since the deadlines are fixed and the amount of inference compute required is increasing. The schedulers are eventually forced to prioritize requests as the deadlines become tighter. The grouped schedulers perform the best in this scenario and maintain the highest average utility. LO-Priority is the next best, since it also prioritizes requests based on the variance in model performance and the anticipated deadline. LO-EDF is deadline-aware, but does not prioritize requests to account for the increased workload.

Figure 13b examines the computational overhead for scheduling as the number of requests increases. For all approaches, we observe an increase in scheduling overhead as the number of requests increases. This is not surprising, since all of our proposed approaches perform work that is linear in the number of requests. We also



(a) Step Function Penalty **(b) Linear Penalty**

Figure 14: Schedule utility across penalty functions.

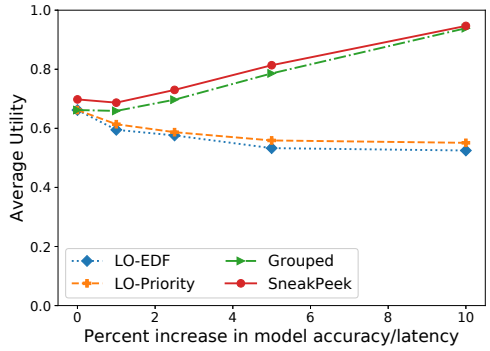


Figure 15: Schedule utility with increasingly heterogeneous model variants.

observe a slightly higher overhead for clustering (especially with data-awareness) since the scheduler is able to brute force small numbers of clusters.

6.4.4 Penalty Function. We now examine how various penalty functions affect average scheduling utility.

Figure 14a shows the resulting utilities for a step function. We observe that our data-aware group scheduler outperforms all the other approaches. The data oblivious group scheduler also strongly outperforms the baselines when deadlines are short, and is comparable to the baselines when deadlines are longer. LO-Priority and LO-EDF perform worse in this setting, since they fail to exploit inference batching for early requests. Then the deadlines for later requests are exceeded, which results in a utility of zero.

Figure 14b shares a similar pattern with the sigmoid penalty. SneakPeek consistently outperforms the data-oblivious baselines. Data-oblivious grouped scheduling also outperforms the baselines when deadlines are short.

6.4.5 Model Variants. We now explore how different sets of model variants affect schedule utility. The goal of this experiment is to understand how model heterogeneity in $\mathcal{M}_{a_i}$ (in terms of accuracy and latency) influences system performance. We seek to derive practical insights that can guide practitioners in training and selecting model variants for deployment in data-aware inference serving systems. We exclude the short-circuit approach from this evaluation, as it obscures the direct effects of model heterogeneity when it defers to the SneakPeek model. We first created test models that randomly return the correct label following a pre-specified accuracy. Then, we generate three models for each application: one which has the mean accuracy and inference latency of all models in $\mathcal{M}_a$, and two that we use to increase the variance in model performance for that application. At each point, we alter the average accuracy (and latency) by a specified percentage. For example, if the average model in $\mathcal{M}_a$ has 80% accuracy with an inference latency of 20ms, and we want to increase the variance by 1%, we will create two models with accuracies of 79.2% and 80.8%, and corresponding latencies of 19.8 ms and 20.2 ms.

Figure 15 shows the results for this experiment. We observe that our proposed grouped approaches benefit substantially as the variance in model performance increases. The grouped approaches are able to exploit inference batching, which allows for more time to execute high utility models and meet more deadlines. This suggests that including a diverse set of model variants gives systems the most flexibility, which can help improve utility. The approaches that do not incorporate grouping obtain lower utilities as the variance increases. As higher accuracy (and latency) models become available, these locally-optimal approaches select them for the first few requests, but then fail to meet the deadlines for the requests that are later in the ordering.

6.5 Summary of Results

Our evaluation highlights the benefits of *grouped* scheduling, where incorporating inference batching decisions in the scheduling process has tangible benefits. We also showed that leveraging SneakPeek models for data-awareness provides additional benefits, by selecting more accurate models for each request.

