

# Cost-Efficient VM Selection for Cloud-Based LLM Inference with KV Cache Offloading

Kihyun Kim\*, Jinwoo Kim\*, Hyunsun Chung\*, Myung-Hoon Cha<sup>†</sup>, Hong-Yeon Kim<sup>†</sup>, Youngjae Kim\*<sup>‡</sup>

\*Sogang University, Seoul, Republic of Korea, <sup>†</sup>ETRI, Daejeon, Republic of Korea  
{kion777, jinwookim, hchung1652, youkim}@sogang.ac.kr, {mhcha, hykim}@etri.re.kr

**Abstract**—LLM inference is essential for applications like text summarization, translation, and data analysis, but the high cost of GPU instances from Cloud Service Providers (CSPs) like AWS is a major burden. This paper proposes INFERSAVE, a cost-efficient VM selection framework for cloud-based LLM inference. INFERSAVE optimizes KV cache offloading based on Service Level Objectives (SLOs) and workload characteristics, estimating GPU memory needs, and recommending cost-effective VM instances. Additionally, the Compute Time Calibration Function (CTCF) improves instance selection accuracy by adjusting for discrepancies between theoretical and actual GPU performance. Experiments on AWS GPU instances show that selecting lower-cost instances without KV cache offloading improves cost efficiency by up to 73.7% for online workloads, while KV cache offloading saves up to 20.19% for offline workloads.

**Index Terms**—Cloud Computing, LLM Inference, Service Level Objective (SLO) Management, KV Cache Offloading

## I. INTRODUCTION

Large Language Models (LLMs) have become a core technology in modern Natural Language Processing (NLP), demonstrating outstanding performance in various applications such as text summarization, machine translation, and conversational AI [1]. LLMs built on Transformer-based architectures, such as GPT [2] and LLaMA [3], leverage multi-layer self-attention mechanisms and large-scale pretraining to achieve near-human-level language understanding and generation capabilities. Thanks to their superior performance, LLMs are widely used across industries, providing high accuracy and natural responses in a wide range of tasks, including text summarization, question answering, and document analysis.

However, to efficiently design an LLM inference system, it is essential to consider task-specific Service Level Objectives (SLOs). For instance, in online inference tasks, such as real-time conversational services or question answering, latency must be minimized to ensure a seamless user experience. Minimizing inference latency is a key challenge in these scenarios.

On the other hand, in batch processing tasks [4, 5] such as text summarization for large datasets, log analysis, and document clustering, latency requirements are generally less strict. Instead, maximizing throughput is critical, as these tasks involve processing large volumes of input data at once. In such batch processing environments, handling large batches can easily lead to GPU memory shortages. Due to the auto-regressive nature of LLM inference, the Key-Value (KV) cache, which stores past token information, continuously grows. As a result,

GPU memory usage increases sharply with sequence length and batch size.

A common technique to mitigate this issue is KV cache offloading, which offloads KV cache data exceeding GPU memory limits to CPU memory or disk. This enables large-batch processing without running out of memory [6, 7, 8, 9]. However, if the additional latency introduced by offloading is not properly managed, throughput can significantly degrade, potentially failing to meet the required SLOs.

**Cost Efficiency of LLM Inference in Cloud Environments:** Major cloud service providers such as AWS, GCP, and Azure offer a variety of GPU instance options with different performance levels and cost structures, providing flexibility in resource utilization [10]. However, selecting a cost-efficient GPU instance in a cloud environment is a complex task that is difficult for users to perform manually. The challenge arises because GPU instances vary significantly in price and performance (Refer to Table I), and workload often require flexible KV cache offloading strategies, making optimal selection difficult.

Given this complexity, an optimized approach must integrate the following two key factors:

- GPU instance selection based on workload characteristics
- Efficient KV cache offloading strategies

Balancing throughput targets and cost efficiency by combining these two factors remains a critical challenge that must be addressed.

**Limitations of Existing Research:** Previous studies on cost efficiency in cloud environments [11, 12, 13, 14, 15] have focused primarily on image processing or general machine learning workloads. As a result, they do not capture the unique characteristics of large-scale LLM inference. Moreover, recent research on cost-efficient LLM inference has largely concentrated on real-time inference scenarios [16, 17, 18, 19, 20], neglecting large-scale data processing environments where KV cache offloading could be effectively leveraged. Furthermore, these studies do not comprehensively analyze cost efficiency in relation to Service Level Objectives (SLOs).

To address these challenges, this paper proposes INFERSAVE, the first framework that automatically recommends the most economical VM instance for LLM serving in cloud environments by integrating KV cache offloading decisions with VM selection optimization. INFERSAVE operates as a decision-making layer that works in conjunction with existing KV cache offloading frameworks [6, 9], determining when to

<sup>‡</sup>Y. Kim is the corresponding author.

apply offloading strategies and which VM instance provides the most cost-effective execution environment while meeting SLO requirements. This integration of offloading decisions with VM selection enables comprehensive optimization of LLM inference deployments in cloud environments.

The INFERSAVE framework operates as follows: First, it calculates the required GPU memory based on the specified SLO and workload size, analyzing the feasibility of KV cache offloading to determine candidate instances. Next, using pre-collected performance data, it predicts the performance and cost of each instance through a modeling step. Finally, it evaluates these predictions to recommend the most cost-efficient instance that meets the user’s SLO constraints.

Experimental results show that applying INFERSAVE achieves significant cost savings compared to traditional maximum-performance-based policies, with reductions of up to 73.7% for online workloads and 20.19% for offline workloads. In addition, it is designed to be flexible across various AWS instances and cloud environments, providing a practical and efficient approach to operating LLM inference services.

## II. BACKGROUND AND MOTIVATION

### A. LLM Architecture and Inference

Large-scale language models (LLMs), such as OpenAI’s GPT [2] and Meta’s LLaMA [3], are built on the Transformer [1] architecture. These models consist of a multi-layer structure incorporating self-attention mechanisms and Feed-Forward Networks, enabling their broad applicability across various natural language processing (NLP) tasks.

The LLM inference process is divided into two stages: *Prefill* and *Decode*. In the Prefill stage, the input prompt is processed in parallel to generate the initial output tokens. During this process, query, key, and value vectors are computed for each token in the input prompt, capturing contextual information through token-wise interactions. Simultaneously, the computed key and value tensors are stored in the GPU memory as a Key-Value (KV) cache to alleviate computational overhead in subsequent operations.

The KV cache is essential for preventing redundant computations in Self-Attention, thereby enhancing inference speed and resource efficiency. For instance, if the Prefill stage computes and stores the key and value tensors for the input "I am a," the Decode stage can reuse them to rapidly generate the next token, "man," without redundant computations.

In the Decode stage, new tokens are sequentially generated in an Auto-Regressive manner based on previously generated output tokens. Here, the stored KV cache is reused to reduce the computational burden of repeated Self-Attention operations and improve processing speed. However, the size of the KV cache increases significantly with the input length and model size. For example, as shown in Fig. 1, in the OPT-2.7B model running on an AWS `g4dn.xlarge` with 1,024 input tokens, the KV cache consumes approximately 0.332 GB of KV cache at a batch size of 2. When the batch size increases to 32, the KV cache expands to 5.312 GB, which can lead to GPU memory exhaustion. This memory constraint

can degrade overall system throughput and reduce resource utilization efficiency [1, 21].

### B. Memory Optimization for LLM Inference via KV Cache Offloading

During LLM inference, the increasing size of the KV cache can lead to GPU memory exhaustion, resulting in an Out-of-Memory (OoM) issue. To address this, KV cache offloading techniques have been proposed [6, 7, 8, 9]. These techniques operate by offloading KV cache data that exceeds the GPU memory capacity to CPU memory or disk and retrieving it to the GPU when required for computation. This approach effectively alleviates GPU memory pressure, enabling the processing of long sequences and large batch sizes. In addition, it allows efficient inference on lower-end GPUs without requiring additional high-performance GPUs, thus reducing deployment costs.

