

Wiretapping LLMs: Network Side-Channel Attacks on Interactive LLM Services

Mahdi Soleimani
Yale University
mahdi.soleimani@yale.edu

Grace Jia
Yale University
grace.jia@yale.edu

In Gim
Yale University
in.gim@yale.edu

Seung-seob Lee
Yale University
seung-seob.lee@yale.edu

Anurag Khandelwal
Yale University
anurag.khandelwal@yale.edu

Abstract

Recent server-side optimizations like speculative decoding significantly enhance the interactivity and resource efficiency of Large Language Model (LLM) services. However, we show that these optimizations inadvertently introduce new side-channel vulnerabilities through network packet timing and size variations that tend to be input-dependent. Network adversaries can leverage these side channels to learn sensitive information contained in *encrypted* user prompts to and responses from public LLM services.

This paper formalizes the security implications using a novel indistinguishability framework and introduces a novel attack that establishes the insecurity of real-world LLM services with streaming APIs under our security framework.

Our proposed attack effectively deconstructs encrypted network packet traces to reveal the sizes of underlying LLM-generated tokens and whether the tokens were generated with or without certain server-side optimizations. Our attack can accurately predict private attributes in real-world privacy-sensitive LLM applications in medicine and finance with 71–92% accuracy on an open-source vLLM service and 50–90% accuracy on the commercial ChatGPT service. Finally, we show that solutions that hide these side channels to different degrees expose a tradeoff between security and performance — specifically, interactivity and network bandwidth overheads.

1 Introduction

Recent advances in Large Language Models (LLMs) [1–4] allow them to provide personalized responses to user prompts based on user-provided context and documents. Such context can often include privacy-sensitive data such as medical records [5, 6] and proprietary financial insights [7]. Consequently, many recent studies have focused on identifying and mitigating vulnerabilities that may allow adversaries with direct or indirect access to the LLM from learning such sensitive information under diverse threat models, including membership inference [8–10] and attribute inference [11, 12]. Our work instead focuses on previously unexplored *network side-channel* vulnerabilities of LLM serving systems.

Our key finding is that recent performance optimizations in LLM serving systems expose new network side-channel vulnerabilities. The driving factor for such optimizations is responsiveness — popular LLM services provide streaming APIs [13, 14] where the LLM responses are streamed back in small batches to allow users to see responses as the LLM generates them. These public serving systems also employ several optimizations [15–18] to improve the latency of these response streams further while also reducing the cost of operation for serving systems by as much as 5–10×. Unfortunately, these optimizations also have an unintended side effect: optimization-driven variations in streaming response times and batch sizes are inadvertently embedded within the communication traffic between users and the LLM service, even when encrypted. We demonstrate this effect on real-world public LLM services [13, 14] in §6.

We focus on speculative decoding optimizations [15, 16, 19] used by many modern LLM serving systems as a representative and pervasive example of such optimizations¹. They speed up response generation for semantically easier-to-predict words² using smaller but faster models while delegating harder-to-predict words to larger but slower ones (§2.1). When the responses are streamed to a user, either word-by-word or in small batches over encrypted packets, the inter-packet timings and packet sizes can be used to learn which words were generated with speculation and which were not. Even more concerning is our observation that this information can be tied to specific private information within the encrypted user prompts and LLM responses with high accuracy.

Our study is the first to formalize, deconstruct, and explore possible mitigation techniques for such network side-channel leakages. Our security model assumes a practical network adversary that only observes a timestamped sequence of *encrypted* packets between the target user-LLM service pair

¹Gemini uses speculative decoding to improve responsiveness across many products [20]. Several other public services also use speculative decoding [21], as unofficially confirmed by their developers [22].

²We refer the unit of LLM generated data here as ‘word’ for simplicity; §2 establishes more precise terminology.

and knows publicly available information about the LLM service and user prompts. This is a reasonably practical adversarial model since such information is readily available by eavesdropping on the encrypted TLS sessions [23–26]. We introduce a novel security definition to capture the security of LLM servers against such an adversary. Intuitively, the definition, named Indistinguishability under Chosen Prompt Attack (or IND-CPrA, §3), requires that the transcript of network communications between the LLM service and the user is independent of user prompts and LLM responses. Our definition adapts to a wide range of streaming models (§4) possible for LLM services, capturing the granularities at which they leak information (*e.g.*, per word, for batches of words, or the entire response) as well as the amount of information they leak (*e.g.*, sizes and timing of generated words, etc.).

We develop a novel attack that demonstrates existing public and private LLM services are indeed insecure under IND-CPrA (§5). The critical step in the attack is deconstructing the transcript of network communications — comprising packet timestamps and sizes — to identify which of the generated words in the LLM response leveraged optimizations and which ones did not. The attack leverages this information to (i) learn the mapping between optimization patterns and potentially sensitive elements within typical prompts to/responses from the LLM and (ii) use the mapping to predict user private information within actual user prompts/responses. Our evaluation (§6) studies the impact of the attack on both proprietary systems like ChatGPT [13] and open-source systems like vLLM [27], demonstrating prediction accuracy as high as 50–92%, compared to 1–8% accuracy of random guesses.

We also study the impact of different streaming models (§4) on attack accuracy. We show that the granularity and amount of leaked information expose the expected tradeoff between security and performance, *i.e.*, protocols that leak less information at coarser granularities reduce the attack’s prediction accuracy but also incur higher network overheads and worse responsiveness. We empirically characterize this tradeoff for real-world use cases and LLM deployment modes in §7 to help practitioners pick appropriate design points based on their security and performance needs. While this work focuses on network side channels stemming from speculative decoding, this is only a first step in characterizing such leakages due to the plethora of similar optimizations in LLM serving. To that end, we close the paper by outlining several avenues of future research (§9).

2 Background and Motivation

We provide a brief overview of interactive LLM serving systems in §2.1, followed by an overview of how network side channels in such systems can reveal sensitive information about the user requests and system responses in §2.3.

2.1 Interactive LLM Serving Systems

Our work focuses on causal LLMs, the most common kind across both commercial and open-source use cases (*e.g.*,

GPT [28], Llama [1], Phi [29], etc.). These models are trained to predict the most probable next *token* given an input sequence of tokens, referred to as a *prompt*. A token is the smallest unit of text the model processes, such as a word, subword, or character. Causal LLMs generate output by iteratively predicting and appending the next token autoregressively: they compute the distribution of probable next tokens, sample one (*e.g.*, with the highest probability), append it to the prompt, and repeat until a termination token is encountered. This phase of token generation is also referred to as *decoding*.

Streaming for interactivity. With large-scale LLMs using over a thousand billion parameters [1–4], they tend to be hosted in the cloud with public-facing APIs. These APIs permit users to issue prompts to these models and receive responses from them over the Internet. To enhance interactivity and responsiveness, commercial LLM services and open-source LLM serving frameworks typically expose streaming APIs over HTTPS, where generated tokens are streamed back to the user as soon as they are produced [13, 14].

Speculative decoding. Optimizing LLM inference is crucial to both reduce operational costs stemming from high computational demands and improve response latency to meet user interactivity requirements [20]. *Speculative decoding* [15–17] is a common technique to accelerate the inference of causal LLMs by using a smaller, faster approximation model to generate token sequences that the original model then verifies. The benefits of speculative decoding are twofold: (i) a smaller language model can quickly generate a draft sequence that likely overlaps with the output of the original LLM, and (ii) the original model can verify all the tokens in the draft in parallel which is faster than sequential generation.

Speculative decoding employs a *draft-and-verify* mechanism. At each generation step, the input — which includes the original prompt and the tokens generated so far — is fed into an *approximation model*. This approximation model is a computationally efficient version of the original LLM, typically created through methods like distillation [16, 30], early-exiting [31], or adding auxiliary prediction heads [32]. It produces a sequence of μ tokens, where μ is a predefined parameter. The original model then computes the next token distributions for each token in the draft sequence. A draft token is considered *verified* if the most probable next token from the original model matches the draft token, indicating successful speculation. If a draft token fails verification, that token and all subsequent tokens are discarded. Verified tokens are appended to the prompt, and the process repeats.