7 MULTI-WORKER SETTING

7.1 Problem Definition

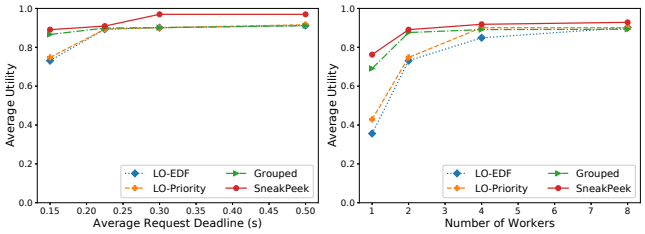
Our problem formulation currently focuses on a system with a single GPU, but our proposed approaches can also be generalized to settings with multiple, heterogeneous workers. The schedule can be augmented to include a worker in the selection process (indexed by k). For a set of workers $\mathcal{W}$, the (global) optimization objective becomes:

$$\max_{S} \quad \frac{1}{|\mathcal{R}|} \sum_{i=1}^{|\mathcal{R}|} \sum_{j=1}^{|\mathcal{M}_{a_i}|} \sum_{k=1}^{|\mathcal{W}|} \mathbb{1}_{s_{ijk}>0}\, u_{a_i}(m_j, d_i, t_i) \tag{15}$$

In this setting, each model variant would be profiled on every candidate worker, which would create latency functions that depend on the model and worker.

7.2 Preliminary Results

We briefly examine how our approach performs in the multi-worker environment by simulating multiple workers on the same hardware used in the single-worker experiments. We ensure that each worker operates independently to minimize resource contention. In our first experiment (figure 16a), we compare schedule utilities with two workers and vary the average deadline. Similar to the single worker case, we observe strong benefits associated with grouped scheduling, which is able to explicitly leverage inference batching. Incorporating data-awareness (SneakPeek) also provides substantial benefit, especially with higher request deadlines.



(a) Utility with two workers **(b) Varying workers**

Figure 16: Preliminary comparison of resulting schedule utility in the multi-worker setting.

In figure 16b, we compare schedule utilities while varying the number of workers. We use an average request deadline of 150 ms. We observe that as the number of workers increases, the relative benefit of our grouped scheduling decreases. This is expected, since the benefit of inference batching decreases if there is less resource contention and fewer requests are assigned to each worker. We also note that the benefit of data-awareness increase slightly with the number of workers. Since deadlines are tight, having more executors allows us to avoid exclusively selecting the fastest model to service the request.

8 DISCUSSION

Our evaluation highlights the benefits of *data-awareness*, which can be incorporated seamlessly into existing schedulers and allows us to select more accurate models for each request. We also observed that *grouped* scheduling provides additional benefits by incorporating inference batching decisions into the scheduling process.

Our system performs SneakPeek modeling on an edge system, but a future work could push these operations closer to the data-generating devices. For example, if this computation can be performed on local IoT devices, data transfers could be avoided, and short-circuit inference could be leveraged when appropriate. Cloud offloading was excluded from this analysis, since WAN transfers can be costly and applications in healthcare and other domains may be restricted in their ability to offload data. However, offloading inference to the cloud can also be incorporated into our framework by modifying the latency function $\ell(m_j)$ to include any data transfer time required to run a model in a different location.

Additional modifications may be required to further improve efficiency in the multi-worker setting. For grouped scheduling, we need to enforce a maximum group size and split groups. Having one very large group could lead to load imbalance, so we may require periodic load balancing. Furthermore, data plane optimizations could also be explored to ensure video frames are always available for inference by the time requests are dispatched to workers.

9 RELATED WORK

Inference serving (or model serving) has received substantial research attention, with a special focus on cloud deployments [4, 16, 23, 36, 40, 45, 46, 55]. The InFaaS system introduced the concept of *model-less* inference serving, where model variants are dynamically selected by the system in addition to serving the request [46]. Other systems such as MArk attempt to forecast demand and use autoscaling to ensure that performance is consistent across fluctuating workloads [55]. Resource elasticity is a common component in these designs, which is not available in all application settings. In addition, offloading data to the cloud may be infeasible due to network constraints or privacy requirements.

Several optimizations have been proposed to improve various aspects of the inference serving model (for both edge and cloud deployments). Several techniques employ *dynamic batching* to reduce inference latency when the scheduler assigns the same model to adjacent requests [1, 11, 45]. However, inference batching can be exploited further when it is incorporated directly in scheduling decisions (e.g. with a grouped scheduler). A complimentary research direction attempts to improve efficiency by having multiple requests share hardware resources [24, 37, 39, 42]. Additional works have considered mechanisms for exploiting multi-modal data sources to improve system efficiency [28, 54]. Systems that process multi-modal data often have models with extremely diverse characteristics (accuracy and latency) which can offer additional flexibility to the system [54]. Efficient interaction with inference serving systems is another important research area, given the overhead of transmitting data and models [34]. These optimizations are largely complimentary and can be incorporated directly into our system.