However, the latency introduced by data transfers between the GPU and external storage (e.g., CPU memory or disk) remains a major limitation of KV cache offloading. If the frequency of KV cache transfers is high, the increased latency can lead to bandwidth bottlenecks, ultimately degrading inference performance. Therefore, for effective deployment of KV cache offloading, it is essential to optimize the process by considering LLM inference characteristics (e.g., sequence length, batch size) and user-defined Service Level Objectives (SLOs), such as maximum allowable response time.

### C. Challenges of LLM Inference and KV Cache Offloading in the Cloud

Cloud service providers (CSPs), such as Amazon AWS, offer a wide range of GPU virtual machine (VM) instances. As shown in Table I, their prices vary significantly, ranging from \$0.379 (`g4ad.xlarge`) to \$40.96 (`p4de.24xlarge`), depending on GPU type, memory capacity, and network bandwidth [22].

Moreover, when applying KV cache offloading to LLM inference, the trade-off between inference performance and actual cost introduces a complex dilemma. To maximize cost-efficiency, users must carefully optimize their choice of VM and offloading strategy based on: (i) Model size, (ii) Sequence length, and (iii) Service Level Objectives (SLOs), such as maximum response time. However, a systematic framework for making these decisions is currently lacking. As a result, users must experiment with multiple VM options and offloading policies manually to determine an optimal configuration, which adds significant overhead [6, 9].

In this paper, we outline the key dilemmas of KV cache offloading for LLM inference in the cloud as follows.

- **Dual Nature of KV Cache Offloading:** KV cache offloading mitigates GPU memory shortage issues, allowing for the processing of larger batch sizes (e.g., greater than 16). However, it increases latency due to data transfer between CPU and GPU (e.g., up to 20% latency increase in FlexGen [6]). Specifically, when the sequence length exceeds 4096, the KV Cache size grows significantly (e.g.,

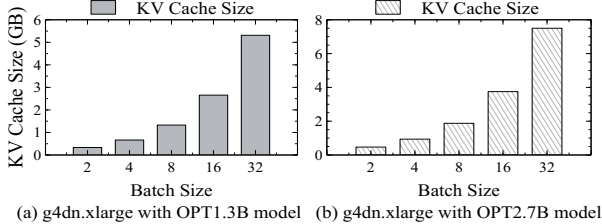


Fig. 1. KV cache size increases with batch size across different LLMs.

exceeding 3.2GB), making offloading essential. This, however, increases the likelihood of failing to meet Service Level Objectives (SLOs) such as a 100ms response time.

- Complexity of Cloud VM Selection:** As shown in Table I, the performance and cost between instances like `g4dn.xlarge` (\$0.526, 16 GiB GPU Memory) and `p4de.24xlarge` (\$40.96, 7680 GiB GPU Memory) vary significantly. The optimal VM selection depends on the model requirements (e.g., memory usage, computation speed). High-performance VMs reduce the need for KV cache offloading, while lower-end VMs increase reliance on offloading.
- Difficulty of SLO-Based Optimization:** High-performance VMs (e.g., `g6.48xlarge`) solve the Out-of-Memory (OoM) problem but may lead to GPU utilization dropping below 50% when the inference load is low, resulting in wasted costs. On the other hand, lower-end VMs (e.g., `g4ad.xlarge`) have lower initial costs but suffer from frequent KV cache offloading due to VRAM limitations, causing latency to increase by more than double [9]. This results in a dilemma of (i) resource wastage with high-cost VM selection, and (ii) performance degradation with low-cost VM selection.
- Lack of Automated Optimization Systems:** Currently, there is a lack of guidelines for automating the selection of VMs and KV cache offloading in cloud environments. Users must manually test various VMs (e.g., `g5` vs. `g6` series) and offloading settings, which increases time and cost burdens.

This study proposes the necessity of a framework that automatically recommends optimal VM and KV cache offloading strategies based on SLO, and introduces a model (Solver) that can balance cost and performance.

#### D. Existing Approaches and Their Limitations

Recent work on LLM inference optimization falls into three main categories: (1) KV cache-oriented memory management, (2) resource allocation methods, and (3) architectural partitioning strategies. While each has led to progress in specific areas, few approaches integrate memory-aware techniques with cost-efficient resource selection—particularly in heterogeneous cloud environments.

**Memory Management Frameworks:** FlexGen [6] and DeepSpeed-Inference [9] provide effective KV cache offloading mechanisms to support inference in memory-constrained environments. These frameworks focus on optimizing execution within predetermined hardware configurations and do not address the problem of selecting cost-effective VM instances

TABLE I  
VARIOUS TYPES OF INSTANCES PROVIDED BY AWS (RETRIEVED ON FEBRUARY 4, 2025, N. VIRGINIA REGION).

| Name                       | GPU Type | On-Demand (\$) | GPU (#) | FLOPS (TFLOPS) | vCPU (#) | GPU Mem (GiB) | Mem (Gbps) | Network (Gbps) |
|----------------------------|----------|----------------|---------|----------------|----------|---------------|------------|----------------|
| <code>g4dn.xlarge</code>   | T4       | 0.526          | 1       | 8,141          | 4        | 16            | 16         | - 25           |
| <code>g4ad.xlarge</code>   | V520 Pro | 0.379          | 1       | 7,373          | 4        | 8             | 16         | - 10           |
| <code>g5.xlarge</code>     | A10G     | 1.006          | 1       | 31.52          | 4        | 24            | 16         | - 10           |
| <code>g5g.xlarge</code>    | T4G      | 0.42           | 1       | 8,141          | 4        | 16            | 8          | - 10           |
| <code>g6.xlarge</code>     | L4       | 0.805          | 1       | 30.29          | 4        | 24            | 16         | - 10           |
| <code>g6.4xlarge</code>    | L4       | 1.323          | 1       | 30.29          | 16       | 24            | 64         | - 25           |
| <code>g4dn.12xlarge</code> | T4       | 3.912          | 4       | 8,141          | 48       | 64            | 192        | 50             |
| <code>g4dn.metal</code>    | T4       | 7.824          | 8       | 8,141          | 96       | 128           | 384        | 100            |
| <code>g4ad.16xlarge</code> | V520 Pro | 3.468          | 4       | 7,373          | 64       | 32            | 256        | 25             |
| <code>g5.12xlarge</code>   | A10G     | 5.672          | 4       | 31.52          | 96       | 96            | 192        | 40             |
| <code>g5g.16xlarge</code>  | T4G      | 2.744          | 2       | 8,141          | 64       | 32            | 128        | 25             |
| <code>g6.12xlarge</code>   | L4       | 4.602          | 4       | 30.29          | 48       | 96            | 192        | 40             |
| <code>g6.48xlarge</code>   | L4       | 13.35          | 8       | 30.29          | 192      | 196           | 768        | 100            |
| <code>p4de.24xlarge</code> | A100     | 40.96          | 8       | 19.49          | 96       | 7680          | 640        | 400            |

tailored to workload requirements. Consequently, users must manually determine appropriate infrastructure configurations, frequently resulting in suboptimal resource utilization and cost inefficiencies.

**Cloud Resource Allocation Methodologies:** Melange [16] optimizes cost-performance trade-offs by mixing GPU types based on workload-level profiling, including request characteristics and SLO constraints. However, it does not explicitly model per-model memory requirements or support KV cache offloading, both of which are critical for LLM workloads with large batch sizes or long input/output sequences where memory is often the primary bottleneck. Without memory-aware modeling, Melange may recommend configurations that appear cost-efficient but fail at runtime due to Out-of-Memory (OoM) errors, particularly for large models or long contexts. This limitation becomes especially problematic in relaxed-latency scenarios, where KV cache offloading can enable the use of lower-cost GPU instances, an opportunity that is missed when memory is not treated as an explicit optimization parameter. Aladdin [17], a cluster-level scheduler, coordinates LLM inference requests across a fixed set of GPU instances to meet SLO constraints. Although it considers KV cache usage, it assumes a static cluster and does not support instance type selection or cost-aware resource optimization.