Figure 1 (bottom) demonstrates speculative decoding for a diagnostic text completion example (similar to current medical LLMs use-cases [5, 6], §2.2), with $\mu = 5$, where a user queries the LLM with the prompt, “What is a probable cause for a severe headache?” and specifies the output prefix, “A severe headache may”. With speculative decoding, the draft model generates the next $\mu = 5$ tokens as “occur after brain

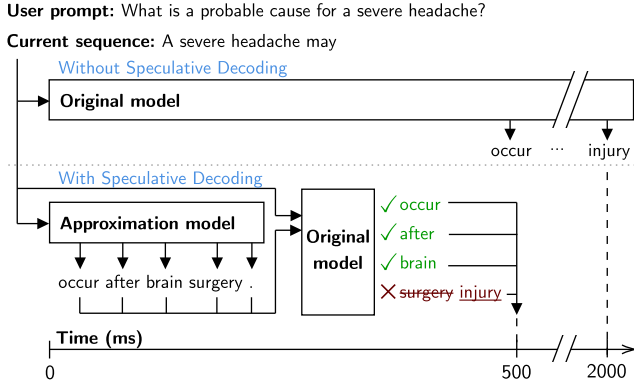


Fig. 1: **Speculative ($\mu = 5$) vs. baseline decoding.** The original model verifies the approximation model-suggested tokens every round and accepts a subset (green checkmarks).

surgery.” The original model accepts the first three tokens, rejects the rest (i.e., “surgery” and “.”), and inserts the token “injury” into the sequence to complete the response. Compared to the execution without speculative decoding (Figure 1 (top)) — where the slower original model sequentially generates all four tokens — speculative decoding generates the first three tokens much faster than the original model decoding round via the approximation model, using the slower original model to verify these token in parallel and generate the last token. In practice, this reduces the average time-per-token by $\sim 5 \times$ [16, 17, 20], significantly improving responsiveness for streaming APIs with high accuracy, and speed up response generation by up to $3 \times$ [20].

2.2 Privacy-Sensitive LLM Use Cases

Our work focuses on domain-specific LLM-based services with privacy-sensitive use cases. These applications employ predefined input prompt templates [33] that are engineered to augment the user’s input with domain-specific knowledge and restrict the response to generate a reliable and valid result. Throughout this paper, we focus on a medical use case, ChatCAD [5, 6], and a financial use case, AWS stock price forecasting [34], as target applications since they communicate private information in both input prompts and responses. While these applications require input prompts to follow a specified template, they do not restrict LLM outputs.

Medical use case. ChatCAD [5] and ChatCAD+ [6] leverage public LLM services for reliable computer-aided diagnosis (CAD). They employ LLMs for interactive multi-modal medical diagnosis and temporal time-series forecasting, involving multi-modal private information such as patient descriptions and medical images. ChatCAD uses a prompt template that compiles disease descriptions and patient reports from multiple sources. The LLM’s response template is designed to produce a detailed diagnostic report with a disease prediction. While many privacy-sensitive elements exist in both the prompts and responses, we focus primarily on unveiling the

LLM-generated disease prediction, which contains critical medical information that must remain confidential.

Financial use case. AWS stock forecasting [34] utilizes prompt templates that instruct the LLM to forecast the next week’s stock return for a given stock symbol based on the company profile, historical weekly news summaries, keywords, stock returns, and optionally examples from similar companies. The predicted trend is categorized using bins like “D3” and “U5+”, where each represents a specific range of price change (e.g., “D3” means a drop of $\sim 3\%$). The LLM’s response templates include placeholders for the predicted summary, keywords, and predicted trend, explaining the reasoning behind the forecast. We focus on unveiling the queried stock symbols and the LLM-predicted trends, as these tend to be proprietary, necessitating confidentiality.

Appendix A describes the input prompt templates and potential responses for both use cases in detail.

2.3 Network Side Channels in LLM Serving

While streaming APIs and server-side LLM optimizations, such as speculative decoding, improve interactivity and responsiveness for user-facing applications, they also inadvertently introduce new network side channels. As an example, consider an LLM service that streams each token back to the user as soon as it becomes available; if the system employs speculative decoding, the inter-token arrival times (t_{inter}) will vary depending on whether the approximation model (shorter t_{inter}) or the original model (longer t_{inter}) generated the token. Such a timing channel is easily visible to a network adversary that snoops on traffic between a target user and the LLM service, even if the traffic itself is encrypted.

As a more concrete hypothetical example, in a medical use case, the approximation model may be able to diagnose common diseases like flu, enabling low-latency responses. In contrast, rare and complex diseases (e.g., glioblastoma) may be delegated to target model under speculative decoding, requiring more time to generate tokens. This allows an adversary to correlate lower token generation times with the more probable diseases like flu and use such correlations to make high-accuracy predictions about sensitive information (e.g., potential diagnoses) contained in actual user prompts and corresponding LLM responses.

One way to hide the timing channel is to stream tokens in batches; however, if not done carefully, batching may still reveal just as much information. For instance, to preserve the latency benefits of speculative decoding, the LLM serving system may batch together tokens generated in the same decoding round — i.e., a sequence of tokens generated by the approximation model, followed by a token generated by the original model. However, the sizes of these batches would be correlated to the number of tokens generated in the corresponding round, which a network adversary can observe to deduce the same information as before.

While these are just two points in the design space, both

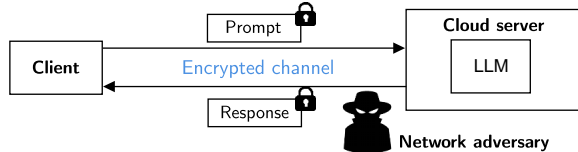


Fig. 2: **Threat model.** The adversary observes a transcript of encrypted packets between the LLM service and a client.

streaming models described above are used in real-world LLM serving systems [27, 35]. Our work formalizes the security model under which such network side channels may be exploited (§3) and explores a diverse set of streaming models (including the two above) that may reveal these side channels (§4). Moreover, we find that these side channels have strong correlations with privacy-sensitive fields in corresponding user prompts and LLM responses; we leverage this observation to develop a practical attack (§5) that can predict the identity of such fields in real-world medical and financial use cases with high accuracy, both for real-world LLM services (§6) and in our own evaluation setup (§7).

3 Security Model

This section defines the security model for network side-channel attacks on interactive LLM serving systems. We begin by describing our threat model (§3.1) and conclude with a formal definition of security (§3.2). Our system model follows those for interactive LLM services described in §2.1: users send input prompts (following the application-specific template) over the network to the system, which runs LLM inference on each prompt and streams the output response. The communication channels between the system and its users are encrypted, *e.g.*, using TLS, so no entity in the network can observe the prompts or LLM responses in plaintext.

3.1 Threat Model

Our threat model assumes a network adversary (Figure 2) that can view encrypted communications between the user and the LLM service, but not any computations or communications within the service provider’s infrastructure. Moreover, the adversary is *passive persistent*, *i.e.*, it neither injects any traffic of its own nor tampers with the user traffic. Practical examples of such network adversaries include compromised browsers on clients, Wi-Fi access points, routers, ISPs, tapped network cables [36, 37], and TLS-session eavesdropping [23–26].

Adversarial knowledge. The adversary aims to infer the private fields within the user’s prompt or the LLM’s response. We assume that it knows the application of the LLM, specifically the format of the input prompts used and the set of potential classes of the private attributes in the LLM responses (as outlined in §2.1) — *e.g.*, predicted disease or stock symbol in our medical and financial use cases, respectively (§2.2). We also assume that the adversary has black-box access to the LLM service, *i.e.*, it can send prompts to the LLM ser-

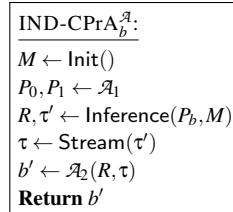


Fig. 3: IND-CPrA with adversary \mathcal{A} and random bit b .

vice and receive responses but cannot view the LLM’s architecture or parameters. This is often the case for user-facing services [13, 14, 38, 39], where the adversary can pose as a regular user to extract information about how specific sensitive attribute values affect the network traces (§5). The adversary can also leverage prior knowledge of domain-specific data distributions, *e.g.*, for the medical use case, it may know the distribution of common diseases. This is a practical assumption for real-world cases where distributional information about the population or application domain is common knowledge [40–42]. We also consider how the adversary fares without these distributions in §7.

Finally, the adversary can collect traces of the precise sizes and receive times of the encrypted packets transmitted between the LLM server and clients. These traces capture side-channel leakage from the system and reveal information about the execution of the LLM inference and, consequently, private information within the prompts and responses.

3.2 Security Definition

We formalize our notion of privacy for an LLM serving system using a game-based definition called ‘Indistinguishability under Chosen Prompt Attack’ (IND-CPrA). IND-CPrA follows the concrete security approach [43] to analyze whether an interactive LLM serving scheme is secure under the network adversary outlined in §3.1. At a high level, in the IND-CPrA game, the adversary generates two user prompts, and the LLM service randomly picks one to perform inference on it. The adversary observes the encrypted network traffic sent back by the LLM service and uses this information (along with prior knowledge outlined in §3.1) to “guess” which of the two prompts the LLM service ran the inference on. The scheme is secure if the adversary’s guess is no better than a random coin toss and insecure otherwise.

