Latency-aware Robust LLM Inference Workload Allocation under Precision-Dependent Uncertainty

Jiaming Cheng, *Student Member, IEEE* and Duong Tung Nguyen, *Member, IEEE*

***Abstract*—Large language model (LLM) inference entails intricate trade-offs among latency, accuracy, and cost. Higher-tier GPUs and higher-precision computation improve accuracy but increase processing delay and rental expense, while lower-tier or quantized configurations reduce cost at the expense of service quality. To manage these interdependencies, we propose a latency-aware robust optimization framework for LLM inference workload allocation under precision-dependent uncertainty. We develop a novel uncertainty model that explicitly links hardware tier and precision level to processing delay and inference error. The proposed formulation jointly optimizes model selection, GPU-tier provisioning, and workload allocation within a given budget, ensuring robustness against runtime variability. We derive tractable reformulations and scalable algorithms suitable for large heterogeneous GPU clusters. Simulation results on representative LLM and GPU settings demonstrate the three key trade-offs captured by our model—among latency, accuracy, and cost—highlighting its ability to realistically achieve lower worst-case latency and error rates while maintaining cost efficiency.**

***Index Terms*—LLM inference, GPU composition, delay uncertainty, error rate uncertainty, robust optimization.**

I. Introduction

Large language models (LLMs) have become the core of modern natural language processing (NLP) systems, achieving human-level performance in text generation, reasoning, translation, and code synthesis [1]. Pre-trained transformers such as GPT, LLaMA, and Mistral now underpin diverse AI applications, from conversational assistants to domain-specific copilots. An LLM's operation typically consists of two distinct phases: training and inference. During the training phase, the model learns language representations from large-scale datasets using extensive computational resources over weeks or months. In contrast, the inference phase applies the trained model to process new user queries and generate outputs in real time. Since inference occurs continuously and directly interacts with users, its efficiency critically influences service-level objectives (SLOs), user experience, and operational cost. Moreover, the computational footprint of inference scales with both the model size and the token length of each request—larger models require more GPU memory, higher precision, and longer processing pipelines [2]. As model scales and user demand continue to grow, these factors combine to create significant challenges in resource utilization, cost management, and service reliability.

In an LLM system, each user generates a query, or request, comprising a specific number of tokens. Input tokens represent the text or data provided by the user, such as questions, sentences, or code, which is tokenized for processing. Output tokens are the responses or results generated by the language model. Input tokens represents the text or data provided by the users such as questions, image snapshot, or code - which are tokenized and embedded for the model processing. The LLM then produces a sequence of output tokens that forms the generated response. To process these queries, the LLM service provider (SP) routes each incoming request to a suitable base language model hosted within a data center (DC). This base model, pre-trained on large language model (e.g., LLaMA-13B, LLaMA-70B, GPT-5-class models), performs a forward pass through multiple transformer layers, executing matrix multiplications and attention computation overall input tokens.

To serve a large volume of diverse requests, the SP rents heterogeneous GPUs from the cloud—ranging from low-tier devices such as NVIDIA A6000 to high-tier accelerators such as A100 and H100—each offering distinct compute power, memory capacity, precision support, and rental cost. When a query arrives, the SP routes it to an appropriate model–GPU configuration and performs inference through a forward pass across multiple transformer layers. This computation involves intensive matrix multiplications and attention operations executed on the selected GPUs. However, memory capacity quickly becomes a critical bottleneck in this process. Each transformer layer generates and stores intermediate key–value (KV) cache tensors for the attention mechanism, whose size scales linearly with the number of processed tokens and the precision level used. For large models or long-context queries, the combined memory of model weights and KV caches often exceeds the capacity of a single GPU. To mitigate this limitation, the SP adopts tensor parallelism (TP) and pipeline parallelism (PP). TP partitions large matrix operations across multiple GPUs to reduce per-device memory, while PP divides the model layers into sequential segments executed on different GPUs. Although these operations enable the execution of massive models, they introduce additional communication and synchronization overhead, which in turn increases inference latency.

To deliver low-latency inference, the SP must therefore jointly decide which GPU configuration to rent, which foundation model to deploy, and how to allocate workloads across these heterogeneous resources. However, these configuration decisions are inherently coupled and induce multiple, non-linear trade-offs that significantly influence performance and cost. Different GPU configurations—characterized by their computing power, memory capacity, and supported numerical precision—exhibit fundamentally different performance–cost behaviors. Higher-tier GPUs (e.g., A100, H100) deliver faster arithmetic throughput and larger memory bandwidth, allowing the deployment of larger models or higher-precision computation. However, these advantages come at a steeper rental cost and increased power consumption. These factors are interdependent. A more powerful GPU can offset part of the latency induced by high-precision computation, but it does so at a significantly

All authors are with the Ira A. Fulton Schools of Engineering, Arizona State University, Tempe, AZ, USA. Email: {jiaming,duongnt}@asu.edu.

higher economic cost; likewise, lowering precision can alleviate memory pressure but risks violating accuracy requirements, especially for complex or long-context queries. Moreover, higher-precision computation (e.g., FP32 or BF16) improves numerical accuracy and stability but requires more operations per token and consumes more memory for both model weights and KV cache storage, which in turn increases processing latency. Conversely, lower-tier GPUs or lower-precision modes (e.g., INT8, INT4) reduce per-token delay and cost, yet suffer from higher quantization error, leading to degraded service quality. As a result, the SP must navigate a multi-dimensional trade-off: increasing precision improves inference accuracy but prolongs latency; upgrading to more powerful GPUs accelerates processing but inflates cost; and lowering precision or hardware tier reduces expenses but degrades output quality. This intricate interdependence among latency, accuracy, and economic cost defines the core optimization challenge addressed in this study.