**Architectural Partitioning Strategies:** SplitWise [19] and ThunderServe [18] improve cost efficiency by partitioning LLM inference into prefill and decode stages, with each stage assigned to specialized GPU resources. While this approach enables targeted optimization, it fundamentally differs from our VM selection framework in several key aspects. These systems rely on pre-provisioned heterogeneous GPU clusters and focus on intra-cluster scheduling, rather than selecting optimal VM configurations. Furthermore, they typically depend on high-bandwidth interconnects such as NVLink, which limits their applicability in public cloud environments where only standard PCIe connections are available.

### III. PROBLEM DEFINITION

#### A. Definition of Service Level Objective (SLO) Metrics

In cloud environments, large language model (LLM) inference involves a complex trade-off between memory constraints, cost, and service quality. Depending on the type

of inference task, users may have different Service Level Objectives (SLOs).

In this paper, we define two types of inference tasks: Online Inference and Offline Inference.

- **Online Inference** (e.g., chatbots, voice assistants) prioritizes low response latency (e.g., within 100 ms) over query throughput, as real-time responsiveness is crucial. Thus, response time serves as the primary SLO metric.
- **Offline Inference** (e.g., batch processing of large datasets) prioritizes high query throughput over response latency, making throughput the primary SLO metric.

This classification aligns with industry practice, as providers like OpenAI and AWS offer distinct APIs for online and offline inference [4, 23, 24]. Despite differing SLO metrics, unified resource management is needed to consistently evaluate both workload types.

To encompass both of these metrics under a unified framework, we define Tokens Per Second (TPS) as the SLO metric. TPS represents the number of tokens processed per second, including both input tokens ( $L_{in}$ ) and output tokens ( $L_{out}$ ). LLM inference is typically performed in batches, where a batch consists of multiple queries ( $BS$ ). Given that the total processing time for a batch is denoted as  $T_{E2E}$ , TPS is defined as follows:

$$TPS = \frac{BS \times (L_{in} + L_{out})}{T_{E2E}} \quad (1)$$

This TPS formulation effectively captures both throughput capacity and response latency in a single metric. For a fixed workload size, TPS and  $T_{E2E}$  maintain an inverse relationship, allowing this unified metric to address both online and offline inference requirements.

### B. Definition of Cost Efficiency

In this study, our primary objective is to minimize user costs while ensuring that inference tasks meet their designated SLOs. To achieve this, we define a cost efficiency metric based on the previously introduced Tokens Per Second (TPS) metric.

Let  $TPS_{SLO}$  denote the target TPS required by the user to meet the SLO, and let  $TPS_{actual}$  represent the actual throughput achieved during inference. We define the effective TPS as:  $TPS_{effective} = \min(TPS_{actual}, TPS_{SLO})$ .

This minimum function reflects a fundamental economic principle in resource provisioning: exceeding performance requirements incurs additional cost without proportional utility gains. In practice, both online inference (constrained by human perceptual limits of 4–6 words per second [25, 26]) and offline inference (bounded by operational deadlines, such as 24-hour completion windows [4, 23]) exhibit diminishing returns beyond their respective TPS thresholds. By capping effective throughput at the SLO level, our metric ensures that computing resources are evaluated based on practical utility rather than theoretical performance potential.

Given this, the total time required to process a batch of queries, denoted as  $T_{task}$ , is calculated as:

$$T_{task} = \frac{BS \times (L_{in} + L_{out})}{TPS_{effective} \times 3600} \quad (2)$$

In cloud environments, GPU usage is typically billed on an hourly basis. Therefore, we apply a ceiling function to  $T_{task}$  to account for the actual billable time.

Based on this, we define SLO-based cost efficiency (CE) as a metric to evaluate the cost-effectiveness of a given inference task while ensuring compliance with the SLO. Let VM Price represent the hourly cost of the virtual machine (in dollars per hour). The cost efficiency metric is then defined as:

$$CE_{task} = \frac{TPS_{effective} \times 3600}{\lceil T_{task} \rceil \times VM\ Price} \quad (3)$$

This metric provides a quantitative measure of how efficiently a system meets the required SLO while optimizing costs in a cloud-based inference environment.

### C. Preliminary Results

As shown in Table I in Section II, cloud VM instances exhibit significant differences in both performance and cost. This variability makes it challenging for users to select the most cost-efficient instance for LLM inference tasks. To validate the complexity of this decision-making process, we evaluated the Cost Efficiency (CE) of two representative VM instances (g4dn.xlarge and g5.xlarge) under varying batch sizes and SLO requirements. The experiments were conducted for both cases: with and without KV cache offloading, assessing its impact on cost efficiency. The results are presented quantitatively in Fig. 2.

**In a strict SLO environment (100 TPS)**, g5.xlarge demonstrated higher cost efficiency than g4dn.xlarge even at small batch sizes ( $BS < 16$ ). This is because g5.xlarge delivers higher performance under high-throughput requirements, allowing it to maintain superior cost efficiency over g4dn.xlarge even at smaller batch sizes. At Batch Size 16, g4dn.xlarge faced GPU memory constraints, necessitating KV cache offloading, which further reduced its cost efficiency. In contrast, g5.xlarge had sufficient memory to operate without offloading, maintaining consistently high cost efficiency as the batch size increased.

**In a relaxed SLO environment (10 TPS)**, g4dn.xlarge exhibited higher cost efficiency than g5.xlarge at smaller batch sizes ( $BS < 16$ ). This is because, under relaxed SLO conditions, instance cost became a more critical factor than raw performance. At Batch Size 16, despite g4dn.xlarge requiring KV cache offloading due to GPU memory limitations, the performance degradation caused by offloading was not a major issue under the relaxed SLO constraints. As a result, g4dn.xlarge, with its lower instance cost, achieved higher cost efficiency compared to g5.xlarge.

To sum up, cost efficiency varies significantly depending on SLO settings and GPU memory utilization strategies, demonstrating that using a high-performance GPU is not always the optimal choice. Particularly in offline inference tasks, where response time constraints are less stringent, KV cache offloading techniques can enable cost-efficient inference even on lower-cost GPUs. These findings highlight that the optimal GPU instance selection depends on the user’s SLO requirements and the characteristics of the inference task.

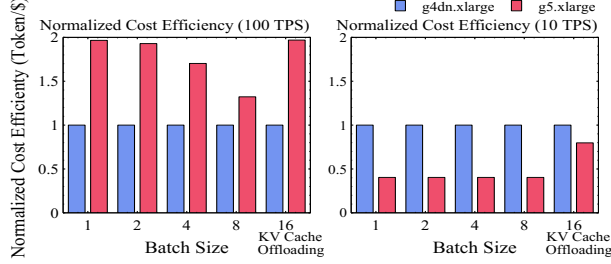


Fig. 2. Cost efficiency comparison between AWS `g4dn.xlarge` and `g5.xlarge` instances across different batch sizes and SLO constraints. Results are based on the OPT-2.7B model with 512-token input and 128-token output.

#### IV. DESIGN OF INFERSAVE

##### A. INFERSAVE: A Cost-Efficient VM Selection Framework

Selecting a cost-efficient VM instance in a cloud environment is a challenging task for users. To address this issue, we propose INFERSAVE, a software tool designed to assist users in making cost-efficient VM selections. The INFERSAVE framework operates in the following four stages:

- Stage 1 Requirement Analysis and Parameter Extraction:** The user provides input parameters, including cost constraints, model characteristics, and performance requirements.
- Stage 2 Resource Suitability Assessment and Candidate Instance Identification:** Based on the provided parameters, the framework calculates the required memory capacity, analyzes the feasibility of KV cache offloading, and identifies a set of suitable GPU instance candidates.
- Stage 3 Performance-Cost Prediction Modeling:** Leveraging pre-profiled performance data, the framework predicts the TPS of each candidate GPU instance and evaluates its cost efficiency.
- Stage 4 SLO-Based Optimization and Instance Selection:** The framework recommends the most cost-efficient GPU instance that satisfies the SLO constraints.

Table II summarizes the input and intermediate parameters used throughout the four stages of the INFERSAVE framework.

##### B. Requirement Analysis and Parameter Extraction

This stage involves collecting key input parameters necessary for LLM inference tasks. The most critical parameter is the maximum willingness-to-pay price ( $P_{\max}$ ), which represents the maximum cost (\$/hour) that the user is willing to pay. This value serves as a fundamental constraint in the subsequent stages of the algorithm, determining the range of GPU instances that can be considered.

