Change the Channel: Timing Side Channel Exploits for LLM Server Caches

Kirtan Shah
University of California, San Diego

Madurya Suresh
University of California, San Diego

Abstract

LLMs are rapidly gaining widespread adoption across a range of use cases. As LLM service providers seek to reduce costs, caching has become a popular way to decrease computational load and latency. However, cache implementation may expose users to novel security risks. Our work extends previous research on exploiting cache timing side channels – particularly the Peeping Neighbor Attack (PNA) in semantic caches and Prompt-Stealing Attack in Key-Value caches. Our contributions include theoretical validation of the PNA, empirical confirmation through simulation, a more realistic threat model for cache timing attacks, and a discussion on the improbability of successful KV cache timing attacks in practical scenarios.

1 Introduction

As users increasingly trust LLMs with day-to-day tasks, it's important that the systems that serve them are efficient, accurate, and secure. One of the ways that LLM server hosts reduce costs is by utilizing *caching*. Some popular mechanisms include **Key-Value Caching** [1], **Semantic Caching** [2], and **Context Caching** [3]. However, the convenience of caching also comes with security implications and trade-offs. Specifically, user prompts can contain private data or personally identifiable information (PII). Caching introduces the potential for leakage, and therefore there is an emerging research area exploring: **cache timing side channel attacks**.

Cache timing side-channel attacks involve exploiting a cache shared between multiple users of an LLM server to recover a victim's prompt, answer, or private attributes. In this attack, an attacker need only gain general user querying privileges in order to time response and infer information that might be stored in the cache from previous user queries.

Specifically, we explore the plausibility of exploiting KV cache timing differences to glean useful information, and present threat models wherein an attacker is more likely to infer information from the semantic cache.

Contributions. In our work, we replicate the semantic cache-based Peeping Neighbor Attack presented in (*Unveiling Timing Side Channels in LLM Serving Systems*, Song et al.) [4]. We also extend this work by presenting a framework in which an attacker can cover a much larger prompt space in attempting to infer another user's prompt. Since timing attacks are sensitive to real-world conditions, we propose a refined threat model grounded in strong but specific assumptions. Specifically, our contributions include:

- Partially corroborate theoretical results of the Peeping Neighbor Attack (PNA)
- Corroborate results of the PNA empirically with a simulated setup with a real cache/LLM
- Propose a more realistic threat model with stronger assumptions
- Present an argument for the implausibility of KV cache side channel attacks given real world conditions

2 Background

Semantic caching stores embeddings of prompt-answer pairs from previous requests [5]. Upon receiving a new prompt, the system checks for semantic similarity with cached prompts. If a match is found, the cached answer is returned; otherwise, the model processes the prompt and adds the new prompt-answer pair to the cache. This reduces response latency, and decreases computational costs.

KV caching caches key-value pairs generated during the model's attention mechanism [1]. These pairs are reused in subsequent steps, reducing the need to recompute them. This speeds up inference, especially for long sequences, by optimizing memory usage and reducing computational overhead.

3 Previous Research

Peeping Neighbor Attack. The existing threat model in *Unveiling Timing Side Channels* [4] outlines an attack scenario for inferring another user's private prompt through a semantic cache timing side channel. It assumes a prompt template, for example "Compose a meeting agenda for an interdisciplinary team discussing the treatment plan for [name] with [medical condition]." The victim prompt must be *semantically similar* to the prompt template and contain the same private attributes (name, medical condition) in order for an attack prompt to register a cache "hit."

Encrypted network packet side channel. Multiple papers [6], [7] have demonstrated the real-world plausibility of deducing information from encrypted network packets transmitted between a user and LLM service providers. These include inferring conversation topic [6] as well as inferring a completion response from packet lengths [7].

