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 involves 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 decision-dependent uncertainty (DDU) model that links hardware tier and precision level to processing delay and inference error. The 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 efficient algorithms scalable to large heterogeneous GPU clusters. Simulation results on representative LLM and GPU settings show that the proposed approach achieves significantly lower worst-case latency and error rates than deterministic or heuristic baselines while maintaining cost efficiency.**

***Index Terms*—Edge computing, processing delay variability, uncertain delay, network hardening, robust optimization, decision-dependent uncertainty.**

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 LLM SP rents heterogeneous GPUs from the cloud—ranging from low-tier devices such as NVIDIA T4 and A10 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 footprint 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 load, 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, nonlinear 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

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

interdependent. A more powerful GPU can offset part of the latency induced by high-precision computation, but it does so at a significantly 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., FP16 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.

Contribution: To address these interdependent trade-offs, we introduce a novel precision-dependent decision-dependent uncertainty (DDU) framework that captures the multi-endogenous relationships between configuration decisions and inference performance. In our formulation, both processing latency and inference error rate are modeled as decision-dependent uncertainty sets whose bounds vary with GPU tier and precision level. We formulate a latency-aware robust optimization problem that jointly optimizes model selection, GPU-tier provisioning, and workload distribution across heterogeneous resources. The proposed model explicitly accounts for the interdependence between accuracy, latency, and economic cost, enabling the SO to achieve robust and cost-efficient inference performance under uncertain runtime conditions. We derive tractable reformulations of the decision-dependent 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.

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 (e.g., A10, A100, H100) and numerical precision level (e.g., FP16, INT8, INT4). 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 SSD storage capacity at the DC is denoted by C^{SSD}. 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 resource 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 (e.g., INT8 or INT4 quantization) introduces quantization errors, increasing the inference error rate e_i. 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 due to heavier computation. Both d_i and e_i 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 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 $\mathrm{TP}_{j,k}$ and a pipeline-parallelism degree $\mathrm{PP}_{j,k}$ 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 number 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 provi-

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

sioning, 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.

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

Specifically, TP of different GPU tier k is typically selected from a certain range in practice. 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 $w_{j,k}^n \in \{0,1\}$ indicates whether model j uses TP degree n on GPU tier k.

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 represent the per-token processing delay for query type i. The processing delay is:

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

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

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

Thus, we can formulate the optimization problem as follows:

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

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

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

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

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

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

$$z_{i,j}^k \le q_{j,k}, \ \forall i,j,k \tag{5g}$$

$$\frac{B_j}{TP_{j,k}} + \sum_j \beta_j (h_i + f_i) \lambda_i T_{i,j,k}^{\mathsf{res}} \frac{x_{i,j}^k}{y_{j,k}} \le C_k^{\mathsf{GPU}} y_{j,k}, \ \forall j,k \tag{5h}$$

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

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

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

$$\sum_{j\in\mathcal{J}} \sum_{k\in\mathcal{K}} e_i \lambda_i x_{i,j}^k \le \epsilon_i, \ \forall i, \boldsymbol{e} \in \mathcal{E} \tag{5l}$$

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

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

$$0 \le x_{i,j}^k \le z_{i,j}^k, \ \forall i,j,k \tag{5o}$$

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

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 $w_{j,k}^n \in \{0,1\}$ 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 $w_{j,k}^n = 0$. 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 $w_{j,k}^n$ 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). Constraint $5g$ guarantees logical consistency between query routing and deployment decisions. Each GPU has limited C_k^{GPU} for tier-k configuration, as shown in (5h). The first term $\frac{B_j}{TP_{j,k}}$ indicates the model weight shard, which the portion of model weights stored on each GPU, while the second term means the total memory needed to store attention keys and values (KV-cache). By utilizing relationship from ((5e)) - ((5g)), $\forall j,k$, we have the following new memory capacity constraints:

$$\sum_{n\in\mathcal{N}_k} \frac{B_j}{n} w_{j,k}^n + \left(\sum_{n\in\mathcal{N}_k} \frac{\beta_j}{n} w_{j,k}^n \right) \sum_i r_i T_{i,j,k}^{\mathsf{res}} x_{i,j}^k \le C_k^{\mathsf{GPU}} q_{j,k} \tag{6}$$

where let $r_i = (h_i + f_i)\lambda_i, \forall i$. Constraint (5j) shows that all instances access the same storage pool, limited by maximum storage capacity C^{s}. Let α_i be the average compute