Additionally, the user specifies the target LLM model (e.g., OPT-2.7B, LLaMA-7B), and based on this selection, the system automatically extracts key model parameters such as model size, number of attention heads, head dimensions, feed-forward network (FFN) dimensions, and activation size. Other essential input parameters include the average input token length, average output token length, batch size, and the required SLO in terms of TPS (Tokens Per Second).

TABLE II  
NOTATION AND FORMULAS FOR MODEL AND MEMORY COMPUTATION.

| User Input Parameters       |  |
|-----------------------------|--|
| Variable                    | Description and Formula  |
| $BS$                        | Batch size   |
| $L_{in}$                    | Input token length   |
| $L_{out}$                   | Output token length  |
| $P_{\max}$                  | User max price willingness   |
| $TPS_{SLO}$                 | User SLO requirement   |
| Model Parameters            |  |
| $h_1$                       | Hidden size (model dimension)  |
| $h_2$                       | Intermediate size (projection)   |
| $nh$                        | Number of attention heads  |
| $L$                         | Transformer layers   |
| $C_{off}$                   | KV cache offloading ratio  |
| $Precision_{bytes}$         | Bytes per parameter (e.g., FP16 = 2B)  |
| $Mem_{model}$               | Number of parameters $\cdot$ $Precision_{bytes}$                               |
| $Mem_{KVcache}$             | $2 \cdot BS \cdot (L_{in} + L_{out}) \cdot nh \cdot Precision_{bytes} \cdot L$ |
| $Mem_{KVcache, per\_layer}$ | $Mem_{KVcache} / L$  |
| $Mem_{activation}$          | $2 \cdot (L_{in} + L_{out}) \cdot BS \cdot h_1$                                |
| Instance Specifications     |  |
| $FLOPS_{GPU}$               | GPU's theoretical FLOPS  |
| $BW_{gpu \rightarrow cpu}$  | GPU-to-CPU bandwidth   |
| $BW_{cpu \rightarrow gpu}$  | CPU-to-GPU bandwidth   |

This stage plays a crucial role in transforming user requirements into quantitative parameters, establishing the foundation for resource suitability assessment and performance prediction. Ultimately, it is essential for selecting the most cost-efficient GPU instance that meets both performance objectives and budget constraints.

##### C. Resource Suitability Assessment and Candidate Instance Identification

At this stage, the system evaluates memory requirements based on user parameters and assesses the feasibility of KV cache offloading to identify suitable GPU instances. It first computes the total memory requirement  $Mem_{total}$  for the given Transformer model and input-output parameters, defined as:  $Mem_{total} = Mem_{model} + Mem_{activation} + Mem_{KVcache}$ . The base memory requirement is also defined as  $Mem_{base} = Mem_{model} + Mem_{activation}$ . The system then evaluates GPU instance suitability based on three criteria and applies Algorithm 1.

**Case1) No Offloading Required:** If the available GPU memory is greater than or equal to the total memory requirement, i.e.,  $GPU_{memory}^i \geq Mem_{total}$  then the instance can fully accommodate the model without KV cache offloading. Here,  $i$  refers to the current particular running instance. In this case, the offloading coefficient is set to  $C_{off}^i = 0$  and the instance is added to the candidate pool.

**Case2) Offloading Not Feasible:** An instance is deemed unsuitable if it meets any of the following conditions:

- If the available GPU memory is smaller than the model weights:  $GPU_{memory}^i < Mem_{model}$ .
- If the KV cache size per layer exceeds the available memory:  $Mem_{KVcache, per\_layer} > Mem_{avail}^i$ .

This condition arises because attention operations are performed on the GPU, requiring KV cache to remain in GPU

memory. When the available memory is insufficient, an Out of Memory (OoM) error occurs, preventing execution.

**Case3) KV Cache Offloading Required:** If an instance does not fall into either of the previous categories, KV cache offloading is required. In this case, the offloading coefficient is computed as:  $C_{\text{off}}^i = 1 - \frac{\text{Mem}_{\text{avail}}^i}{\text{Mem}_{\text{KVcache}}}$

Finally, the selected instances are sorted in ascending order based on cost, and the results are used as input for the performance-cost prediction modeling stage. This systematic approach ensures that the most cost-efficient GPU instance is selected within the user’s budget while accurately evaluating the feasibility and cost-efficiency of KV cache offloading.

#### D. Instance Performance Prediction

At this stage, the system predicts Tokens Per Second (TPS) for the candidate GPU instances identified in the previous step. This is achieved through mathematical modeling that leverages model parameters, hardware profiling information (FLOPS, bandwidth, etc.) of each candidate instance, and the offloading coefficient to quantitatively estimate the task processing time.

The total task processing time  $T_{\text{task}}$  consists of the Prefill and Decode stages and is calculated as follows [6]:

- **Prefill Stage:** This stage processes the entire input sequence. The processing time per layer ( $T_{\text{pre}}$ ) is multiplied by the number of layers ( $n$ ).
- **Decode Stage:** This stage generates each output token sequentially. The processing time per layer ( $T_{\text{dec}}$ ) is multiplied by the number of layers ( $n$ ) and the number of generated tokens ( $L_{\text{out}} - 1$ ), since the first output token is already processed in the Prefill stage.

Thus, the total task processing time  $T_{\text{task}}$  is expressed as follows:

$$T_{\text{task}} = \underbrace{T_{\text{pre}} \cdot n}_{\text{Prefill Time}} + \underbrace{T_{\text{dec}} \cdot n \cdot (L_{\text{out}} - 1)}_{\text{Decode Time}} \quad (4)$$

a) *Prefill Stage Processing Time:* The Prefill stage processing time  $T_{\text{pre}}$  consists of computation time and KV cache storage time. Since GPU computation and KV cache offloading occur in parallel, the total delay is determined by the process with the longest execution time:

$$T_{\text{pre}} = \max(CTCF(T_{\text{compute}}^p), T_{\text{trans}}^p) \quad (5)$$

The computation time  $T_{\text{compute}}^p$  in the Prefill stage is divided into Linear Layer computation and Self-Attention computation, both of which are calculated by dividing the required floating-point operations (FLOPs) by the GPU’s theoretical FLOPS capacity:

$$T_{\text{compute}}^p = \frac{BS(8L_{\text{in}}h_1^2 + 4L_{\text{in}}h_1h_2)}{FLOPS_{\text{GPU}}} + \frac{BS(4L_{\text{in}}^2h_1)}{FLOPS_{\text{GPU}}} \quad (6)$$

Linear Layer compute time                      Self-Attention compute time

The KV cache transfer time  $T_{\text{trans}}^p$  in the Prefill stage represents the time required to offload the generated KV cache from GPU to CPU memory and is computed as:

$$T_{\text{trans}}^p = \frac{C_{\text{off}} \cdot \{2 \cdot (L_{\text{in}} + 1) \cdot h_1 \cdot \text{Precision}_{\text{bytes}}\} \cdot BS}{BW_{\text{gpu} \rightarrow \text{cpu}}}$$