4 Threat Model

4.1 Semantic Cache

Semantic cache exploits assume that an attacker knows the semantic space and set of possible private attributes, which is unrealistic in most scenarios. On a general LLM server with diverse use cases, attackers would face a prohibitively large search space, requiring them to test numerous combinations of sentences and attributes. Thus, semantic cache exploits may be less feasible than they appear; the effort to extract private information often outweighs the benefits. We propose a refined threat model for semantic cache side-channel attacks:

Scenario 1. Consider an LLM server with a narrow use case and a small set of users. For example, a Human Resources chatbot that uses semantic caching used by only members of the same company. An attacker could much more efficiently search through the set of names since it is a small set (employees of the company). Instead of discovering private attributes from scratch, attackers would use a "guess-and-check" approach to confirm that someone with known attributes P is discussing known topic T.

Scenario 2. Consider a large LLM server with many users and many use cases. In this scenario, the search space for semantic cache attacks becomes prohibitively large. Therefore, an attacker would likely have to pair the semantic cache side channel attack with another LLM side channel attack. In *Remote Timing Attacks* [6], the authors demonstrate an attacker successfully retrieving the user's topic T of conversation using only encrypted network packets. Once an attacker knows the topic of conversation, they can generate a set of attack prompt templates that cover a different semantic space within the topic's semantic space. However, they would still have to guess the private attributes of their victim speaking about topic T. So, this scenario comes with stronger assumptions than #1, but is more specific in its execution. More research would have to be done in this area. We will focus on scenario #1 in our experiments.

4.2 KV Cache

We look at the set of *all* possible threat models relating to KV cache exploits, where the attacker has black-box access to an LLM service provider's completions API. Specifically the attacker has the following capabilities and restrictions:

- Can query the LLM service provider with an arbitrary prompt an arbitrary number of times
- Can accurately time requests
- Is located remotely (i.e. separated from the LLM host through an internet connection)

5 Verifying Previous Research

We aim to verify the results from the *Unveiling Timing Side Channels* [4] paper. The authors kindly provided their code for us to replicate their experiments.
We used the prompt template "Compose a meeting agenda for [Name] with [medical condition]", populated with the python package's NameDataset and a Medical Condition Dataset, respectively [8]. We generated $N = 100$ semantically similar templates using gpt-3.5-turbos. 20% of these sentences are used as "True Label" sentences, or victims prompts that we are trying to retrieve. The rest are used as attack sentences. We then replaced the private attributes among these sentences to construct the "False Label" set. The attack sentences should not be semantically similar to "False label" sentences since private attributes differ.

Next, we used *distilbert-base-uncased* to compute embeddings of attack sentences. We then sort them by embeddings that are closest to the mean embedding by euclidean distance.
Finally, we compute the semantic similarity of every attack sentence to every other. If a "worse" attack sentence is semantically similar to a "better" attack sentence, the "worse" attack sentence is removed from the set of attack sentences. This is to avoid poisoning the cache with an attacker's own prompts in a real-world attack.

To simulate the attack, we use the ONNX Embedding Evaluation model [9] with a 0.8 threshold. Each attack sentence is tested against sentences from the True and False

Label sets. If the sentences are semantically similar above the threshold, it's a "cache hit"; otherwise, a "cache miss." The goal is for attack sentences to hit True Label sentences and miss False Label sentences.

For our best attack sentences (the first row in 1), we observe comparable TPR/FPR with the *Unveiling Timing Side Channels* paper. However, we found that additional attack sentences did not improve results.

We reason that the candidates generated by gpt-3.5-turbo have built in randomness. Therefore in the general case, the top-ranked attack sentence yields a good classifier, but additional attack sentences are not likely to represent the space of semantically similar sentences very well.

# Trials	Our TPR	Our FPR	Their TPR	Their FPR
1	85.2	5.5	85.3	3.2
2	81.3	3.4	91.9	3.3
3	80.0	2.8	95.1	3.6
4	79.3	2.1	96.2	3.9

Table 1: Theoretical attack efficiency on medical meeting agenda prompt (%).

