Resource-Constrained LLM Inference in Edge Computing Systems with Provable Guarantees

Shiqiang Wang, *Senior Member, IEEE*, Jake B. Perazzone, Changchang Liu, Kevin Chan, *Senior Member, IEEE* .

Abstract—Large language models (LLMs) have become increasingly popular nowadays with a wide range of promising applications. However, a key challenge is that the resource consumption of such models can be high even at the inference stage. This problem becomes especially exacerbated in resource-constrained mobile edge computing systems. In this paper, we propose a novel approach for optimizing LLM inference on edge servers with resource budget constraints. By observing that similar input prompts usually generate similar LLM outputs in expressive LLM applications, our method caches representative input-output pairs. When a new input is close to a cached input prompt, the system directly returns the cached output without querying the LLM, thereby reducing the resource consumption while still being able to provide accurate results. We formulate a stochastic optimization problem to minimize the embedding distance between new and cached inputs subject to a given resource budget. The problem is solved using a novel way of integrating the Lyapunov drift-plus-penalty framework with resource cost estimation and caching. Our theoretical analysis shows provable upper bounds of both constraint satisfaction and optimality of the solution given by our proposed algorithm, while providing useful insights on the trade-off between them. In addition, our experiments with real datasets and models show up to 7.2% improvement in the model accuracy compared to baseline approaches when the resource budget is the same.

Index Terms—Large language models, machine learning, mobile edge computing, resource control, stochastic optimization

1 Introduction

Large language models (LLMs) have become a cornerstone of modern natural language processing (NLP), enabling a wide range of applications such as text classification, generation, translation, etc. [1]. These models exhibit remarkable performance and versatility, making them essential components in various emerging applications.

This research was partly sponsored by the U.S. Army Research Laboratory under Agreement Number W911NF-16-3-0002. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

S. Wang and C. Liu are with IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, USA (e-mail: shiqiang.wang@ieee.org, changchang.liu33@ibm.com). J. Perazzone was with DEVCOM Army Research Laboratory, Adelphi, MD 20783, USA when he contributed to this work (e-mail: jake.perazzone@gmail.com). K. Chan is with DEVCOM Army Research Laboratory, Adelphi, MD 20783, USA (e-mail: kevin.s.chan.civ@army.mil)

Most LLM inference services are hosted in the cloud nowadays. As LLM-based applications become more popular, it is foreseeable that more of such services will be provided at edge servers and end devices, to meet the users' needs for low-latency and robust access. However, a significant challenge with deploying LLMs at the edge is their substantial resource consumption [2], [3]. Performing inference with these models demands considerable computational power and energy. This issue is particularly pronounced where the LLMs are served in resource-constrained edge computing environments, where computation, storage, and energy supply are limited. Therefore, we ask the following important question:

How to maximize the inference performance of LLMs under a constrained resource budget in edge computing environments?

There are challenges in addressing this question. First, varying resource availability and unpredictability in user requests make optimizing real-time inference with LLMs a difficult task. Second, ensuring high inference accuracy while adhering to strict resource constraints involves complex trade-offs that are not straightforward to manage, especially when LLM inference requires substantial computational power and energy. To solve such a problem, we need an algorithm that is capable of making adaptive decisions while leveraging lightweight inference mechanisms. The solution must optimize resource usage without compromising the inference accuracy.

To address these challenges, we formulate the problem in a stochastic optimization framework and propose a novel solution that integrates vector embeddings and similarity caching at edge servers to reduce the number of expensive LLM calls. While our method is general, in this work we focus on *expressive LLM applications*, such as summarization, paraphrasing, and dialogue support, where descriptive quality rather than strict factual precision determines usefulness. Our approach leverages the idea that certain user prompts seeking the same response can be worded differently, e.g., "How can I track my credit card's delivery?" versus "Is there a way to track the card that was just sent to me?", and may be asked repeatedly. In addition, in LLM-based applications with retrieval augmented generation (RAG) [4], such as explaining concepts from a policy document or user manual, the retrieved context is usually very similar for questions expecting the same response. For

such repeating requests with different wording, a traditional cache would consider these to be completely different and result in a miss, whereas a similarity cache would recognize them as similar and return the expected response. By equipping an NLP system with a similarity cache, needless calls to the LLM can be avoided, thus saving a significant amount of resources.

Our approach starts by converting the user input prompt (in the form of text) into dense vector representations, known as embeddings [5]–[7], which effectively capture the semantic similarity between inputs. Then, equipped with a cache storing representative input-output pairs and a distance metric to measure similarity, our system can quickly retrieve and return cached outputs when new inputs are sufficiently similar to those already in the cache. Naturally, in the event that a new input does not closely match any cached input, the LLM is queried. This process provides a systematic way to limit the amount of requests to the LLM, while only inducing a slight degradation in response accuracy.

The decision to use either the cached output or the LLM is driven by a *stochastic optimization problem* that seeks to minimize the embedding distance between the current user input and the input corresponding to the served output (possibly from the cache), while adhering to a predefined *resource budget constraint*. This optimization ensures that resource consumption is efficiently managed without significantly reducing the accuracy of the outputs. We propose a novel way of integrating the Lyapunov drift-plus-penalty framework [8] with resource cost estimation and caching to balance the resource-accuracy trade-off in real-time, guiding the system to make (near-)optimal decisions that maintain high inference accuracy under constrained resources.

Our problem has a *unique challenge* that the resource cost function is *unknown* and must be learned *online*. In addition, it also entails caching strategies in which the cache *affects future system states* and evolves solely based on previous LLM requests and responses. *Due to these challenges, we cannot directly apply existing Lyapunov optimization techniques to our new problem.* Leveraging the structure of our problem, we prove a finite upper bound on the length of the virtual queue, which captures the accumulated constraint violation. This upper bound does not grow in time, which improves the constraint satisfaction guarantee over existing Lyapunov optimization results. This improvement also makes it possible for us to prove a theoretical upper bound on the optimality gap in the presence of our problem's unique challenges. This gap monotonically decreases as the number of time steps or cache size increases. A control parameter V effectively balances the trade-off between constraint satisfaction and optimality.

The main contributions in this paper are summarized as follows.

1) We formulate a stochastic optimization problem that decides whether to use cached outputs or query the LLM based on input embedding distance, subject to a constraint on the time-averaged resource cost.
2) We solve the problem in an online manner, while incorporating key aspects of our problem including resource cost estimation and caching, to dynamically balance the trade-off between accuracy and resource consumption.
3) We provide a thorough theoretical analysis, establishing finite upper bounds on the virtual queue length and demonstrating the trade-offs between constraint satisfaction and optimality.
4) We validate our approach through extensive experiments with real datasets and models running on actual hardware, confirming its effectiveness in reducing resource consumption while maintaining high accuracy.

The remainder of this paper is organized as follows. We review the related work in Section 2. Section 3 presents our system design along with its motivation, followed by the optimization problem formulation. Section 4 describes our online decision making algorithm. Next, the theoretical performance analysis on constraint satisfaction and optimality is presented in Section 5. Experimental results are presented in Section 6, and Section 7 draws conclusion.

2 Related Work

Recent advances in NLP have established the concept of vector embedding [5]–[7] where the semantic meaning of data can be distilled into a real vector, in which nearby vectors have similar semantic meaning. A few recent works have utilized vector embeddings to create caches for LLMs using the term "semantic caching" [9]–[11]. Under the assumption that semantically similar queries can be served by the same response, previous query/response pairs stored in a cache can be used to serve future similar queries. Comparing this to traditional caching where only exact repeated queries will be served, semantic caches can significantly save on computation and communication costs by avoiding expensive LLM calls. In these works, a distance or similarity measure in the embedding space is used to quantify semantic similarity in the input query space. Thus, when a new query comes in, it is compared with all items in the cache and if the nearest one is within a certain similarity threshold, the cached response is returned to the user. Otherwise, the LLM is called, incurring additional cost. Works such as [9], [10], [12] provide heuristic caching policies and empirical results while [13] proposes cache-specific fine-tuning of the embedding model to improve cached responses. The only work that provides the theoretical analysis via regret bounds of their proposed caching policy is [11], but their results only consider exact caching.

Caching, in general, is a wide-ranging topic that has received significant research interest over the years, with numerous applications [14]. In much of the work on caching, though, only exact matches from repeated requests are served by the cache. As with LLMs, however, it is acceptable in some applications to return responses that are near the intended output, perhaps with some loss in utility. The first works to generally consider such caching systems include [15]–[17] which introduced the concept as "similarity caching." These works also assume that there exists some distance metric that is able to quantify similarity in the input space such that inputs/queries that are close can be served by the same result. Most of the works on this topic propose heuristic caching algorithms and perform empirical analysis

[15], [17], [18], but some theoretical treatments of similarity caching also exist [16], [19], [20]. While the theoretical works include competitive analysis, they do not consider resource constraints and make some strict assumptions for certain analyses, e.g., having a separate content retrieval process uncorrelated with the actual requests, a cache of size 1, or fixed retrieval cost.

Recently, in models with transformer architectures, which is the case for almost all the LLMs nowadays, key-value (KV) cache has been implemented to reduce the internal computations of the transformer during inference. Different exact [21]–[23] and approximate [24]–[26] KV cache mechanisms have been developed. However, implementing new variants of KV cache requires low-level changes to the inference stack, which can be undesirable in practice. Meanwhile, such existing KV cache methods are usually based on heuristics that do not provide rigorous guarantees on optimality or constraint satisfaction.