---

#### Algorithm 1: Resource Suitability Evaluation and Instance Selection (Price Priority)

---

**Input: Memory Requirements:**

$\text{Mem}_{\text{model}}$  — Model weight memory requirement

$\text{Mem}_{\text{activation}}$  — Activation memory requirement

$\text{Mem}_{\text{KVcache}}$  — Total KV Cache memory requirement

$\text{Mem}_{\text{KVcache, per\_layer}}$  — KV Cache memory per layer

**For each GPU instance  $i$ :**

$\text{GPU}_{\text{memory}}^i$  — Total GPU memory

$\text{GPU}_{\text{price}}^i$  — GPU price

**User-defined maximum price:**  $P_{\text{max}}$

**Output:** GPU candidates that satisfy both price and resource conditions

```

1 Candidates  $\leftarrow \emptyset$  // Initialize candidate set
2  $\text{Mem}_{\text{total}} \leftarrow \text{Mem}_{\text{model}} + \text{Mem}_{\text{activation}} + \text{Mem}_{\text{KVcache}}$ ;
3  $\text{Mem}_{\text{base}} \leftarrow \text{Mem}_{\text{model}} + \text{Mem}_{\text{activation}}$ ;
4 for each GPU instance  $i$  do
5   if  $\text{GPU}_{\text{price}}^i \leq P_{\text{max}}$  then
6     // Apply Price Filter
7      $\text{Mem}_{\text{avail}}^i \leftarrow \text{GPU}_{\text{memory}}^i - \text{Mem}_{\text{base}}$  // Calculate Available Memory
8     if  $\text{GPU}_{\text{memory}}^i \geq \text{Mem}_{\text{total}}$  then
9       // Offloading Not Required
10       $C_{\text{off}}^i \leftarrow 0$ ;
11      Add  $(i, C_{\text{off}}^i)$  to Candidate Set;
12    else
13      if  $\text{GPU}_{\text{memory}}^i < \text{Mem}_{\text{model}}$  OR
14         $\text{Mem}_{\text{KVcache, per\_layer}} > \text{Mem}_{\text{avail}}^i$  then
15          // Offloading Not Possible
16          Mark GPU  $i$  as Unsuitable // Exclude from candidates
17        end
18      else
19        // KV Cache Offloading Required
20         $C_{\text{off}}^i \leftarrow 1 - \frac{\text{Mem}_{\text{avail}}^i}{\text{Mem}_{\text{KVcache}}}$ ;
21        if  $\text{Mem}_{\text{KVcache, per\_layer}} \leq \text{Mem}_{\text{avail}}^i$  then
22          // Layer-Level Constraint Check
23          Add  $(i, C_{\text{off}}^i)$  to Candidate Set;
24        end
25        else
26          Mark GPU  $i$  as Unsuitable // Exclude from candidates
27        end
28      end
29    end
30  end
31 end
32 Sort Candidate Set by  $\text{GPU}_{\text{price}}^i$  in ascending order;
33 return Candidate Set Candidates;

```

---

b) *Decode Stage Processing Time:* The Decode stage processing time  $T_{\text{dec}}$  includes computation time and KV cache retrieval time. If the KV cache fully resides in the GPU, only computation time is considered. However, if offloading occurs, additional latency is introduced due to data transfer from CPU to GPU. The Decode time is therefore expressed as:

$$T_{\text{dec}} = CTCF(T_{\text{compute}}^d) + T_{\text{trans}}^d \quad (7)$$

The Decode computation time  $T_{\text{compute}}^d$  consists of Linear Layer and Self-Attention computation, and is computed as follows:

$$T_{\text{compute}}^d = \frac{BS \cdot (8h_1^2 + 4h_1 \cdot h_2)}{FLOPS_{\text{GPU}}} + \frac{4 \cdot BS \cdot (L_{\text{in}} + \frac{L_{\text{out}}}{2}) \cdot h_1}{FLOPS_{\text{GPU}}}$$

Linear Layer compute time                      Self-Attention compute time

The KV cache transfer time  $T_{\text{trans}}^d$  in the Decode stage refers to the time required to load KV cache stored in CPU memory back into GPU memory and is computed as:

$$T_{\text{trans}}^d = \frac{(C_{\text{off}} \cdot 2 \cdot (L_{\text{in}} + 1) + L_{\text{out}}) \cdot h_1 \cdot \text{Precision}_{\text{bytes}} \cdot BS}{\text{BW}_{\text{cpu} \rightarrow \text{gpu}}}$$

This modeling approach accounts for scenarios where offloading is either required or not, effectively incorporating GPU memory constraints and computational performance. By modeling both computation latency and KV cache offloading overhead, it enables a quantitative analysis of trade-offs between computation and memory access time across the Prefill and Decode stages.

Using this modeling framework, Tokens Per Second (TPS) can be estimated to guide the selection of an optimal GPU instance for a given inference task. While this theoretical model offers a solid foundation, theoretical FLOPS values from GPU vendors may not accurately reflect real-world LLM inference performance. Section IV-F discusses these limitations and introduces the Compute Time Calibration Function (CTCF) to correct such discrepancies.

#### E. Step 4: Final Instance Selection Based on SLO

Based on the TPS (Tokens Per Second) values computed for each GPU instance in the previous stage, this step selects the most cost-efficient instance while ensuring that the user’s Service Level Objective (SLO) is met. The selection process follows these steps:

First, instances that fail to satisfy the user-defined SLO constraint ( $\text{TPS} \geq \text{TPS}_{\text{SLO}}$ ) are first excluded from consideration. The cost efficiency metric is then computed for each remaining instance using (3). The instance with the highest cost efficiency is selected, with ties broken in favor of the one offering higher throughput (TPS).

The final selection result is presented to the user along with comprehensive details, including instance type, expected TPS, cost, and KV cache offloading configuration. Additionally, the system provides alternative options and a performance-cost trade-off analysis, enabling users to make an informed decision that is optimized for their specific LLM inference workload.

#### F. Compute Time Calibration Function (CTCF)

The theoretical FLOPS values provided by GPU manufacturers do not accurately reflect real-world performance in LLM inference workloads. Fig. 3 illustrates the discrepancy between the FLOPS values advertised by the manufacturer and those actually utilized in computation across three different GPU instances. This discrepancy arises from factors such as memory bottlenecks, reduced GPU utilization, and variations in computation patterns, which manifest differently in the Prefill and Decode stages of LLM inference. As a result, selecting a GPU instance solely based on theoretical FLOPS can lead to significant performance mismatches, causing users to incur unnecessary costs. To address this issue, it is essential to introduce a calibration method that aligns theoretical FLOPS values with actual computational performance.

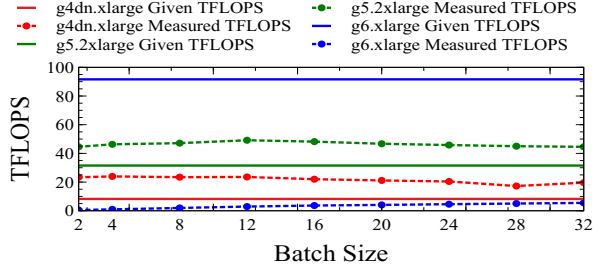


Fig. 3. Comparison of FLOPS provided by the GPU manufacturer (NVIDIA) and the actual FLOPS utilized when calculating Prefill time on AWS GPU VMs. The results present TFLOPS measurements for three different GPU VMs using the OPT-2.7B model with an input size of 512 tokens and an output size of 128 tokens as batch size grows.

- CTCF Modeling:** We conducted preliminary experiments across various batch sizes to analyze the relationship between LLM inference time and batch size. The results consistently showed a linear increase in inference time for both the prefill and decode stages. This linear trend was observed across different GPU architectures, including T4, A10G, L4, and L40s, with consistent patterns across multiple sizes of Transformer-based models. The observed linearity stems from the computational characteristics of Transformer architectures: as batch size increases, both self-attention mechanisms and feed-forward layers exhibit computation costs that scale proportionally, while GPU memory access patterns also show similar linear growth. This persistent scaling behavior across diverse GPU environments motivated the adoption of a regression-based CTCF model.

CTCF is a linear transformation function that adjusts theoretical computation time to match actual execution time. It is defined as follows:

$$CTCF(T_{\text{compute}}) = \alpha \cdot T_{\text{compute}} + \beta \quad (8)$$

where  $\alpha$  is a scaling factor that corrects overestimation or underestimation of theoretical computation time and  $\beta$  is a fitted offset that accounts for systematic deviations between theoretical and actual execution time, such as GPU-level latency, memory behavior, and scheduling effects. These parameters are optimized using the least squares method and are determined through pre-profiling experiments.

Through extensive pre-profiling,  $\alpha$  and  $\beta$  values were computed for all AWS GPU instances across various batch sizes and stored as reference data. As shown in Table III, applying these per-instance  $\alpha$  and  $\beta$  values significantly reduces the prediction error, bringing the adjusted execution time very close to the actual measurement. Based on this, INFERSAVE profiles  $\alpha$  and  $\beta$  values for all available AWS GPU instances, enabling precise FLOPS-based execution time predictions and recommending the optimal instance for users.

