Stanford CS 191W Senior Project: Timing Attacks on Prompt Caching in Language Model APIs

Chenchen Gu [1] **Xiang Lisa Li** [1] **Rohith Kuditipudi** [1] **Percy Liang** [1] **Tatsunori Hashimoto** [1]

Abstract

Prompt caching in large language models (LLMs) enables cache hits between prompts with matching prefixes, reducing response times. This results in data-dependent timing variations, as cached prompts are processed faster than non-cached prompts, introducing the risk of side-channel timing attacks. For example, if the prompt cache is shared across users, then an attacker could detect cache hits from timing differences to learn information about other users' prompts, as a prompt being cached implies that a user recently sent it. We develop and conduct statistical audits to detect prompt caching on real-world LLM API providers. We detect cache sharing across users and organizations in seven API providers, including OpenAI, resulting in privacy leakage about users' prompts. We also extract model architecture information by detecting caching. Namely, we find that OpenAI's embedding model is a decoder-only Transformer, which was previously not publicly known.

1. Introduction

As Transformer large language models (LLMs) grow larger, they become costlier and slower to run. In response, recent work has developed optimizations to make LLM inference and serving more efficient, such as prompt caching (Zheng et al., 2024; Gim et al., 2024). In prompt caching, reuse of the attention key-value (KV) cache across requests enables cache hits between prompts with matching prefixes, reducing response times. Recently, Anthropic and OpenAI officially released prompt caching features in August and October 2024, respectively.

However, prompt caching results in data-dependent timing variations—cached prompts will be processed faster than non-cached prompts, introducing the risk of side-channel timing attacks. For example, an attacker could detect cache hits by looking for fast API response times. If the prompt cache is shared across users, then detecting cache hits can leak private information about other users' prompts, as a prompt being cached implies that a user recently sent it. In general, timing differences between cache hits and cache misses have been widely exploited in computer security, such as in the infamous Meltdown (Lipp et al., 2018) and Spectre attacks (Kocher et al., 2019).

In this paper, we investigate timing attacks on prompt caching in real-world LLM APIs. First, we develop a rigorous auditing procedure using statistical hypothesis testing to determine if an API is caching prompts and the level of cache sharing across users. To do so, we construct and sample response times from two procedures: one that attempts to produce cache hits, and one that produces cache misses. Under the null hypothesis of no prompt caching, where only cache misses are possible, these procedures produce identical distributions of times. Accordingly, we detect caching if we find a statistically significant difference between these distributions.

We conducted audits on 17 LLM API providers, detecting prompt caching in eight API providers. In seven of these providers, the prompt cache was shared across users and organizations (when we conducted our audits in September and October 2024). On these APIs, an attacker can detect cache hits from timing differences to infer that another user sent a prompt that shares a prefix with a given prompt.

We also extract model architecture information by detecting caching. Cache hits between prompts that share a prefix but have different suffixes are possible only in autoregressive decoder-only Transformers, where each token attends only to previous tokens. Therefore, detecting cache hits on such prompts indicates that the model has a decoder-only architecture. Virtually all chat models are decoder-only, but top open-weights embedding models include both encoder and decoder models. As such, extracting architectural information can be interesting for proprietary embedding models. By detecting prompt prefix caching, we find that OpenAI's text-embedding-3-small has a decoder-only architecture, which was previously not publicly known.

Responsible disclosure. In October 2024, we shared our findings with each API provider in which we detected

[1]Stanford University. Correspondence to: Chenchen Gu <cygu@cs.stanford.edu>, Tatsunori Hashimoto <thashim@stanford.edu>.

prompt caching. After this disclosure, we waited 60 days before publicly releasing our findings. During this window, multiple providers made changes to mitigate vulnerabilities, e.g., disabling cache sharing across organizations.

2. Preliminaries and Threat Model

First, we briefly describe prompt caching, our threat model, and our terminology of API users and organizations.

2.1. Prompt caching

Recent works have proposed prompt caching in Transformer (Vaswani et al., 2017) LLM serving by reusing the attention key-value (KV) cache across requests (Zheng et al., 2024; Gim et al., 2024). In these methods, a prompt is cached by storing the prompt's attention KV cache. Then, if a subsequent prompt has a matching prefix with a cached prompt, the KV cache for the matching prefix can be retrieved from the cache, instead of computing the KV cache from scratch. As a result, cache hits will tend to have a faster time to first token (TTFT), which is the time taken to process the prompt and generate the first response token. In decoder-only Transformers, where each token only attends to previous tokens, reusing the KV cache for matching prefixes exactly preserves model behavior, even when the prompt suffixes differ.

Several API providers have recently released prompt caching features, including Anthropic, DeepSeek, Fireworks AI, and OpenAI. These providers do not state technical details of prompt caching, but all these providers enable cache hits for (and only for) exact prefix matches between prompts. For our purposes, the precise implementation of prompt caching is largely unimportant. The properties of prompt caching that we exploit are:

1. Cache hits occur on prefix matches between prompts.
2. Cache hits tend to be faster than cache misses.

2.2. Threat Model

We assume that an attacker can send arbitrary prompts to the API (possibly subject to some maximum length) and measure either client-side or server-side API response times, or both. The client-side timing is obtained simply by measuring the time elapsed between when the attacker sends the API request and when the attacker receives the API response. The server-side timing can be measured if it is contained somewhere in the API response.[1]

In addition, we assume that the attacker is able to specifically measure the time to first token, instead of the time taken to generate the entire LLM output. An attacker can achieve this by setting the maximum tokens parameter to 1, which restricts the LLM output to only contain 1 token. This parameter is supported by most, if not all, real-world LLM APIs. Alternatively, the attacker can enable response streaming, where the server sends the LLM response in chunks as soon as each chunk is generated, as opposed to sending the entire response after the LLM has finished generation. Then, the attacker can approximate the time to first token by measuring the response time of the first chunk, which should contain only a few tokens at most. Response streaming is supported by most real-world LLM APIs.

[1]We can measure server-side timing in more than half of the APIs we test, often from undocumented fields in the HTTP headers of the API response.

2.3. Terminology: Users and Organizations

To facilitate our discussion of privacy leakages in APIs, we define our terminology of users and organizations. A **user** is one person that uses the API. Each user has a unique email/username and login password. An **organization** contains many users, but shares a billing system, usage limits, centralized membership management, etc. Organizations are useful for collaborative environments, such as companies or research groups. Many, but not all, API providers support organizations, although they are sometimes called other terms, such as teams or accounts. For consistency and simplicity, we refer to them all as organizations. We differentiate between cache sharing within users of an organization versus across organizations, as they lead to different severities of privacy leakage.