consumption per token (FLOPS per token) and P_k^{GPU} be FLOPs / s per GPU for tier k. Constraint (5i) enforces that the total computational demand (FLOPs/s) from all incoming queries assigned to model j on GPU tier k does not exceed the total computational capacity (FLOPs/s) provided by the rented GPU. Constraint (5j) 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 (5k). Constrains (5l) shows that the weighted average error rate for type i queries must be within ϵ_i. Constraints (5m) - (5n) enforce that each query type i is assigned to exactly one base model and exactly one GPU configuration (i.e., a single precision level and GPU type). Note that this problem is within the class of RO problem, where (5k) - (5l) are robust constraints that must satisfy for all uncertain realizations within predefined uncertainty sets $\mathcal{D}$ and $\mathcal{E}$.

Endogenous uncertainty: In practice, both the processing delay d_i and inference error rate e_i 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 B. Since $\boldsymbol{d}$ and $\boldsymbol{e}$ are endogenous to the SP's decisions: 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 this endogeneity, we introduce a decision-dependent uncertainty set 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_i 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 decision-dependent uncertainty set of processing delay can be characterized as follows:

$$\mathcal{D}(\hat{d}, \bar{d}, \mathbf{z}, \Gamma) := \Big\{ d_i : \ 0 \le g_i \le 1 + \sum_{j\in\mathcal{J}} \sum_{k\in\mathcal{K}} \gamma_j^k z_{i,j}^k, \tag{7a}$$

$$d_i = \bar{d}_i + \hat{d}_i g_i, \forall i; \ \sum_i g_i \le \Gamma_d \Big\} \tag{7b}$$

where $\bar{d}_i$ denotes the nominal baseline delay, and $\hat{d}_i$ represents its deviation. The coefficient γ_j^k captures the impacts of the resource-tier selection on the variability of the delay. Specifically, higher-tier configurations (e.g., models operating at higher precision levels or with more complex GPU setups) generally increase processing time. Within this uncertainty set, the upper bound of g_i increases from 1 to $1 + \sum_{j,k} \gamma_j^k z_{i,j}^k$, illustrating that appropriate configuration decisions can partially constrain delay variation. 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_i 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.,

$$\mathcal{E}(\hat{e}, \bar{e}, \mathbf{z}, \Gamma) := \Big\{ e_i : \ 0 \le s_i \le 1 - \sum_{k\in\mathcal{K}} \sum_{j\in\mathcal{J}} \eta_j^k z_{i,j}^k, \tag{8a}$$

$$e_i = \bar{e}_i + \hat{e}_i s_i, \forall j; \ \sum_i s_i \le \Gamma_e \Big\} \tag{8b}$$

where $\bar{e}_i$ is denoted by the nominal (baseline) error rate corresponding to the lowest-level GPU configuration, and $\hat{e}_i$ is denoted by the potential deviation of the error rate from this baseline. The coefficient η_j^k quantifies the impact of resource-tier selection on the inference accuracy. Higher-tier hardware configurations (indexed by k) generally provide stronger error-rate reduction capabilities. Consequently, the maximum uncertainty scaling factor s_i can be reduced from 1 to $1 - \sum_{j,k} \gamma_j^k z_{i,j}^k$, reflecting the performance gain achieved through improved resource selection. This formulation highlights that an appropriate resource tier not only enhances inference accuracy but also mitigates the worst-case processing delay. While higher-tier resources improves the accuracy, the expected and worst-case delays both increase due to heavier computational loads. 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. The endogenous uncertainty presents substantial challenges in solving (5) as the uncertainty set (8) since it introduce decision variables that need to be determined in the outer minimization problem. 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 [**?**]. Additionally, the primary difficulty is that the uncertain parameter d_i and e_i appears in the robust constraints (5k) and (5l), while d_i also appears in the objective components ($\mathcal{C}_3$). Without variables in the uncertainty set, conventional methods facilitate the reformulation of the problem into an equivalent mixed-integer linear programming (MILP) by leveraging the strong duality [3]. To address this challenge, we introduce a ***two-step*** transformation to convert (5k) and (5l) into an equivalent and solvable MILP form, which enables us to utilize off-the-shelf solvers to solve the problem efficiently.