The CTCF-based correction method effectively compensates for the inherent limitations of theoretical FLOPS values provided by GPU manufacturers, leading to more accurate LLM inference performance predictions.

TABLE III  
VALUES OF  $\alpha$  AND  $\beta$  TO CALCULATE ADJUSTED  $T_{\text{PREFILL}}$  FOR OPT-2.7B MODEL

| Instance Type (GPU Model) | $\alpha$ | $\beta$ | avg. error rate (%) |
|---------------------------|----------|---------|---------------------|
| g4dn.xlarge (T4)          | 0.450    | -0.229  | 4.47                |
| g5.2xlarge (A10G)         | 0.709    | -0.030  | 2.60                |
| g6.xlarge (L4)            | 0.775    | 0.015   | 2.23                |

## V. IMPLEMENTATION

We implemented INFERSAVE in Python (3.10.14), using NumPy (1.24.3) for numerical computation and statistical analysis. The system builds on FlexGen, a state-of-the-art LLM inference framework with KV cache offloading support. A key strength of INFERSAVE is its ability to identify cost-efficient configurations without performing runtime profiling or full-model execution, which minimizes decision latency. Given user parameters and SLOs, INFERSAVE efficiently recommends suitable GPU instances by predicting TPS and computing cost-efficiency. The full source code is available at: <https://github.com/las-lab/InferSave>

## VI. EVALUATION

### A. Experimental setup

For our evaluation, we conducted two contrasting inference tasks representative of online and offline scenarios to assess the impact of offloading strategies on cost and performance across various cloud-based GPU instances. The goal is to quantitatively analyze the effect of offloading on both cost and performance efficiency, and to identify the optimal instance given a target SLO. Online inference aims to meet strict SLO requirements while minimizing cost, whereas offline inference relaxes latency constraints, allowing offloading and the use of lower-cost instances. Each experiment was repeated three times per instance, and the average result was used for analysis to ensure consistency.

**Workload Definition:** To evaluate INFERSAVE’s ability to select optimal instances across diverse scenarios, we define two contrasting inference workloads.

- **Online inference workload:** To simulate a real-time chatbot scenario, we use 128 input tokens and 512 output tokens per request. This reflects typical interactions where users ask brief questions and receive detailed answers. The workload consists of 3,000 requests.
- **Offline inference workload:** To simulate batch processing tasks such as document summarization or data wrangling, we use 1,024 input tokens and 128 output tokens per request. The workload evaluates system performance over 1,000 requests.

**AWS Cloud Experiment Setup:** To maintain uniform experimental conditions and minimize potential disruptions caused by fluctuating cloud workloads, all experiments were conducted on AWS in the us-east-1 (Northern Virginia) region between 9:00 AM and 10:00 PM KST, during the period from December 2024 to March 2025. To avoid performance variation due to regional resource contention, testing was evenly distributed across availability zones us-east-1a through us-east-1f. For GPU-based VMs, we used g4dn.xlarge

TABLE IV  
SPECIFICATIONS OF VM INSTANCES, INCLUDING 4 GPU-VMs BASED ON AWS SPECIFICATIONS.

| Instance Type (GPU Model) | On-Demand Price (\$/hr) | GPU Memory (GB) | FP16 (TFLOPS) | PCIe B/W (GB/s) |
|---------------------------|-------------------------|-----------------|---------------|-----------------|
| g6e.xlarge (L40s)         | 2.699                   | 48              | 91.61         | 12              |
| g6.xlarge (L4)            | 1.167                   | 24              | 30.29         | 12              |
| g5.xlarge (A10G)          | 1.466                   | 24              | 31.52         | 12              |
| g4dn.xlarge (T4)          | 0.710                   | 16              | 8.24          | 6               |

(NVIDIA T4), g5.xlarge (NVIDIA A10G), g6.xlarge (NVIDIA L4), and g6e.xlarge (NVIDIA L40s). Detailed specifications of these instances are provided in Table IV. To validate the effectiveness of INFERSAVE, we evaluated major Transformer-based LLMs including OPT-1.3B, OPT-2.7B, and OPT-6.7B using an in-house benchmark suite. To identify optimal configurations, we varied the batch size from 1 to 64 under different single-GPU settings.

**Policy to Select Instance:** As noted in Section II-D, there are no well-established methods for GPU instance selection in LLM inference. Accordingly, we compare INFERSAVE against two baseline approaches in our evaluation.

- **Most powerful instance (Max-Performance):** This policy selects the GPU instance with the highest raw performance, aiming to minimize latency and maximize throughput. However, it ignores cost, which can lead to unnecessarily high operational expenses.
- **Simple performance prediction (INFERSAVE without KV cache offloading):** This policy selects instances based on theoretical performance metrics such as FLOPS and memory bandwidth. However, it does not account for KV cache offloading effects, and may therefore miss more cost-efficient options, similar to Melange [16].

### B. CTCF Validation

INFERSAVE proposes the Compute Time Calibration Function (CTCF) to accurately determine the optimal instance based on user requirements. To validate the accuracy of CTCF, experiments were conducted on two GPU instances, g4dn.xlarge and g6.xlarge. The experiments utilized the OPT-2.7B model, with an input token length of 512 and an output token length of 128. The model’s key computational units, including a hidden size of 2560 and an intermediate size of  $2560 \times 4$ , were configured, and the total number of layers (32) was incorporated to measure computation time. For FLOPS estimation, the theoretical FLOPS values provided by GPU manufacturers were used: g4dn.xlarge with NVIDIA T4 (8.24 TFLOPS) and g6.xlarge with NVIDIA L4 (30.29 TFLOPS).

After applying CTCF, the corrected prediction times were computed and compared with actual measurements to analyze the error rate. As shown in Fig. 4, the CTCF-adjusted values closely matched the actual measurements. Specifically, in the Decode stage of g4dn.xlarge, the corrected values exhibited an average error rate of 1% compared to actual measurements, while in the Prefill stage of g6.xlarge, the average error rate was 2%. These results demonstrate that

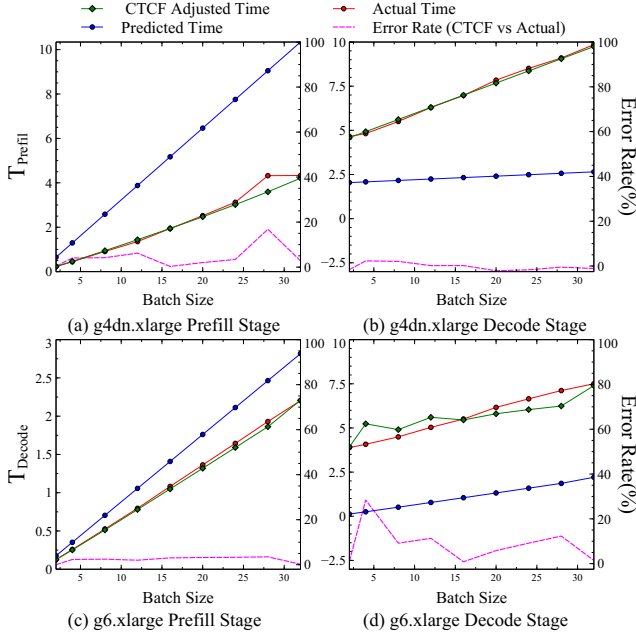


Fig. 4. CTCF accuracy analysis. The results illustrate the predicted time (blue), actual time (red), and CTCF-adjusted values (green) for Prefill and Decode times as batch size increases on two different GPU VMs. Additionally, the Error Rate between the CTCF-adjusted time and actual time is presented.

the CTCF-adjusted computation time aligns well with real-world measurements, thereby verifying that INFERSAVE can accurately recommend the most suitable GPU instance for users.