6 Experimental Setup - PNA

Next, we aim to test the *Unveiling Timing Side Channels* paper's theoretical results on a real cache. This way, we can see if an attacker can actually retrieve a victim's answer (and therefore confirm their private attributes and the semantic space of their prompt), using *only* the methods of *querying* and *timing responses*.

Our experimental setup involves setting up an LLM server with a semantic cache. We used the LangChain framework to set up a LLM serving application, and then integrated GPTCache into the application as a caching mechanism [10] [5]. When our application is queried, the ONNX evaluation model computes the query's similarity to all cached queries. If there is a cache hit, we return the corresponding cached answer immediately to the user, without querying the LLM at all. If we get a cache miss, then we query the LLM using a request to the OpenAI gpt-3.5-turbo-instruct. Then, the response from the model is what gets returned to the user.

7 Experiments - Semantic Cache Exploit

Our two main goals in contributing to the research done in *Unveiling Timing Side Channels* were to (1) determine if an attacker could reliably judge whether or not their attack sentence was a cache hit or miss by solely timing responses, and (2) determine if the attack sentences generated via the methodology in *Unveiling Timing Side Channels* can actually recover a victim's cached answer in a real cache. In other words, we want to empirically prove (or disprove) their theoretical results.

We first generated the True Label, False Label, and attack sentences in the same manner described in our **Verifying Previous Research** section. Consider a set of attack sentences, which corresponds to a set of True Label sentences and a set of False Label sentences. First, for each True Label sentence (we predict a cache hit for these), we query the server to simulate a victim prompt, and record the response time. Then, we query 1 of the attack sentences, and record the response time. We also record whether or not the answers that the victim and attacker received are the same, as this can confirm if the attacker actually received a cached answer. Then, we flush the cache. We then ask the *same* victim prompt from the True Label set, then ask the next attack sentence. We flush the cache, and repeat until we have tried each attack sentence individually on the same victim prompt. Once this is done, we move on to the next True Label sentence in the True Label set, testing each attack sentence against it and recording response times and answers.

Then, we follow a similar process with the False Label set and attack sentence set. We query from the False Label set (we predict a cache miss for these, as the private attributes don't match the attack sentence private attributes) to simulate a victim prompt, and then query from the attack sentence set to simulate a (failed) attacker prompt. We record the response times and answers, and flush the cache in between each trial.

We repeat this process for all sets of True Label sentences, False Label sentences, and attack sentences in our dataset. Because we had limited computing resources, we tested this process on an arbitrary subset (N=698) of the entire dataset generated from the theoretical study.

We then fit a Logistic Regression classifier trained on **response times** against **ground truth cache hit**, which we can determine based on whether the victim and attacker received the same answer or not. We found that such a classifier performed extremely well, with an AUC of .99 (see 2). This is likely because the average cache hit response time was **0.19417**, with a standard deviation of 0.04094, and the average cache miss response time was **2.8436**, with a standard deviation of 0.75604. So, the average cache hit and cache miss time differed by around 2000 milliseconds. We conclude that an attacker could reliably tell whether or not their attack sentence actually hit or missed the cache by solely recording response times.

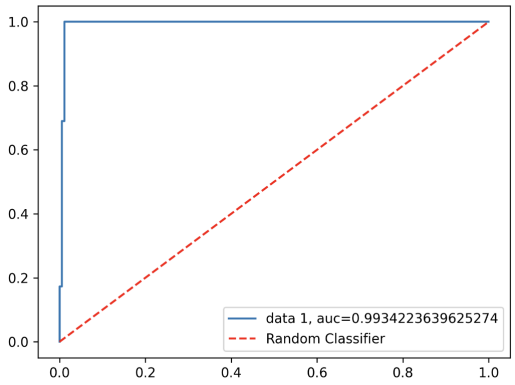

Figure 1: ROC curve for classifying response times as cache hit or cache miss

Next, we wanted to determine if, on a real cache, the attack sentences derived from the theoretical methodology of *Unveiling Timing Side Channels* that were *predicted* to be cache hits were actually cache hits, and those attack sentences that we predicted to be cache misses were actually cache misses. This will show us the effectiveness of the existing attack sentence generation methodology on a real LLM server using a semantic cache. We did this by fitting a Logistic Regression classifier trained on cache hit/miss predictions (whether or not the victim prompt has the same private attributes as the attack prompt or not) and ground truth cache hit/miss for the given pair of victim and attack prompts.