Security game (Figure 3). We formalize an LLM serving scheme as a tuple of algorithms (Init, Inference, Stream). Init initializes the LLM model and system parameters, including optimizations (specifically, speculative decoding); Init’s output M encapsulates the post-initialization system state. Inference takes as input a user prompt (P , represented as a sequence of m tokens) and Init’s output (M) to generate an encrypted response (R). Inference also generates a transcript τ' of the tokens in R and the timestamps at which they are generated by the LLM. While this intermediate transcript is not visible to the adversary, it serves as input to Stream, which

captures LLM serving scheme’s response streaming model (§4), i.e., how it places the generated tokens in packets and sends them to the user. The adversary can observe the output of Stream — a network transcript that captures the scheme’s outgoing traffic, modeled as a sequence of n encrypted packets, where $\tau[i].\text{time}$ denotes the timestamp at which the i^{th} packet was captured and $\tau[i].\text{size}$ denotes its size.

The game is parameterized by an adversary \mathcal{A} — formalized as a pair of algorithms \mathcal{A}_1 and \mathcal{A}_2 — and a bit $b \in \{0, 1\}$ (kept secret from \mathcal{A}). In the game, the adversary generates two prompts P_0 and P_1 using algorithm \mathcal{A}_1 . The scheme runs LLM inference on P_b (determined by the random bit b) and produces the encrypted inference result R and the transcript τ corresponding to the observed network traffic. The adversary \mathcal{A} uses these two outputs to run algorithm \mathcal{A}_2 to guess the bit b , i.e., which prompt was chosen, and outputs this guess as b' . We express adversary \mathcal{A} ’s ability to distinguish between the scheme’s executions on P_0 compared to P_1 as its IND-CPrA *advantage*. More formally, the advantage is the absolute difference in probabilities that the adversary guesses $b' = 1$ when the game is initialized with $b = 0$ and $b = 1$, i.e.,

$$\text{Adv}^{\text{IND-CPrA}}(\mathcal{A}) = |\Pr[\text{IND-CPrA}_0^{\mathcal{A}} \Rightarrow 1] - \Pr[\text{IND-CPrA}_1^{\mathcal{A}} \Rightarrow 1]|$$

A scheme where the adversarial advantage is non-zero is insecure under IND-CPrA; this definition can thus serve as a framework for evaluating the security of mitigations against any attacks that exploit the network packet timings of an LLM service. In §5, we will present a practical realization of the adversary to guess the chosen prompt and show in §5.4 that when applied to a wide range of real-world and hypothetical streaming models for interactive LLM serving systems, it provides a non-zero adversarial advantage.

4 LLM Response Streaming Models

We consider a range of real and hypothetical streaming models for interactive LLM services that differ in how they stream responses from the LLM service back to the user. For each model, we show how the network adversary (§3) can extract low-level information embedded within the network traffic typically correlated with LLM optimizations. We will discuss how this information can be used to learn sensitive fields in user prompts and LLM responses in §5. Although many more design points are possible, we consider six representative streaming models to demonstrate the different granularities of information they leak to the network adversary.

4.1 Token-by-token Streaming

Our first considered category of streaming models streams one token back to the user at a time, with each token sent back in a separate packet as soon as it is available at the LLM service. We consider three models:

PacketPerToken describes a streaming model where the generated tokens are converted into strings (i.e., detokenized)

before they are packetized, encrypted, and sent to the user. It encompasses real-world open-source LLM serving systems like vLLM [27] and represents one extreme of the design space, where the network adversary can extract the most information via network timing and size channels.

TokenPadding is a variant where each detokenized string is padded to the maximum string encoding size before it is sent in a packet to the user to hide the sizes of individual tokens from a network observer.

NoSpecDecoding builds atop TokenPadding (i.e., employs per-token padding), and in addition, also disables speculative decoding. This is an extreme where no information about token optimizations or sizes is leaked to an adversary, but the responses are still streamed to the user for responsiveness.

Leaked information. As outlined in §2.3, optimizations like speculative decoding introduce noticeable variations in token generation times. Intuitively, tokens generated via the approximation model in speculative decoding have observably lower latency in this steaming model category than those generated by the original model. Specifically, the transcript τ reveals this latency gap through its timing channel since the inter-token arrival times t_{inter} are strongly correlated with inter-packet arrival times, i.e., $t_{inter}^i \approx \tau[i].\text{time} - \tau[i-1].\text{time}$. The differences in inter-token arrival times can then be used to identify which tokens are generated by the approximation vs original models, as shown in Figure 4(a).

The adversary can use statistical methods like HDBSCAN³ [44] to group tokens into different classes based on their arrival times. HDBSCAN extracts clusters in inter-packet delays, which is always two for speculative decoding. While Figure 5 shows that the two categories of tokens are distinguishable with high confidence for real-world medical and financial use cases, we evaluate the precision of HDBSCAN in grouping them into different classes in §7.2.

If no padding is used, in addition to the timing channel, the adversary can also learn token sizes, i.e., the length of the detokenized string, by subtracting various header sizes from each packet size $\tau[i].\text{size}$. All in all, this gives us two key pieces of information:

- *Optimization vector.* A bit vector where i^{th} bit is 1 if the approximation model generates the i^{th} token, and 0 otherwise.
- *Token sizes.* The size of each detokenized string (typically UTF8-encoded) in the LLM response.

In TokenPadding the adversary cannot learn the token sizes, but it can still infer the optimization vector from the timing channel. In contrast, NoSpecDecoding prevents the adversary from learning optimization vector as well, only revealing the total number of tokens in the response.

³Hierarchical Density-Based Spatial Clustering of Applications w/ Noise

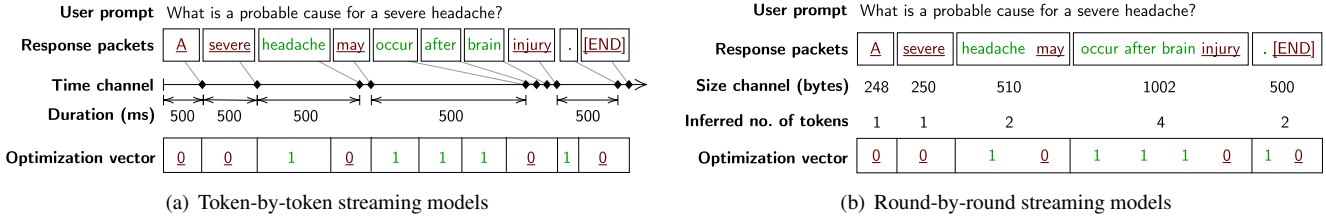


Fig. 4: **Reconstructing the optimization vector** via (a) the timing channel in token-by-token streaming models (§4.1), and (b) the size channel in round-by-round streaming models (§4.2).

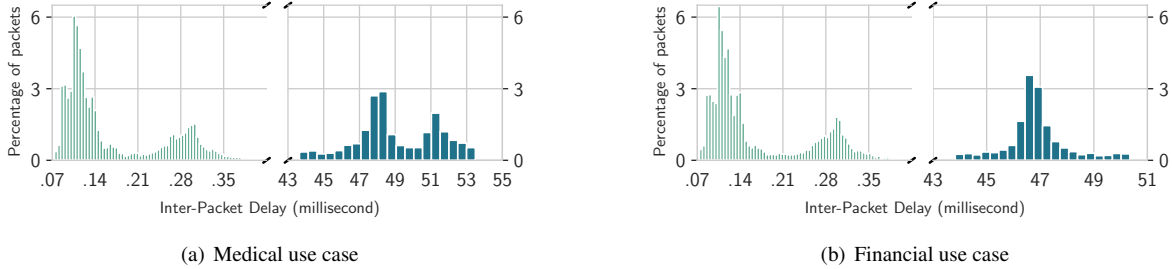


Fig. 5: **Histogram of inter-packet delays.** The streaming delays of PacketPerToken are distributed along two different clusters, corresponding to tokens generated by the original model (Phi3-14B, which yields the higher inter-packet delay cluster) and those generated by the approximation model (Phi3-3.8B, which yields the lower inter-packet delay cluster). The x-axis scale breaks in (a) and (b) demarcate regions with different scales (sub-millisecond vs hundreds of milliseconds) and shows that these clusters are clearly separable for a network observer. See §7 for details on the experimental setup.

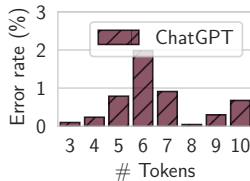


Fig. 6: **Error rates in estimating token counts per packet across packets containing varying numbers of tokens in ChatGPT.** The estimation errors remain $< 3\%$ for TLS response traffic from ChatGPT. The x-axis is limited to < 10 tokens since not enough packets have over 10 tokens to yield statistically significant results.

4.2 Round-by-round Streaming

In this streaming model category, tokens generated in a single speculative decoding round are aggregated into a single packet before being sent back to the user. We consider two models:

PacketPerRound concatenates all tokens generated in a decoding round and sends them to the user in a single response packet. This model is used in real-world offerings, *e.g.*, ChatGPT [13], vLLM [45], and Gemini [14, 46] (evaluated in §6).

PacketPadding pads each token to the maximum string encoding size *and* injects fake tokens into the response packet, ensuring that the same number of tokens ($\mu + 1$) are sent to the user for each decoding round.