### C. Evaluation results

To evaluate the effectiveness of INFERSAVE and the proposed methodology, we conducted experiments on both online and offline workloads. While comprehensive tests were performed across various model and batch sizes, we focus our analysis on representative results using the OPT-2.7B model with a batch size of 32. This configuration clearly illustrates performance differences across GPU instances and serves as a balanced point in terms of performance and resource utilization. We set the maximum cost per hour ( $P_{\max}$ ) to \$3.00/hr. This value was chosen because g6e.xlarge, the most powerful instance in our experiments, has an on-demand price of \$2.699/hr, and a slightly higher threshold allows for a fair comparison across all instances.

1) *Online Inference Workload Results:* Table V and Fig. 5 show the instances selected by each policy under the SLO requirements for the online inference workload, along with their corresponding performance and cost comparisons. We analyze the top two instance selections made by INFERSAVE under two TPS constraints (400 and 600 TPS) and compare them to the Max-Performance policy’s selection. Note that the results of INFERSAVE without KV cache offloading were identical to those of Max-Performance, and are therefore omitted from Table V. This outcome is due to the relatively small workload size in this experiment, which allowed all KV cache data to fit within GPU memory. As a result, offloading

TABLE V  
COMPARISON OF INSTANCE SELECTION RESULTS UNDER 400 AND 600 TPS SLOS.

| SLO     | Policy        | VM          | TPS     | Cost (\$) |
|---------|---------------|-------------|---------|-----------|
| 400 TPS | INFERSAVE-1st | g4dn.xlarge | 620.17  | 0.71      |
|         | INFERSAVE-2nd | g6.xlarge   | 802.19  | 1.167     |
|         | Max-Perf.     | g6e.xlarge  | 1506.54 | 2.699     |
| 600 TPS | INFERSAVE-1st | g6.xlarge   | 800.15  | 1.167     |
|         | INFERSAVE-2nd | g5.xlarge   | 1206.12 | 1.466     |
|         | Max-Perf.     | g6e.xlarge  | 1505.37 | 2.699     |

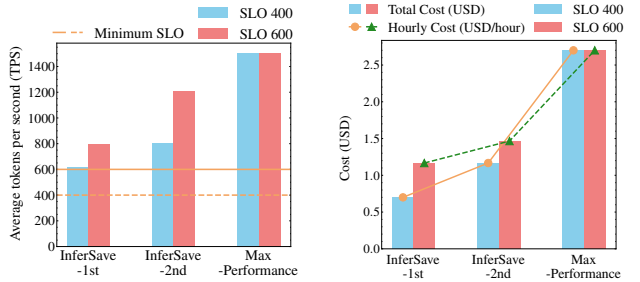


Fig. 5. Comparison of average TPS and cost for different INFERSAVE configurations and the baseline configuration under varying SLO constraints for online inference workloads (Left: Average TPS, Right: Cost).

had no effect on performance, and the selected instances remained the same. Additionally, since the total runtime was less than one hour, the hourly and total costs were equivalent in this experiment.

With an SLO requirement of 400 TPS, INFERSAVE selected g4dn.xlarge as its first choice, offering the lowest cost of \$0.71 while achieving 620.17 TPS. In contrast, the Max-Performance policy selected g6e.xlarge, which delivered the highest throughput (1506.54 TPS) at a cost of \$2.699, approximately 280% more expensive than INFERSAVE’s choice. A similar trend was observed under the 600 TPS constraint, where INFERSAVE selected g6.xlarge, meeting the requirement at a 56.75% lower cost compared to g6e.xlarge.

These results indicate that the Max-Performance policy significantly exceeds the required TPS, leading to under-utilized resources and increased operational costs. In contrast, INFERSAVE accurately predicts performance and selects cost-effective instances that meet the SLO without over-provisioning.

2) *Offline inference workload results:* Table VI and Fig. 6 show the instance selections made by each policy under the SLO requirements for the offline inference workload, along with their corresponding performance and cost comparisons. Due to the large input token size, all instances except g6e.xlarge required KV cache offloading. Without offloading, only one instance could meet the memory requirements, leading INFERSAVE without offloading to select the same instance as the Max-Performance policy.

Given an SLO requirement of 100 TPS, INFERSAVE selected g4dn.xlarge as its top choice, providing approximately 160 TPS at the lowest total processing cost of \$2.13. In contrast, both the Max-Performance policy and INFERSAVE without offloading selected g6e.xlarge, which achieved significantly higher throughput (approximately 7600 TPS) at a total cost of \$2.699, representing an increase of about 26.7%.

TABLE VI  
COMPARISON OF INSTANCE SELECTION RESULTS  
UNDER 100 AND 200 TPS SLOS.

| SLO     | Policy                       | VM          | $C_{\text{off}}$ (%) | TPS     | Cost (\$) |
|---------|------------------------------|-------------|----------------------|---------|-----------|
| 100 TPS | INFERSAVE-1st                | g4dn.xlarge | 100                  | 169.17  | 2.13      |
|         | INFERSAVE-2nd                | g6.xlarge   | 60                   | 415.04  | 2.344     |
|         | Max-Perf., INFERSAVE(w/o KV) | g6e.xlarge  | 0                    | 1506.54 | 2.699     |
| 200 TPS | INFERSAVE-1st                | g6.xlarge   | 60                   | 414.28  | 2.334     |
|         | INFERSAVE-2nd                | g5.xlarge   | 60                   | 414.01  | 2.932     |
|         | Max-Perf., INFERSAVE(w/o KV) | g6e.xlarge  | 0                    | 1505.37 | 2.699     |

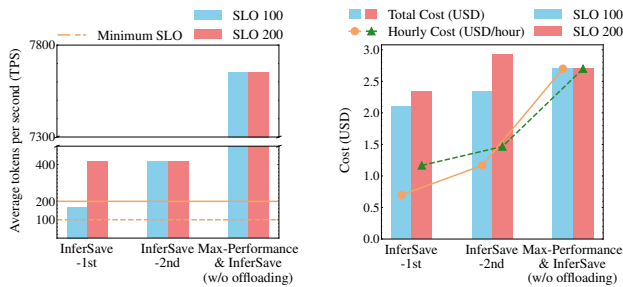


Fig. 6. Comparison of average TPS and cost for different INFERSAVE configurations and the baseline configuration under varying SLO constraints for offline inference workloads (Left: Average TPS, Right: Cost).

This instance achieves maximum throughput by storing the entire KV cache in GPU memory, eliminating the need for offloading. However, despite satisfying the SLO, its high cost results in lower overall cost efficiency.

With an SLO requirement of 200 TPS, INFERSAVE selected `g5.xlarge` as its top choice, since `g4dn.xlarge` no longer met the performance requirement. This instance provides approximately 400 TPS while maintaining a total cost of \$2.344. In contrast, the Max-Performance policy again selected `g6e.xlarge`, which achieved about 7600 TPS at a total cost of \$2.699, representing roughly a 15% increase in cost. This result illustrates that, without considering offloading, an unnecessarily high-performing and expensive instance may be selected. This leads to excessive costs and reduced overall cost efficiency.

3) *Overall Analysis*: By evaluating the experimental results from both online and offline inference workloads, we derive key insights for the efficient operation of LLM inference systems.

- (i) **The impact of workload I/O patterns on optimal infrastructure selection**: The input and output token lengths of online inference and batch processing workloads differ significantly. These differences serve as key factors in determining optimal instance types and offloading strategies.
- (ii) **The significance of selectively applying KV cache offloading**: KV cache offloading is not a universally effective strategy and yields the greatest cost savings when applied selectively based on workload characteristics. For offline batch processing workloads with long inputs, cost reductions of up to 28% were achieved while still satisfying SLO requirements. In contrast, for online inference workloads, applying KV cache offloading was

often more beneficial when both cost and performance were considered.

- (iii) **Finding the optimal configuration through INFERSAVE**: INFERSAVE balances cost and performance by jointly considering SLO requirements and workload characteristics. Instead of selecting the instance with the highest raw performance, it chooses the most cost-efficient option that satisfies the SLO.

These results demonstrate that INFERSAVE optimizes cost and performance by analyzing workload characteristics and selecting the most efficient instance for a given SLO. We next discuss its system role, applicability, and future directions.