Additional works directly address machine learning inference in hardware-constrained settings, such as the edge [1, 2, 22, 35, 42, 47, 48, 51, 54, 56, 57]. A common strategy is to consider accuracy scaling when hardware-scaling is unavailable [1, 2, 18, 38, 42, 54]. The Proteus system maximizing model accuracy, subject to a constraint on the minimum system throughput [1]. Other edge systems attempt to offload requests when insufficient resources are available locally [42, 52]. The Neurosurgeon system dynamically partitions DNN models to allow parts of the model to execute locally at the edge while the remaining computation is offloaded to the cloud [31]. The LayerCake system performs locally-optimal model selection at the edge, while also enumerating candidate models that can be run in the cloud [42]. Inference pipelines are also commonly deployed at the edge. Video analytics pipelines such as Chameleon and Ekya leverage spatial-awareness and continuous learning to improve efficiency and accuracy at the edge [5, 29].

10 CONCLUSION

We proposed a scheduling algorithm for hardware-constrained inference serving which implements accuracy scaling and greedily incorporates inference batching into the scheduling process. We also showed that making decisions based on profiled model accuracy is suboptimal, since it is a static quantity computed over a test dataset and may not reflect the characteristics of out-of-sample data. To address this, we proposed SneakPeek: a data-aware approach which attempts to dynamically improve the estimates of model accuracy in real-time. Our evaluation shows that these techniques obtain higher utility schedules and higher SLO attainment.

ACKNOWLEDGEMENTS

We thank our shepherd Francisco Romero and the anonymous reviewers for their valuable feedback on this work.

REFERENCES

[1] Sohaib Ahmad, Hui Guan, Brian D. Friedman, Thomas Williams, Ramesh K. Sitaraman, and Thomas Woo. 2024. Proteus: A High-Throughput Inference-Serving System with Accuracy Scaling. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 318–334.

[2] Sohaib Ahmad, Hui Guan, and Ramesh K. Sitaraman. 2024. Loki: A System for Serving ML Inference Pipelines with Hardware and Accuracy Scaling. In *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing* (Pisa, Italy) *(HPDC '24)*. Association for Computing Machinery, New York, NY, USA, 267–280. https://doi.org/10.1145/3625549.3658688

[3] Ganesh Ananthanarayanan et al. 2017. Real-Time Video Analytics: The Killer App for Edge Computing. *Computer* (2017).

[4] Tiago Da Silva Barros et al. 2024. Scheduling with Fully Compressible Tasks: Application to Deep Learning Inference with Neural Network Compression. In *2024 IEEE/ACM 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 327–336.

[5] Romil Bhardwaj et al. 2022. Ekya: Continuous Learning of Video Analytics Models on Edge Compute Servers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 119–135.

[6] S. C. Bollepalli, R. K. Sevakula, W. M. Au-Yeung, M. B. Kassab, F. M. Merchant, G. Bazoukis, R. Boyer, E. M. Isselbacher, and A. A. Armoundas. 2021. Real-Time Arrhythmia Detection Using Hybrid Convolutional Neural Networks. *J Am Heart Assoc* 10, 23 (Dec 2021), e023222.

[7] John Bruno and Peter Downey. 1978. Complexity of Task Sequencing with Deadlines, Set-Up Times and Changeover Costs. *SIAM J. Comput.* 7, 4 (1978), 393–404. https://doi.org/10.1137/0207031

[8] Hoyt Burdick, Carson Lam, Samson Mataraso, Anna Siefkas, Gregory Braden, R. Phillip Dellinger, Andrea McCoy, Jean-Louis Vincent, Abigail Green-Saxena, Gina Barnes, Jana Hoffman, Jacob Calvert, Emily Pellegrini, and Ritankar Das. 2020. Prediction of respiratory decompensation in Covid-19 patients using machine learning: The READY trial. *Computers in Biology and Medicine* 124 (2020), 103949.