The computational demands and operational efficiency of LLM systems are fundamentally challenged by diverse optimization objectives arising from multiple dimensions, including stringent latency requirements, energy constraints, environmental sustainability goals, and geographic distribution complexities. Various aspects of resource allocation for inference workloads have been studied, including latency aware scheduling with SLO guarantees [3, 4, 5, 6], energy-efficient inference workload allocation [7, 8], carbon-aware workload distribution leveraging renewable energy availability [9, 10, 11], and locality-based optimization considering spatial-temporal request movement patterns [12, 13, 14]. These research directions collectively demonstrate a paradigm shift from single-objective performance optimization toward multi-dimensional frameworks that co-optimize across the entire computational stack while integrating sustainability or efficiency as first-class design constraints. However, one aspect often neglected in the literature is system uncertainties and their impacts on decisions. The inherent unpredictability of autoregressive generation processes, variable-length outputs, and heterogeneous service-level agreements (SLA) introduce additional complexities that traditional resource allocation strategies cannot adequately address. ***Contribution:*** To address these interdependent trade-offs, we introduce a novel robust optimization framework that explicitly captures the multi-interdependencies between configuration decisions and inference performance, including inference error rate and processing delay. In our formulation, both processing latency and inference error rate are modeled as uncertainty sets. This robust optimization problem jointly optimizes model selection, GPU-tier provisioning, and workload distribution across heterogeneous resources to account for the interdependence between accuracy, latency, and economic cost, achieving robust and cost-efficient inference performance under uncertain runtime conditions. We derive tractable reformulations of the robust optimization problem and design efficient solution algorithms that exploit problem structure to ensure scalability for large heterogeneous GPU clusters. The proposed approach maintains robustness while preserving computational efficiency suitable for practical inference orchestration. We validate our framework through extensive simulations using representative LLM configurations and GPU hardware profiles. Results demonstrate that our robust scheme significantly reduces worst-case latency and accuracy degradation compared to deterministic and heuristic allocation policies.

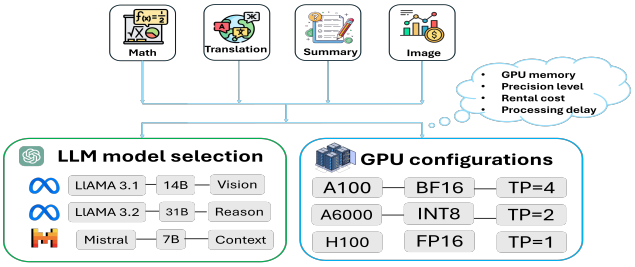


Fig. 1: System model

II. System Model and Problem Formulation

In this problem, the LLM service provider (SP) does not own dedicated computing infrastructure; instead, it rents GPU resources from a cloud infrastructure provider or a third-party platform to perform inference. Each user query belongs to a type $i \in \mathcal{I}$, representing distinct LLM tasks such as summarization, code generation, or image/video generation. The SP maintains a set of base language models $j \in \mathcal{J}$, which differ in size B_j, number of parameters, and fine-tuning checkpoints. Each query type i requires an average input of h_i tokens and produces an output of approximately f_i tokens after inference. To handle diverse workloads efficiently, the SP employs a pre-trained classifier that categorizes incoming prompts and routes them to appropriate model-GPU configurations. Inference job is executed on a resource tier $k \in \mathcal{K}$, where each tier represents a particular GPU configuration characterized by its hardware type and numerical precision level. Higher-tier configurations generally provide larger GPU memory capacity C_k^{GPU} and support higher-precision computation, but at a higher rental cost and longer processing time due to increased computational load. The total available storage at the DC is denoted by C^s. For each model j, the average key–value (KV) cache memory consumption per token is represented by β_j, while α_i denotes the average computational cost per token for processing query type i that converts token count into computational workloads, usually measured in floating point operation (FLOPs/token) for query i. During the inference operation, each input query must be processed by a particular base language model (e.g., LLaMA-13B, Mistral-7B) using a specific configuration tier, where a tier represents a combination of hardware configuration and numerical precision level (e.g., FP16, INT8, INT4).

The GPU configuration directly affects the inference accuracy and delay through several mechanisms. Lower-precision computation introduces quantization errors, increasing the inference error rate $e_{i,j}^k$. Limited memory capacity may also constrain attention fidelity and batch size, causing higher delay variability. Conversely, high-precision configurations reduce inference error but tend to increase per-token processing delay $d_{i,j}^k$ due to heavier computation. Both $d_{i,j}^k$ and $e_{i,j}^k$ are uncertain and are modeled through *decision-dependent uncertainty sets*, capturing the endogenous influence of SP's resource-allocation decisions on system performance. In this problem, the SP must make three interrelated decisions that jointly determine

TABLE I: Illustration of GPU configuration tiers and precision capabilities.