3. Auditing APIs for Prompt Caching

Some API providers have publicly stated that they perform prompt caching, such as OpenAI and Anthropic, but some API providers may be performing prompt caching without announcing it. Also, even if a provider announces prompt caching, they may not clarify whether the cache is shared across users or organizations. Therefore, we propose and conduct an audit procedure to determine whether an API provider is caching prompts and the level of cache sharing across users and organizations. Our audit uses statistical hypothesis testing and outputs true p-values with respect to the null hypothesis of no caching, allowing for guarantees on the false positive rate.

3.1. Formulation: Statistical Hypothesis Testing

We formulate our audit as a statistical hypothesis test using the following null and alternative hypotheses:

$$H_0 : \text{API is not caching prompts,}$$
$$H_1 : \text{API is caching prompts.}$$

The caching in H_0 does not refer only to prompt caching via the KV cache reuse described earlier. More verbosely, H_0 can be written as "when the API receives a prompt x, the API does not store any information about x that affects the response times of future prompts."

To test these hypotheses, we construct procedures that attempt to produce and measure the response times of cache hits and cache misses. At a high level, to attempt to produce a cache hit, we send the same prompt to the API multiple times to try to cache the prompt, then we send the prompt again to try to hit the cache. To produce a cache miss, we simply send a random prompt.

Let $\mathcal{D}_{\text{hit}}$ and $\mathcal{D}_{\text{miss}}$ be the distributions of response times from these cache hit and cache miss procedures, respectively. Under the null hypothesis H_0 of no caching, $\mathcal{D}_{\text{hit}} = \mathcal{D}_{\text{miss}}$, as both procedures will produce only cache misses. In contrast, under the alternative hypothesis H_1 of caching, we would expect the cache hit times to tend to be faster than the cache miss times, so $\mathcal{D}_{\text{hit}} \neq \mathcal{D}_{\text{miss}}$. Now, we can reformulate our hypotheses as

$$H_0 : \mathcal{D}_{\text{hit}} = \mathcal{D}_{\text{miss}},$$
$$H_1 : \mathcal{D}_{\text{hit}} \neq \mathcal{D}_{\text{miss}}.$$

Given this reformulation, to perform our audit, we first sample response times using the cache hit and cache miss procedures. Then, we run a statistical test for whether our samples came from the same distribution, e.g., the two-sample Kolmogorov-Smirnov test, producing a p-value with respect to the null hypothesis of no caching.

3.2. Audit Procedure Details

Next, we describe our audit procedure in more detail.

Parameters. The procedure uses the following configuration parameters: PROMPTLENGTH, RANDOMSUFFIXLENGTH, NUMVICTIMREQUESTS, and NUMSAMPLES. We explain the meanings of these parameters in the procedure descriptions below.

Cache miss. To produce a cache miss, we simply generate a random prompt of PROMPTLENGTH tokens and send it to the API. We set the temperature to 1 and maximum output tokens to 1, so we can measure the time to first token by simply measuring the total response time. The prompt consists of PROMPTLENGTH random letters from the English alphabet, lowercase and uppercase, each separated by space characters, e.g., "`m x N j R`". Because all commonly used byte pair encoding (BPE) tokenizers (Gage, 1994; Sennrich et al., 2016) split on whitespace during pre-tokenization, this results in a prompt that is exactly PROMPTLENGTH tokens.[2] Since the prompt consists of random letters, there is a negligible probability that a noticeable prefix has already been cached, so this procedure accurately measures a baseline of cache miss times.

Cache hit. First, we generate a random prompt x with PROMPTLENGTH tokens using the same procedure as above. Then, from the "victim" user, we send x to the API NUMVICTIMREQUESTS times consecutively, with the goal of placing x into the prompt cache. We do not need to measure the time to first token for the victim requests, so we set the maximum output tokens to 100 (arbitrarily chosen) and temperature to 1.

Then, we construct a modified prompt x' by replacing the last RANDOMSUFFIXLENGTH tokens of x with the same number of random tokens. We generate this random suffix using the same method as for random prompt generation. To ensure that x and x' have a shared prefix length of exactly PROMPTLENGTH $-$ RANDOMSUFFIXLENGTH tokens, we ensure that the first token of the random suffix in x' differs from the token at that position in the original x. When RANDOMSUFFIXLENGTH > 0, the purpose of the random suffix is to test for cache hits between prompts with the same prefix but different suffixes. Such cache hits are possible only under attention KV cache reuse or a similar mechanism, and not simply caching the final LLM outputs.

To attempt to produce a cache hit, from the "attacker" user, we send x' to the API and measure the time to first token, as in the cache miss procedure. The choice of the victim and attacker users determines the level of cache sharing being audited. For example, the victim and attacker can be set to the same user to determine if there is prompt caching at all, whereas the victim and attacker can be set to users in different organizations to test for cross-organization caching. We will discuss the choice of victim and attacker more thoroughly in the experiment setup.

Statistical testing. Putting these pieces together, to perform the audit, we collect NUMSAMPLES timings each from the cache hit and cache miss procedures. We randomize the order in which we collect the timings. Then, we test for a statistically significant difference between the distributions of times from the two procedures. We use the SciPy implementation (Virtanen et al., 2020) of the two-sample Kolmogorov-Smirnov (KS) test (Hodges Jr, 1958), which is a nonparametric test for equality of distributions. The test statistic is the maximum difference between the empirical cumulative distribution functions at any point. More

[2] Many APIs add a small number of tokens to the user prompt due to the default system prompt, special tokens for prompt and role formatting, etc. However, these additional tokens are unimportant for our procedure, as the number of additional tokens is small and remains constant across prompts to a given model API.

Table 1. Results of auditing APIs for prompt caching, conducted in September and October 2024. APIs are grouped by the level of cache sharing detected and order alphabetically within each group. ✓ denotes caching was detected, ✗ denotes caching was not detected, a blank space denotes that the test was not conducted because caching was not detected in an earlier stage, and — denotes that cache sharing within an organization was not tested, either because the API did not support organizations or because we did not have access to the organizations feature. For APIs where we detected caching sharing either within or across organizations, we report the average precision for classifying times from the cache hit procedure. We report the average precision for client-side timing and server-side timing separately, with — denoting that the given timing method is unavailable for that API.