Step 1: Robust constraint reformulation: We focus on transforming the robust constraints (5k) and (5l) into a set of linear constraints that satisfy all uncertain realizations within the predefined uncertainty set $\mathcal{D}$. Specifically, based on (5k), the robust delay constraints (5k) can be rewritten as:

$$\Delta_i \ge \sum_{j,k} d_i (f_i + h_i) \lambda_i x_{i,j}^k \quad \forall i, \forall \boldsymbol{d} \in \mathcal{D} \tag{9}$$

Let $r_i = (f_i + h_i)\lambda_i, \forall i$. Thus, for all i, the last term in (9) can be expressed as:

$$\max_{\boldsymbol{g}\ge 0} \quad \sum_{j,k} \bar{d}_i r_i x_{i,j}^k + \hat{d}_i r_i x_{i,j}^k g_i \tag{10a}$$

$$\text{s.t.} \quad \sum_j g_i \le \Gamma_d, \qquad (\tau^s) \tag{10b}$$

$$g_i \le 1 + \sum_{j,k} \gamma_j^k z_{i,j}^k, \ \forall i, \qquad (\sigma_{i,0}) \tag{10c}$$

where, for each area i, τ_0 and $\sigma_{i,0}$ are dual variables associated with constraints (10b) and (10c), respectively. By employing LP duality [4], we can express (5k) as follows:

$$\Delta_i \geq \sum_{j,k} \bar{d}_i r_i x^k_{i,j} + \Gamma_d \tau_0 + \sum_i \sigma_{i,0} - \sum_{j,k} \gamma^k_j \boldsymbol{z^k_{i,j}\sigma_{i,0}}, \forall i \quad (11a)$$

$$\tau_0 + \sigma^0_i \geq \hat{d}_i r_i x^k_{i,j}, \ \forall i,j,k; \quad \tau_0 \geq 0; \ \sigma_{i,0} \geq 0, \ \forall i \quad (11b)$$

where $\boldsymbol{z^k_{i,j}\sigma_{i,0}}$ is the bilinear term that needs to be linearized in the second step. Similarly, we repeat the same step for the error rate constraints in (5l), we will obtain the following constraints:

$$\epsilon_i \geq \sum_{j,k} \bar{e}_i \lambda_i x^k_{i,j} + \Gamma_e \tau_1 + \sum_i \sigma_{i,1} - \sum_{j,k} \eta^k_j \boldsymbol{z^k_{i,j}\sigma_{i,1}}, \forall i, \ (12a)$$

$$\tau_1 + \sigma_{i,1} \geq \hat{e}_i \lambda_i x^k_{i,j}, \quad \forall i,j,k; \quad \tau_1 \geq 0; \ \sigma_{i,1} \geq 0, \ \forall i \ (12b)$$

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

$$\min_{\mathbf{x},\mathbf{u},\mathbf{y}} \ \mathcal{C}_1 + \mathcal{C}_2 + \mathcal{C}_4 + \varrho \quad (13a)$$

$$\text{s.t } \varrho \geq \max_g \sum_{i,j,k} \rho_i r_i \big(\bar{d}_i x^k_{i,j} + \hat{d}_i g_i x^k_{i,j}\big) \quad (13b)$$

The last term in (13b) can be written explicitly as follows:

$$\max_{\boldsymbol{g}\geq 0} \sum_{i,j,k} \rho_i r_i \hat{d}_i g_i x^k_{i,j} \quad (14a)$$

$$\text{s.t.} \quad \sum_i g_i \leq \Gamma_d, \qquad (\tau_2) \quad (14b)$$

$$g_i \leq 1 - \sum_{j,k} \gamma^k_j z^k_{i,j}, \ \forall i, \qquad (\sigma_{i,2}) \quad (14c)$$

where τ_2 and $\sigma_{i,2}$ are the dual variables associated with constraints (14b) and (14c), respectively. By employing LP duality, we can express (14) :

$$\varrho \geq \ \sum_{i,j,k} \Gamma_d \tau_2 + \sum_i \sigma_{i,2} - \sum_{j,k} \gamma^k_j \boldsymbol{z^k_{i,j}\sigma_{i,2}} \quad (15a)$$

$$\tau_2 + \sigma_{i,2} \geq \rho_i \hat{d}_i r_i x^k_{i,j}, \forall i,j,k; \quad \tau_2 \geq 0; \ \sigma_{i,2} \geq 0, \forall i. \ (15b)$$

where $\boldsymbol{z^k_{i,j}\sigma_{i,2}}$ is another bilinear term.