[9] Bo Chen, Chris N. Potts, and Gerhard J. Woeginger. 1998. *A Review of Machine Scheduling: Complexity, Algorithms and Approximability*. Springer US, Boston, MA, 1493–1641. https://doi.org/10.1007/978-1-4613-0303-9_25

[10] Hyeongju Choi, Apoorva Beedu, Harish Haresamudram, and Irfan Essa. 2022. Multi-Stage Based Feature Fusion of Multi-Modal Data for Human Activity Recognition. arXiv:2211.04331 [cs.CV]

[11] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 613–627.

[12] Stefania Cristina, Vladimir Despotovic, Rodrigo Pérez-Rodríguez, and Slavisa Aleksic. 2024. Audio- and Video-Based Human Activity Recognition Systems in Healthcare. *IEEE Access* 12 (2024), 8230–8245.

[13] Angus Dempster, Daniel F. Schmidt, and Geoffrey I. Webb. 2021. MiniRocket: A Very Fast (Almost) Deterministic Transform for Time Series Classification *(KDD '21)*. Association for Computing Machinery, New York, NY, USA, 248–257.

[14] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. (2024). arXiv:2401.08281 [cs.LG]

[15] Zhou Fang, Dezhi Hong, and Rajesh K. Gupta. 2019. Serving deep neural networks at the cloud edge for vision applications on mobile platforms. In *Proceedings of the 10th ACM Multimedia Systems Conference* (Amherst, Massachusetts) *(MMSys '19)*. Association for Computing Machinery, New York, NY, USA, 36–47. https://doi.org/10.1145/3304109.3306221

[16] Zhou Fang, Tong Yu, Ole J. Mengshoel, and Rajesh K. Gupta. 2017. QoS-Aware Scheduling of Heterogeneous Servers for Inference in Deep Neural Networks. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management* (Singapore, Singapore) *(CIKM '17)*. Association for Computing Machinery, New York, NY, USA, 2067–2070.

[17] C. Feichtenhofer. 2020. X3D: Expanding Architectures for Efficient Video Recognition. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 200–210.

[18] Andrea Fresa and Jaya Prakash Champati. 2021. Offloading Algorithms for Maximizing Inference Accuracy on Edge Device Under a Time Constraint. *CoRR* abs/2112.11413 (2021). arXiv:2112.11413 https://arxiv.org/abs/2112.11413

[19] Bruce Friedman, Daniel Fuckert, Mary Jahrsdoerfer, Rochelle Magness, Emily S. Patterson, Rehman Syed, and John R. Zaleski. 2019. Identifying and Monitoring Respiratory Compromise: Report from the Rules and Algorithms Working Group. *Biomedical Instrumentation & Technology* 53, 2 (2019), 110–123.

[20] A L Goldberger, L A Amaral, L Glass, J M Hausdorff, P C Ivanov, R G Mark, J E Mietus, G B Moody, C K Peng, and H E Stanley. 2000. PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals. *Circulation* 101, 23 (June 2000), E215–20.

[21] Margherita Grandini, Enrico Bagli, and Giorgio Visani. 2020. Metrics for Multi-Class Classification: an Overview. arXiv:2008.05756 [stat.ML] https://arxiv.org/abs/2008.05756

[22] Peizhen Guo, Bo Hu, and Wenjun Hu. 2022. Sommelier: Curating DNN Models for the Masses. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1876–1890. https://doi.org/10.1145/3514221.3526173

[23] M. Halpern, B. Boroujerdian, T. Mummert, E. Duesterwald, and V. Janapa Reddi. 2019. One Size Does Not Fit All: Quantifying and Exposing the Accuracy-Latency Trade-Off in Machine Learning Cloud Service APIs via Tolerance Tiers. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. Los Alamitos, CA, USA.

[24] Ziyi Han, Ruiting Zhou, Chengzhong Xu, Yifan Zeng, and Renli Zhang. 2024. InSS: An Intelligent Scheduling Orchestrator for Multi-GPU Inference With Spatio-Temporal Sharing. *IEEE Transactions on Parallel and Distributed Systems* 35, 10 (2024), 1735–1748. https://doi.org/10.1109/TPDS.2024.3430063

[25] Mohammed K. Hassan, Ali I. El Desouky, Sally M. Elghamrawy, and Amany M. Sarhan. 2019. *Big Data Challenges and Opportunities in Healthcare Informatics and Smart Hospitals*. Springer International Publishing, Cham.