		Same prompt	Same prefix, different suffix			Avg. precision	
Provider	Model	Same user	Same user	Within org.	Cross org.	Client	Server
Azure	text-embedding-3-small	✓	✓	—	✓	0.80	—
Deep Infra	Llama 3.1 8B Instruct	✓	✓	—	✓	0.84	—
Fireworks	Llama 3.1 8B Instruct	✓	✓	✓	✓	0.77	0.79
Lepton	Llama 3.1 8B Instruct	✓	✓	—	✓	0.71	0.70
OpenAI	text-embedding-3-small	✓	✓	✓	✓	0.78	0.79
Perplexity	Llama 3.1 8B Instruct	✓	✓	—	✓	0.90	—
Replicate	Llama 3 8B Instruct	✓	✓	—	✓	—	1.00
Anthropic	Claude 3 Haiku	✓	✓	✓	✗	0.84	—
OpenAI	GPT-4o mini	✓	✓	✓	✗	0.79	0.86
Amazon	Claude 3 Haiku	✗					
Azure	GPT-4o mini	✗					
Cohere	Command R	✗					
Cohere	embed-english-v3.0	✗					
DeepSeek	DeepSeek Chat	✗					
Google	Gemini 1.5 Flash	✗					
Google	text-embedding-004	✗					
Groq	Llama 3 8B Instruct	✗					
Hyperbolic	Llama 3.1 8B Instruct	✗					
Mistral	Mistral Nemo	✗					
Mistral	Mistral Embed	✗					
OctoAI	Llama 3.1 8B Instruct	✗					
Together	Llama 3.1 8B Instruct	✗					

specifically, since we expect cache hits to be faster under the alternative, we perform a one-sided test, so the test statistic is the maximum difference in the direction of cache hits being faster. The KS test outputs a p-value, which we can use to reject or not reject the null hypothesis of no prompt caching at a given significance level α.

4. Privacy Leakage in Real-World APIs

Next, we audit real-world LLM APIs to identify APIs that cache prompts and determine the level of cache sharing across users and organizations. Cache sharing results in a privacy leakage, as an attacker could then detect hits from timing data to learn information about other users' prompts.

4.1. Audit Setup and Parameters

API providers and models. We audit 17 API providers: Anthropic, Amazon Bedrock, Microsoft Azure OpenAI, Cohere, Deep Infra, DeepSeek, Fireworks AI, Google, Groq, Hyperbolic, Lepton AI, Mistral, OctoAI, OpenAI, Perplexity, Replicate, and Together AI. The model APIs that we audit for each provider are included in Table 1. For API providers that primarily serve open-weights models, we audit their Llama 3 or 3.1 8B Instruct API (Dubey et al., 2024). For providers that serve proprietary models, we audit the cheapest chat model in their most recent family of models. In addition, we audit APIs for proprietary embedding models, where available. We do not audit APIs for open-weights embedding models because all the ones we found use an encoder-only Transformer architecture, making prefix caching impossible, as each token attends to all other tokens in the prompt.

Parameters and procedure. For our audits, we use $\textsc{PromptLength} = 5000$ and $\textsc{NumSamples} = 250$. We run four stages of audits of increasing privacy leakage. At each stage, we only continue to audit APIs if we detect caching during the previous stage. We use a significance

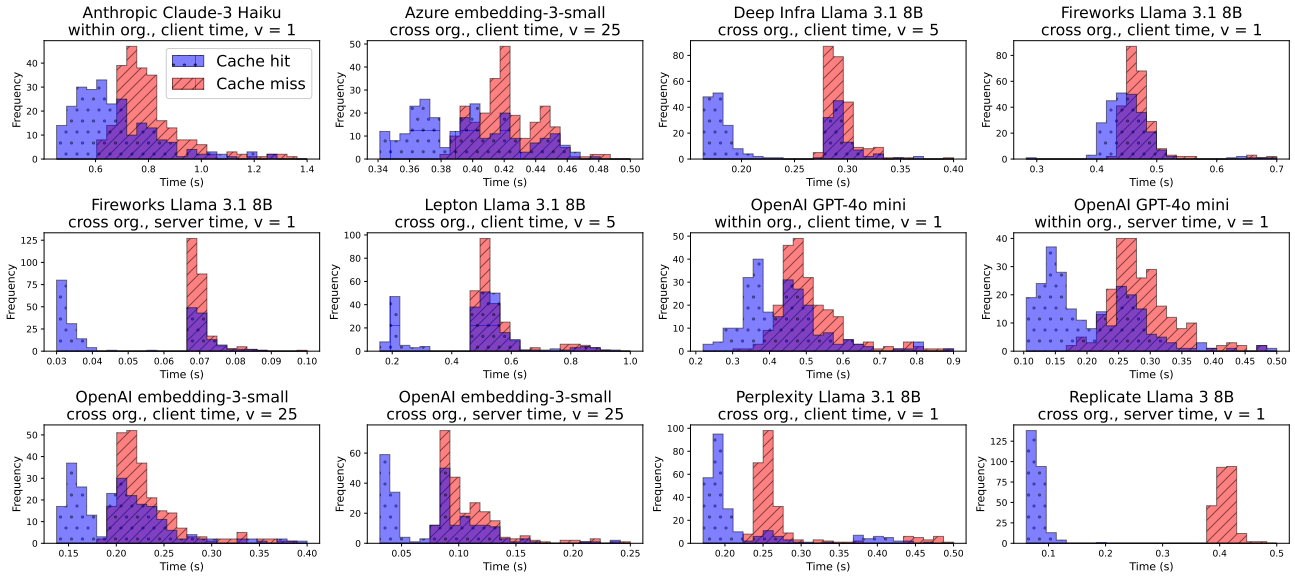

Figure 1. Histograms of response times from the cache hit and cache miss procedures in APIs where we detected caching. The distributions of times are clearly distinguishable, with cache hits tending to be faster. Each histogram title states the API provider, model, level of cache sharing (within org. or cross org.), timing source (client-side or server-side timing), and the NUMVICTIMREQUESTS used, denoted V.

level of $\alpha = 10^{-8}$.

To narrow down our list of providers, the first stage tests for the simplest level of prompt caching:

1. **Same prompt, same user.** We test for exact prompt matches by setting RANDOMSUFFIXLENGTH $= 0$. We set the victim and attacker to be the same user, and we set NUMVICTIMREQUESTS $= 25$.

In the remaining three stages, we test for cache hits between prompts that have the **same prefix but different suffix** by setting RANDOMSUFFIXLENGTH $= 250$. In each stage, we test for increasing levels of cache sharing by appropriately setting the victim and attacker:

2. **Same user.** The victim and attacker are the same user, as in the first stage.
3. **Within organization.** The victim and attacker are different users within the same organization. For APIs without organizations, we skip this stage.
4. **Cross organization.** The victim and attacker are different users in different organizations. For APIs without organizations, the victim and attacker are simply different users.

In stages 2–4, to determine how many victim requests are needed to detect caching, we run tests using NUMVICTIMREQUESTS $\in \{1, 5, 25\}$ in increasing order, stopping after the first significant p-value. To account for multiple testing, we perform a Bonferroni correction by dividing the significance threshold for each individual test by three. In addition, for APIs where both client-side and server-side timing is available, we run tests using both timing methods and perform another Bonferroni correction, dividing by two this time. For other APIs, we use whichever timing method is available.