In summary, none of the aforementioned works consider the *optimal* caching for LLMs *with a given resource budget constraint*. Additionally, the threshold-based methods [9], [10], [17], [27] empirically determine the best threshold after experimentation, which is complex to manage and can be impractical when the system conditions and input patterns change dynamically. In contrast, our decision process dynamically adapts depending on both the quality of the cached response and the resource consumption at a given point. While we primarily consider the LLM as a "black-box" and focus on caching input-output pairs for expressive LLM applications in this work, our control algorithm can be extended to support KV caches by integrating our caching mechanism more tightly into LLM inference engines.

3 System Model

3.1 Motivation

To operate within the constrained resource budget while maintaining inference accuracy, we propose to reuse previous LLM outputs when the input prompts are similar. To quantify the similarity, we compute the distance over an embedding space derived from the original text, which is common practice in NLP [5]–[7]. We motivate our solution based on experimental findings on the WMT14 English-German translation dataset [28] in the following.

Input and Output Similarities Are Correlated. For a given user input, our ultimate goal of caching is to produce an output that is similar to the actual LLM output but without having to process this input with the LLM. In this case, we can compute the embedding of the input prompt, which allows us to quantify the similarity between this new input and those in the cache, but the corresponding similarity in their outputs/responses is not known. To investigate their dependency, we compute the similarities in both the input and output embedding spaces and plot them in Fig. 1. The embeddings are computed using the paraphrase multilingual MiniLM-L12-v2 model from Sentence-BERT [29], [30], where each embedding is a vector of length 384. We observe a positive, approximately linear correlation between the two, indicating that similarity in the input space is a good proxy for similarity in the output space. Thus, we determine whether we can use a cached output instead of querying the LLM based on the distance between the new input embedding and cached input embeddings.

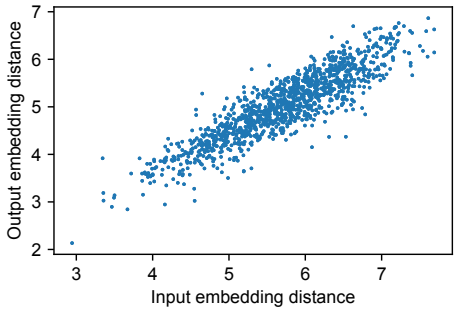


Fig. 1. Correlation between input and output embedding distances (for the first 50 samples on WMT14 English-German dataset, corresponding to $50^2 = 2,500$ distance values). Pearson correlation coefficient: $r = 0.90$.

TABLE 1
Comparison of energy consumption and running time of embedding computation and LLM inference, showing mean $\pm$ standard deviation

Computation	Energy (Joules)	Running time (seconds)
Embedding	0.17 ± 0.03	$6.2 \times 10^{-3} \pm 0.2 \times 10^{-3}$
LLM inference	328.0 ± 176.6	1.3 ± 0.7

Embedding Computation Consumes Much Less Resources Than LLM Inference. In Table 1, we compare the energy consumption and running time of computing embeddings versus a full LLM inference. For the LLM, we use a Phi-3-mini instruction-tuned model containing 3.8 billion parameters [31], and we keep the same embedding model as mentioned above. The models are run on a machine with Intel Core i9-10900X CPU @ 3.70GHz and NVIDIA RTX 3090 GPU representing an edge server where the energy and running time measurements are taken. The results show that the energy and time cost of embedding computation are several orders of magnitude smaller. Clearly, avoiding expensive LLM calls can greatly reduce resource consumption, justifying our proposed approach.

3.2 System Model

Motivated by the aforementioned findings, we present our system model as depicted in Fig. 2. For the ease of presentation, we focus on a single edge server in this paper, while noting that multiple edge servers will have their individual caches and the procedure at each edge server will be the same. We assume that the user to edge server association is given, since how such associations are made is a separate research problem that has been studied in various existing works [32]–[35].

A user requests the services of an LLM hosted at an edge server and submits a text prompt. Its corresponding embedding is computed and then sent to a scheduler who has access to a cache containing a subset of previous requests' embeddings and their corresponding outputs. Depending on the amount of consumed resources, overall resource budget, and proximity to the nearest item inside the cache, a decision is made whether to return a cached response or forward the input prompt to an LLM. In the latter case, more resources are consumed and the LLM returns a new output for the current user input, based on which the cached content can be updated.

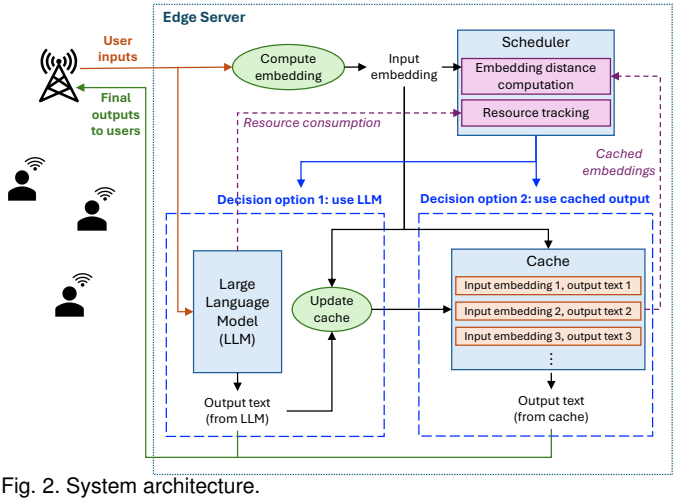


Fig. 2. System architecture.

3.3 Problem Formulation

We assume a slotted system where, at each time $t \in \{1, 2, 3, \ldots, T\}$, a new input text $\mathbf{m}_t$ of length (number of characters) ℓ_t arrives at the edge server. The edge server then computes its associated embedding vector $\mathbf{p}_t \in \mathbb{R}^d$ with dimension d. We note that although ℓ_t is generally different for different t, the dimension d of the embedding computed from the embedding model remains the same. We assume that we have a cache of size K in which up to K pairs of input embeddings and output text are stored. More specifically, we denote the cache at time t by $\mathcal{C}_t := \{\mathbf{c}_t^i \in \mathbb{R}^d : i \in \{1, 2, \ldots, K_t\}\}$, where ":=" means "defined to be equal to," each $\mathbf{c}_t^i$ denotes an input embedding in the cache, and $K_t \leq K$ is the number of elements in the cache at time t. Note that we do not explicitly define the output text in the cache, because there is a direct mapping from the input embedding to it. We also define a distance function[1] between vectors in the embedding space, denoted by $\rho(\mathbf{p}, \mathbf{c}_i)$, and $f(\mathbf{p}, \mathcal{C}) := \min_i \rho(\mathbf{p}, \mathbf{c}_i)$ is the nearest-neighbor distance of $\mathbf{p}$ with respect to all the cached input embeddings. We define $f(\mathbf{p}, \emptyset) = \infty$ and $0 \cdot f(\mathbf{p}, \emptyset) = 0$.

Once an input is received, a decision is made whether to return a response from the cache or to query the LLM. This decision is denoted by the identity function $\mathbb{1}_t$, which is 1 when the LLM is queried and 0 when the cache is used. If the LLM is queried, a resource cost (e.g., amount of consumed energy) of $g(\ell_t)$ is incurred which depends on the length of the input (ℓ_t). The value of $g(\ell_t)$, however, is *unknown* at decision time t and is only revealed after the LLM is queried, because it is difficult to know a priori the exact resource consumption of processing an LLM inference request. There is a time-averaged cost budget of $\overline{g}$ that is pre-specified by a system admin. In practice, this cost budget can be determined by the overall spending that is planned for running the system, the capacity of the energy supply, etc.

The overall goal of the scheduler is to provide the best responses for the user while staying within the given cost budget. As a proxy for quality of response to the user, we minimize the distance between the user's input embedding and the input embedding corresponding to the response served to the user. This distance is zero when the LLM is used ($\mathbb{1}_t = 1$), and it is $f(\mathbf{p}_t, \mathcal{C}_t)$ when a cached output is used ($\mathbb{1}_t = 0$). More specifically, the optimization problem is

$$\textbf{P1:} \quad \min_{\{\mathbb{1}_t\}, \{\mathcal{C}_t\}} \quad \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\left[(1 - \mathbb{1}_t) \cdot f(\mathbf{p}_t, \mathcal{C}_t)\right] \tag{1a}$$

$$\text{s.t.} \quad \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\left[\mathbb{1}_t \cdot g(\ell_t)\right] \leq \overline{g} \tag{1b}$$

$$\mathbb{1}_t \in \{0, 1\}. \tag{1c}$$

Challenges. It is impractical to solve this problem directly over the entire time horizon of T steps in an offline manner, because it is generally impossible to predict future input requests from users. Therefore, we need to devise solutions that make decisions in an *online* manner at each time step t, based on observations up to the current time t. The presence of both a time-averaged objective function and constraint lends itself nicely to an online stochastic optimization framework called Lyapunov optimization [8]. In essence, Lyapunov optimization captures the degree of constraint satisfaction in a virtual queue. It defines a per-slot objective that includes an upper bound of the Lyapunov drift of the virtual queue and a penalty term capturing the objective of the original problem. By minimizing the drift-plus-penalty objective for each slot t, the original problem with a time-averaged objective and constraint can be solved approximately, with provable guarantees of the time-averaged objective being nearly optimal and the time-averaged constraint being satisfied as the total time $T \to \infty$. However, our problem **P1** has the following *unique challenges so that we cannot directly apply existing algorithms or results in Lyapunov optimization*:

1) The exact function of $g(\ell_t)$ is unknown at time t before the decision is made and must be learned in an online fashion.
2) The cache $\mathcal{C}_t$ affects the future in a possibly unknown way and its evolution is via responses from the LLM when $\mathbb{1}_t = 1$.