GPU Tier (k)	Compute Power (TFLOPs)	Memory (GB)	Supported Precision	Rental Cost ($/hr)
T4 (Low-tier)	8.1	16	FP16 / INT8	0.35
A10 (Mid-tier)	31.2	24	FP16 / INT8 / INT4	0.60
A100 (High-tier)	156	40	FP32 / FP16 / BF16 / INT8	2.50
H100 (Premium-tier)	197	80	FP32 / FP16 / BF16 / INT8 / INT4	3.90

its inference strategy. ***First***, in the GPU provisioning stage, the SP decides how many GPU instances of each configuration tier to rent from the platform. We define $z_{i,j}^k$ as the placement decisions where it equals 1 if type i queries are assigned to model j at tier k. The number of tier k GPUs rented for running model j is denoted by $y_{j,k}$. ***Second***, in the model deployment stage, the SP selects which foundation LLMs to load onto the provisioned GPUs for different query categories. This choice governs both the inference capability and the resource footprint of each model. In addition, each model j deployed on tier-k GPUs can further specify a tensor-parallelism degree $\text{TP}_{j,k}$ and a pipeline-parallelism degree $\text{PP}_{j,k}$ to balance computation and memory bottleneck. ***Finally***, in the workload allocation stage, given the deployed models and available GPU tiers, the SP distributes incoming queries among these configurations to achieve the desired balance between cost, delay, and inference accuracy. We define $x_{i,j}^k$ as the workload allocation variables which indicates the portion of type-i queries inferred by model j using the tier-k GPU configuration.

The SP's goal is to determine the optimal combination of $(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{u})$ and parallelism settings $(\mathbf{TP}, \mathbf{PP})$ that minimizes the total operating cost while satisfying delay, accuracy, and budget constraints. The overall cost includes GPU rental expenses, penalties associated with processing delay, and penalties for inference errors that degrade user experience. Formally, we will formulate the proposed optimization problem that **jointly considers *(i) model selection*, *(ii) GPU-tier provisioning*, and *(iii) workload distribution*** across heterogeneous resources, discussed as follows:

1) Resource rental: Given that the LLM SP lacks proprietary computing infrastructures, it must procure computational resources from external cloud platforms. Let p_k^{c} be the hourly rental rate for tier-k and Δ_T be the rental period.

$$\mathcal{C}_1(\mathbf{y}) = \Delta_T \sum_{j\in\mathcal{J}} \sum_{k\in\mathcal{K}} p_k^{\mathsf{c}} y_{j,k} \tag{1}$$

2) Storage cost: This component captures the storage expenses associated with hosting foundation language models. As most large language models are publicly available through open source library (e.g., Ollama), the primary deployment expense stems from storage requirements. Recall that B_j is denoted by the size of base model and θ_i by the average unit token size of type i queries, and p^s represents the hourly storage rate.

$$\mathcal{C}_2(\mathbf{x}, \mathbf{z}) = \Delta_T \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}} \sum_{k\in\mathcal{K}} p^s \left(B_j z_{i,j}^k + \theta_i (h_i + f_i) \lambda_i x_{i,j} \right) \tag{2}$$

3) Processing delay penalty Let $d_{i,j}$ represent the per-token processing delay for query type i by base model j. The processing delay is:

$$\mathcal{C}_3(\mathbf{x}) = \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}} \sum_{k\in\mathcal{K}} \rho_i d_{i,j}^k (h_i + f_i) \lambda_i x_{i,j}^k \tag{3}$$

4) Unmet demand penalty The objective captures the cost of unversed requests, where ϕ_i represents the penalty coefficient and u_i denotes the volume of unmet demand for query type i

$$\mathcal{C}_4(\mathbf{u}) = \sum_{i\in\mathcal{I}} \phi_i \lambda_i u_i \tag{4}$$

Thus, we can formulate the optimization problem as follows:

$$\mathcal{P}_1 : \min_{\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{u},TP} \max_{\boldsymbol{e}\in\mathcal{E},\boldsymbol{d}\in\mathcal{D}} \quad \mathcal{C}_1(\mathbf{y}) + \mathcal{C}_2(\mathbf{x},\mathbf{z}) + \mathcal{C}_3(\mathbf{x}) + \mathcal{C}_4(\mathbf{u}) \tag{5a}$$

$$s.t \quad \sum_{j\in\mathcal{J}} \sum_{k\in\mathcal{K}} x_{i,j}^k + u_i = 1, \ \forall i \tag{5b}$$

$$\Delta_T \sum_{i,j,k} \left[p_k^{\mathsf{c}} y_{i,j}^k + p^s \left(B_j z_{i,j}^k + \theta_i (h_i + f_i) \lambda_i x_{i,j}^k \right) \right] \leq \delta \tag{5c}$$

$$\sum_{n\in\mathcal{N}_k} w_{j,k}^n = q_{j,k}, \forall j, k \tag{5d}$$

$$TP_{j,k} = \sum_{n\in\mathcal{N}_k} n w_{j,k}^n, \forall j, k \tag{5e}$$

$$y_{j,k} = TP_{j,k} q_{j,k} = \sum_{n\in\mathcal{N}_k} n \cdot w_{j,k}^n q_{j,k}, \forall j, k \tag{5f}$$