Cost per test. When NUMVICTIMREQUESTS $= 25$, one test uses roughly 34 million prompt tokens. The number of response tokens used is much smaller because we set the maximum response tokens parameter. For the chat APIs we audit, the prices per million prompt tokens are 0.05–0.25 USD, resulting in a cost per test of 1.69–8.44 USD. The tests are cheaper when NUMVICTIMREQUESTS is smaller.

4.2. Audit Results

We conducted our audits in September and early October 2024 using clients located in California. The audit results are shown in Table 1. We detected cache sharing across organizations in seven API providers. This means that an attacker can potentially learn information about other users' prompts by detecting cache hits from timing data. To quantify an attacker's ability to detect if another user sent a prompt that shares a prefix with a given prompt, we compute the average precision (Zhu, 2004) for classifying times from the cache hit procedure.[3] Average precision is equal

[3]The cache hit procedure attempts to produce cache hits but cannot guarantee cache hits (e.g., due to server routing), so some times in the cache hit distribution may actually be cache misses.

to the area under the precision-recall curve (the precision is averaged over the recall scores). As shown in Table 1, the average precisions are mostly around a moderately high value of 0.8. Figure 2 contains selected precision-recall curves, showing that an attacker can achieve near perfect precision up to moderate recall scores. Figure 4 in the appendix shows the full set of relevant precision-recall curves.

Figure 1 displays histograms of times from the cache hit and cache miss procedures. The distributions of times are clearly distinguishable, with cache hits tending to be faster. Each histogram title states the minimum NUMVICTIMREQUESTS (denoted V in the titles) that resulted in a significant p-value. In most of the APIs where we detected caching, only NUMVICTIMREQUESTS $= 1$ was needed to detect caching. Only the OpenAI and Azure text-embedding-3-small APIs required NUMVICTIMREQUESTS $= 25$ to achieve a significant p-value. In Appendix B, we report all the p-values from our audits. In many APIs, the p-values are many orders of magnitude smaller than our significance level of $\alpha = 10^{-8}$. In all APIs where we detected caching, both client-side and server-side timing (if available) resulted in significant p-values.

In the Anthropic Claude 3 Haiku and OpenAI GPT-4o mini APIs, we detected cache sharing within organizations, but not across organizations. This exact level of cache sharing is stated in their prompt caching documentations, confirming the efficacy of our audit procedure. Since OpenAI and Anthropic officially document cache sharing within organizations, we do not consider it a security vulnerability. Cache sharing across organizations in the OpenAI text-embedding-3-small API was a potential vulnerability, but has been patched following our responsible disclosure prior to the release of this paper.

Although DeepSeek has a prompt caching feature and returns the number of cache hit tokens in API responses, which we used to confirm that we produced cache hits, we were unable to detect caching from response times. There was no statistically significant difference between the distributions of cache hit and cache miss times, even in two-sided tests. DeepSeek states that the cache is not shared across users, and we ran tests which confirmed that this is the case.

4.3. Ablations

We run ablations to determine the effects of PROMPTLENGTH, RANDOMSUFFIXLENGTH, and model size on the average precision, shown in Figure 3.

We use the APIs in which we detected cross-organization caching with NUMVICTIMREQUESTS $= 1$, i.e., the Llama 3/3.1 8B Instruct APIs of Fireworks, Perplexity, and Replicate. In Figures 3a and 3b, we vary the PROMPTLENGTH in the same prompt and same prefix but different suffix settings,

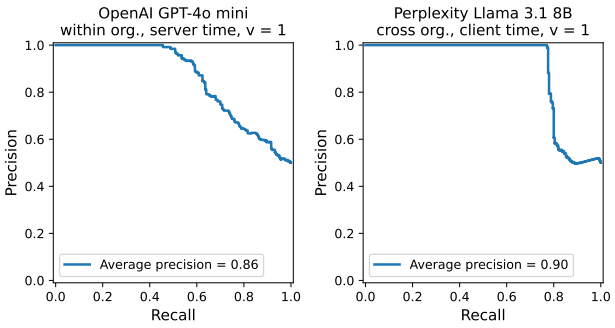

Figure 2. Selected precision-recall curves for distinguishing between times from the cache hit and cache miss procedures. Cache hits are the positive class. The curves show that cache hits can be detected with near perfect precision up to moderate recall scores. Figure 4 in the appendix contains curves for other APIs.

respectively. When the PROMPTLENGTH is moderately high ($\gtrapprox 1000$), the average precision is relatively high and stable. However, as the PROMPTLENGTH approaches zero, the average precision decreases to random chance. In Figure 3c, we vary the RANDOMSUFFIXLENGTH while holding the PROMPTLENGTH constant. As the length of the matching prefix (PROMPTLENGTH $-$ RANDOMSUFFIXLENGTH) decreases, the average precision again decreases to random chance. In Figure 3d, we vary the model size on the Fireworks API, which supports all models in the Llama 3.1 and 3.2 families. We detected caching in all model sizes, with no clear relationship between model size and average precision.

4.4. Prompt Extraction Attacks?

Given that cache hits can be detected via timing, one natural idea is to extract other users' prompts token by token using a breadth-first search style approach. At each step, the goal is to determine which continuation tokens are cached and thus part of other users' prompts.

However, we were unable to perform practical prompt extraction attacks for multiple reasons. The branching factor of the search is very large, with potentially thousands of possible continuation tokens at each step. Therefore, a successful attack requires extremely accurate detection of cached tokens. Just one incorrect token causes complete failure due to the exact prefix match required for a cache hit. In preliminary experiments, we were unable to reliably detect the presence of one additional cached token. Increasing the number of tokens guessed at each step may increase accuracy, but exponentially increases the branching factor.

Another idea is to make repeated measurements to boost accuracy. However, this approach faces several difficulties. To detect whether a prompt is cached, the attacker must send the prompt to the API. Then, future measurements may detect a cache hit not because another user sent the prompt, but because the attacker themself sent it. At a minimum,