To address these challenges, we present a novel algorithm in Section 4 to (approximately) solve **P1**, followed by its analysis in Section 5.

4 New Algorithm for Online Decisions

In this section, we describe our proposed online decision making process. We start with explaining the main concepts in Section 4.1, then present the key steps in Section 4.2.

4.1 Main Concepts

4.1.1 Estimated Resource Cost

The decision of whether to use the cached output or query the LLM depends on the cost $g(\ell_t)$, but its exact value is often unknown before a decision of querying the LLM is made, due to system dynamics of the edge server, such as concurrent workloads. We need to have a way of estimating the cost so that the system can make a "reasonably good" decision. To do so, we first investigate how the energy consumption changes in the input length ℓ_t. In Fig. 3, we plot

1. When the distance function is an ℓ^2 norm and the vectors are normalized, minimizing the distance is equivalent to maximizing the cosine similarity. We use distance to represent the (negated) similarity for the ease of theoretical analysis in Section 5.

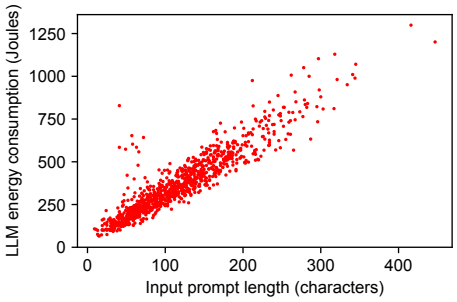


Fig. 3. Correlation between energy consumption and input length (for the first $1,000$ samples on WMT14 English-German dataset). Pearson correlation coefficient: $r = 0.93$.

the energy consumption of the LLM as a function of input length ℓ_t, obtained in the same experimental setup as in Section 3.1. We observe an approximately linear relationship between the two.

Motivated by this experimental observation, we use linear regression to approximate $g(\ell_t)$ in our decision algorithm. In particular, we define the *estimated cost* $\hat{g}(\ell_t, a_t, b_t) := a_t\ell_t + b_t$, where a_t and b_t are determined in an online manner throughout the decision making process, so that $\hat{g}(\ell_t, a_t, b_t) \approx g(\ell_t)$ for any ℓ_t. This estimated cost is then used in determining $\mathbb{1}_t$ in each step t.

If $\mathbb{1}_t = 1$, i.e., the LLM is queried, the system observes the true cost $g(\ell_t)$ by measuring the resource consumption during LLM inference *after* the decision is made. Then, the algorithm updates the regression parameters to a_{t+1} and b_{t+1}, so that a better cost estimate is available in the next step $t + 1$.

4.1.2 Virtual Queue

As commonly done in Lyapunov optimization, we convert the time-averaged constraint (1b) into a virtual queue. The virtual queue with length Q_t evolves as

$$Q_{t+1} = \max\{0, Q_t + \mathbb{1}_t \cdot g(\ell_t) - \overline{g}\}, \tag{2}$$

where $Q_1 = 0$. The queue length Q_t captures an upper bound of the aggregated (i.e., without time average) violation of constraint (1b) before time t.

To capture the exact (instead of estimated) resource consumption for satisfying constraint (1b), we apply the queue update in (2) at the end of time slot t *after* observing the exact resource consumption of LLM inference if $\mathbb{1}_t = 1$. Therefore, in (2), either the exact value of $g(\ell_t)$ is known when $\mathbb{1}_t = 1$, or $\mathbb{1}_t = 0$ and $\mathbb{1}_t \cdot g(\ell_t) = 0$, where in the latter case the value of $g(\ell_t)$ has no effect on the queue update.

4.2 Key Steps

4.2.1 Overall Procedure

The overall procedure is outlined in Algorithm 1. Upon receiving a new input in every time step t, the algorithm first determines $\mathbb{1}_t$ (Line 5). If $\mathbb{1}_t = 1$, the cost estimation parameters (Line 8) and cached content (Line 9) are updated. Finally, the virtual queue capturing the resource cost incurred over time gets updated (Line 10). We provide details on some of the specific steps as follows.

Algorithm 1: Online decision making for resource-constrained LLM inference

Parameters: $V > 0$, $\{\eta_t > 0\}$
Output: $\{\mathbb{1}_t\}$, $\{\mathcal{C}_t\}$
1 Initialize $\mathcal{C}_1 \leftarrow \emptyset$, $Q_1 \leftarrow 0$, and choose $a_1 \in [A_0, A_1]$, $b_1 \in [B_0, B_1]$;
2 **for** $t \leftarrow 1, \ldots, T$ **do**
3 Receive new input text $\mathbf{m}_t$ of length ℓ_t;
4 Compute embedding $\mathbf{p}_t$ of $\mathbf{m}_t$;
5 Compute $\mathbb{1}_t$ from (4);
6 **if** $\mathbb{1}_t = 1$ **then**
7 Receive true cost $g(\ell_t)$ and output from LLM;
8 Compute a_{t+1}, b_{t+1} from (7);
9 Compute $\mathcal{C}_{t+1}$ from policy in Section 4.2.4;
10 Compute Q_{t+1} from (2);

4.2.2 Minimizing Drift-Plus-Penalty to Determine $\mathbb{1}_t$

We define a control parameter $V > 0$ which controls the trade-off between optimality and constraint satisfaction. For a given cache $\mathcal{C}_t$, the algorithm minimizes the following drift-plus-penalty objective for each slot t:

P2:

$$\min_{\mathbb{1}_t \in \{0,1\}} V(1 - \mathbb{1}_t) \cdot f(\mathbf{p}_t, \mathcal{C}_t) + Q_t(\mathbb{1}_t \cdot \hat{g}(\ell_t, a_t, b_t) - \overline{g}). \tag{3}$$

The notion of drift here refers to the Lyapunov drift, which is the conditional expectation in the difference of squared queue lengths in two adjacent time slots $t + 1$ and t [8]. In the original Lyapunov optimization framework, the second term in the objective (2) of **P2** is an upper bound of the Lyapunov drift. However, we have replaced the exact cost $g(\ell_t)$ in (2) with the estimated cost $\hat{g}(\ell_t, a_t, b_t)$ in (3), because the exact cost is unknown at this step of Algorithm 1. We will see in Section 5 that, although we have made this replacement, we can still obtain theoretical performance guarantees.

The penalty term is the first term in the objective (3) of **P2**, which corresponds the per-slot objective in **P1** multiplied by the control parameter V.

It is easy to see that **P2** has a simple solution:

$$\mathbb{1}_t = \begin{cases} 0, & \text{if } V \cdot f(\mathbf{p}_t, \mathcal{C}_t) \le Q_t \cdot \hat{g}(\ell_t, a_t, b_t) \\ 1, & \text{otherwise.} \end{cases} \tag{4}$$

When there is no ambiguity, *we use $\mathbb{1}_t$ to denote this optimal solution to **P2** hereafter*.

4.2.3 Cost Estimation

For estimating the resource cost of any given input length ℓ_t, we define a squared error function:

$$h(\ell_t, a, b) := (\hat{g}(\ell_t, a, b) - g(\ell_t))^2 = (a\ell_t + b - g(\ell_t))^2. \tag{5}$$

We further define constants A_0, A_1, B_0, B_1, so that $[a, b]$ remains within the convex set of $\Omega := \{a, b : A_0 \le a \le A_1$ and $B_0 \le b \le B_1\}$. This is useful for stabilizing the cost estimation procedure and guaranteeing convergence.

Our goal of resource cost estimation is to minimize the mean squared error:

$$\textbf{P3:} \quad \min_{a,b} \quad \mathbb{E}\left[h(\ell_t, a, b)\right] \tag{6a}$$

$$\text{s.t.} \quad [a, b] \in \Omega. \tag{6b}$$

In (6a), the expectation is over the possible input lengths ℓ_t and their corresponding costs $g(\ell_t)$, for all t.

If all the samples of $\{[\ell_t, g(\ell_t)]\}$ were available, **P3** has a closed-form solution. However, in our scenario, the samples are revealed one-by-one, so the closed-form solution is not applicable here. Instead, we perform one step of projected gradient descent every time when we receive a sample of $[\ell_t, g(\ell_t)]$, which happens in every slot t with $\mathbb{1}_t = 1$. The projected gradient descent update step is as follows:

$$\begin{bmatrix} a_{t+1} \\ b_{t+1} \end{bmatrix} = \text{Proj}_{\Omega} \left\{ \begin{bmatrix} a_t \\ b_t \end{bmatrix} - \eta_t \nabla_{a,b} h(\ell_t, a, b)\big|_{a=a_t, b=b_t} \right\}$$
$$= \begin{bmatrix} \max\{A_0, \min\{A_1, a_t - 2\eta_t \ell_t (a_t \ell_t + b_t - g(\ell_t))\}\} \\ \max\{B_0, \min\{B_1, b_t - 2\eta_t (a_t \ell_t + b_t - g(\ell_t))\}\} \end{bmatrix}, \quad (7)$$

where $\eta_t > 0$ is the gradient descent step size at time t, and the last equality is from the projection onto convex set.