$$\sum_{n\in\mathcal{N}_k} \frac{B_j}{n} w_{j,k}^n + \left(\sum_{n\in\mathcal{N}_k} \frac{\beta_j}{n} w_{j,k}^n \right) \sum_i r_i T_{i,j,k}^{\mathsf{res}} x_{i,j}^k \leq C_k^{\mathsf{GPU}} q_{j,k} \tag{5g}$$

$$\sum_i \alpha_i (f_i + h_i) x_{i,j}^k \leq P_k^{\mathsf{GPU}} y_{j,k}, \quad \forall j, k \tag{5h}$$

$$\sum_{j\in\mathcal{J}} \sum_{k\in\mathcal{K}} B_j z_{i,j}^k \leq C^{\mathsf{s}} \tag{5i}$$

$$\sum_{j\in\mathcal{J}} \sum_{k\in\mathcal{K}} \frac{d_{i,j}^k (h_i + f_i) x_{i,j}^k}{TP_{j,k}} \leq \Delta_i, \quad \forall i, \boldsymbol{d} \in \mathcal{D} \tag{5j}$$

$$\sum_{j\in\mathcal{J}} \sum_{k\in\mathcal{K}} e_{i,j}^k x_{i,j}^k \leq \epsilon_i, \ \forall i, \boldsymbol{e} \in \mathcal{E} \tag{5k}$$

$$0 \leq u_i \leq \zeta_i, \ \forall i \tag{5l}$$

$$\sum_{k\in\mathcal{K}} \sum_{j\in\mathcal{J}} z_{i,j}^k = 1, \quad \forall i \tag{5m}$$

$$0 \leq x_{i,j}^k \leq z_{i,j}^k, \quad z_{i,j}^k \leq q_{j,k}, \ \forall i, j, k \tag{5n}$$

$$TP_{j,k}, y_j^k \in \mathbb{Z}^{J\times K}, \ q_{j,k} \in \mathbb{B}_+^{J\times K}, z_{i,j}^k \in \mathbb{B}^{I\times J\times K}. \tag{5o}$$

Constraint (5b) shows that each type-i LLM queries can be either served or dropped. Economic budget constraints in (5c) ensure that the total expense for the rental and storage cost must be less than the certain budget threshold δ. In practice,

TP of different GPU tier k is typically selected from a certain range. Let $\mathcal{N}_k$ be the set of feasible TP for GPU tier k, where each value in $\mathcal{N}_k$ indicates allowed TP degrees on tier k GPU. Binary decision $w^n_{j,k} \in \{0,1\}$ indicates whether model j uses TP degree n on GPU tier k. Constraints (5d) ensures mutual exclusivity among feasible TP choices. If model j is not deployed on tier k, i.e., $q_{j,k}$, then all $w^n_{j,k} = 0$. If $q_{j,k} = 1$, exactly one TP configuration $n \in \mathcal{N}_k$ must be selected for that deployment. In Constraint (5e), the effective TP value is computed as the weighted sum of feasible options because only one $w^n_{j,k}$ can be active, $TP_{j,k}$ equals the selected degree n. In our model, we only allow one replica per model per GPU tier in the current setup. Thus, one replica of the model uses $TP_{j,k}$ GPUs. To this end, we can construct the relationship between TP and variable $y_{j,k}$ in (5f). Each GPU has a limited memory capacity C^{GPU}_k as stated in (5g). The total memory consumption can be intuitively understood as the sum of two main components. The first term $\frac{B_j}{TP_{j,k}}$ indicates the *model weight shard* stored on each GPU after tensor parallel splitting. The second term captures the memory used to store the attention keys and values, which grows with sequence length, batch size, and model depth. Let $T^{res}_{i,j,k}$ denote the residency time which is the average time that KV cache data remains in the GPU memory for a query of type i processed by model j on GPU tier k. Together, these ensure that the total memory used by model weights and KV-cache does not exceed the GPU's available capacity.

Constraint (5i) shows that all instances access the same storage pool, limited by maximum storage capacity C^{s}. The parameter α_i is the average compute consumption per token (FLOPS per token) and P^{GPU}_k is the FLOPs of tier k GPU. Constraint (5h) enforces that the total computational demand from all incoming queries assigned to model j on GPU tier k does not exceed the total computational capacity provided by the rented GPU. Constraint (5i) ensures that all model weight parameters can be well-saved in the storage system. To ensure a satisfactory user experience, the SP can impose a certain threshold Δ_i on the average processing delay, as shown in (5j). Constrains (5k) shows that the weighted average error rate for type i queries must be within ϵ_i. Constraints (5m) enforce that each query type i is assigned to exactly one base model and exactly one GPU configuration (i.e., single precision level and GPU type). Note that this problem is within the class of RO problem, where (5j) - (5k) are robust constraints that must satisfy for all uncertain realizations within uncertainty sets $\mathcal{D}$ and $\mathcal{E}$. To further control the unmet portion, the SP may proactively impose unmet portion to be less than certain threshold ζ_i as shown in (5l). Workload allocation variables $\mathbf{x} \in [0,1]$ can be only allocated to the active model and GPU configuration and logical consistency between query routing and deployment decisions, shown in (5n), while the rest of integer and binary variables are defined in (5o).