#### D. Discussion

**Decision Layer for KV Cache Offloading**: INFERSAVE is designed to function as a decision layer that complements existing KV Cache offloading frameworks such as FlexGen [6] and DeepSpeed-Inference [9]. While these frameworks provide mechanisms for implementing offloading, INFERSAVE determines optimal VM selection and offloading strategy based on workload characteristics and SLO requirements.

**Toward Multi-GPU and Distributed Optimization**: The current implementation focuses on single-VM optimization. Future work could extend to multi-GPU and distributed environments, addressing inter-GPU communication costs and synchronization overhead.

**Robust Performance Across LLM Workloads**: Experiments with both online and offline workloads show that INFERSAVE performs well across diverse inference scenarios. These results demonstrate the effectiveness of combining KV cache offloading with VM selection for a wide range of LLM applications.

## VII. CONCLUSION

In this study, we propose INFERSAVE, which utilizes SLO-aware performance prediction to automatically select cost-efficient VM instances in the cloud and validate it across both online and offline workloads. We identify opportunities to enhance cost efficiency and leverage cheaper, less powerful GPU instances while meeting SLO requirements by exploiting techniques such as KV cache offloading. Through extensive evaluation, our results confirm that INFERSAVE effectively exploits these opportunities and achieves up to 73.7% lower operational cost while maintaining SLO compliance. These results suggest that LLM service providers can balance cost and performance more effectively by selecting instances based on SLOs and strategically applying offloading techniques.

#### ACKNOWLEDGEMENT

This work was partly supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (RS-2025-00564249), and by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2022-0-00498, Development of high-efficiency AI computing software core technology for high-speed processing of large-scale learning models).

## REFERENCES

- [1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, E. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017, pp. 5998–6008.
- [2] A. Radford and K. Narasimhan, "Improving language understanding by generative pre-training," <https://openai.com/research/language-unsupervised>, 2018, technical Report.
- [3] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, and et al., "Llama: Open and efficient foundation language models," <https://arxiv.org/abs/2302.13971>, 2023, arXiv:2302.13971.
- [4] Anthropic, "Message batches api: Run jobs up to 50% cheaper," [Online]. Available: <https://www.anthropic.com/news/message-batches-api>, 2024, [Accessed: Dec. 30, 2024].
- [5] OpenAI, "Openai api reference: Batch api," [Online]. Available: <https://platform.openai.com/docs/api-reference/batch>, 2024, [Accessed: Dec. 30, 2024].
- [6] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, B. Chen, P. Liang, C. Ré, I. Stoica, and C. Zhang, "Flexgen: High-throughput generative inference of large language models with a single gpu," in *Proc. of the 40th Int. Conf. on Machine Learning (ICML)*. PMLR, 2023, pp. 31 094–31 116.
- [7] Y. Xiong, H. Wu, C. Shao, Z. Wang, R. Zhang, Y. Guo, J. Zhao, K. Zhang, and Z. Pan, "Layerkv: Optimizing large language model serving with layer-wise kv cache management," *arXiv preprint arXiv:2410.00428*, 2024.
- [8] X. Pan, E. Li, Q. Li, S. Liang, Y. Shan, K. Zhou, Y. Luo, X. Wang, and J. Zhang, "Instinfer: In-storage attention offloading for cost-effective long-context llm inference," <https://arxiv.org/abs/2409.04992>, 2024, arXiv:2409.04992.
- [9] R. Y. Aminabadi, S. Rajbhandari, A. A. Awan, C. Li, D. Li, E. Zheng, O. Ruwase, S. Smith, M. Zhang, J. Rasley, and Y. He, "DeepSpeed-inference: Enabling efficient inference of transformer models at unprecedented scale," in *Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE Press, 2022, pp. 46:1–15.
- [10] Google Cloud, "AWS, Azure, and Google Cloud service comparison," <https://cloud.google.com/docs/get-started/aws-azure-gcp-service-comparison>, [Online]. Accessed: Dec. 26, 2024.
- [11] A. Harlap, A. Tumanov, A. Chung, G. R. Ganger, and P. B. Gibbons, "Proteus: Agile ML elasticity through tiered reliability in dynamic resource markets," in *Proc. of the 12th European Conf. on Computer Systems (EuroSys)*. Belgrade, Serbia: ACM, 2017, pp. 589–604.
- [12] Y. Kim, K. Kim, Y. Cho, J. Kim, A. Khan, K.-D. Kang, B.-S. An, M.-H. Cha, H.-Y. Kim, and Y. Kim, "Deepvm: Integrating spot and on-demand vms for cost-efficient deep learning clusters in the cloud," in *Proc. of the 24th IEEE Int'l Symp. on Cluster, Cloud and Internet Computing (CCGrid)*, 2024, pp. 227–235.
- [13] G. Fragiadakis, V. Liagkou, E. Filiopoulou, D. Fragkakis, C. Michalakis, and M. Nikolaidou, "Cloud services cost comparison: A clustering analysis framework," *Computing*, vol. 105, no. 10, pp. 2061–2088, 2023.
- [14] A. Andrzejak, D. Kondo, and S. Yi, "Decision model for cloud computing under sla constraints," in *Proc. of the IEEE Int'l Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2010, pp. 257–266.
- [15] P. Kokkinos, T. A. Varvarigou, A. Kretsis, P. Soumplis, and E. A. Varvarigos, "Cost and utilization optimization of amazon ec2 instances," in *Proc. of the IEEE Sixth Int'l Conf. on Cloud Computing (CLOUD)*, 2013, pp. 518–525.
- [16] T. Griggs, X. Liu, J. Yu, D. Kim, W.-L. Chiang, A. Cheung, and I. Stoica, "Mélange: Cost efficient large language model serving by exploiting gpu heterogeneity," *arXiv preprint arXiv:2404.14527*, 2024.
- [17] C. Nie, R. Fonseca, and Z. Liu, "Aladdin: joint placement and scaling for slo-aware llm serving," *arXiv preprint arXiv:2405.06856*, 2024.
- [18] Y. Jiang, F. Fu, X. Yao, T. Wang, B. Cui, A. Klimovic, and E. Yoneki, "Thunderserve: high-performance and cost-efficient llm serving in cloud environments," *arXiv preprint arXiv:2502.09334*, 2025.
- [19] P. Patel, E. Choukse, C. Zhang, A. Shah, Í. Goiri, S. Maleki, and R. Bianchini, "Splitwise: efficient generative llm inference using phase splitting," in *Proc. of the 51st ACM/IEEE Annu. Int. Symp. on Computer Architecture (ISCA)*. IEEE, 2024, pp. 118–132.
- [20] Z. Wang, S. Li, Y. Zhou, X. Li, R. Gu, N. Cam-Tu, C. Tian, and S. Zhong, "Revisiting slo and goodput metrics in llm serving," *arXiv preprint arXiv:2410.14257*, 2024.
- [21] R. Pope, S. Douglas, A. Chowdhery, J. Devlin, J. Bradbury, J. Heek, K. Xiao, S. Agrawal, and J. Dean, "Efficiently scaling transformer inference," in *Proc. of the 5th Conf. on Machine Learning and Systems (MLSys)*, 2023, pp. 606–624.
- [22] Amazon Web Services, "Amazon ec2 instance types," <https://aws.amazon.com/ec2/instance-types/>, 2025, accessed: Mar. 10, 2025.
- [23] OpenAI, "Openai api pricing," [Online]. Available: <https://openai.com/api/pricing>, 2024, [Accessed: May 17, 2025].
- [24] Amazon Web Services, "Amazon bedrock pricing," [Online]. Available: <https://aws.amazon.com/bedrock/pricing/>, 2024, [Accessed: May 17, 2025].
- [25] K. Rayner, E. R. Schotter, M. E. J. Masson, M. C. Potter, and R. Treiman, "So much to read, so little time: how do we read, and can speed reading help?" *Psychological Science in the Public Interest*, vol. 17, no. 1, pp. 4–34, 2016.
- [26] H. Li, Y. Liu, Y. Cheng, S. Ray, K. Du, and J. Jiang, "Eloquent: A more robust transmission scheme for llm token streaming," in *Proc. 2024 SIGCOMM Workshop on Networks for AI Computing*, 2024, pp. 34–40.