4.2.4 Cache Update

For every t with $\mathbb{1}_t = 1$, when $|\mathcal{C}_t| < K$, where $|\cdot|$ denotes the cardinality of set, the new element, including input embedding and output text from the LLM, is always added to the cache. When $|\mathcal{C}_t| = K$, the cache has reached its capacity, and a cache eviction policy decides which element to remove from the total of $K+1$ elements (including K elements in the cache and one new element). The remaining elements are then kept in the cache $\mathcal{C}_{t+1}$ used in the next time step $t+1$. Common eviction policies such as least recently used (LRU) and least frequently used (LFU) can be used. In the experiments in Section 6, we will show that the performance of our algorithm is not sensitive to specific caching policies.

4.3 Comparison to Online Learning in Lyapunov Optimization

We note that there exist works where Lyapunov optimization is extended to the online learning setting, with the penalty and constraints being revealed after the decision in each slot t is made [36], [37]. The setup in these works is related to ours. However, our problem has some unique characteristics: 1) The true resource cost $g(\ell_t)$ is only revealed when $\mathbb{1}_t = 1$ and not revealed when $\mathbb{1}_t = 0$. 2) The caching decision in each time t depends on decisions made in previous time steps and also affects the algorithm performance and decisions in subsequent steps. Due to these characteristics, existing approaches are not directly applicable to our problem.

5 Performance Analysis

In this section, we analyze the performance of our proposed algorithm theoretically, with focus on constraint satisfaction and optimality. We consider an arbitrary T in our analysis, which includes and generalizes the special case of $T \to \infty$ that is primarily considered in [8].

We first introduce a boundedness assumption that is common in Lyapunov optimization literature.

Assumption 1. There exist finite constants $\ell_{\max} > 0$, $D \geq 0$, $G_0 > 0$, and $G_1 \geq G_0$, such that $\ell_t \leq \ell_{\max}$, $f(\mathbf{p}_t, \mathcal{C}_t) \in [0, D]$, and $\hat{g}(\ell_t, a_t, b_t), g(\ell_t), \overline{g} \in [G_0, G_1]$.

5.1 Bounded Constraint Violation

We begin by analyzing the constraint violation bound. The next theorem gives a finite upper bound on the queue length.

Theorem 1. *The queue length* $Q_t \leq \frac{VD}{G_0} + G_1$.

Proof. From (3) and Assumption 1, we know that if $Q_t > \frac{VD}{G_0}$, then $Q_t > \frac{V \cdot f(\mathbf{p}_t, \mathcal{C}_t)}{\hat{g}(\ell_t, a_t, b_t)}$ and we will have $\mathbb{1}_t = 0$, thus $Q_{t+1} \leq Q_t$. When $Q_t \leq \frac{VD}{G_0}$, we have $Q_{t+1} \leq Q_t + G_1$. The result then follows by induction over all t. □

Using the queue update relation (2), we immediately have the following upper bound on constraint violation.

Corollary 1. *The time-averaged cost satisfies*

$$\frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left[\mathbb{1}_t \cdot g(\ell_t)\right] \leq \overline{g} + \frac{\mathbb{E}\left[Q_{T+1}\right]}{T} \leq \overline{g} + \frac{VD}{G_0 T} + \frac{G_1}{T}. \quad (8)$$

Remark. Different from the result for generic Lyapunov optimization where the queue length upper bound grows in T, such as [8, Eq. (4.7)] and [36, Theorem 1], our Theorem 1 gives a finite upper bound that does not grow in T for our problem. This is an improvement over existing results. We can obtain this result because of the unique structure of our problem, where for any slot t, either the objective (1a) is non-zero (when $\mathbb{1}_t = 0$) or the constraint variable, i.e., the left-hand side of (1b), is non-zero (when $\mathbb{1}_t = 1$). Due to this improvement, our time-averaged constraint violation bound in Corollary 1 improves $\mathcal{O}\left(\frac{V}{\sqrt{T}}\right)$ in [8], [36] to $\mathcal{O}\left(\frac{V}{T}\right)$.

The finite queue length upper bound also makes it possible to analyze the optimality gap despite the fact that the resource cost estimation parameters and the cache are updated only when $\mathbb{1}_t = 1$. This is because $\mathbb{1}_t = 1$ occurs "frequently enough," as shown in the next subsection.

We also note that we only need Assumption 1 for Theorem 1 and Corollary 1 to hold. This means that the constraint violation is bounded regardless of the resource cost estimation error. While this result may appear surprising, it is true because the virtual queue capturing constraint violation is updated based on the actual cost $g(\ell_t)$ instead of the estimated cost, as stated in (2). As long as both the actual and estimated costs are bounded (Assumption 1), the constraint violation is bounded and goes to zero as $T \to \infty$, as shown in Corollary 1.

5.2 Bounded Optimality Gap

We compare the result of Algorithm 1 to the best result that can be obtained within the class of stationary (possibly randomized) policies. We use $\mathbb{1}_t^*$ and $\mathcal{C}^*$ to denote the outcomes of an optimal stationary policy that solves **P1**, which are independent of the outcomes of Algorithm 1, where $\mathbb{1}_t^*$ can vary depending on the input embedding $\mathbf{p}_t$, but $\mathcal{C}^*$ does not change in t. We assume that the optimal policy knows the exact value of the resource cost $g(\ell_t)$.

This consideration of a stationary policy is common in Lyapunov optimization [8], which intuitively means that the optimal policy has prior knowledge of the statistics (distribution) of input arrivals, but it does not have prior knowledge of the individual random outcomes. When $\mathbf{p}_t$ is independent and identically distributed (i.i.d.) across t,

$\mathbb{1}_t^*$ follows the same distribution for all t although its value can depend on $\mathbf{p}_t$, thus it is stationary. The consideration of $\mathcal{C}^*$ not changing in t is because the optimal cache content depends on the outcomes of $\mathbf{p}_t$ in multiple slots, i.e., for multiple values of t. Since it is not possible to predict future $\mathbf{p}_t$, the policy has to decide the cache content based on the statistical distribution instead of individual outcomes, which is also commonly done in the analysis of online learning algorithms [38]. We begin with introducing the following assumption on the user inputs.

Assumption 2. The user inputs are i.i.d. across different time steps $t \in \{1, 2, 3, \ldots\}$, with the embedding $\mathbf{p}_t$ being uniformly distributed within a d-dimensional hypercube with side length $2R$, for some $R > 0$. The input length ℓ_t and the corresponding resource cost $g(\ell_t)$ are independent of $\mathbf{p}_t$ and also i.i.d. across t.

This assumption facilitates the optimality gap analysis. It is needed due to the unique challenges in our problem, as summarized in Section 3.3. We note that uniform distribution of the input is also commonly assumed in the analysis of similarity caching works [19], [27]. Our experiments in Section 6 show that our proposed method also works well with practical datasets and models where Assumption 2 may not hold. We first introduce an upper bound of the objective function in **P1**.

Theorem 2. *Let y^* denote the optimal value of the objective function in **P1** under a stationary policy. We have*

$$\begin{aligned}&\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[(1-\mathbb{1}_t)\cdot f(\mathbf{p}_t,\mathcal{C}_t)\right]\\&\leq y^*+\frac{G_1^2}{2V}+2\left(\frac{D}{G_0T}+\frac{G_1}{VT}\right)\sum_{t=1}^{T}\mathbb{E}\left[\,|\hat{g}(\ell_t,a_t,b_t)-g(\ell_t)|\right]\\&\quad+\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\,|f(\mathbf{p}_t,\mathcal{C}_t)-f(\mathbf{p}_t,\mathcal{C}^*)|\right].\end{aligned}\tag{9}$$

Proof. See Appendix A. □

The first two terms in Theorem 2 show an additive error of $\mathcal{O}\left(\frac{1}{V}\right)$, which is commonly observed in Lyapunov optimization [8]. The last two terms are related to the error in resource cost estimation and the embedding distance that depends on the cached content, which are unique to our problem. We derive the upper bounds of these two terms in the following.

Since the resource cost estimation parameters and the cache content are only updated when $\mathbb{1}_t = 1$, we define T' as the total number of slots for which $\mathbb{1}_t = 1$ from Algorithm 1 within T steps. We first find a lower bound of T'.

Lemma 1. *Let $\gamma := \left\lceil \frac{VD}{G_0\overline{g}} + \frac{G_1}{\overline{g}} \right\rceil$. For any t, there exists at least one $\tilde{t} \in \Gamma_t := \{t, t+1, \ldots, t+\gamma\}$ such that $\mathbb{1}_{\tilde{t}} = 1$. This implies that for a total time of T, we have $\frac{T}{\gamma+1} \leq T' \leq T$.*

Proof. Suppose that $\mathbb{1}_{\tilde{t}} = 0$ for all $\tilde{t} \in \Gamma_t$. We note that $|\Gamma_t| = \gamma + 1$ by definition and $Q_t \leq \frac{VD}{G_0} + G_1$ from Theorem 1. From (2), we know that $Q_{t+\gamma} = 0$ because Q_t is decreased by $\overline{g}$ for γ times (lower capped by zero) due to $\mathbb{1}_{\tilde{t}} = 0$. From (3), we have $\mathbb{1}_{t+\gamma} = 1$ when $Q_{t+\gamma} = 0$, proving a contradiction. □

We now bound the last two terms in Theorem 2.