Step 2 - Addressing Bilinear terms: Our goal is to linearize the following bilinear terms:

$$Z^s_{i,j,k} = z^k_{i,j}\sigma_{i,s}, \quad \forall i,j,k,s \in \{0,1,2\} \quad (16)$$

where $Z^s_{i,j,k}$ is the product of a continuous variable ($\sigma_{i,s}$) and a binary variable ($z^k_{i,j}$). For clarity in notation, we use the superscript s to denote variables corresponding to different sets of constraints, where 0 - 2 corresponds to associated dual variables mentioned in *Step 1*. By utilizing the McCormick linearization, the bilinear term $z^k_{i,j}\sigma^s_i$ can be equivalently implemented as follows:

$$Z^s_{i,j,k} \leq M z^k_{i,j}, Z^s_{i,j,k} \leq \sigma_{i,s}, \forall i,j,k,s \quad (17a)$$

$$0 \leq Z^s_{i,j,k} \geq \sigma_{i,s} - M(1 - z^k_{i,j}), \ \forall i,j,k,s \quad (17b)$$

where M is a sufficiently large number. The choice of M can be either determined by its natural upper bound or the trial-and-error method (choosing a sufficiently large M) if there is no available upper bound. However, choosing the big-M constant too small can result in a suboptimal solution of the original problem. On the other hand, excessively large values may produce "solutions" that are infeasible for the original problem. This occurs because the products in the linearization constraints, involving large coefficients and binary variables, can be relaxed within a certain tolerance by solvers.

Many existing studies do not address the selection of constants or rely on trial-and-error methods, which provide no guarantee of correct reformulation. To enhance computational efficiency, we propose an effective technique using the type-1 SOS (Special Ordered Sets) method to handle bilinear terms more efficiently during implementation [5]. Based on the SOS-1 method, this complementary constraint becomes:

$$\text{SOS-1}(\boldsymbol{z},\boldsymbol{\sigma}) = \begin{cases} z^k_{i,j} = 1 \ \Rightarrow \ Z^s_{i,j,k} = \sigma_{i,s} \\ z^k_{i,j} = 0 \ \Rightarrow \ Z^s_{i,j,k} = 0 \end{cases}, \forall i,j,k,s \quad (18)$$

where SOS-1 is the indicator constraint by branching $\boldsymbol{z} = 1$ or $\boldsymbol{z} = 0$, which can be automatically reformulated by modern mixed integer solvers (e.g., Gurobi, CPLEX). Finally, we can reformulate ($\mathcal{P}_1$) in (5) as the following MILP, which can be efficiently solved by solvers.

$$\textbf{RDDU:} \min_{\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{u},TP} \quad \mathcal{C}_1(\mathbf{y}) + \mathcal{C}_2(\mathbf{x},\mathbf{z}) + \mathcal{C}_4(\mathbf{u}) + \varrho \quad (19a)$$

$$\text{s.t. } (5b)\text{–}(5j), (5m)\text{–}(5o), (11), (12), (15), (\mathbf{17})\text{or } (\mathbf{18}). \quad (19b)$$

It should be noted that (17) can be equivalently replaced by the SOS-1 constraints proposed in (18).

Remark 1. *(Computational efficiency of SOS-1 constraints) The use of SOS-1 constraints enhances the computational efficiency of solving mixed integer programs (MIPs) by leveraging the solver's built-in branching and presolve capabilities. Instead of relying on large Big-M constraints that can introduce numerical instability and weaker relaxation, SOS-1 constraints explicitly enforce mutually exclusive relationships between variables. When the binary variable $\boldsymbol{z}$ takes a value of 1 or 0, the solver can immediately deduce the corresponding assignment of the continuous variable $\boldsymbol{z}$, effectively eliminating unnecessary search space and improving convergence. Furthermore, modern solvers such as Gurobi and CPLEX have specialized branching strategies to handle SOS constraints efficiently, reducing the number of required branching operations and enhancing the quality of the linear programming relaxation.*