We found that when we used the best attack sentence, which casts the widest net around its semantic space, we were able to successfully retrieve the victim prompt around 84% of the time. Our logistic regression classifier had an AUC of **0.9549** (Figure 2).

We fit a Logistic Regression classifier similarly for attacks with 2 attack sentences (AUC=0.7797), 3 attack sentences (AUC=0.8042), and then 4 attack sentences (AUC=0.7610). We found that when we used more attack sentences (which are the "weaker" ones), our attack didn't necessarily become more effective at retrieving a victim answer. This is likely because these weaker attack sentences cover less of the embedding space, and are therefore less likely to actually hit the victim prompt, resulting in more false negatives, as we incorrectly predicted a cache hit. This aligns with our theoretical findings, where we did *not* find that adding more attack sentences increased our overall TPR (see 1).

Overall, we found that an attacker can generally judge whether their attack sentence was a cache hit or miss solely

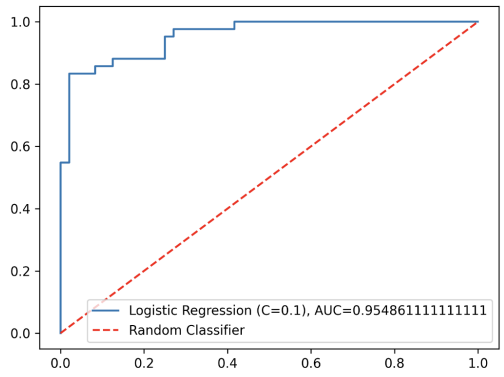

Figure 2: ROC curve for predicting if an attack sentence will cause a desired cache hit or desired cache miss.

by querying the model and timing responses. Additionally, we found that using the methodology to derive attack sentences described in *Unveiling Timing Side Channels* was generally successful in retrieving desired victim answers, and avoiding cache hits for prompts with different private attributes. Finally, we found that using an attack sentence that represented the largest amount of semantic space was the most effective in retrieving a victim's cached answers in our empirical experiments.

We additionally discovered a potential private information leakage risk while using GPTCache with our LLM server. In addition to the ONNX Similarity Evaluation Model, we tested GPTCache's default evaluation settings, though the exact model used was unclear or inaccurate in the documentation. We found that, with a cleared cache, a user could query "*Compose a meeting agenda for Amanda with the cough*", and receive some answer *A*. After this, another user could query "*Compose a meeting agenda for Max with the cold*", and they will receive the cached answer *A* about "Amanda with the cough." This demonstrates a risk of private information leakage in GPTCache's default settings, unlike the ONNX Evaluation Model which doesn't exhibit this behavior. Future research could explore the leakage profiles of different embedding similarity models.

8 Experiments - Generalizing PNA

We generalized the Semantic Cache based PNA by creating a framework for an attacker to very simply provide a prompt template at sets of private attributes to quickly generate attack sentence candidates.

```
orig_template = "Generate a compensation report for
    employee {name} with base pay {salary}"

private_attr_sets = [
```

```
    names['US']['M'] + names['US']['F'],
    hr_ds['Salary'].values.tolist()
]

perform_attack(orig_template, private_attr_sets)
```

An attacker could use an LLM to generate prompt templates to run the PNA over a large space of prompts.

# Trials	TPR (%)	FPR (%)
1	87.17	0.25
2	81.40	0.84
3	74.12	1.38
4	62.9	1.45

Table 2: Theoretical attack efficiency on employee salary prompt.