[26] Kim Hazelwood, Sarah Bird, David Brooks, Soumith Chintala, Utku Diril, Dmytro Dzhulgakov, Mohamed Fawzy, Bill Jia, Yangqing Jia, Aditya Kalro, James Law, Kevin Lee, Jason Lu, Pieter Noordhuis, Misha Smelyanskiy, Liang Xiong, and Xiaodong Wang. 2018. Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 620–629. https://doi.org/10.1109/HPCA.2018.00059

[27] Shenda Hong, Yanbo Xu, Alind Khare, Satria Priambada, Kevin Maher, Alaa Aljiffry, Jimeng Sun, and Alexey Tumanov. 2020. HOLMES: Health OnLine Model Ensemble Serving for Deep Learning Models in Intensive Care Units. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Virtual Event, CA, USA) *(KDD '20)*. Association for Computing Machinery, New York, NY, USA, 1614–1624.

[28] Bodun Hu, Le Xu, Jeongyoon Moon, Neeraja J. Yadwadkar, and Aditya Akella. 2023. MOSEL: Inference Serving Using Dynamic Modality Selection. arXiv:2310.18481 [cs.LG]

[29] Junchen Jiang et al. 2018. Chameleon: Scalable Adaptation of Video Analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary) *(SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 253–266.

[30] Tae Joon Jun, Hoang Minh Nguyen, Daeyoun Kang, Dohyeun Kim, Daeyoung Kim, and Young-Hak Kim. 2018. ECG arrhythmia classification using a 2-D convolutional neural network. *CoRR* abs/1804.06812 (2018). http://arxiv.org/abs/1804.06812

[31] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (Xi'an, China) *(ASPLOS '17)*. Association for Computing Machinery, New York, NY, USA, 615–629.

[32] Liu Ke, Udit Gupta, Mark Hempstead, Carole-Jean Wu, Hsien-Hsin S. Lee, and Xuan Zhang. 2022. Hercules: Heterogeneity-Aware Inference Serving for At-Scale Personalized Recommendation. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 141–154. https://doi.org/10.1109/HPCA53966.2022.00019

[33] Quan Kong, Ziming Wu, Ziwei Deng, Martin Klinkigt, Bin Tong, and Tomokazu Murakami. 2019. MMAct: A Large-Scale Dataset for Cross Modal Human Action Understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

[34] Adithya Kumar, Anand Sivasubramaniam, and Timothy Zhu. 2023. SplitRPC: A Control + Data Path Splitting RPC Stack for ML Inference Serving. *Proc. ACM Meas. Anal. Comput. Syst.* 7, 2, Article 30 (May 2023), 26 pages. https://doi.org/10.1145/3589974

[35] ChonLam Lao, Jiaqi Gao, Ganesh Ananthanarayanan, Aditya Akella, and Minlan Yu. 2024. HawkVision: Low-Latency Modeless Edge AI Serving. arXiv:2405.19213 [eess.SY] https://arxiv.org/abs/2405.19213

[36] Baolin Li, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. 2023. Kairos: Building Cost-Efficient Machine Learning Inference Systems with Heterogeneous Cloud Resources. In *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing* (Orlando, FL, USA) *(HPDC '23)*. Association for Computing Machinery, 3–16.

[37] Lixian Ma, Haoruo Chen, En Shao, Leping Wang, Quan Chen, and Guangming Tan. 2024. ElasticRoom: Multi-Tenant DNN Inference Engine via Co-design with Resource-constrained Compilation and Strong Priority Scheduling. In *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing* (Pisa, Italy) *(HPDC '24)*. Association for Computing Machinery, 1–14.

[38] Vicent Sanz Marco, Ben Taylor, Zheng Wang, and Yehia Elkhatib. 2020. Optimizing Deep Learning Inference on Embedded Systems Through Adaptive Model Selection. *ACM Trans. Embed. Comput. Syst.* 19, 1, Article 2 (feb 2020), 28 pages.

[39] Daniel Mendoza, Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. Interference-Aware Scheduling for Inference Serving. In *Proceedings of the 1st Workshop on Machine Learning and Systems* (Online, United Kingdom) *(EuroMLSys '21)*. Association for Computing Machinery, New York, NY, USA, 80–88. https://doi.org/10.1145/3437984.3458837

[40] 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 (EuroSys '24)*. Association for Computing Machinery, New York, NY, USA, 1016–1038.