Uncertainty modeling: In practice, both the processing delay $d^k_{i,j}$ and inference error rate $e^k_{i,j}$ are uncertain parameters due to hardware variability, and precision levels effects. The SP must therefore choose resource configurations that balance cost, latency and accuracy within budget δ. Higher-tier hardware can reduce delay variability and errors but increases cost, while higher precision improves accuracy at the expense of longer processing time. To model these uncertainties, we aim to capture both processing delay and inference error rate by using uncertainty sets embedded in a robust optimization framework, enabling the SP to minimize operating cost while adhering to service-level commitments and resilience under worst-case conditions.

1) Processing delay uncertainty: The actual processing delay of $d^k_{i,j}$ are often uncertain and exhibits significant variability over time. Assuming that processing delays are fixed and known may lead to suboptimal solutions, potentially degrading user experience. This delay uncertainty set of processing delay can be characterized as follows:

$$\mathcal{D}(\hat{d}, \bar{d}, \Gamma_d) := \Big\{ d^k_{i,j} :\ 0 \le g^k_{i,j} \le 1, \forall i,j,k \tag{6a}$$

$$d^k_{i,j} = \bar{d}^k_{i,j} + \hat{d}^k_{i,j} g^k_{i,j}, \forall i,j,k;\ \sum_{i,j,k} g^k_{i,j} \le \Gamma_d \Big\} \tag{6b}$$

where $\bar{d}^k_{i,j}$ denotes the average nominal delay for processing type i query on model j based on tier k GPU configuration, and $\hat{d}^k_{i,j}$ represents its deviation. The parameter Γ_d represents the uncertainty budget controlling the conservativeness of the delay model. A larger Γ_d enlarges the feasible uncertainty region, yielding more robust yet conservative decisions.

2) Error rate uncertainty: The actual error rate $e^k_{i,j}$ can vary due to hardware heterogeneity and precision level configurations. We define $\mathcal{E}$ as a decision-dependent uncertainty set capturing this variability, i.e.,

$$\mathcal{E}(\hat{e}, \bar{e}, \Gamma_d) := \Big\{ e^k_{i,j} :\ 0 \le s^k_{i,j} \le 1,\ \forall i,j,k \tag{7a}$$

$$e^k_{i,j} = \bar{e}^k_{i,j} + \hat{e}^k_{i,j} s^k_{i,j}, \forall i,j,k;\ \sum_{i,j,k} s^k_{i,j} \le \Gamma_e \Big\} \tag{7b}$$

where $\bar{e}^k_{i,j}$ is denoted by the average nominal error rate for processing type i input query on model j and tier k GPU configuration, and $\hat{e}^k_{i,j}$ is denoted by the deviation of the error rate. The parameter Γ_e represents the uncertain budget controlling the conservativeness of the error rate uncertainty. The SP can jointly adjust Γ_d and Γ_e to trade off between robustness and performance.

III. Solution Approach

This section develops an exact reformulation for solving the robust model (5) efficiently Due to its nonlinear nature, this form poses challenges in reformulating this problem in a tractable manner. In fact, it has been shown that this robust model ($\mathcal{P}_1$) is NP-complete [15]. Additionally, the primary difficulty is that the uncertain parameter $d^k_{i,j}$ and $e^k_{i,j}$ appears in the robust constraints (5j) and (5k), while $d^k_{i,j}$ also appears in the objective components ($\mathcal{C}_3$). To address this challenge, we introduce an exact approach to solve the proposed robust problem, where the main idea is to convert (5j) and (5k) into an equivalent and solvable MILP form, which enables us to utilize off-the-shelf solvers.

We focus on transforming the robust constraints (5j) and (5k) into a set of linear constraints that satisfy all uncertain realizations within the predefined uncertainty set $\mathcal{D}$. Specifically,

based on (5j) and (5e), the robust delay constraints (5j) can be rewritten as:

$$\Delta_i \sum_{\substack{n\in\mathcal{N}_k,\\ j\in\mathcal{J},k\in\mathcal{K}}} n\cdot w^n_{j,k} \geq \sum_{j,k} d^k_{i,j}(f_i+h_i)x^k_{i,j} \quad \forall i, \forall \boldsymbol{d}\in\mathcal{D} \quad (8)$$