Lemma 2. *There exists a learning rate sequence $\{\eta_t\}$, so that*

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\,|\hat{g}(\ell_t,a,b)-g(\ell_t)|\right] \leq \mathcal{O}\left(\sqrt{h^*}+\frac{V}{\sqrt{T}}\right),\tag{10}$$

*where h^* is the true optimal solution to **P3** assuming full knowledge of system statistics.*

Proof. See Appendix B. □

Lemma 3. *For a large enough K, when Algorithm 1 uses a caching policy that caches the first K items received when $\mathbb{1}_t = 1$ and keeps the cache unchanged afterwards, we have*

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\,|f(\mathbf{p}_t,\mathcal{C}_t)-f(\mathbf{p}_t,\mathcal{C}^*)|\right] \leq \mathcal{O}\left(\frac{VK}{T}+\sqrt[d]{\frac{\log K}{K}}\right).\tag{11}$$

Proof. See Appendix C. □

Note that despite the specific non-evicting caching policy considered in Lemma 3 for the purpose of analysis, the experimental results in Section 6 show that our proposed method also works well with more realistic caching policies.

We now state the final bound on the optimality gap.

Theorem 3. *Under the conditions of Lemmas 2 and 3,*

$$\begin{aligned}&\mathbb{E}\left[(1-\mathbb{1}_t)\cdot f(\mathbf{p}_t,\mathcal{C}_t)\right]\\&\leq y^*+\mathcal{O}\left(\frac{1}{V}+\sqrt{h^*}+\frac{V}{\sqrt{T}}+\frac{VK}{T}+\sqrt[d]{\frac{\log K}{K}}\right).\end{aligned}\tag{12}$$

Proof. The result is obtained by plugging the upper bounds in Lemmas 2 and 3 into the upper bound in Theorem 2. □

Remark. Ignoring the logarithmic factor, the additive error upper bound in Theorem 3 is $\tilde{\mathcal{O}}\left(\frac{1}{V}+\sqrt{h^*}+\frac{V}{\sqrt{T}}+\frac{VK}{T}+\frac{1}{\sqrt[d]{K}}\right)$. When $T \to \infty$, the optimality gap becomes $\tilde{\mathcal{O}}\left(\frac{1}{V}+\sqrt{h^*}+\frac{1}{\sqrt[d]{K}}\right)$, where the first term is commonly observed in Lyapunov optimization literature [8], the second term is due to the unique need of resource cost estimation in our work, and the third term is because caching decisions affect the performance in future time slots.

5.3 Further Discussion

Comparing the results in Corollary 1 and Theorem 3, we can clearly see that the control parameter V controls the trade-off between constraint satisfaction and optimality. A smaller V leads to better constraint satisfaction but worse optimality, and vice versa.

If we further choose $V = T^{\frac{1}{4}}$ and $K = \sqrt{T}$, we can see that as $T \to \infty$ the *constraint violation approaches zero* (Corollary 1) and *optimality gap approaches* $\mathcal{O}\left(\sqrt{h^*}\right)$ (Theorem 3). When the relationship between resource cost and input length is close to linear, then the true minimum estimation error h^* is small and thus the optimality gap is small too. It is particularly interesting to emphasize that the constraint violation is guaranteed to approach zero as $T \to \infty$ regardless of h^*, thanks to the way that the virtual queue update is designed in our Algorithm 1. In summary,

when the parameters K and V are chosen appropriately, we have proven that *our proposed Algorithm 1 gives a near-optimal solution to **P1** asymptotically*, which satisfies the constraint and has an additive optimality gap related to the best-possible quality of the resource cost predictor; the optimality gap is zero when the resource cost can be predicted perfectly from the user's input text.

We also note that while the optimal policy that we use as the theoretical baseline requires some known statistics of inputs, such as parameter R in Assumption 2 as can be seen in the proof of Lemma 3 (in Appendix C), our proposed Algorithm 1 does not require such knowledge.

6 Experiments

6.1 Setup

Same as in Section 3.1, we use the Phi-3-mini instruction-tuned model [31] as the LLM that runs on a machine with Intel Core i9-10900X CPU @ 3.70GHz and NVIDIA RTX 3090 GPU representing the edge server. We use the amount of energy consumed per input request as the resource cost. The energy measurement is done by programmatically calling the `nvidia-smi` tool for GPU power measurement and the `psutil` library for CPU power estimation from the utilization percentage. The energy is then computed by multiplying the power by the measurement interval. The input embeddings are computed using the paraphrase multilingual MiniLM-L12-v2 model from Sentence-BERT [29], [30]. We note that the LLM we use is a *general purpose* LLM, where we only apply some simple prompting so that the LLM understands the given task. This aligns with the practical scenario where many of the LLM applications leverage common APIs to a standard model, but its performance shall not be compared to models that are fine tuned to each specific task.[2]

For datasets, we consider some exemplar tasks for *expressive LLM applications* (as mentioned in Section 1): 1) *WMT14* English-German translation dataset that includes paragraphs translated from English to German [28], 2) *SAMSum* dataset for text summary of dialogues [39], and 3) *Banking77* dataset for classifying banking customer requests into one out of 77 classes [40]. We use the *BLEU-1* [41], *ROGUE-1* [42], and *classification accuracy* as the accuracy metrics for quantifying the correctness of the outputs for these datasets, respectively. Since we focus on the inference stage instead of training, we use the test splits of all datasets. For the WMT14 and SAMSum datasets, we simulate similar user inputs by duplicating each original input/output pair and then supplementing it with a rephrased version of each pair. For the larger Banking77 dataset, which contains many examples of similar queries, we only duplicate the originals and do not add any additional rephrased data points. This duplication is mainly to avoid the scenario of the exact caching baseline being completely useless. The inputs are then randomly sampled from each dataset in our experiments. We study the performance after processing $T = 1,500$ inputs, for WMT14 and SAMSum datasets, and $T = 3,000$ inputs, for Banking77 dataset, where the number of inputs T is determined by the size of each dataset.

As baselines, we consider 1) *similarity caching* without stochastic optimization [12], [15], [16], [18]–[20], 2) *exact caching* [11], [14], and 3) *no caching*. For fair comparison, we align the target time-averaged resource budgets for our proposed algorithm and these three baselines. For the baselines, this is achieved by randomly querying the LLM at a given probability related to the resource budget. In addition, we also consider a *thresholding policy baseline* without resource control, as in [9], [10], [17], [27], in the last part of our experiments (Section 6.2.4).

We choose the cache size $K = 100$, the convex set (denoted by Ω as defined earlier) parameters $A_0 = B_0 = 0, A_1 = 3, B_1 = 500$, and initialize $a_1 = 0.5$ and $b_1 = 200$. For the gradient descent step size η_t used in resource cost estimation, we use two different step sizes for updating a, with $\eta_{t,a} = 10^{-6}$, and b, with $\eta_{t,b} = 10^{-3}$, for all t, because the sensitivities of a and b in the mean squared error function $h(\ell_t, a, b)$ are very different. This is a slight extension to the description earlier in the paper due to practical considerations, but the theoretical results still hold after this extension. The hyperparameters above were chosen based on only some very coarse tuning. In the main experiments, we set the control parameter $V = 10^5$ and use LRU as the cache eviction policy. Results for alternative cache eviction policies and values of V are given in Sections 6.2.2 and 6.2.3, respectively. We run each experiment with 5 different random seeds and show the mean and standard deviation values.

Note that *the additional overhead for embedding computation and nearest neighbor search is negligible* compared to LLM inference, as discussed in Section 3.1 with results shown in Table 1. Therefore, we do not specifically study such overhead in the experimental results presented in this section.

We further note that while we consider energy as the resource type in our experiments, our proposed algorithm and its analysis is generic and applicable to any type of resources that is additive over time, such as processing time/cycles, monetary cost, etc.

6.2 Results

6.2.1 Main Findings

Table 2 shows the results of our proposed method and the baselines, for different datasets and under different time-averaged per-input energy budgets. We observe that all the methods approximately conform to the given energy budget, while our proposed method gives a better inference accuracy compared to the baselines especially when the budget is limited, with up to 7.2% improvement in the mean value. This confirms that *our proposed method makes a better use of the available energy to boost the inference performance*. We also note that, when the budget is very high for the given task, all the methods achieve similar performance while using an amount of energy that is substantially below the budget, e.g., in the last row of Table 2.

6.2.2 Cache Eviction Policies

To study the effect of different cache eviction policies, we compare the results obtained from different policies for our

2. The performance can also be improved if we use a larger LLM. However, the current compute hardware we use cannot support a model that is substantially larger than the Phi-3-mini model, which is also a way to resemble the resource constraint at the edge.