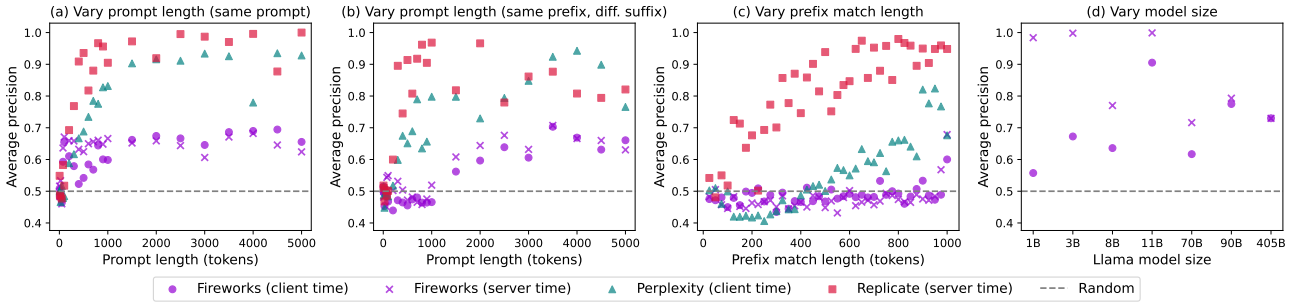

Figure 3. Ablations on the effects of PROMPTLENGTH, RANDOMSUFFIXLENGTH, and model size on the average precision. In (a), we vary PROMPTLENGTH in the range 5–5000, using RANDOMSUFFIXLENGTH $= 0$. (b) is the same as (a), except setting RANDOMSUFFIXLENGTH equal to 5% of each PROMPTLENGTH value. In (c), we set PROMPTLENGTH $= 1000$ and vary RANDOMSUFFIXLENGTH in the range 0–975. We plot the length of the matching prefix, given by PROMPTLENGTH $-$ RANDOMSUFFIXLENGTH. In (a) and (b), when the PROMPTLENGTH is moderately high ($\gtrapprox 1000$), the average precision is relatively high and stable. However, in (a)–(c), as the prompt or prefix match length approaches zero, the average precision decreases to random chance. In (d), we test various model sizes in the Llama 3.1 and 3.2 families, from 1B to 405B, using PROMPTLENGTH $= 2000$ and RANDOMSUFFIXLENGTH $= 100$. We detected caching across all model sizes, with no clear relationship between model size and average precision.

for valid repeated measurements, the victim must have sent the prompt between each measurement. Consequently, this approach relies on the victim repeatedly sending a prompt many times. Lastly, some APIs enable cache hits only in certain conditions. For example, OpenAI enables cache hits only in multiples of 128 tokens that exceed 1024 tokens, making prompt extraction attacks all but impossible.

Note that we do not claim that prompt extraction attacks are necessarily impossible. Such attacks face difficulties, but future work may yet develop successful, practical attacks. In addition, in more restricted sets of target prompts, e.g., known prompt templates with places for users to enter private personal information, it may be easier to overcome these difficulties.

5. Extracting Architecture Information

In decoder-only Transformer models, reuse of the attention KV cache enables cache hits between prompts with matching prefixes, even if the suffixes differ, since each token attends only to previous tokens. Such prefix caching is not possible in encoder-only or encoder-decoder Transformer models, where each token in the prompt attends to all other tokens in the prompt. Therefore, detecting cache hits between prompts with the same prefix but different suffixes indicates that the model cannot have a bidirectional encoder architecture. Virtually all current Transformer-based chat models are decoder-only, but top open-weights embedding models include both encoder-only and decoder-only models. As such, extracting architectural information can be interesting for proprietary embedding models.

In our audits (Table 1), we detected prompt caching in OpenAI's text-embedding-3-small API when prompts had the same prefix but different suffixes. Assuming that text-embedding-3-small is Transformer-based, this indicates that text-embedding-3-small is a decoder-only Transformer. This is new information, as OpenAI has not released any information about the architecture of their embedding models.

We confirm that when the prompt suffix is changed, the returned embedding also changes, indicating that the caching mechanism involves the attention KV cache, instead of simply caching embedding outputs. In addition, when we send the exact same prompt multiple times, when the response time is noticeably faster, indicating a cache hit, the returned embedding differs slightly from the "normal" embedding in most other responses. This behavior is consistent across different random prompts. These differences are small, on the order of 10^{-5} in each coordinate. We hypothesize that these differences may arise if the reused KV cache is stored in a lower precision, resulting in slight discrepancies when the attention KV is computed from scratch in cache misses versus when it is retrieved from the cache in cache hits. Interestingly, when the response time is noticeably slower, the returned embedding also differs slightly from both the "normal" and cache hit embeddings. We do not have an explanation for this behavior.

6. Mitigations

To prevent an API from leaking information about users' prompts, we recommend disabling cache sharing across organizations. This way, an attacker will not be able to produce cache hits on prompts sent by users in different organizations. Since it is unlikely that different users will send prompts with the exact same prefix, this mitigation should retain most of the performance improvements as global prompt caching.

We believe that cache sharing within organizations is reason-

able, as long as users are made aware of it. It is important that users know how their data is handled and who could potentially learn information about their data. This way, users can make informed decisions about how they use an LLM API and what information they are willing to send. For example, if a company knows that an API shares the prompt cache between different users within an organization, the company may decide to create separate organizations for different groups of employees to reduce the risk of unauthorized information access, e.g., a software engineer should not be able to access sensitive legal documents. Overall, we strongly encourage API providers to be transparent about caching policies and how private user data is handled.

For information leakage that only requires caching within the same user, such as extracting architecture information, we believe that the only full mitigation is to disable prompt caching. Note that leakage about model architecture is not a privacy concern, so it does not necessarily need to be mitigated, although higher-severity attacks may be discovered in the future. Another option is to intentionally delay the response time for cache hits so that they look like cache misses. This eliminates the benefits of prompt caching for users, but API providers could still benefit, as cached prompts require less GPU processing time. A partial mitigation to reduce information leakage is to avoid providing server-side timing data to users. Although we found that both client-side and server-side timing can be used to detect cache hits, server-side timing provides stronger signal, since it do not include noise from network latency.

7. Related Work

Prompt caching. Many recent works have developed optimizations for inference and serving of Transformer language models. In particular, various methods involve reuse of the attention KV cache, improving latency and throughput for shared prompt prefixes (Kwon et al., 2023; Zheng et al., 2024; Gim et al., 2024; Ye et al., 2024a;b; Qin et al., 2024; Juravsky et al., 2024). Recall that we do not assume any particular implementation of prompt caching in our attacks. Indeed, we do not know technical details about the caching mechanisms used by the APIs we audited. Other caching methods do not preserve exact model behavior, such as retrieving cached responses for semantically similar prompts (Bang, 2023) or reusing the KV cache even when the prefixes do not exactly match (Gim et al., 2024; Yao et al., 2024; Hu et al., 2024). We do not study such methods, but they are also likely susceptible to similar cache timing attacks, and our audit procedure can easily be adapted to detect other types of caching.

Cache timing attacks. In computer security, many side-channel timing attacks have extracted information by using timing differences to distinguish between cache hits and cache misses, e.g., in the CPU cache or web cache. For example, cache timing attacks have been used to extract AES keys (Bernstein, 2005; Osvik et al., 2006; Bonneau & Mironov, 2006; Tromer et al., 2010; Gullasch et al., 2011; Yarom et al., 2017), a user's private web information (Felten & Schneider, 2000; Bortz & Boneh, 2007; Van Goethem et al., 2015), and sensitive data from other processes on a machine (Percival, 2005; Yarom & Falkner, 2014; Liu et al., 2015), as in the well-known Meltdown (Lipp et al., 2018) and Spectre attacks (Kocher et al., 2019).