Let $r_i = f_i + h_i, \forall i$. Thus, for all i, the last term in (8) can be expressed as:

$$\max_{\boldsymbol{g}\geq 0} \quad \sum_{j,k} \bar{d}^k_{i,j} r_i x^k_{i,j} + \hat{d}_{i,j} r_i x^k_{i,j} g_{i,j} \quad (9a)$$

$$\text{s.t.} \quad \sum_{i,j,k} g^k_{i,j} \leq \Gamma_d, \qquad (\tau_{i,0}) \quad (9b)$$

$$g^k_{i,j} \leq 1, \ \forall i,j,k, \qquad (\sigma^k_{i,j,0}) \quad (9c)$$

where $\tau_{i,0}$ and $\sigma^k_{i,j,0}$ are dual variables associated with constraints (9b) and (9c), respectively. By employing LP duality [15], we can express (5j) as follows:

$$\Delta_i \sum_{\substack{n\in\mathcal{N}_k\\ j\in\mathcal{J},k\in\mathcal{K}}} n\cdot w^n_{j,k} \geq \sum_{j,k} \bar{d}^k_{i,j} r_i x^k_{i,j} + \Gamma_d \tau_{i,0} + \sum_{j,k} \sigma^k_{i,j,0}, \forall i \quad (10a)$$

$$\tau_{i,0} + \sigma^k_{i,j,0} \geq \hat{d}^k_{i,j} r_i x^k_{i,j}, \forall i,j,k \quad (10b)$$

$$\tau_{i,0} \geq 0, \ \forall i; \quad \sigma^k_{i,j,0} \geq 0, \ \forall i,j,k \quad (10c)$$

Similarly, we repeat the same step for the error rate constraints in (5k), we will obtain the following constraints:

$$\epsilon_i \geq \sum_{j,k} \bar{e}_{i,j} x^k_{i,j} + \Gamma_e \tau_{i,1} + \sum_j \sigma_{i,j,1}, \forall i \quad (11a)$$

$$\tau_{i,1} + \sigma_{i,j,1} \geq \hat{e}_{i,j} x^k_{i,j}, \quad \forall i,j,k \quad (11b)$$

$$\tau_{i,1} \geq 0, \ \forall i; \quad \sigma^k_{i,j,1} \geq 0, \ \forall i,j,k \quad (11c)$$

Similarly, we follow the same procedure for ($\mathcal{C}_3$) in objective function, which can be rewritten by the following form:

$$\min_{\mathbf{x},\mathbf{u},\mathbf{y},\mathbf{z},\mathbf{w}} \quad \mathcal{C}_1 + \mathcal{C}_2 + \mathcal{C}_4 + \varrho \quad (12a)$$

$$\text{s.t } \varrho \geq \max_g \sum_{i,j,k} \rho_i r_i \big(\bar{d}_i x^k_{i,j} + \hat{d}_i g_i x^k_{i,j}\big) \quad (12b)$$

The last term in (12b) can be written explicitly as follows:

$$\max_{\boldsymbol{g}\geq 0} \sum_{i,j,k} \rho_i r_i \hat{d}^k_{i,j} g^k_{i,j} x^k_{i,j} \quad (13a)$$

$$\text{s.t.} \quad \sum_{i,j,k} g^k_{i,j} \leq \Gamma_d, \qquad (\tau_{i,2}) \quad (13b)$$

$$g^k_{i,j} \leq 1, \ \forall i,j,k. \qquad (\sigma^k_{i,j,2}) \quad (13c)$$

where $\tau_{i,2}$ and $\sigma^k_{i,j,2}$ are the dual variables associated with constraints (13b) and (13c), respectively. By employing LP duality [15], we can express (13) :

$$\varrho \geq \sum_i \Gamma_d \tau_{i,2} + \sum_{i,j,k} \sigma^k_{i,j,2} \quad (14a)$$

$$\tau_{i,2} + \sigma^k_{i,j,2} \geq \rho_i \hat{d}^k_{i,j} r_i x^k_{i,j}, \forall i,j,k \quad (14b)$$

$$\tau_{i,2} \geq 0, \ \forall i; \ \sigma^k_{i,j,2} \geq 0, \forall i,j,k. \quad (14c)$$

Finally, we can reformulate ($\mathcal{P}_1$) in (5) as the following MILP, which can be efficiently solved by solvers.

$$\mathcal{P}_{RO}: \min_{\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{u},TP} \quad \mathcal{C}_1(\mathbf{y}) + \mathcal{C}_2(\mathbf{x},\mathbf{z}) + \mathcal{C}_4(\mathbf{u}) + \varrho \quad (15a)$$