[41] Vinod Nigade, Pablo Bauszat, Henri Bal, and Lin Wang. 2022. Jellyfish: Timely Inference Serving for Dynamic Edge Networks. In *2022 IEEE Real-Time Systems Symposium (RTSS)*. 277–290. https://doi.org/10.1109/RTSS55097.2022.00032

[42] Samuel Ogden and Tian Guo. 2023. LayerCake: Efficient Inference Serving with Cloud and Mobile Resources. In *2023 23nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*.

[43] Iyiola E. Olatunji and Chun-Hung Cheng. 2019. *Video Analytics for Visual Surveillance and Applications: An Overview and Survey*. Springer.

[44] Khansa Rasheed et al. 2021. Machine Learning for Predicting Epileptic Seizures Using EEG Signals: A Review. *IEEE Reviews in Biomedical Engineering* 14 (2021), 139–155.

[45] Kamran Razavi, Saeid Ghafouri, Max Mühlhäuser, Pooyan Jamshidi, and Lin Wang. 2024. Sponge: Inference Serving with Dynamic SLOs Using In-Place Vertical Scaling. In *Proceedings of the 4th Workshop on Machine Learning and Systems (EuroSys '24)*. ACM, 184–191. https://doi.org/10.1145/3642970.3655833

[46] 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.

[47] Wonik Seo, Sanghoon Cha, Yeonjae Kim, Jaehyuk Huh, and Jongse Park. 2021. SLO-Aware Inference Scheduler for Heterogeneous Processors in Edge Platforms. *ACM Trans. Archit. Code Optim.* 18, 4, Article 43 (jul 2021), 26 pages. https://doi.org/10.1145/3460352

[48] Yechao She et al. 2023. On-demand Edge Inference Scheduling with Accuracy and Deadline Guarantee. In *2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*. 1–10.

[49] Mohammad Khubeb Siddiqui et al. 2020. A review of epileptic seizure detection using machine learning classifiers. *Brain Informatics* 7, 1 (25 May 2020), 5. https://doi.org/10.1186/s40708-020-00105-1

[50] Raphael Tang, Jaejun Lee, Afsaneh Razi, Julia Cambre, Ian Bicking, Jofish Kaye, and Jimmy Lin. 2020. Howl: A Deployed, Open-Source Wake Word Detection System. In *Proceedings of Second Workshop for NLP Open Source Software (NLP-OSS)*. Association for Computational Linguistics, 61–65. https://doi.org/10.18653/v1/2020.nlposs-1.9

[51] Achilleas Tzenetopoulos et al. 2024. Seamless HW-accelerated AI serving in heterogeneous MEC Systems with AI@EDGE. In *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing* (Pisa, Italy) *(HPDC '24)*. Association for Computing Machinery, New York, NY, USA, 377–380.

[52] Xuezhi Wang et al. 2022. Dynamic DNN Model Selection and Inference off Loading for Video Analytics with Edge-Cloud Collaboration. In *Proceedings of the 32nd Workshop on Network and Operating Systems Support for Digital Audio and Video*. Association for Computing Machinery, New York, NY, USA, 64–70.

[53] Pete Warden. 2017. Speech Commands: A public dataset for single-word speech recognition. (2017).

[54] Joel Wolfrath, Anirudh Achanta, and Abhishek Chandra. 2024. Leveraging Multi-Modal Data for Efficient Edge Inference Serving. In *2024 IEEE/ACM 24th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 408–417.

[55] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. 2019. MArk: Exploiting Cloud Services for Cost-Effective, SLO-Aware Machine Learning Inference Serving. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 1049–1062.

[56] Ziyang Zhang, Yang Zhao, and Jie Liu. 2023. Octopus: SLO-Aware Progressive Inference Serving via Deep Reinforcement Learning in Multi-tenant Edge Cluster. In *Service-Oriented Computing*, Flavia Monti, Stefanie Rinderle-Ma, Antonio Ruiz Cortés, Zibin Zheng, and Massimo Mecella (Eds.). Springer Nature Switzerland, Cham, 242–258.

[57] Kongyange Zhao et al. 2023. EdgeAdaptor: Online Configuration Adaption, Model Selection and Resource Provisioning for Edge DNN Inference Serving at Scale. *IEEE Transactions on Mobile Computing* 22, 10 (2023), 5870–5886.