IV. Numerical Results

A. Simulation Setting

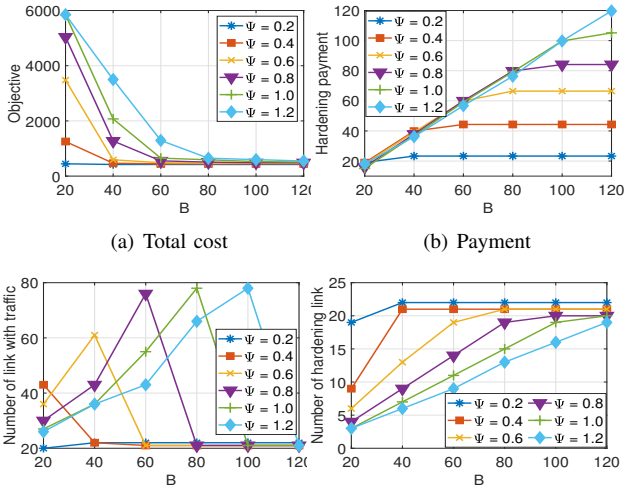
The link hardening cost parameters (i.e., $h_{i,j}, \forall i,j$) are randomly generated from a uniform distribution within the interval $[5, 5.5]$. The parameters γ_j represent the impact of the hardening decision on the link delay, which is set to be 0.9 as default. The resource capacities of ENs are randomly selected based on EC2 . The unmet demand penalty parameters s_i are generated following a uniform distribution on $[40, 50]$. In our **default setting**, the values of other system parameters are as follows: $\rho = 0.1$, $B = 100$, $\Gamma = 15$, and $\Delta_i = \Delta = 30$. During our sensitivity analysis, we will also vary these important system parameters. All the experiments are implemented

in MATLAB environment using CVX[1] and Gurobi 9.12 on a desktop with an Intel Core i7-11700KF and 32 GB of RAM.

B. Sensitivity analysis

This section presents sensitivity analyses to assess the influence of key system parameters on the optimal solution. These parameters include the total budget (B), and impact parameters (γ,$\boldsymbol{\eta}$). To evaluate the impact of varying parameters to the system performance, a scaling factor Ψ is introduced for the cost, where Ψ is equal to 1 in the default setting. Specifically, the base value of $\mathbf{h}$ generated in Section IV-A is multiplied by Ψ to scale up or down the hardening cost. A higher value of Ψ indicates a higher cost for hardening each link. For the purpose of sensitivity analyses, we solely focus on the proposed robust model *RDDU*. The optimal objective value expresses the total cost.

1) Impacts of the hardening cost and budget: Figs.1(a) and 1(b) demonstrate that increasing the harden budget B leads to a total cost reduction since the LLM SP is able to choose more links for hardening. It is important to note that the delay constraints (**??**) prioritize the closest ENs to meet the demand of each area, leading to only a subset of logical links being utilized. Clearly, the LLM SP should only harden links with traffic traversing through them. As shown in Fig. 1(a), the cost becomes saturated after a certain budget value since the LLM SP has already chosen the most critical links for hardening and has no incentive to harden more links even if the budget is redundant. Those conclusions can be further verified by Fig. 1(c) and 1(d). The LLM SP tends to harden as many links as possible if the budget allows. Fig. 1(a) further shows that the total cost increases as the link hardening cost scaling factor Ψ increases. Moreover, the saturation occurs early when Ψ is small. This means that the LLM SP demonstrates a strong inclination to invest in link hardening, resulting in an initial increase of the payment. However, this increase eventually reaches a saturation point, where all traffic links are hardened after a specified budget value, as confirmed in Fig. 1(b).



(a) Total cost (b) Payment

(c) Number of link with traffic (d) Number of link with hardening

Fig. 1: Impacts of Ψ and B on the system performance

[1]CVX: http://cvxr.com/cvx/

V. Conclusion

This paper presents a novel framework for robust link-hardening in edge computing, designed to mitigate the adverse effects of delay uncertainty on service quality. The primary contribution lies in introducing a decision-dependent uncertainty set that captures the interdependence between uncertain link delays and link-hardening decisions, offering a more realistic representation of network dynamics. To solve this challenging problem, we developed an exact algorithm that efficiently computes the optimal solution. Extensive numerical evaluations demonstrate the benefits of incorporating endogenous uncertainties, highlighting the superior performance of the proposed model compared to benchmark approaches.

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," *arXiv preprint arXiv:2407.04014*, 2024.

[3] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski, *Robust optimization*. Princeton university press, 2009.

[4] D. Bertsimas and J. N. Tsitsiklis, *Introduction to linear optimization*. Belmont, MA: Athena Scientific, 1997, vol. 6.

[5] T. Kleinert and M. Schmidt, "Why there is no need to use a big-m in linear bilevel optimization: A computational study of two ready-to-use approaches," *Computational Management Science*, vol. 20, no. 1, p. 3, 2023.