Leaked information. In the round-by-round streaming model, the adversary can no longer extract the optimization

vector via inter-packet delays. However, it can still leverage the size in PacketPerRound to reconstruct the optimization vector. To do so, it uses the observation that with speculative decoding, all tokens except the last generated in a decoding round come from the approximation model. By correctly estimating the number of tokens n_i generated for every round i , the adversary can reconstruct the optimization vector as shown in Figure 4(b).

To estimate the number of tokens in each round, the adversary exploits the positive correlation between the number of tokens generated in a decoding round and the size of the packet containing those tokens. Specifically, knowing the average size of token encodings (t_{avg}) across packets, the adversary can estimate the number of tokens n_i for round i from the packet size $\tau[i].size$ as $n_i \approx \lfloor \frac{\tau[i].size}{t_{avg}} \rfloor$. We note, however, that this estimation is subject to estimation errors stemming from high variations in token sizes, *i.e.*, if the token sizes vary significantly, an observer may not get an accurate account of the number of packets per round. Interestingly, in real-world systems, we find that the token encoding schemes employed in LLM responses end up ensuring that encoded token sizes have a very small variance, *i.e.*, the average encoded token size permits accurate estimation of n_i per packet (§6.2). Figure 6 shows that this approach achieves very low token count estimation errors for packets containing varying numbers of tokens in OpenAI’s ChatGPT [13] service. While we omit results for vLLM [45] and Gemini [46] for brevity, they observe similarly low estimation errors using a similar approach.

PacketPadding ensures that the adversary observes a constant number of padded tokens sent per round, so packet size $\tau[i]$.size is the same constant for every round i . The only additional information the adversary can learn is the number of decoding rounds the LLM takes to generate its response, equal to the number of packets recorded in τ .

4.3 Non-Streaming Model

Finally, we consider the model that sends only one packet per response, NonStreaming: it waits until the LLM has generated all the tokens in the response and then sends them back to the user in a single packet. This model represents another extreme in the design space: preventing the network adversary from extracting useful information by forgoing interactivity.

Leaked information. In NonStreaming, τ contains a single entry, where the timestamp corresponds to the end-to-end latency of the LLM response and packet size corresponds to the sum of encoded token sizes in the response. As a result, only the total number of tokens in the response is revealed, estimated using the average encoded token size (as in PacketPerRound). As with the estimation techniques described in §4.2, this estimation is also subject to errors depending on the variance in token sizes.

5 A Novel Network Side-Channel Attack

We now present a novel attack that leverages information available to the network adversary (defined in §3) for a wide range of LLM service streaming models (outlined in §4) to learn privacy-sensitive fields within encrypted user prompts and LLM responses with high accuracy. This attack analytically establishes the lack of security under IND-CPrA (§5.4) for LLM services that use any of the streaming models in §4.

5.1 Attack Design

Goals. As noted in §2, in the privacy-sensitive use-cases of LLMs, the user prompts and LLM responses contain sensitive user attributes, such as preferred stock ticker symbols or predicted trends in the financial use cases, or symptoms and diagnoses in medical use cases [5, 6, 34]. Our attack aims to extract these private attributes by leveraging LLM optimization information leaked in the network communications between the LLM service and the user (§4). As discussed in §3.1, our attack assumes that the adversary has black-box access to the LLM service and knows the target privacy-sensitive use-case (i.e., their prompt templates, §2.2).

Overview. Our attack operates in two phases: exploration phase and exploitation phase. In the exploration phase, the adversary tries to learn a mapping between the side-channel information embedded in network side channels (discussed in §4) and the private fields in user prompts/LLM responses; it does so with *black-box access* to the LLM service, i.e., by pretending to be an ordinary user. In the exploitation phase, the adversary captures the encrypted network traffic from the

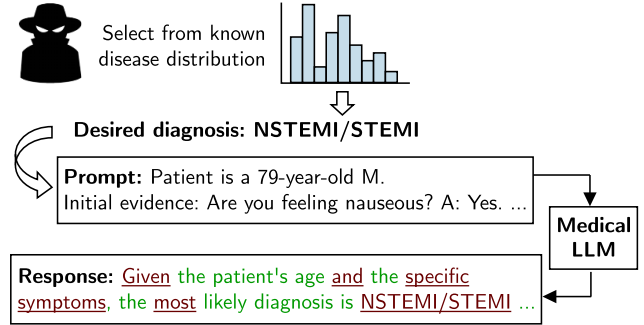


Fig. 7: The adversary generates a set of prompts for the exploration phase by (i) selecting the desired value of a private attribute (e.g., diagnosis=NSTEMI/STEMI) drawn from a known distribution and (ii) then crafting an input prompt that would generate a response containing that attribute.

LLM service and predicts the private attribute values using the mapping learned in the exploration phase.

5.2 Exploration phase

This phase learns how to map information embedded within the network traces to privacy-sensitive attributes and comprises two steps. In the first step, the adversary issues a curated set of prompts to the LLM service, capturing both LLM responses and network side channels. In the second step, it trains a neural network over the captured data to learn the mapping between side channel information and the privacy-sensitive attributes.

Step 1: Collecting side channel information and private fields for a curated set of prompts. The adversary first curates a set of input prompts that follow the prompt template and attribute distribution of the targeted privacy-sensitive use case (§2.1). Figure 7 (top) shows an example of a prompt for the medical use case. To generate this prompt, the adversary first selects the diagnosis (a privacy-sensitive attribute) it wants to see in the LLM response (NSTEMI / STEMI in the figure) and then creates a patient description in the input prompt based on the selected diagnosis. The way these prompts are curated can be critical for the effectiveness of the attack. In particular, in many cases, privacy-sensitive attributes (e.g., medical diagnoses) follow well-known distributions. As such, creating a set of prompts that follow this distribution improves the accuracy with which the attack can predict the attribute value for real user prompts in exploitation phase. However, if such information is unavailable to an adversary (e.g., a proprietary distribution of preferred stock symbols for a financial use case), it can always sample the values for these attributes following a uniform random distribution over the attribute domain. We study the impact of this information’s availability on the attack’s efficacy in §7.3.

The adversary then issues its curated prompts to the LLM service, recording the corresponding responses and packet traces. For each prompt, it extracts (i) the privacy-sensitive

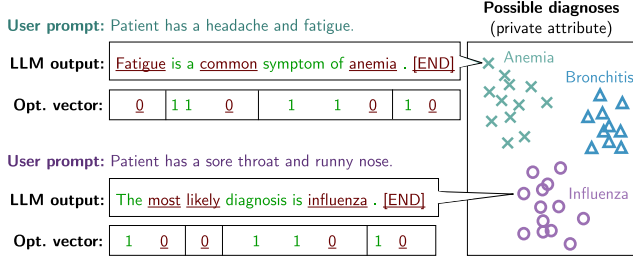


Fig. 8: User prompts for different privacy-sensitive attributes (e.g., anemia vs. influenza) result in responses with separable ‘clusters’ of optimization vectors, which our classifier model learns in the exploration phase.

fields in user prompts and LLM responses and (ii) information leaked in the packet traces (e.g., token sizes, optimization vector, etc.) as described in §4. This information serves as input features for training a classifier in the next step.

Step 2: Mapping leaked information to privacy-sensitive attributes. In the second step, the adversary trains a classifier model using the extracted information (leaked in packet traces) as input and the privacy-sensitive attributes as the classification labels. The success of this classifier hinges on its ability to classify prompt-response pairs into distinct classes with the same privacy-sensitive attribute based on the leaked information. Figure 8 shows an example of this clustering across different prompts in the medical use case, using only the optimization vector as the sole source of leakage. We use a simple Multilayer Perceptron (MLP) [47] to capture the relationship between these input vectors and private attributes with high accuracy across various domains (§7.1). We note that our attack is modular and can employ a wide variety of other models like random forests [48] and ensemble methods [49, 50] to determine the final label. We analyze the impact of model selection on training accuracy in §7.3.

5.3 Exploitation phase

In this phase, the adversary uses the trained classification model to predict privacy-sensitive attributes for victim user-LLM service interactions based on the captured encrypted communications between them. In particular, for each victim user prompt, the encrypted packet traces for the LLM response are analyzed to extract the leaked information (as outlined in §4). This information is used to construct an input feature vector and fed to the trained classifier model, which yields a classification label, i.e., a value for the privacy-sensitive attribute. The accuracy of this prediction is the basis for showing the lack of security of existing interactive LLM services under IND-CPrA, as we show next.

5.4 Security Impact

Our IND-CPrA adversary \mathcal{A} is defined as follows: \mathcal{A}_1 runs the exploration phase using its black-box access to the LLM and generates P_0 and P_1 based on its prompt curation strategy. After the LLM serving scheme executes inference on

P_b , \mathcal{A}_2 runs the exploitation phase on the generated transcript τ_b . \mathcal{A} guesses $b' = 0$ if the classifier’s output matches the label of P_0 , and $b' = 1$ if the classifier outputs the label of P_1 . If the classifier’s output does not match either prompt, \mathcal{A} guesses randomly. Our evaluation (§7, §6) confirms that the classifier achieves (often significantly) better accuracy than random, i.e., \mathcal{A} ’s advantage under IND-CPrA is non-zero ($\text{Adv}^{\text{IND-CPrA}}(\mathcal{A}) > 0$) for existing LLM serving systems, confirming their lack of security.