$$\text{s.t. } (5b)-(5i),(5m)-(5o),(10),(11),(14). \quad (15b)$$

IV. Numerical Results

A. Simulation Setting

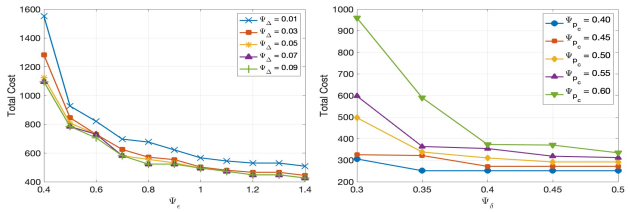
We evaluate a large scale LLM inference SP that severs query types by dynamically selecting suitable foundation models and GPU configurations from a GPU resource pool comprising heterogeneous configurations. The system supports supports $I = 6$ query types with demand λ_i following empirical distribution observed in workloads [2]. Text-based tasks exhibit high demand $\lambda_i = U[80, 140], \forall i$ queries per hour, while multi-model tasks show moderate but latency critical demand $\lambda_i = [10, 60]$ queries per hour. In addition, there are totally $J = 6$ types Llama-3 models are considered, ranging from Llama-3.2-1B (2GB) to Llama-3.1-70B (140GB). The GPU pool comprises $K = 10$ heterogeneous configurations including RTX-4090, A6000, A100, and H100 devices supporting BF16/INT8/INT4 precision. The computing and memory capacity are generated based on the specifications of selected GPUs, with memory ranging from $P^{\mathsf{GPU}}_k = [24, 80]GB$ and computing from $C^{\mathsf{GPU}}_k = [40.7, 1483.5]$ TFLOPs. GPU rental costs p^c_k is generated from $0.35\$ - 2.5\$$ per hour with based on vast.ai [1]. Tensor parallelism of degree $\mathcal{N} = \{1, 2, 4, 8\}$ is supported for distributed inference. Average unit memory consumption β_j is set to be $\{0.02, 0.05, 0.15, 1.4, 8, 0.25\}$ GB per token and unit computing consumption $\alpha_i = [0.3, 4.5]$ TFLOPS per 1000 tokens capture workload intensity. Followed by previous work [2, 11], we generate input and output tokens from ranges $h_i = [32, 512]$ and $f_i = [128, 2048]$ respectively. QoS threshold are specified as average delay limits $\Delta_i = [1, 1.5, 0.8, 2, 4, 5]$ s and error thresholds $\epsilon_i = [4, 8, 3, 10, 12, 15]\%$. Penalty parameter include delay cost $\rho_i = 0.0003 - 0.006\$/ms$ and unmet demand penalty $\phi_i = U[5, 30]$ / query. The total operational budget is $\delta = \$3000$ for scheduling horizon $\Delta_T = 12$, including storage cost $p_s = U[0.003, 0.008]$ (GB $\cdot$ hour). Uncertain parameters (Γ_d, Γ_e) are set to 50 and a random seed of 42 for reproducibility. Due to limited space, all detailed parameter setup and implementations can be found in https://github.com/JJmingcc/LLM-RO. All the experiments are implemented in Python environment using Gurobipy on an M3 Macbook pro and 16 GB of RAM.

B. Trade-off evaluations

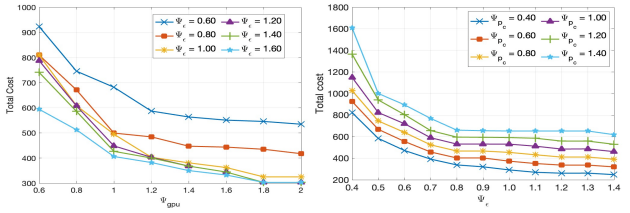
This section will highlight three trade-offs highlighted of the model: *1) error rate vs. processing delay*, *(2) Model complexity vs. latency* and *(3) rental cost vs. budget* . To access how system behavior shifts under different operating conditions, a scaling factor $\Psi_\cdot$ is introduced for the evaluated system parameters, where the subscript indicates the parameter aiming to scale.

1) Trade-off: error rate vs. processing delay: Figs.2(a) shows that relaxing QoS constraints substantially reduce cost, with diminishing returns beyond moderate threshold levels. As the error rate/average delay threshold increases, the total cost drops markedly because the SP can meet the relaxed accuracy requirement without employing higher-tier GPU configurations. Consequently, the SP can reduce GPU rental expenses and avoid substantial penalties associated with unmet demand.

[1] https://cloud.vast.ai/



(a) Varying Δ_i and ϵ_i (b) Varying δ and p_k^c



(c) Varying C_k^{GPU} and ϵ_i (d) Varying p_k^c and ϵ_i

Fig. 2: Impact of system parameters

When the accuracy requirement becomes more stringent than delay, the SP must use larger base models to maintain acceptable error rates, increasing GPU rental and storage costs. When delay is the binding constraint, additional GPUs or faster interconnects are required to meet latency targets, even if accuracy is easily satisfied. When both requirements becomes more stringent, the total cost usually behave higher because of higher unmet penalty. At intermediate QoS levels, both factors jointly influence cost and resource allocation, often leading to a non-monotonic GPU provisioning pattern.

2) Trade-off: Rental cost vs. Budget: Fig.2(b) illustrate how total cost respond to variations in the operational budget δ and GPU rental cost p_k^c. Results indicate a nonlinear relationship characterized by three regimes: i) a penalty-dominated under provisioned regime where unmet demand drives total cost; the SP prioritizes only the cheapest GPUs and small model variants (Llama-3.2-1B/3B). ii) a transition regime where increasing budget sharply reduces total cost through additional GPU rentals, reflecting the switch from resource scarcity to full capacity utilization; iii) a saturation regime where costs flatten once capacity becomes sufficient or the rental cost is low.

3) Trade-off: error rate vs. Memory capacity: Figs.2(c) - 2(d) reveals sensitivity analysis on GPU memory capacity (C_k^{GPU}) and GPU rental cost (p_k^c) with varying error rate threshold. Increasing per-GPU memory consistently decreases total cost since the SP may deploy large base models without aggressive quantization precision level. When error rate constraints are tighter, the SP often seeks for higher-performance GPUs with higher order of Tensor processing capability, enabling model parallelization while increasing storage. Furthermore, Table II presents the cost breakdown for selected scenarios under a fixed memory capacity and varying error-rate thresholds. All computation instances complete within one second. As the error-rate requirement becomes more stringent, the SP tends to select a TP degree of 8 to prevent memory overflow. We also observe that unmet penalty decreases monotonically as ϵ_i is relaxed. However, the rental cost does not necessarily follow this trend, since the SP may opt to pay higher rental fees to maintain QoS, given that the unmet penalty carries greater weight than the GPU unit rental cost.