The results of the PNA attack on the above prompt template and the Human Resources Dataset of salaries [11] are shown in Figure 2

9 KV Cache Exploits

We provide an argument for the implausibility of KV cache exploits under real world conditions and under the set of threat models outlined in Section 4.2.

9.1 Experimental Setup

We used the SGLang LLM server [12] running TheBloke/Llama-2-7B-GPTQ [13] on a RTX 2070 MaxQ GPU. We executed the following procedure $N = 1000$ times:

- Flush SGLang cache
- Query LLM with "Python is a programming language that developers across the world use to"
- t_{hit} = Time-to-first-token LLM with "Python is a programming language that"
- t_{miss} = Time-to-first-token LLM with "Java was a programming language that"

We make the generous assumption that the attacker is able to guess the first 6 tokens of the victim prompt. Collecting the set of T_{hit}, T_{miss}, we fit a Logistic Regression classifier and observe ROC curve shown in Figure 3.

9.2 Attack Plausibility Argument

$AUC = .71$ is acceptable, however, this would assume the attacker is on the same local network as the LLM service provider, violating a condition of the proposed threat model.

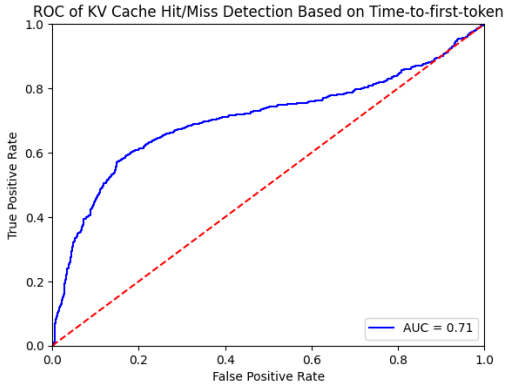

Figure 3: ROC of ideal hit/miss classifier

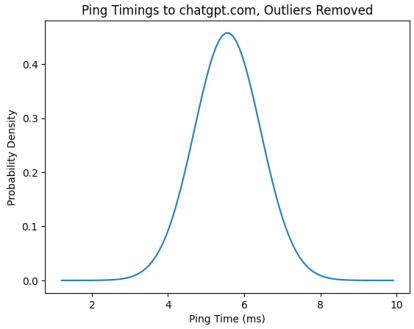

Figure 4: Ping Distribution to chatgpt.com

Figure 4 shows a distribution of timed pings to chatgpt.com with outliers removed, revealing a lower bound standard deviation of $0.87ms$). After adding a random variable $X \sim \mathcal{N}(0, 0.871)$ representing noise, the ROC curve (Figure 5) indicates a classifier that is indistinguishable from random. Note that the attacker's ability to infer information from the private cache is contingent on its ability to accurately distinguish between a cache hit/miss. Therefore, we argue that KV cache timing side channels are implausible given real world attacker threat models.

10 Discussion

Based on our experiments, we conclude that semantic cache timing attacks are a possibility. They are made easier when an attacker knows the general embedding space that a victim's prompt is inside of, and a set of private attributes that could be inside of the victim's prompt. Otherwise, the attack becomes unreasonable from a computing resource and time perspective.

In practice, semantic caches can be added as a layer to one's LLM-serving application. We find that the use of semantic caching as a layer to increase an application's

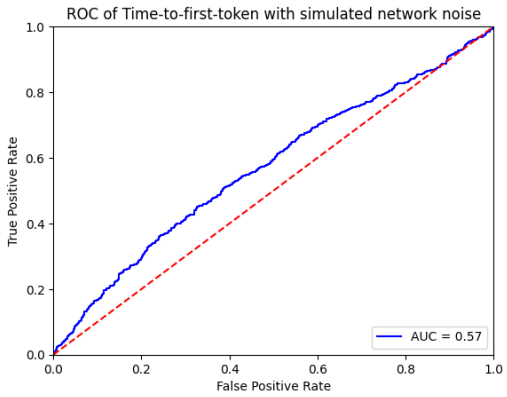

Figure 5: ROC of hit/miss classifier w/ simulated network latency noise

latency is generally in a preliminary stage of deployment. As it gains more popularity, LLM-serving applications should seriously consider the trade-offs and likelihoods of private information leakages through the semantic cache, especially if the applications have a small use case or user base.