6 Attacking Real-world LLM Services

We study the feasibility of our attack (§5) on three real-world LLM services. We seek to answer the following questions:

- Is the attack effective on black-box LLM services where the adversary lacks knowledge of their internal operation and optimization techniques? (§6.1)
- What information do real-world LLM services leak in their network side channels? (§6.2)

Targeted services. We target (i) ChatGPT’s GPT-4 models [13, 35] (gpt-4o, gpt-4o-mini) (ii) Google’s Gemini model 1.5-pro [14], and (iii) vLLM framework [27, 45, 51] from [52] using Phi3 14B and 3.8B models as original and approximation models, respectively.

While ChatGPT and Gemini internals are not public, an analysis of captured packet traces for their streaming APIs suggests the use of round-by-round streaming (§4.2). For instance, in ChatGPT, each token is encoded as a JSON object (adding 244B metadata to 3–15B tokens), and several of these tokens (presumably for the same decoding round) are batched into a response packet.

Since vLLM’s infrastructure code is open sourced, we have better insight into its internal operation. Specifically, it uses speculative decoding and follows the token-by-token streaming model (§4.1), a variant of PacketPerToken. However, each packet includes not only the newly generated token but also all the previously generated tokens for that prompt, e.g., successive response packets for a medical prompt might contain “A,” “A severe,” “A severe headache,” and so on.

The adversary observes the encrypted packet traces between the user and the LLM service for all systems, including the TLS handshake, cipher exchange packets, and data packets. During the exploration phase, the attack filters out non-data packets and extracts the leaked information from the packet transcript based on each service’s closest streaming model (as discussed in §4). We discuss the implications of JSON wrapping and cumulative encoding of tokens in packets on extracting the leaked information in §6.2.

Curating prompts for targeted use cases. Due to budget limitations, we restrict the size of the adversary’s curated prompts for large models (gpt-4, gpt-4o and gemini). Specifically, we use 10k prompts for the financial use case, restricting focus to the 10 largest NASDAQ [53] companies for stock symbols. For each symbol, we focus on trends for 350 days,

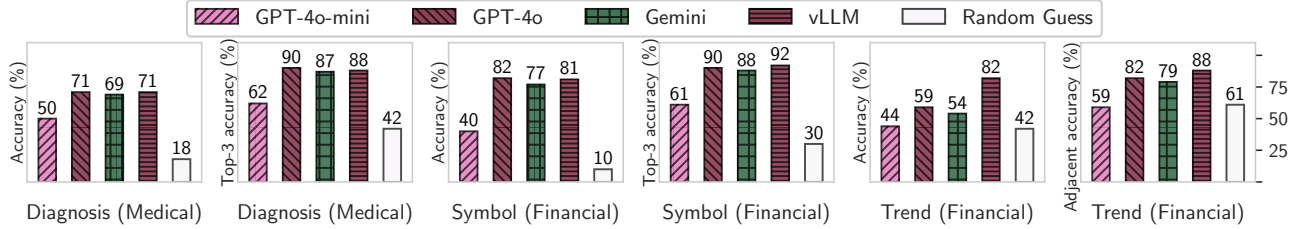


Fig. 9: **Attack accuracy for real-world LLM services** on medical and financial use cases. The adversary achieves high accuracy by extracting payload sizes and optimization vector from the PacketPerRound streaming model used by ChatGPT [13, 35] and Gemini [46]. vLLM demonstrates accuracy comparable to the results presented in §7.1.

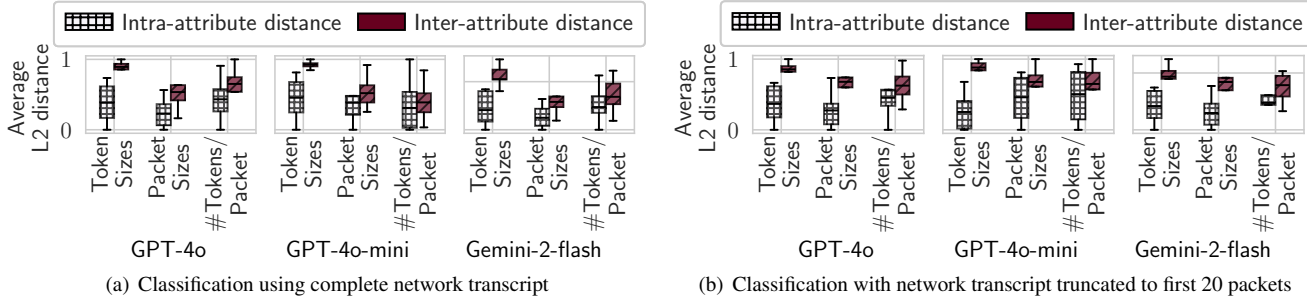


Fig. 10: **Intra- and inter-attribute distances in GPT-4o models for different leakages.** Each box marks 25th and 75th percentile, while the horizontal line in it marks median and whiskers mark minimum and maximum values. See §6.2 for details.

with three possible news events affecting the outcome of the LLM’s stock predictions. For the medical use case, we focus on the 10 most common diseases and use 10k prompts.

6.1 Attack Efficacy

Our attack experiments follow the same structure as those in §7.1; we now summarize our key takeaways for Figure 9.

The attack is effective on real-world LLM services. Our attack achieves between 54–92% prediction accuracy for privacy-sensitive attributes on the evaluated financial and medical use cases, which is much higher than that for random guesses (*e.g.*, 0.96% for stock symbols, 8.5% for diseases). This establishes adversarial advantage under IND-CPrA.

GPT-4o observes higher attack accuracy than 4o-mini. Interestingly, the attack observes higher accuracy against the gpt-4o model than the gpt-4o-mini model. As we will show in §6.2, this is because the leaked information (*e.g.*, optimization vector and packet sizes) for gpt-4o-mini have a weaker correlation with the privacy-sensitive attribute class that the prompt-response pair corresponds to, when compared to gpt-4o. While we do not know to what extent either of these models employs speculative decoding (or similar) optimizations, the difference in correlations suggests a more aggressive use of optimizations in gpt-4o than in 4o-mini.

Gemini observes lower accuracy than GPT. On average, gemini observes 3–5% lower accuracy than gpt-4o. We attribute this to the slightly higher error rate in extracting the optimization vector from network traces in Gemini.

Mounting the attack is relatively inexpensive (\leq \$100). Since the exploration phase requires the adversary to issue prompts to the LLM service, the cost of this phase of the attack is measured as the cumulative cost of API calls to the respective models. Our measurements indicate that it costs \$94 and \$3.4 for the gpt-4o, and gpt-4o-mini models, respectively. This shows that a careful curation of prompts can lead to significant accuracy at a relatively low cost, even when targeting the most expensive and powerful LLM models. Additionally, the adversary needs to train the MLP as its classifier, which takes around 8 mins on an Nvidia L4 GPU, which costs $<$ \$1 even by conservative estimates.

6.2 Information Leaked in Network Traces

We now analyze the accuracy and impact of the information extracted from network traces for real-world LLM services.

Extracting leaked features from packet traces. The PacketPerRound streaming model (§4.2) in ChatGPT and Gemini exposes two side channels: the token count per packet and the size of each packet. Interestingly, per-token JSON encoding in the responses enables a more accurate estimation of the number of tokens per packet. This is because such JSON encoding reduces variance in the size of each encoded token (from 3–15B in PacketPerRound to 248–260B in ChatGPT responses), reducing the error in estimating token counts based on the average encoded token size.

On the other hand, each of vLLM’s response packets contains both the new and all previous tokens; as such, every subsequent packet is larger than the last. This can introduce noise in the timing channel due to variability in packet sizes

and, therefore, transmission delays. However, the measured error of the optimization vector generation is negligible (0.12%) since the additional transmission delay introduced by a longer response string ($< 300\mu\text{s}$) is at least two orders of magnitude smaller than the difference in token generation times between the original and approximation models ($> 50\text{ms}$).

Mapping leaked features to private attributes. As discussed in §5.1, the accuracy of our attack’s classifier model depends on the correlation of leaked features to the private attributes: for high accuracy, prompt-response pairs with the same attribute value should have similar leaked features, and vice versa. We quantify this similarity using the normalized L2 distance between leaked features represented as integer vectors. Specifically, *intra-attribute* distance corresponds to the average L2 distance between two feature vectors for the *same* attribute. In contrast, *inter-attribute* distance represents the same for feature vectors for *different* attributes. A good candidate for leaked features would ensure that the gap between intra- and inter-attribute distance for the corresponding feature vectors remains high for high clustering accuracy.