Attacks on language model APIs. Several recent works have attacked language model APIs. Carlini et al. (2024) and Finlayson et al. (2024) show that logits and logprobs leak information from an LLM API, including the model's hidden dimension size and final layer weights. Weiss et al. (2024) partially extract encrypted and streamed LLM responses by inferring and analyzing token lengths from packet sizes. Carlini & Nasr (2024) and Wei et al. (2024) exploit speculative decoding (Leviathan et al., 2023; Chen et al., 2023) and similar methods to extract LLM responses with higher success by measuring delays between packets.

Most related to our work is the concurrent work of Song et al. (2024), which also studies timing attacks and privacy leakages arising from prompt caching, including both KV cache reuse and semantic caching. Our work differs in developing an audit procedure that is practical and provides statistical guarantees, using these audits to precisely identify different levels of privacy leakage, and extracting information about model architecture. Song et al. (2024) demonstrate prompt extraction attacks in a simulated setting, but the attack is run locally without network latency, uses knowledge of the distribution of prompts, requires explicit clearing of the cache to make repeated measurements, and makes an average of 234 measurements for each extracted token. As we discussed earlier, we believe that these simulated attacks are currently unlikely to be real-world privacy threats.

8. Conclusion

As LLMs and other machine learning systems become more widely deployed and used in the real world, it is increasingly important to consider security and privacy aspects of these systems. Towards that end, in this paper, we show that prompt caching in LLM APIs leaks private and proprietary information through timing differences. We perform rigorous statistical audits on real-world APIs, finding that multiple APIs leak information about other users' prompts.

Cache timing attacks have been extensively studied in the computer security literature, and we simply apply them to the new domain of prompt caching in LLM APIs. More broadly, we believe that leveraging established security re-

search will be helpful in developing secure LLM systems. We hope that future work will continue to evaluate and audit the security and privacy of machine learning systems, ensuring their robustness and trustworthiness.

Acknowledgments

CG was supported by the Stanford CURIS program. XL was supported by a Two Sigma PhD Fellowship. PL was supported by NSF Award Grant no. 1805310 and an Open Philanthropy Project Award. TH was supported by gifts from Open Philanthropy, Amazon, Google, Meta, and a grant under the NSF CAREER IIS-2338866.

Conflicts of Interest

PL is a co-founder of Together AI. However, this work was done in his Stanford capacity. The methods, providers audited, and results were not influenced by or shared with Together prior to the public release of this paper. All API providers were audited using the same procedure, including Together. None of the other authors have conflicts of interest with the providers audited in this paper.

References

Bang, F. GPTCache: An open-source semantic cache for LLM applications enabling faster answers and cost savings. In Tan, L., Milajevs, D., Chauhan, G., Gwinnup, J., and Rippeth, E. (eds.), *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*, pp. 212–218, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.nlposs-1.24. URL https://aclanthology.org/2023.nlposs-1.24.

Bernstein, D. J. Cache-timing attacks on aes. 2005. URL https://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

Bonneau, J. and Mironov, I. Cache-collision timing attacks against aes. In *Cryptographic Hardware and Embedded Systems-CHES 2006: 8th International Workshop, Yokohama, Japan, October 10-13, 2006. Proceedings 8*, pp. 201–215. Springer, 2006.

Bortz, A. and Boneh, D. Exposing private information by timing web applications. In *Proceedings of the 16th international conference on World Wide Web*, pp. 621–628, 2007.

Carlini, N. and Nasr, M. Remote timing attacks on efficient language model inference. *arXiv preprint arXiv:2410.17175*, 2024.

Carlini, N., Paleka, D., Dvijotham, K. D., Steinke, T., Hayase, J., Cooper, A. F., Lee, K., Jagielski, M., Nasr, M., Conmy, A., Wallace, E., Rolnick, D., and Tramèr, F. Stealing part of a production language model. In Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F. (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5680–5705. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/carlini24a.html.

Chen, C., Borgeaud, S., Irving, G., Lespiau, J.-B., Sifre, L., and Jumper, J. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Felten, E. W. and Schneider, M. A. Timing attacks on web privacy. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, pp. 25–32, 2000.

Finlayson, M., Ren, X., and Swayamdipta, S. Logits of API-protected LLMs leak proprietary information. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=oRcYFm8vyB.

Gage, P. A new algorithm for data compression. *The C Users Journal*, 12(2):23–38, 1994.

Gim, I., Chen, G., Lee, S.-s., Sarda, N., Khandelwal, A., and Zhong, L. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338, 2024.

Gullasch, D., Bangerter, E., and Krenn, S. Cache games–bringing access-based cache attacks on aes to practice. In *2011 IEEE Symposium on Security and Privacy*, pp. 490–505. IEEE, 2011.

Hodges Jr, J. The significance probability of the smirnov two-sample test. *Arkiv för matematik*, 3(5):469–486, 1958.

Hu, J., Huang, W., Wang, H., Wang, W., Hu, T., Zhang, Q., Feng, H., Chen, X., Shan, Y., and Xie, T. Epic: Efficient position-independent context caching for serving large language models. *arXiv preprint arXiv:2410.15332*, 2024.

Juravsky, J., Brown, B., Ehrlich, R., Fu, D. Y., Ré, C., and Mirhoseini, A. Hydragen: High-throughput llm inference with shared prefixes. *arXiv preprint arXiv:2402.05099*, 2024.

Kocher, P., Horn, J., Fogh, A., , Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., and Yarom, Y. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Leviathan, Y., Kalman, M., and Matias, Y. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Fogh, A., Horn, J., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., and Hamburg, M. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

Liu, F., Yarom, Y., Ge, Q., Heiser, G., and Lee, R. B. Last-level cache side-channel attacks are practical. In *2015 IEEE symposium on security and privacy*, pp. 605–622. IEEE, 2015.

Osvik, D. A., Shamir, A., and Tromer, E. Cache attacks and countermeasures: the case of aes. In *Topics in Cryptology–CT-RSA 2006: The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2005. Proceedings*, pp. 1–20. Springer, 2006.

Percival, C. Cache missing for fun and profit. BSDCan Ottawa, 2005. URL https://www.daemonology.net/papers/htt.pdf.

Qin, R., Li, Z., He, W., Zhang, M., Wu, Y., Zheng, W., and Xu, X. Mooncake: A kvcache-centric disaggregated architecture for llm serving. *arXiv preprint arXiv:2407.00079*, 2024.

Sennrich, R., Haddow, B., and Birch, A. Neural machine translation of rare words with subword units. In Erk, K. and Smith, N. A. (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://aclanthology.org/P16-1162.