TABLE 2
Inference accuracy metric (BLEU-1 for WMT14, ROGUE-1 for SAMSum, and classification accuracy for Banking77) and per-input energy consumption (time-averaged, per input request), showing mean ± standard deviation

Dataset	Method	Proposed		Similarity caching [12], [15], [16], [18]–[20]		Exact caching [11], [14]		No caching	
	Budget (J)	Perf. metric (%)	Energy (J)	Perf. metric (%)	Energy (J)	Perf. metric (%)	Energy (J)	Perf. metric (%)	Energy (J)
WMT14	$\overline{g}=50$	**22.3**±0.4	51.4±0.1	19.2±0.4	50.4±2.5	10.1±0.5	47.1±3.1	6.8±0.4	54.0±1.9
	$\overline{g}=100$	**29.3**±0.7	101.0±0.1	24.2±0.4	101.7±5.1	16.4±0.5	98.3±6.5	13.0±0.4	103.2±2.7
	$\overline{g}=200$	**38.8**±0.8	200.6±0.2	32.0±0.9	193.7±4.9	27.5±1.0	196.3±5.9	25.0±0.5	195.7±4.8
	$\overline{g}=300$	**44.9**±0.7	287.3±2.5	40.6±1.0	294.3±4.6	39.1±1.1	298.1±4.1	37.3±0.8	290.3±5.9
SAMSum	$\overline{g}=50$	**21.6**±0.3	50.9±0.1	19.1±0.6	52.8±3.6	8.9±0.6	48.1±2.8	6.3±0.2	54.6±1.4
	$\overline{g}=100$	**26.7**±0.3	100.8±0.1	23.3±0.4	104.6±3.8	14.9±0.4	101.8±4.8	12.3±0.2	105.7±2.4
	$\overline{g}=200$	**33.9**±0.5	200.4±0.2	29.8±0.3	197.1±3.8	25.1±0.8	198.1±6.1	23.1±0.4	195.5±2.4
	$\overline{g}=300$	**38.3**±0.3	277.0±2.9	36.7±0.3	299.6±3.6	35.7±0.7	301.8±6.3	35.2±0.3	299.6±3.7
Banking77	$\overline{g}=25$	**43.5**±2.2	25.9±0.0	37.2±1.3	25.7±1.0	12.8±0.8	24.2±0.8	11.2±0.3	26.2±0.9
	$\overline{g}=50$	**50.0**±1.8	50.9±0.0	42.8±1.3	51.1±1.1	23.3±0.9	50.0±1.9	21.9±0.3	51.7±1.8
	$\overline{g}=100$	**57.3**±1.9	100.7±0.1	51.4±1.2	98.1±0.7	43.3±1.9	99.7±2.0	41.2±1.6	97.9±0.8
	$\overline{g}=150$	**59.2**±1.8	131.0±0.9	**59.3**±1.8	140.4±0.9	**59.3**±1.8	140.4±0.9	**59.3**±1.8	140.4±0.9

TABLE 3
Classification accuracy and per-input energy consumption for different caching policies, on Banking77 with budget of $\overline{g}=100$ J

Method	Cache eviction policy	Accuracy (%)
Proposed	LRU	57.3±1.9
	LFU	57.4±1.8
	No eviction	56.6±1.9
Similarity cache	LRU	51.4±1.2
	LFU	50.9±1.8
	No eviction	51.3±1.2
Exact cache	LRU	43.3±1.9
	LFU	43.0±1.8
	No eviction	43.2±2.0

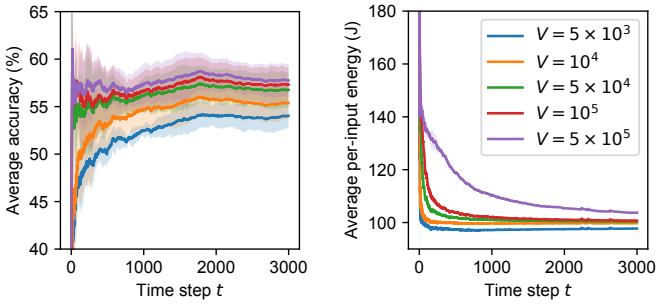
proposed method and the cache-enabled baselines. The results are shown in Table 3, where "no eviction" corresponds to the condition in Lemma 3 used in our theoretical analysis. We observe that for each method, *the performances when using different cache eviction policies are very similar*, confirming our discussion in Section 4.2.4 and the practical relevance of our analysis in Section 5.2.

6.2.3 Control Parameter V

On the effect of the control parameter V, as suggested by the theoretical analysis in Section 5, a larger V reduces the input embedding distance, i.e., the objective function in **P1**, which leads to better inference accuracy, but may also cause larger violation in the resource constraint for a finite T. This is validated by the results in Fig. 4, which confirms that *V controls the optimality and constraint satisfaction trade-off*. In practice, the parameter V can be chosen based on how strict the energy budget requirement is. We also note that the results for all datasets and energy budgets in the main experimental results shown in Table 2 use the same V (specified in Section 6.1), which indicates that there is generally no need to fine tune V for specific settings.

6.2.4 Comparison with a Thresholding Policy on Embedding Distance

Finally, we compare to a decision policy based on a fixed distance threshold, which always queries the LLM when the embedding distance exceeds the given threshold and uses the cached output otherwise, as in existing works [9], [10], [17], [27]. By comparing the two plots in Fig. 5, we observe that, if one would collect the statistics of embedding



(a) (b)

Fig. 4. Results of our proposed method with different control parameters V, on Banking77 dataset with time-averaged per-input energy budget of $\overline{g}=100$ J; (a) average accuracy until time step t, (b) average per-input energy until time step t.

distances over a sufficiently long time period and compute a threshold based on the energy budget (see Fig. 5a), the average accuracy obtained using this thresholding method with this pre-computed threshold is similar to that obtained from our proposed method (see Fig. 5b). For example, we see from the blue curve in Fig. 5a that a distance threshold of approximately 3.8 conforms to the resource budget of $\overline{g}=50$ J. In Fig. 5b, we see that for the same distance threshold of 3.8, the accuracy obtained using this thresholding method is approximately 50%. When using our proposed method, for the same resource budget of $\overline{g}=50$ J, we can obtain a very similar performance, as shown in the green dashed lines in both plots in Fig. 5.

However, the advantage of our proposed method is that it *does not* require prior knowledge on the statistics of energy consumption or embedding distances nor does it require historical empirical data to guarantee satisfaction of any given budget $\overline{g}$, whereas the thresholding policy requires such knowledge to determine the optimal threshold. This shows that *our method performs closely to an optimally tuned threshold-based algorithm, while we do not require the prior knowledge that would be needed for threshold tuning*.

7 Conclusion

In this paper, we have studied the problem of optimizing LLM inference in resource-constrained environments. We have identified some unique challenges in this problem

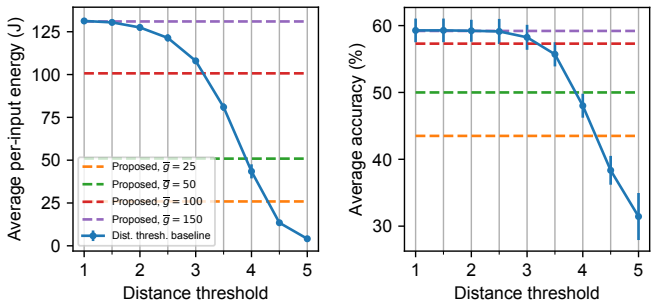


Fig. 5. Fixed distance threshold policy compared to the proposed method, on Banking77 dataset; (a) average accuracy for a given distance threshold, (b) average per-input energy for a given distance threshold. The blue curves show the mean results from the fixed threshold policy, with error bars showing the standard deviation. The dashed horizontal lines are mean values from the proposed method from Table 2, for different budgets $\overline{g}$ (unit: J).

that make standard results of Lyapunov optimization not readily applicable, including that the LLM inference cost is unknown before the inference request is completed and that the effect of caching decisions is coupled across multiple time steps. Addressing these challenges, we have proposed a new online decision making algorithm, which is inspired by the Lyapunov optimization framework but extends beyond the known results. Our theoretical analysis leveraging a unique structure of our problem provides new insights on the trade-off between inference accuracy and resource consumption in this challenging problem setup. Extensive experiments using real datasets, model, and hardware have confirmed our algorithm's effectiveness in improving the inference accuracy while conforming to the resource cost budget. Future work can consider extending our control algorithm to support other caching mechanisms for LLMs, such as KV cache. Our novel analytical methodology can be adapted to other stochastic optimization problems that involve a prediction step as well.

REFERENCES

[1] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, W. Ye, Y. Zhang, Y. Chang, P. S. Yu, Q. Yang, and X. Xie, "A survey on evaluation of large language models," *ACM Trans. Intell. Syst. Technol.*, vol. 15, no. 3, Mar. 2024.

[2] S. Moazeni, "Q&A: UW researcher discusses just how much energy ChatGPT uses," *UW News*, July 2023.

[3] S. Mehta, "How much energy do LLMs consume? unveiling the power behind AI," *Association of Data Scientists*, 2023.

[4] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in neural information processing systems*, vol. 33, pp. 9459–9474, 2020.

[5] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *International Conference on Learning Representations (ICLR)*, 2013.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

[7] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar *et al.*, "Universal sentence encoder for english," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2018, pp. 169–174.

[8] M. Neely, *Stochastic network optimization with application to communication and queueing systems*. Springer Nature, 2022.

[9] F. Bang, "GPTcache: An open-source semantic cache for LLM applications enabling faster answers and cost savings," in *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*, 2023, pp. 212–218.

[10] W. Gill, M. Elidrisi, P. Kalapatapu, A. Anwar, and M. A. Gulzar, "Privacy-aware semantic cache for large language models," *arXiv preprint arXiv:2403.02694*, 2024.

[11] B. Zhu, Y. Sheng, L. Zheng, C. Barrett, M. Jordan, and J. Jiao, "Towards optimal caching and model selection for large model inference," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[12] A. Finamore, J. Roberts, M. Gallo, and D. Rossi, "Accelerating deep learning classification with error-controlled approximate-key caching," in *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2022, pp. 2118–2127.

[13] H. Zhu, B. Zhu, and J. Jiao, "Efficient prompt caching for large language model inference via embedding similarity," in *Machine Learning for Systems 2023*, 2023.

[14] S. Podlipnig and L. Böszörmenyi, "A survey of web cache replacement strategies," *ACM Computing Surveys (CSUR)*, vol. 35, no. 4, pp. 374–398, 2003.

[15] F. Falchi, C. Lucchese, S. Orlando, R. Perego, and F. Rabitti, "A metric cache for similarity search," in *Proceedings of the 2008 ACM Workshop on Large-Scale Distributed Systems for Information Retrieval*, 2008, pp. 43–50.