In Figure 10(a), we compare the inter- and intra-attribute distances for the two leaked features in OpenAI’s streaming model: packet sizes (i.e., sum of token sizes per packet) and the number of tokens per packet. While individual token sizes for ChatGPT and raw packet sizes for Gemini are not leaked for these models, we still include the attribute distances based on token sizes to serve as an upper bound on the achievable gap between the intra- and inter-attribute distances. The figure shows that packet sizes and (to a lesser extent) the number of tokens per packet serve as reliable features for classifying prompt-response pairs based on their private attribute value, for both `gpt-4o` and `gemini`, albeit less so than token sizes (which are not leaked). In our evaluations, the combination of packet sizes and number of tokens permit sufficiently accurate clustering, resulting in high prediction accuracy (§6.1).

For ChatGPT, we also observed that, despite configuring the LLM to minimize randomness in its outputs (via its temperature parameter), responses to the same prompt begin to diverge when they grow beyond a certain number of tokens [54]. This reduces prediction accuracy in the attack for longer responses. One approach to address this is to truncate the network transcript to a fixed number of tokens to remove the randomness introduced by the latter tokens during the attack’s exploration and exploitation phases. Figure 10(b) shows the accuracy with the transcript truncated to the first 20 packets, demonstrating a larger gap between intra- and inter-attribute distance. We leave investigating the ‘optimal’ number of tokens for maximizing attack accuracy to future work.

7 Attacking Controlled LLM Serving Systems with Speculative Decoding

We evaluate our proposed attack (§5) atop various LLM streaming models (§4) in a controlled setup to understand:

Table 1: Summary of Curated Prompts

Use case	# Possible Attribute Values	#Prompts	References
Medical	35 (diagnosis)	949,854	[5, 6, 59]
Financial	104 (symbols), 12 (trends)	1,545,648	[34, 53]

- accuracy of the proposed attack for various privacy-sensitive use cases and streaming models (§7.1),
- impact of noise (§7.2) and adversarial knowledge (§7.3).

Targeted LLM serving setup. We implemented a reference LLM service that processes user prompts sequentially. The service employs Microsoft’s Phi3 model [55] as a representative of open-source LLMs, sharing its architecture with widely used models like Llama [1] and Gemini [14]. We use the 14B [29] and 3.8B [56] models as the original and approximation models, respectively. We run the models on an NVIDIA RTX 4090 GPU with 24GB memory using int 8-bit quantization [57] for the approximation model via the bitsandbytes library [58]. We use deterministic sampling (top-1) for token generation and limit the number of output tokens to 128. Our service supports all streaming models introduced in §4.

Curating prompts for targeted use scenarios (Table 1). We evaluate the medical and financial use scenarios introduced in §2.2. We extract diseases and patient descriptions from the DDXPlus [59] dataset and format them in a ChatCAD prompt template [5, 6]. For the financial service, we generate a dataset following the template used in a recent LLM stock prediction system [34]. Specifically, we extract stock trends from Nasdaq-100, create stock symbol descriptions using `gpt-4o` [13], and for each symbol, select 1 of 12 `gpt-4o`-generated news articles ranging from “very unfavorable” to “very favorable” to consider the potential effect of the news on stock prices. For both use cases, we reserve 10% of the prompts as real user prompts that are not made available to the adversary. We assume that the disease distribution is known to the adversary while prompts for the financial use case are uniformly distributed over stock symbols.

7.1 Feasibility of the Proposed Attack

We use three metrics to evaluate the attack accuracy: accuracy, top-3 accuracy, and adjacent accuracy. Accuracy is the percentage of cases where the adversary correctly identifies the LLM-assigned attribute. Top-3 accuracy is the percentage of cases where the correct attribute value is among the top three predictions from the attack’s classifier. Finally, adjacent accuracy, measured only in the financial use case, is the percentage of cases where the correct trend is either equal to or a neighbor of the predicted trend (e.g., U3 predicted for U2; see §2.2 for a description of trend classes). These more ‘relaxed’ accuracy metrics correspond to practical threat models where the adversary may be satisfied with *approximately* correct predictions. We now present our key takeaways.

The attack achieves high accuracy for streaming mod-

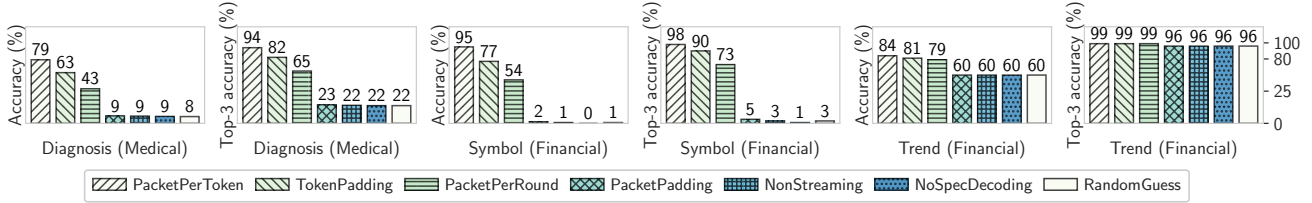


Fig. 11: **End-to-end prediction accuracy for the attack across different streaming models.** As information in the size and timing side channels decreases, attack accuracy approaches random guesses (8.5% for medical diagnoses, 0.9% for stock symbols, and 60% for stock price trends). PacketPerToken and TokenPadding represent real-world systems (§6) where speculative decoding optimization vectors are visible, resulting in high attack accuracy (43%–98%) across attributes.

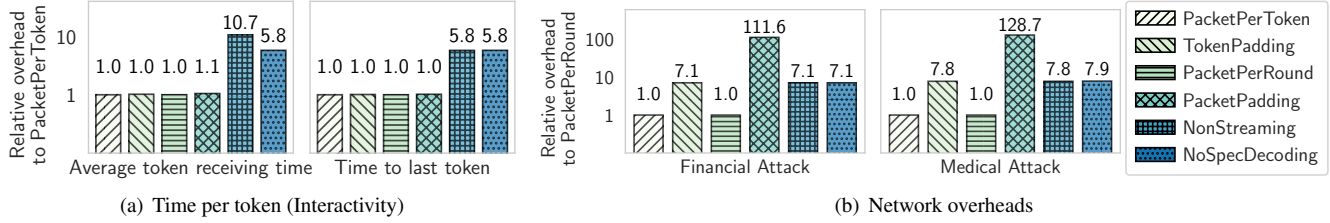


Fig. 12: **End-to-end (a) network overheads and (b) time per token (i.e., interactivity overhead) for different streaming models.** All metrics are normalized to those for the PacketPerToken model. Note that the additional latency introduced by batching tokens from a round into a packet (§4.2) is negligible since the original model verifies these tokens in parallel; only NonStreaming incurs large interactivity penalties since it sends tokens only after all are generated.

els used in the real-world (Figure 11). In particular, for the streaming models that real-world services are based on (PacketPerToken and PacketPerRound; §6), the accuracy is much higher (63–95%) than random guesses (8.5%, 0.9%, and 60% for medical diagnoses, stock symbols, and stock price trends, respectively). The prediction accuracy is even higher (82–98%) for the more relaxed top-3 and adjacent metrics.

Streaming models expose a tradeoff between security and performance (Figure 12). While streaming models that employ batching and padding achieve better security (shown as reduced accuracy in Figure 11), they do so by incurring higher performance penalties. We quantify performance overheads using (i) network bandwidth used for padding and (ii) time per token, which captures interactivity. Figure 12 shows overheads for each streaming model normalized by those for PacketPerRound (which are nominal). We make several interesting observations:

- TokenPadding incurs a $7.5\times$ network overhead to hide token sizes but is ineffective at reducing attack accuracy. This is because the adversary can still use inter-token delays to construct the optimization vector.
- PacketPerRound groups tokens into packets, incurring no interactivity or bandwidth overhead while reducing the attack accuracy relative to TokenPadding. Batching tokens from a decoding round into a packet does not incur additional delays because these tokens are verified by the original in parallel and finalized simultaneously (§2.1).
- PacketPadding is secure in our evaluated use cases since the prediction accuracy is comparable to random guesses,

i.e., the number of decoding rounds does not leak much information. However, it also incurs the highest network bandwidth overhead ($119\times$) by injecting fake tokens with additional padding in each decoding round. Our measurements show that the increased transmission delays due to the larger payloads are negligible relative to LLM overheads and do not affect interactivity.

- Unsurprisingly, NonStreaming is also secure, as it only leaks the sum of all token sizes and the estimated total number of tokens. However, it exhibits significantly higher network overheads and lower interactivity, with the average time per token $11.6\times$ and $9.7\times$ higher for the medical and financial use cases, respectively.
- Similarly, NoSpecDecoding is secure. However, compared to TokenPadding, disabling speculative decoding significantly degrades interactivity (by $\sim 5.8\times$).