$(\Psi_{C_{gpu}}, \Psi_\epsilon)$	Time (s)	C_total	C_rental	C_unmet	TP
(0.8,0.6)	0.49	745.52	362.67	289.84	(1,1,1,2)
(0.8,0.8)	0.46	671.12	393.33	193.24	(1,0,1,3)
(0.8,1.0)	0.39	608.51	387.39	147.46	(1,0,0,3)
(0.8,1.2)	0.58	607.47	386.84	146.02	(1,0,2,2)
(0.8,1.4)	0.62	585.79	370.05	146.02	(1,0,2,2)
(0.8,1.6)	0.39	512.55	436.95	18.09	(1,0,0,3)
(0.8,1.8)	0.27	479.79	422.72	0	(1,1,0,2)

TABLE II: Cost breakdown for selected scenarios

V. Conclusion

This paper introduces a robust optimization framework for allocating LLM inference workloads under accuracy and delay uncertainty. By explicitly modeling the relationship between hardware tier, computation precision, processing delay, and inference error, the proposed approach captures key trade-offs among latency, accuracy, and cost in large-scale GPU clusters. The tractable reformulations enable efficient decision-making even in heterogeneous and uncertain runtime environments. Simulation results validate three key trade-offs captured by the proposed model—among latency, accuracy, and cost—demonstrating its effectiveness in achieving lower worst-case latency and error rates while maintaining cost efficiency.

References

[1] A. A. Chien, L. Lin, H. Nguyen, V. Rao, T. Sharma, and R. Wijayawardana, "Reducing the carbon impact of generative ai inference (today and in 2035)," in *Proceedings of the 2nd workshop on sustainable computer systems*, 2023, pp. 1–7.

[2] G. Wilkins, S. Keshav, and R. Mortier, "Offline energy-optimal llm serving: Workload-based energy models for llm inference on heterogeneous systems," *ACM SIGENERGY Energy*, vol. 4, no. 5, pp. 113–119, 2024.

[3] Y. Zhao, J. Chen, P. Sun, L. Li, X. Liu, and X. Jin, "Seallm: Service-aware and latency-optimized resource sharing for large language model inference," *arXiv preprint arXiv:2504.15720*, 2025.

[4] T. Xia, Z. Mao, J. Kerney, E. J. Jackson, Z. Li, J. Xing, S. Shenker, and I. Stoica, "Skylb: A locality-aware cross-region load balancer for llm inference," *arXiv preprint arXiv:2505.24095*, 2025.

[5] A. K. Kakolyris, D. Masouros, P. Vavaroutsos, S. Xydis, and D. Soudris, "Slo-aware gpu frequency scaling for energy efficient llm inference serving," *arXiv preprint arXiv:2408.05235*, 2024.

[6] T. Wallace, B. Ombuki-Berman, and N. Ezzati-Jivan, "Optimization strategies for enhancing resource efficiency in transformers & large language models," in *Proc. ACM ICPE*, 2025, pp. 105–112.

[7] J. Stojkovic, C. Zhang, Í. Goiri, J. Torrellas, and E. Choukse, "Dynamollm: Designing llm inference clusters for performance and energy efficiency," *arXiv preprint arXiv:2408.00741*, 2024.

[8] Y. Jiang, F. Fu, X. Yao, G. He, X. Miao, A. Klimovic, B. Cui, B. Yuan, and E. Yoneki, "Demystifying cost-efficiency in llm serving over heterogeneous gpus," *arXiv preprint arXiv:2502.00722*, 2025.

[9] G. Wilkins, "Online workload allocation and energy optimization in large language model inference systems," 2024.

[10] J. Stojkovic, E. Choukse, C. Zhang, I. Goiri, and J. Torrellas, "Towards greener llms: Bringing energy-efficiency to the forefront of llm inference. arxiv 2024," *arXiv preprint arXiv:2403.20306*.

[11] J. Cheng and D. T. Nguyen, "Green-llm: Optimal workload allocation for environmentally-aware distributed inference," *arXiv preprint arXiv:2507.09942*, 2025.

[12] X. Zhang, Y. Yang, and D. Wang, "Spatial-temporal embodied carbon models for the embodied carbon accounting of computer systems," in *ACM e-energy*, 2024, pp. 464–471.

[13] K. Kim, J. Kim, H. Chung, M.-H. Cha, H.-Y. Kim, and Y. Kim, "Cost-efficient llm serving in the cloud: Vm selection with kv cache offloading," *arXiv preprint arXiv:2504.11816*, 2025.

[14] W. Borui, Z. Juntao, J. Chenyu, G. Chuanxiong, and W. Chuan, "Efficient llm serving on hybrid real-time and best-effort requests," *arXiv preprint arXiv:2504.09590*, 2025.

[15] A. Ben-Tal, A. Goryashko, E. Guslitzer, and A. Nemirovski, "Adjustable robust solutions of uncertain linear programs," *Mathematical programming*, vol. 99, no. 2, pp. 351–376, 2004.