Google's Gemini [3] model provides a functionality for context caching, which uses KV caching to store large contexts for reuse on prompts with a common prefix [14]. Further research on whether or not context caching side channel attacks are implausible with real network delays (as we demonstrated with KV caching) would be beneficial.

Additionally, we propose that an attacker could combine a networking packet side channel attack to determine a user's topic of conversation, as done in *Remote Timing Attacks*[6], to narrow down a large search space to a reasonably smaller one, and then use a cache timing side channel attack in order to retrieve a victim's prompt or private information. More research in the possibility of combining multiple LLM server side channel attacks could refine this model further.

11 Conclusion

In this paper, we verified the theoretical plausibility of a semantic cache timing attack on an LLM server, and tested its robustness to multiple prompt datasets. We then proved that such attacks were possible on a deployed LLM server using a semantic cache. Finally, we demonstrated that, while KV cache side channel exploitation is theoretically possible, it is unlikely under real world conditions. The existing threat model for semantic cache exploits tends to make sweeping assumptions about attackers' knowledge of victim information – we refined this model by explaining that an attacker would need to either attack in a narrow search space, or combine cache timing attacks with another side channel attack, although further research on this remains to be explored.

12 Individual Contributions

Kirtan

- Created timing harness for SGLang/KV-cache measurement study
- Generalized PNA code to any input prompt template
- Ran attack on new prompt template (HR dataset)

Madurya

- Implemented empirical PNA attack by simulating attack queries
- Setup timing harness for GPTCache
- Developed detailed threat model

13 Code

Github Link: https://github.com/kirtan-shah/LLM_Timing_SideChannels/tree/main

References

[1] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023.

[2] GitHub - zilliztech/GPTCache: Semantic cache for LLMs. Fully integrated with LangChain and llama_index. — github.com. https://github.com/zilliztech/GPTCache.

[3] Context caching. https://ai.google.dev/gemini-api/docs/caching?lang=python.

[4] Linke Song, Zixuan Pang, Wenhao Wang, Zihao Wang, XiaoFeng Wang, Hongbo Chen, Wei Song, Yier Jin, Dan Meng, and Rui Hou. The early bird catches the leak: Unveiling timing side channels in llm serving systems, 2024.

[5] Sajal Regmi and Chetan Phakami Pun. Gpt semantic cache: Reducing llm costs and latency via semantic embedding caching, 2024.

[6] Nicholas Carlini and Milad Nasr. Remote timing attacks on efficient language model inference, 2024.

[7] Roy Weiss, Daniel Ayzenshteyn, Guy Amit, and Yisroel Mirsky. What was your prompt? a remote keylogging attack on ai assistants, 2024.

[8] Medquad. https://huggingface.co/datasets/lavita/medquad, 2019.

[9] GitHub - onnx/onnx: Open standard for machine learning interoperability — github.com. https://github.com/onnx/onnx.

[10] Langchain. https://www.langchain.com/.

[11] Dr. Richard A. Huebner and Dr. Carla Patalano. Human resources data set, 2020.

[12] Sglang. https://sgl-project.github.io/.

[13] Thebloke/llama-2-7b-gptq. https://huggingface.co/TheBloke/Llama-2-7B-GPTQ.

[14] Cunchen Hu, Heyang Huang, Junhao Hu, Jiang Xu, Xusheng Chen, Tao Xie, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, and Yizhou Shan. Memserve: Context caching for disaggregated llm serving with elastic memory pool, 2024.