Issuing curated prompts to the LLM service dominates the attack’s overheads. Our attack’s overheads stem from three key factors: (i) issuing curated prompts to the LLM service and (ii) training the classifier in the exploration phase, and (iii) communication overheads in the exploitation phase. In our evaluation setup, issuing prompts to the LLM service took 43 hours for the medical use case and 56 hours for the financial use case, primarily due to the time the LLM takes to generate responses. Training the MLP took a mere 16 minutes for 100 epochs, while the communication cost in the exploitation phase is negligible; this is because capturing LLM response packets using `tcpdump` does not incur much overhead given their low data rate.

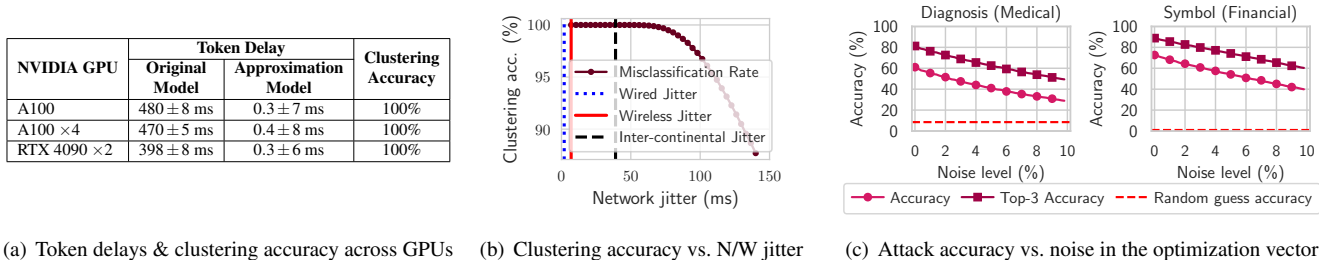


Fig. 13: **Impact of noise.** See §7.2 for details.

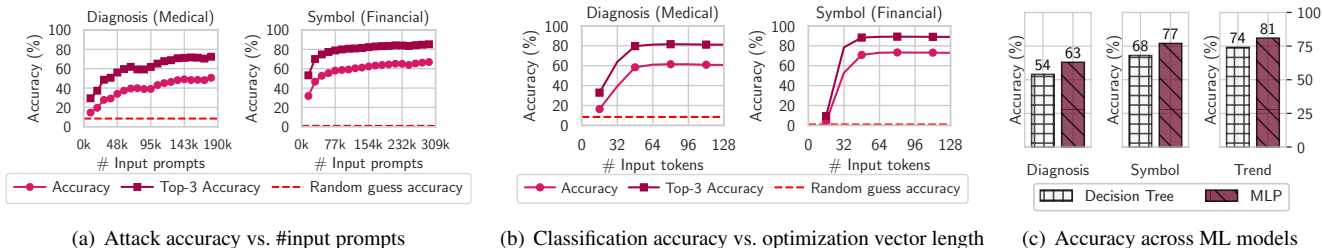


Fig. 14: **Attack’s sensitivity to adversarial knowledge.** See §7.3 for details.

7.2 Impact of Noise

Choice of GPU does not affect the estimation accuracy for the optimization vector (Figure 13(a)). While the exact token generation time varies with the computational power of the GPU, the latency gap between tokens generated by the original and prophet models remains consistently high across different GPU models, i.e., the approximation model is significantly faster than the original model. This permits perfect clustering of tokens using HDBSCAN based on delays.

Network jitter can affect the optimization vector estimation accuracy (Figure 13(b)). We measure the token clustering accuracy via HDBSCAN as a function of network jitter (measured as the standard deviation of network RTTs) to assess its impact on optimization vector estimation. Even with a high network jitter of 100 ms, the clustering accuracy remains around 95%. Although token clustering accuracy decreases noticeably as network jitter increases further, we expect this degradation to be minimal in networks without heavy congestion. To calibrate our findings, we measured the actual network jitter for connections to OpenAI servers, which were ~ 0.10 ms over wired and 2.98ms over wireless channels. We also measured the jitter for a user in Germany accessing the US server over a wireless channel as 26.9. Shown as vertical lines in Figure 13(b), these measurements show that real-world jitters (including effects from cross-traffic) remain well below the threshold that would degrade accuracy.

Attack accuracy decreases as the noise (errors) in the optimization vector increases (Figure 13(c)). We analyze the impact of noise on the end-to-end attack accuracy by directly injecting errors into the optimization vector (by flipping random bits) for the TokenPadding model. The attack accu-

acy remains above random guesses (red horizontal lines), although it does drop down to 35% at 10% noise. Although the accuracy reduction appears significant, we note that direct noise injection into the optimization vector demonstrates the worst-case scenario, e.g., 10% noise corresponds to a network jitter of over 130 ms!

7.3 Sensitivity Analysis

High attack accuracy is achievable with few prompts (Figure 14(a)). We find that attack accuracy decreases sub-linearly with a decrease in the number of prompts that the adversary can train its classifier on — even 10% of the prompts used in Figure 11 can yield comparable attack accuracy. This is notable as the adversary must send each prompt to the LLM service, which typically charges a fixed cost per token. In §6.1 we show that our attack can achieve high accuracy for real-world LLM services with a $< \$100$ budget.

Attack accuracy remains high without knowledge of the distribution of private attributes. In §7.1, the adversary leverages the distribution of diseases in the medical use case to generate prompts and train the classifier. To assess the impact of this knowledge, we repeated the experiment with uniformly distributed diseases. Although the prediction and top-3 accuracy decreased by a small amount — from 63% to 58% and from 82% to 79%, respectively — the result shows the attack can remain effective without this information.

Larger input to the classifier increases the attack accuracy (Figure 14(b)). For both medical and financial use cases, the first 48 tokens contain the most information in the optimization vector; beyond this point, increasing the input size does not improve attack accuracy. Interestingly, in the financial use case, the first 16 tokens emitted by the LLM are nearly

identical (corresponding to a common preamble), containing no valuable information for the attack.

Simpler ML models can achieve comparable accuracy with lower computational cost (Figure 14(c)). While MLPs excel at learning complex patterns in data, simpler ML methods may trade off accuracy for computational cost. We evaluated our attack’s accuracy using a decision tree [60] with a maximum depth of 32 as the classifier. The decision tree reduces accuracy by $\sim 7\text{--}9\%$ compared to the MLP but offers significant computational efficiency; its training takes less than 30 seconds, which is $35\times$ faster than the MLP. This efficiency makes decision trees appealing for adversaries who aim to balance accuracy with the cost of mounting the attack.

8 Related Work

Network side-channel attacks. A long line of network side-channel research has shown that a passive persistent network adversary can learn private user activities [36, 61] and sensitive user information simply by analyzing network traces at various layers, *e.g.*, the application layer [37, 62, 63], IP layer [64] and the physical layer [65, 66]. Our work is the first to formalize the network side channels of interactive LLM services and formulate an attack on them.

Attacks using ML. Machine learning (ML) has become a powerful tool in extracting secrets from noisy or sparse side channels. ML techniques have been successfully applied to side-channel attacks in various settings, including networks [37, 67–69], operating systems [70], and even low-level hardware [71, 72]. The closest to our work are attribute inference attacks that leverage ML models to infer private information about individuals from publicly available data [12, 73]. Our attack builds on this approach to design a high-accuracy attribute inference attack on LLM services, leveraging side-channel vulnerabilities unique to such systems.

Attacks on ML. ML models have been subject to various attacks with different adversarial assumptions and goals. Membership inference attacks [8–10, 74] focus on sensitive information leaked by ML models about their training data. Instead, we focus on extracting private user attributes from the model’s output during inference. Prompt attacks against interactive LLM services use engineered prompts to elicit harmful responses from the LLM [75–77]. In contrast, our work does not aim to change the LLM’s behavior and only queries the LLM service to collect network side-channel information. Closest to our work is recent work on side channels in ML systems [78], although they ignore network side channels. Our work is the first to show that interactivity optimizations inadvertently expose network side channels in LLM services.

9 Discussion

Despite our focus on speculative decoding, side-channel vulnerabilities exist for other optimizations in LLM serving. We

highlight these optimizations below as avenues for future work.

Prompt caching. Prompt caching [18, 79, 80] allows frequently reused prompt segments to be precomputed on the LLM-serving infrastructure and then reused whenever they appear again in a user request. This strategy significantly reduces time-to-first-token (TTFT) latency, enhancing user-perceived responsiveness. However, by precisely measuring TTFT across different requests, an adversary could infer whether a particular prompt segment was served from the cache. Such inferences may leak details about the prompt’s content or reuse patterns — potentially revealing sensitive data if repeated context is detected, as pioneered in [81, 82].

Early-exit LLMs. Early-exit mechanisms for LLMs [83–85] terminate inference once the model becomes sufficiently confident in the next token. This approach parallels speculative decoding: “easier” tokens are generated quickly, while more “difficult” tokens require additional Transformer layers. Consequently, these models exhibit nonuniform token-generation times, exposing yet another timing side channel. By monitoring inter-token delays, an attacker might deduce which tokens are considered easy or difficult, obtaining hints about the underlying text’s structure or content.