Song, L., Pang, Z., Wang, W., Wang, Z., Wang, X., Chen, H., Song, W., Jin, Y., Meng, D., and Hou, R. The early bird catches the leak: Unveiling timing side channels in llm serving systems. *arXiv preprint arXiv:2409.20002*, 2024.

Tromer, E., Osvik, D. A., and Shamir, A. Efficient cache attacks on aes, and countermeasures. *Journal of Cryptology*, 23:37–71, 2010.

Van Goethem, T., Joosen, W., and Nikiforakis, N. The clock is still ticking: Timing attacks in the modern web. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1382–1393, 2015.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Wei, J., Abdulrazzag, A., Zhang, T., Muursepp, A., and Saileshwar, G. Privacy risks of speculative decoding in large language models. *arXiv preprint arXiv:2411.01076*, 2024.

Weiss, R., Ayzenshteyn, D., and Mirsky, Y. What was your prompt? a remote keylogging attack on AI assistants. In *33rd USENIX Security Symposium (USENIX Security 24)*, pp. 3367–3384, Philadelphia, PA, August 2024. USENIX Association. ISBN 978-1-939133-44-1. URL https://www.usenix.org/conference/usenixsecurity24/presentation/weiss.

Yao, J., Li, H., Liu, Y., Ray, S., Cheng, Y., Zhang, Q., Du, K., Lu, S., and Jiang, J. Cacheblend: Fast large language model serving with cached knowledge fusion. *arXiv preprint arXiv:2405.16444*, 2024.

Yarom, Y. and Falkner, K. {FLUSH+ RELOAD}: A high resolution, low noise, l3 cache {Side-Channel} attack. In *23rd USENIX security symposium (USENIX security 14)*, pp. 719–732, 2014.

Yarom, Y., Genkin, D., and Heninger, N. Cachebleed: a timing attack on openssl constant-time rsa. *Journal of Cryptographic Engineering*, 7:99–112, 2017.

Ye, L., Tao, Z., Huang, Y., and Li, Y. ChunkAttention: Efficient self-attention with prefix-aware KV cache and two-phase partition. In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11608–11620, Bangkok, Thailand, August 2024a. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.623. URL https://aclanthology.org/2024.acl-long.623.

Ye, Z., Lai, R., Lu, B.-R., Lin, C.-Y., Zheng, S., Chen, L., Chen, T., and Ceze, L. Cascade inference: Memory bandwidth efficient shared prefix batch decoding, February 2024b. URL https://flashinfer.ai/2024/02/02/cascade-inference.html.

Zheng, L., Yin, L., Xie, Z., Sun, C., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., Barrett, C., and Sheng, Y. SGLang: Efficient execution of structured language model programs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=VqkAKQibpq.

Zhu, M. Recall, precision and average precision. *Department of Statistics and Actuarial Science, University of Waterloo, Waterloo*, 2(30):6, 2004.

A. Precision-Recall Curves

Figure 4 shows precision-recall curves for distinguishing between cache hit and cache miss times in APIs where we detected caching in our audits (Table 1).

B. P-values from Audits

We report all the p-values from our audits on APIs. Table 2 contains p-values from stage 1 of our audits: same prompt, same user. Table 3 contains p-values from stage 2 of our audits: same prefix but different suffix, same user. Table 4 contains p-values from stage 3 of our audits: same prefix but different suffix, within organization. Table 5 contains p-values from stage 4 of our audits: same prefix but different suffix, cross organization.

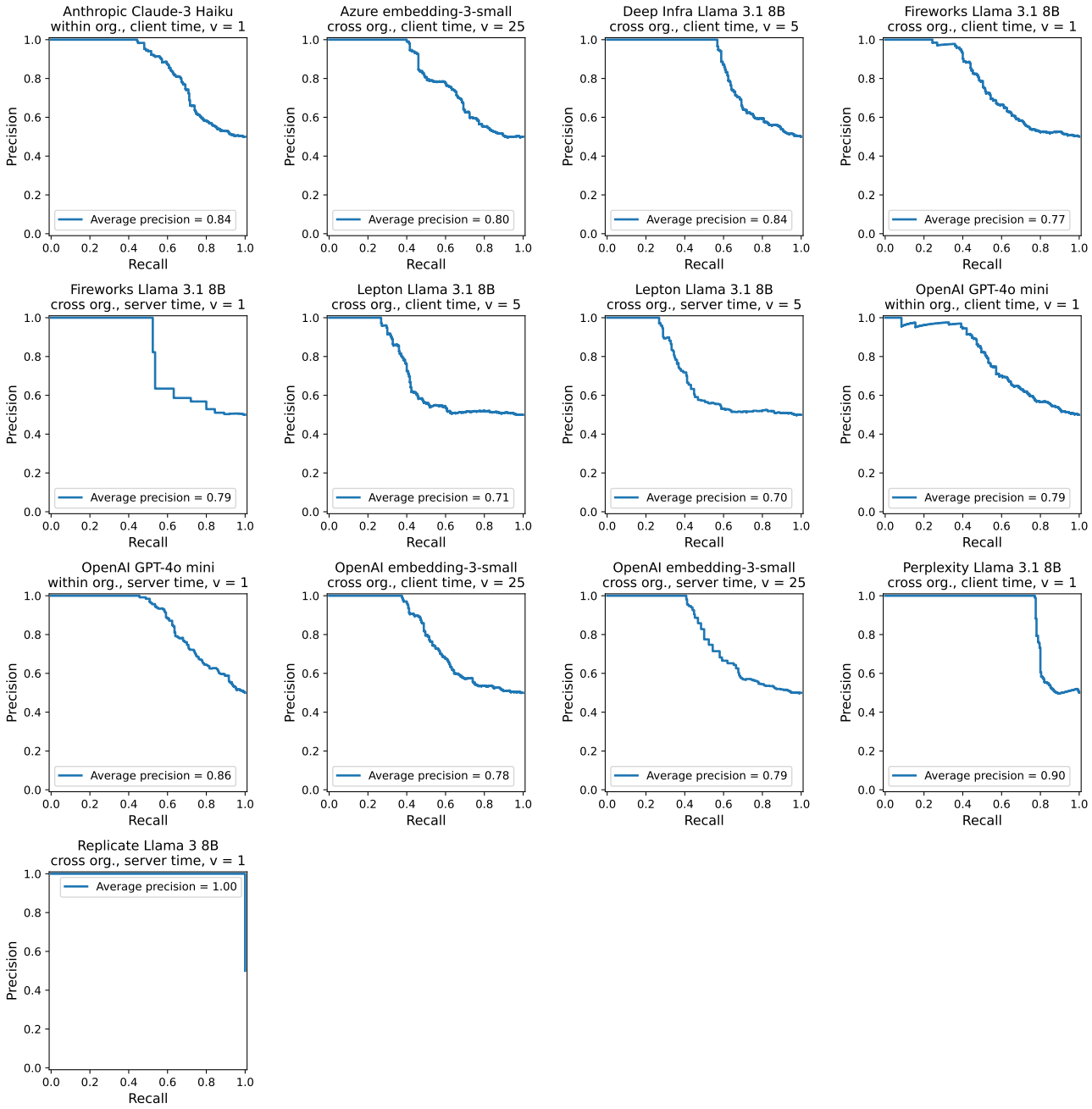

Figure 4. Precision-recall curves for distinguishing between times produced by the cache hit and cache miss procedures in APIs where we detected caching in our audits (Table 1). Cache hits are the positive class, and cache misses are the negative class. The curves show that cache hits can be detected with near perfect precision up to moderate recall scores. Note that our cache hit procedure attempts to produce cache hits but cannot guarantee cache hits (e.g., due to server routing), so some times in the cache hit distribution may actually be cache misses, which would hurt recall scores.