[16] F. Chierichetti, R. Kumar, and S. Vassilvitskii, "Similarity caching," in *Proceedings of the Twenty-eighth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2009, pp. 127–136.

[17] S. Pandey, A. Broder, F. Chierichetti, V. Josifovski, R. Kumar, and S. Vassilvitskii, "Nearest-neighbor caching for content-match applications," in *Proceedings of the 18th International Conference on World Wide Web*, 2009, pp. 441–450.

[18] Y. B. Mazziane, S. Alouf, G. Neglia, and D. S. Menasche, "Computing the hit rate of similarity caching," in *2022 IEEE Global Communications Conference*. IEEE, 2022, pp. 141–146.

[19] G. Neglia, M. Garetto, and E. Leonardi, "Similarity caching: Theory and algorithms," *IEEE/ACM Transactions on Networking*, vol. 30, no. 2, pp. 475–486, 2021.

[20] T. S. Salem, G. Neglia, and D. Carra, "Ascent similarity caching with approximate indexes," *IEEE/ACM Transactions on Networking*, vol. 31, no. 3, pp. 1173–1186, 2022.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[22] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," in *The Twelfth International Conference on Learning Representations*, 2024.

[23] S. Luohe, H. Zhang, Y. Yao, Z. Li, and hai zhao, "Keep the cost down: A review on methods to optimize LLM's KV-cache consumption," in *First Conference on Language Modeling*, 2024.

[24] S. A. Bergman, Z. Ji, A.-M. Kermarrec, D. Petrescu, R. Pires, M. Randl, and M. de Vos, "Leveraging approximate caching for faster retrieval-augmented generation," in *Proceedings of the 5th Workshop on Machine Learning and Systems*, ser. EuroMLSys '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 66–73.

[25] X. Liu, S. He, Q. Wang, R. Li, Y. Song, Z. Liu, L. Li, Q. Liu, Z. Huang, Q. Guo, Z. He, and X. Qiu, "Beyond homogeneous attention: Memory-efficient llms via fourier-approximated kv cache," 2025.

[26] H. Sun, L.-W. Chang, W. Bao, S. Zheng, N. Zheng, X. Liu, H. Dong, Y. Chi, and B. Chen, "ShadowKV: KV cache in shadows for high-throughput long-context LLM inference," in *Forty-second International Conference on Machine Learning*, 2025.

[27] A. Sabnis, T. S. Salem, G. Neglia, M. Garetto, E. Leonardi, and R. K. Sitaraman, "GRADES: Gradient descent for similarity caching," *IEEE/ACM Transactions on Networking*, vol. 31, no. 1, pp. 30–41, 2022.

[28] M. Ott, S. Edunov, A. Baevski, A. Fan, S. Gross, N. Ng, D. Grangier, and M. Auli, "Fairseq: A fast, extensible toolkit for sequence modeling," in *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[29] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using siamese BERT-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[30] ——, "Making monolingual sentence embeddings multilingual using knowledge distillation," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2020.
[31] M. Abdin *et al.*, "Phi-3 technical report: A highly capable language model locally on your phone," 2024.
[32] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2333–2345, 2018.
[33] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *IEEE Conference on Computer Communications*. IEEE, 2019, pp. 10–18.
[34] F. A. Salaht, F. Desprez, and A. Lebre, "An overview of service placement problem in fog and edge computing," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–35, 2020.
[35] V. Farhadi, F. Mehmeti, T. He, T. F. La Porta, H. Khamfroush, S. Wang, K. S. Chan, and K. Poularakis, "Service placement and request scheduling for data-intensive applications in edge clouds," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 779–792, 2021.
[36] H. Yu, M. Neely, and X. Wei, "Online convex optimization with stochastic constraints," *Advances in Neural Information Processing Systems*, vol. 30, 2017.
[37] H. Yu and M. J. Neely, "Learning-aided optimization for energy-harvesting devices with outdated state information," *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1501–1514, 2019.
[38] E. Hazan *et al.*, "Introduction to online convex optimization," *Foundations and Trends® in Optimization*, vol. 2, no. 3-4, pp. 157–325, 2016.
[39] B. Gliwa, I. Mochol, M. Biesek, and A. Wawer, "SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization," in *Proceedings of the 2nd Workshop on New Frontiers in Summarization*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 70–79.
[40] I. Casanueva, T. Temcinas, D. Gerz, M. Henderson, and I. Vulic, "Efficient intent detection with dual sentence encoders," in *Proceedings of the 2nd Workshop on NLP for ConvAI - ACL 2020*, mar 2020.
[41] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.
[42] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*, 2004, pp. 74–81.
[43] A. Rakhlin, O. Shamir, and K. Sridharan, "Making gradient descent optimal for strongly convex stochastic optimization," in *Proceedings of the 29th International Coference on International Conference on Machine Learning*, 2012, p. 1571–1578.

APPENDIX A
PROOF OF THEOREM 2

The Lyapunov drift

$$\begin{aligned}
&\frac{1}{2}\mathbb{E}\left[Q_{t+1}^2 - Q_t^2 \middle| Q_t\right] \\
&\le \frac{1}{2}\mathbb{E}\left[(Q_t + \mathbb{1}_t \cdot g(\ell_t) - \overline{g})^2 - Q_t^2 \middle| Q_t\right] && \text{(Eq. (2))} \\
&= \frac{1}{2}\mathbb{E}\left[2Q_t(\mathbb{1}_t \cdot g(\ell_t) - \overline{g}) + (\mathbb{1}_t \cdot g(\ell_t) - \overline{g})^2 \middle| Q_t\right] \\
&\le \mathbb{E}\left[Q_t(\mathbb{1}_t \cdot g(\ell_t) - \overline{g}) \middle| Q_t\right] + \frac{G_1^2}{2} && \text{(Assumption 1)} \\
&= \mathbb{E}\left[Q_t(\mathbb{1}_t(g(\ell_t) - \hat{g}(\ell_t, a_t, b_t) + \hat{g}(\ell_t, a_t, b_t)) - \overline{g}) \middle| Q_t\right] + \frac{G_1^2}{2} \\
&= \mathbb{E}\left[Q_t \mathbb{1}_t(g(\ell_t) - \hat{g}(\ell_t, a_t, b_t)) \middle| Q_t\right] \\
&\quad + \mathbb{E}\left[Q_t(\mathbb{1}_t \cdot \hat{g}(\ell_t, a_t, b_t) - \overline{g}) \middle| Q_t\right] + \frac{G_1^2}{2} \\
&\le \left(\frac{VD}{G_0} + G_1\right)\mathbb{E}\left[\left|g(\ell_t) - \hat{g}(\ell_t, a_t, b_t)\right| \middle| Q_t\right] \\
&\quad + \mathbb{E}\left[Q_t(\mathbb{1}_t \cdot \hat{g}(\ell_t, a_t, b_t) - \overline{g}) \middle| Q_t\right] + \frac{G_1^2}{2}. && \text{(Theorem 1)}
\end{aligned}$$

For the objective (3) in **P2**, we have

$$\begin{aligned}
&\mathbb{E}\left[V(1 - \mathbb{1}_t) \cdot f(\mathbf{p}_t, \mathcal{C}_t) + Q_t(\mathbb{1}_t \cdot \hat{g}(\ell_t, a_t, b_t) - \overline{g}) \middle| Q_t\right] \\
&\le \mathbb{E}\left[V(1 - \mathbb{1}_t^*) \cdot f(\mathbf{p}_t, \mathcal{C}_t) + Q_t(\mathbb{1}_t^* \cdot \hat{g}(\ell_t, a_t, b_t) - \overline{g}) \middle| Q_t\right] \\
&\qquad (\mathbb{1}_t \text{ minimizes (3) for any instance of } Q_t) \\
&= \mathbb{E}\left[V(1 - \mathbb{1}_t^*)\left(f(\mathbf{p}_t, \mathcal{C}_t) - f(\mathbf{p}_t, \mathcal{C}^*) + f(\mathbf{p}_t, \mathcal{C}^*)\right)\right. \\
&\quad \left. + \; Q_t(\mathbb{1}_t^*(\hat{g}(\ell_t, a_t, b_t) - g(\ell_t) + g(\ell_t)) - \overline{g}) \middle| Q_t\right] \\
&= \mathbb{E}\left[V(1 - \mathbb{1}_t^*)(f(\mathbf{p}_t, \mathcal{C}_t) - f(\mathbf{p}_t, \mathcal{C}^*)) \middle| Q_t\right] \\
&\quad + \mathbb{E}\left[Q_t(\mathbb{1}_t^*(\hat{g}(\ell_t, a_t, b_t) - g(\ell_t)) \middle| Q_t\right] \\
&\quad + \mathbb{E}\left[V(1 - \mathbb{1}_t^*) \cdot f(\mathbf{p}_t, \mathcal{C}^*) \middle| Q_t\right] \\
&\quad + \mathbb{E}\left[Q_t(\mathbb{1}_t^* \cdot g(\ell_t) - \overline{g}) \middle| Q_t\right] \\
&\le V\mathbb{E}\left[\left|f(\mathbf{p}_t, \mathcal{C}_t) - f(\mathbf{p}_t, \mathcal{C}^*)\right| \middle| Q_t\right] \\
&\quad + \left(\frac{VD}{G_0} + G_1\right)\mathbb{E}\left[\left|\hat{g}(\ell_t, a_t, b_t) - g(\ell_t)\right| \middle| Q_t\right] \\
&\quad + \mathbb{E}\left[V(1 - \mathbb{1}_t^*) \cdot f(\mathbf{p}_t, \mathcal{C}^*)\right] \\
&\quad + Q_t\mathbb{E}\left[(\mathbb{1}_t^* \cdot g(\ell_t) - \overline{g})\right] && \text{(Theorem 1 \& cond. indep.)} \\
&\le V\mathbb{E}\left[\left|f(\mathbf{p}_t, \mathcal{C}_t) - f(\mathbf{p}_t, \mathcal{C}^*)\right| \middle| Q_t\right] \\
&\quad + \left(\frac{VD}{G_0} + G_1\right)\mathbb{E}\left[\left|\hat{g}(\ell_t, a_t, b_t) - g(\ell_t)\right| \middle| Q_t\right] + Vy^*,
\end{aligned}$$

where the last inequality is due to stationarity and the definition of y^*, as well as $\mathbb{E}\left[(\mathbb{1}_t^* \cdot g(\ell_t) - \overline{g})\right] = 0$.