Table 2. P-values from stage 1 of our audits: same prompt, same user. Each column shows one combination of NUMVICTIMREQUESTS and timing source (client-side or server-side timing). Green indicates a significant p-value, after performing the appropriate Bonferroni corrections. Red indicates a p-value that is not significant. — indicates that the given timing source was not available for the API. APIs are grouped by whether caching was detected in this stage and sorted alphabetically within the groups.

		NUMVICTIMREQUESTS 25	
Provider	Model	Client	Server
Anthropic	Claude 3 Haiku	7.8e-21	—
Azure	text-embedding-3-small	1.7e-42	—
Deep Infra	Llama 3.1 8B Instruct	9.5e-116	—
Fireworks	Llama 3.1 8B Instruct	2.0e-80	4.7e-109
Lepton	Llama 3.1 8B Instruct	2.2e-138	2.2e-138
OpenAI	GPT-4o mini	2.4e-66	2.9e-105
OpenAI	text-embedding-3-small	7.6e-09	2.3e-10
Perplexity	Llama 3.1 8B Instruct	1.9e-90	—
Replicate	Llama 3 8B Instruct	—	2.2e-140
Amazon	Claude 3 Haiku	0.27	0.51
Azure	GPT-4o mini	0.95	—
Cohere	Command R	0.62	0.72
Cohere	embed-english-v3.0	0.41	0.56
DeepSeek	DeepSeek Chat	0.75	—
Google	Gemini 1.5 Flash	0.17	0.20
Google	text-embedding-004	0.20	0.24
Groq	Llama 3 8B Instruct	0.41	0.51
Hyperbolic	Llama 3.1 8B Instruct	0.72	—
Mistral	Mistral Nemo	0.56	0.96
Mistral	Mistral Embed	0.67	0.91
OctoAI	Llama 3.1 8B Instruct	0.32	0.27
Together	Llama 3.1 8B Instruct	0.51	0.96

Table 3. P-values from stage 2 of our audits: same prefix but different suffix, same user. Each column shows one combination of NUMVICTIMREQUESTS and timing source (client-side or server-side timing). Green indicates a significant p-value, after performing the appropriate Bonferroni corrections. Red indicates a p-value that is not significant. — indicates that the given timing source was not available for the API. A blank cell indicates that the given value of NUMVICTIMREQUESTS was not tested because caching was detected in the API using a smaller value of NUMVICTIMREQUESTS. Caching was detected in all APIs audited in this stage. APIs are sorted alphabetically.

		NUMVICTIMREQUESTS 1		5		25	
Provider	Model	Client	Server	Client	Server	Client	Server
Anthropic	Claude 3 Haiku	9.6e-37	—				
Azure	text-embedding-3-small	0.20	—	6.0e-04	—	6.9e-42	—
Deep Infra	Llama 3.1 8B Instruct	0.03	—	5.0e-22	—		
Fireworks	Llama 3.1 8B Instruct	4.3e-15	5.0e-33				
Lepton	Llama 3.1 8B Instruct	1.00	0.96	7.7e-10	7.7e-10		
OpenAI	GPT-4o mini	9.5e-27	1.5e-39				
OpenAI	text-embedding-3-small	0.03	0.03	0.10	0.17	2.6e-12	4.3e-15
Perplexity	Llama 3.1 8B Instruct	5.4e-68	—				
Replicate	Llama 3 8B Instruct	—	8.6e-150				

Table 4. P-values from stage 3 of our audits: same prefix but different suffix, within organization. Each column shows one combination of NUMVICTIMREQUESTS and timing source (client-side or server-side timing). Green indicates a significant p-value, after performing the appropriate Bonferroni corrections. Red indicates a p-value that is not significant. — indicates that the given timing source was not available for the API. A blank cell indicates that the given value of NUMVICTIMREQUESTS was not tested because caching was detected in the API using a smaller value of NUMVICTIMREQUESTS. Caching was detected in all APIs audited in this stage. APIs are sorted alphabetically.

		NUMVICTIMREQUESTS					
		1		5		25	
Provider	Model	Client	Server	Client	Server	Client	Server
Anthropic	Claude 3 Haiku	1.7e-31	—				
Fireworks	Llama 3.1 8B Instruct	1.3e-21	5.2e-32				
OpenAI	GPT-4o mini	1.1e-19	4.6e-34				
OpenAI	text-embedding-3-small	0.27	0.14	0.27	0.27	8.2e-14	8.2e-14

Table 5. P-values from stage 4 of our audits: same prefix but different suffix, cross organization. Each column shows one combination of NUMVICTIMREQUESTS and timing source (client-side or server-side timing). Green indicates a significant p-value, after performing the appropriate Bonferroni corrections. Red indicates a p-value that is not significant. — indicates that the given timing source was not available for the API. A blank cell indicates that the given value of NUMVICTIMREQUESTS was not tested because caching was detected in the API using a smaller value of NUMVICTIMREQUESTS. APIs are grouped by whether caching was detected in this stage and sorted alphabetically within the groups.

		NUMVICTIMREQUESTS					
		1		5		25	
Provider	Model	Client	Server	Client	Server	Client	Server
Azure	text-embedding-3-small	0.46	—	0.02	—	1.3e-21	—
Deep Infra	Llama 3.1 8B Instruct	6.5e-05	—	7.5e-38	—		
Fireworks	Llama 3.1 8B Instruct	9.0e-17	5.2e-32				
Lepton	Llama 3.1 8B Instruct	0.12	0.07	1.2e-10	1.4e-09		
OpenAI	text-embedding-3-small	0.41	0.36	0.20	0.08	1.1e-19	1.1e-19
Perplexity	Llama 3.1 8B Instruct	5.3e-74	—				
Replicate	Llama 3 8B Instruct	—	8.6e-150				
Anthropic	Claude 3 Haiku	0.24	—	0.77	—	0.87	—
OpenAI	GPT-4o mini	0.41	0.20	0.41	0.62	0.41	0.94