Combining the above, we have

$$\begin{aligned}
&\frac{1}{2}\mathbb{E}\left[Q_{t+1}^2 - Q_t^2 \middle| Q_t\right] + \mathbb{E}\left[V(1 - \mathbb{1}_t) \cdot f(\mathbf{p}_t, \mathcal{C}_t) \middle| Q_t\right] \\
&\le \frac{G_1^2}{2} + \left(\frac{VD}{G_0} + G_1\right)\mathbb{E}\left[\left|g(\ell_t) - \hat{g}(\ell_t, a_t, b_t)\right| \middle| Q_t\right] \\
&\quad + \mathbb{E}\left[V(1 - \mathbb{1}_t) \cdot f(\mathbf{p}_t, \mathcal{C}_t) + Q_t(\mathbb{1}_t \cdot \hat{g}(\ell_t, a_t, b_t) - \overline{g}) \middle| Q_t\right] \\
&\le Vy^* + \frac{G_1^2}{2} + 2\left(\frac{VD}{G_0} + G_1\right)\mathbb{E}\left[\left|g(\ell_t) - \hat{g}(\ell_t, a_t, b_t)\right| \middle| Q_t\right] \\
&\quad + V\mathbb{E}\left[\left|f(\mathbf{p}_t, \mathcal{C}_t) - f(\mathbf{p}_t, \mathcal{C}^*)\right| \middle| Q_t\right].
\end{aligned}$$

Noting $Q_1 = 0$ by definition, after taking total expectation, averaging over t, and rearranging, we obtain the result. □

APPENDIX B
PROOF OF LEMMA 2

It can be easily shown that $h(\ell_t, a, b)$ is smooth and strongly convex with respect to a and b within the convex set Ω. Let $\psi(t) := |\tau \le t : \mathbb{1}_\tau = 1|$ be the number of times of $\mathbb{1}_\tau = 1$ for all $\tau \le t$. Define $h^* := \mathbb{E}\left[h(\ell_t, a^*, b^*)\right]$. Due to the i.i.d. assumption in Assumption 2, together with (7) and according to standard analysis of stochastic gradient descent [43], we have

$$\mathbb{E}\left[h(\ell_t, a, b)\right] - h^* \le \mathcal{O}\left(\frac{1}{\psi(t)}\right).$$

We note that

$$\begin{aligned}
\left(\mathbb{E}\left[\left|\hat{g}(\ell_t, a, b) - g(\ell_t)\right|\right]\right)^2 &\le \mathbb{E}\left[\left(\hat{g}(\ell_t, a, b) - g(\ell_t)\right)^2\right] \\
&= \mathbb{E}\left[h(\ell_t, a, b)\right] \le h^* + \mathcal{O}\left(\frac{1}{\psi(t)}\right),
\end{aligned}$$

where the first inequality is due to Jensen's inequality. Thus,

$$\mathbb{E}\left[\left|\hat{g}(\ell_t, a, b) - g(\ell_t)\right|\right] \le \mathcal{O}\left(\sqrt{h^*} + \frac{1}{\sqrt{\psi(t)}}\right),$$

where we use that h^* is upper bounded by a constant due to Assumption 1. From Lemma 1, there are at most $\gamma + 1$ different values of t with the same value of $\psi(t)$, and the maximum value of $\psi(t)$ is T'. Therefore,

$$\begin{aligned}
\sum_{t=1}^{T}\mathbb{E}\left[\left|\hat{g}(\ell_t, a, b) - g(\ell_t)\right|\right] &\le \mathcal{O}\left(T\sqrt{h^*} + (1+\gamma)\sum_{t'=1}^{T'}\frac{1}{\sqrt{t'}}\right) \\
&= \mathcal{O}\left(T\sqrt{h^*} + (1+\gamma)\sqrt{T'}\right).
\end{aligned}$$

Dividing by T, using $T' \le T$ and $\mathcal{O}(\gamma + 1) = \mathcal{O}(V)$ gives the result. □

APPENDIX C
PROOF OF LEMMA 3

We first derive an upper bound of $\mathbb{E}\left[f(\mathbf{p}_t, \mathcal{C}^*)\right]$. Consider a cache of size $K' := \left(\lfloor \sqrt[d]{K} \rfloor\right)^d \le K$. Due to Assumption 2, when the cache size is K', the optimal cache placement $\mathcal{C}_{K'}^*$ that minimizes $\mathbb{E}\left[f(\mathbf{p}_t, \mathcal{C}_{K'}^*)\right]$ is to fill the hypercube containing the input embedding space with K' non-overlapping hypercubes each with side length $2r$, where $r = \frac{R}{\sqrt[d]{K'}} \le \frac{R}{\sqrt[d]{K}-1}$ and the total volume $K'(2r)^d = (2R)^d$. Such a filling is possible since $R/r = \sqrt[d]{K'} = \lfloor \sqrt[d]{K} \rfloor$ is an integer. The input embedding corresponding to each cached item in the K'-sized cache is the center of each of such small hypercube with side length $2r$. Under this $\mathcal{C}_{K'}^*$, the maximum distance of any input embedding to the embedding of its nearest cached item is $r\sqrt{d}$. Thus,

$$f(\mathbf{p}_t, \mathcal{C}_{K'}^*) \le r\sqrt{d} \le \frac{R\sqrt{d}}{\sqrt[d]{K} - 1},$$

for all $\mathbf{p}_t$. Since $K \ge K'$, for the optimal placement $\mathcal{C}^*$ of the cache with size K, we have

$$\mathbb{E}\left[f(\mathbf{p}_t, \mathcal{C}^*)\right] \le \mathbb{E}\left[f(\mathbf{p}_t, \mathcal{C}_{K'}^*)\right] \le \frac{R\sqrt{d}}{\sqrt[d]{K} - 1}.$$

For any t such that there exist at least K different values of $\tau < t$ where $\mathbb{1}_\tau = 1$, the cache $\mathcal{C}_t$ contains K items.

From Assumption 2, the embeddings of these K items are uniformly distributed within the hypercube that has side length $2R$. For any embedding $\mathbf{p}_t$, when it is covered by a hypercube with side length $2\tilde{r} := 2r\sqrt[d]{\log K}$ with a cached embedding as its center, the maximum distance between $\mathbf{p}_t$ and its nearest cached embedding is $\tilde{r}\sqrt{d}$. The probability that $\mathbf{p}_t$ is not covered by any hypercube with side length $2\tilde{r}$ is

$$\left(1-\frac{(\tilde{r})^d}{R^d}\right)^K = \left(1-\frac{\log K}{K'}\right)^K \leq \left(1-\frac{\log K}{K}\right)^K \leq e^{-\log K} = 1/K,$$

and the maximum distance between any two points in a hypercube with side length $2R$ is $2R\sqrt{d}$. Hence,

$$\mathbb{E}\left[f(\mathbf{p}_t,\mathcal{C}_t)\right] \leq \tilde{r}\sqrt{d} + \frac{2R\sqrt{d}}{K} = R\sqrt{d}\cdot\sqrt[d]{\frac{\log K}{K'}} + \frac{2R\sqrt{d}}{K} \leq R\sqrt{d}\cdot\frac{\sqrt[d]{\log K}}{\sqrt[d]{K}-1} + \frac{2R\sqrt{d}}{K}.$$

Then, we combine the above and note that, when $|\mathcal{C}_t| = K$, we have

$$\mathbb{E}\left[|f(\mathbf{p}_t,\mathcal{C}_t) - f(\mathbf{p}_t,\mathcal{C}^*)|\right] \leq \mathbb{E}\left[f(\mathbf{p}_t,\mathcal{C}_t)\right] + \mathbb{E}\left[f(\mathbf{p}_t,\mathcal{C}^*)\right] \leq R\sqrt{d}\cdot\frac{\sqrt[d]{\log K}}{\sqrt[d]{K}-1} + \frac{2R\sqrt{d}}{K} + \frac{R\sqrt{d}}{\sqrt[d]{K}} = \mathcal{O}\left(\sqrt[d]{\frac{\log K}{K}}\right),$$

since $f(\cdot,\cdot) \geq 0$.

For the case of $|\mathcal{C}_t| < K$, we know from Lemma 1 that there are at most $(\gamma+1)K = \mathcal{O}(VK)$ slots for which $|\mathcal{C}_t| < K$, and we bound $\mathbb{E}\left[f(\mathbf{p}_t,\mathcal{C}_t)\right] \leq 2R\sqrt{d}$ for such t where $|\mathcal{C}_t| < K$.

Then, we combine both cases of the range/value of $|\mathcal{C}_t|$, average over T slots, and obtain the result after absorbing the constants R and d in $\mathcal{O}(\cdot)$. □