Tool-augmented LLMs. Many modern LLM applications integrate external APIs or databases, often through a mechanism called LLM function calling [86–88]. Here, the model emits a function call whenever it needs to access external information or tools. These calls are then interpreted and executed outside the LLM server, with results returned to the LLM in a separate step. A potential side-channel risk emerges because popular LLM services such as ChatGPT and Claude treat function calls as indivisible blocks in streamed responses rather than as standard token-by-token outputs [89, 90]. Developers benefit from this for clarity, but it can leak the length of function call signatures. An adversary monitoring network traffic might detect a sudden increase in payload size correlated with these calls, allowing them to identify whether (and which) external tools are being invoked—a particularly troubling outcome if these calls pertain to sensitive user data or private analytics.

10 Conclusion

Performance optimizations in interactive LLM services, particularly speculative decoding, expose new network-side channels, specifically optimization-dependent packet timings and sizes. Our novel attack on real-world LLM services demonstrates that this leakage is closely tied to private user attributes, presenting an acute privacy concern. We show that defenses against such attacks expose a security-performance tradeoff that may be useful to practitioners designing such systems.

References

- [1] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- [2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [3] Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, Anthony Hartshorn, Elvis Saravia, Andrew Poulton, Viktor Kerkez, and Robert Stojnic. Galactica: A large language model for science. *arXiv preprint arXiv:2211.09085*, 2022.
- [4] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.
- [5] Sheng Wang, Zihao Zhao, Xi Ouyang, Qian Wang, and Dinggang Shen. Chatcad: Interactive computer-aided diagnosis on medical image using large language models. *arXiv preprint arXiv:2302.07257*, 2023.
- [6] Zihao Zhao, Sheng Wang, Jinchen Gu, Yitao Zhu, Lanzhuji Mei, Zixu Zhuang, Zhiming Cui, Qian Wang, and Dinggang Shen. Chatcad+: Towards a universal and reliable interactive cad using llms. *IEEE Transactions on Medical Imaging*, 2024.
- [7] Allen H Huang, Hui Wang, and Yi Yang. Finbert: A large language model for extracting information from financial text. *Contemporary Accounting Research*, 40(2):806–841, 2023.
- [8] Fatemehsadat Mireshghallah, Archit Uniyal, Tianhao Wang, David Evans, and Taylor Berg-Kirkpatrick. An empirical analysis of memorization in fine-tuned autoregressive language models. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1816–1826, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.
- [9] Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Úlfar Erlingsson, Alina Oprea, and Colin Raffel. Extracting training data from large language models. In *USENIX Security*, 2021.
- [10] Matthieu Meeus, Shubham Jain, Marek Rei, and Yves-Alexandre de Montjoye. Did the neurons read your book? document-level membership inference for large language models. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 2369–2385, Philadelphia, PA, August 2024. USENIX Association.
- [11] Shaguftha Mehnaz, Sayanton V. Dibbo, Ehsanul Kabir, Ninghui Li, and Elisa Bertino. Are your sensitive attributes private? novel model inversion attribute inference attacks on classification models. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 4579–4596, Boston, MA, August 2022. USENIX Association.
- [12] Robin Staab, Mark Vero, Mislav Balunović, and Martin Vechev. Beyond memorization: Violating privacy via inference with large language models. *arXiv preprint arXiv:2310.07298*, 2023.
- [13] OpenAI. GPT-4 technical report, 2024.
- [14] Gemini Team. Gemini: A family of highly capable multimodal models, 2024.
- [15] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023.
- [16] Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation, 2023.
- [17] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Rae Ying Yee Wong, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. Specinfer: Accelerating generative llm serving with speculative inference and token tree verification. *arXiv preprint arXiv:2305.09781*, 1(2):4, 2023.
- [18] In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khanelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338, 2024.
- [19] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023.
- [20] Google Research. Looking back at speculative decoding. <https://research.google/blog/looking-back-at-speculative-decoding/>, 2025. Accessed: 2025-01-09.
- [21] OpenAI. Predicted outputs. <https://platform.openai.com/docs/guides/predicted-outputs>.

- [22] Openai uses speculative decoding during inference. <https://x.com/stevendcoffey/status/1853582548225683814>.
- [23] Bogdan Cebere and Christian Rossow. Understanding web fingerprinting with a protocol-centric approach. In *Proceedings of the 27th International Symposium on Research in Attacks, Intrusions and Defenses*, pages 17–34, 2024.
- [24] Wanshuang Lin, Chunhe Xia, Tianbo Wang, Chen Chen, Yuan Zhao, and Weidong Zhou. Reda: Malicious traffic detection based on record length and frequency domain analysis. In *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 668–675, 2023.
- [25] Martin Laštovička, Stanislav Špaček, Petr Velan, and Pavel Čeleda. Using tls fingerprints for os identification in encrypted traffic. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–6, 2020.
- [26] Mathew Hogan, Yan Michalevsky, and Saba Eskandarian. Dbreach: Stealing from databases using compression side channels. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 182–198, 2023.
- [27] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *SOSP*, pages 611–626, 2023.
- [28] Tom B Brown. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.
- [29] Microsoft. Phi-3-medium-4k-instruct. <https://huggingface.co/microsoft/Phi-3-medium-4k-instruct>. Accessed: 2024-08-30.
- [30] Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Ion Stoica, Zhijie Deng, Alvin Cheung, and Hao Zhang. Online speculative decoding. *arXiv preprint arXiv:2310.07177*, 2023.
- [31] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layer skip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*, 2024.
- [32] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.
- [33] Bertalan Meskó. Prompt engineering as an important emerging skill for medical professionals: tutorial. *Journal of medical Internet research*, 25:e50638, 2023.
- [34] Xinli Yu, Zheng Chen, Yuan Ling, Shujing Dong, Zongyi Liu, and Yanbin Lu. Temporal data meets llm-explainable financial time series forecasting. *arXiv preprint arXiv:2306.11025*, 2023.
- [35] OpenAI API, 2020. <https://openai.com/index/openai-api/>.
- [36] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. Website fingerprinting at internet scale. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016.
- [37] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Beauty and the burst: Remote identification of encrypted video streams. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1357–1374, Vancouver, BC, August 2017. USENIX Association.
- [38] Microsoft. Copilot. <https://copilot.microsoft.com>.
- [39] Meta AI. <https://www.meta.ai>.
- [40] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In *Ndss*, volume 20, page 12, 2012.
- [41] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 668–679, 2015.
- [42] Olga Ohrimenko, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Markulf Kohlweiss, and Divya Sharma. Observing and preventing leakage in mapreduce. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1570–1581, 2015.
- [43] M. Bellare, A. Desai, E. Jorjani, and P. Rogaway. A concrete security treatment of symmetric encryption. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 394–403, 1997.
- [44] Leland McInnes, John Healy, Steve Astels, et al. hdbscan: Hierarchical density based clustering. *J. Open Source Softw.*, 2(11):205, 2017.

- [45] vLLM Team. vllm: Easy, fast, and cheap llm serving with pagedattention, 2024. <https://blog.vllm.ai/2023/06/20/vllm.html>.
- [46] Google AI. Gemini api documentation. <https://ai.google.dev/gemini-api/docs/openai>, 2025. Accessed: 2025-01-20.
- [47] Marius-Constantin Popescu, Valentina E Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8(7):579–588, 2009.
- [48] Steven J Rigatti. Random forest. *Journal of Insurance Medicine*, 47(1):31–39, 2017.
- [49] Thomas G Dietterich et al. Ensemble learning. *The handbook of brain theory and neural networks*, 2(1):110–125, 2002.
- [50] Xibin Dong, Zhiwen Yu, Wenming Cao, Yifan Shi, and Qianli Ma. A survey on ensemble learning. *Frontiers of Computer Science*, 14:241–258, 2020.
- [51] Facebook. vllm, 2024. <https://github.com/vllm-project/vllm.git>.
- [52] Facebook. vllm-v0.6.1, 2024. <https://pypi.org/project/vllm/0.6.1.post1/>.
- [53] Kaggle. Nasdaq 100 data. <https://www.kaggle.com/datasets/kalilurrahman/nasdaq100-stock-price-data>. Accessed: 2024-08-30.
- [54] OpenAI developer forum. Why is gpt-4 giving different answers with same prompt & temperature=0?, 2023. <https://community.openai.com/t/why-is-gpt-4-giving-different-answers-with-same-prompt-temperature-0/143513/6>.
- [55] Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.
- [56] Microsoft. Phi-3-mini-4k-instruct. <https://huggingface.co/microsoft/Phi-3-mini-4k-instruct>. Accessed: 2024-08-30.
- [57] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.
- [58] Facebook. Bitsandbytes. <https://github.com/bitsandbytes-foundation/bitsandbytes?tab=readme-ov-file>.
- [59] Arsene Fansi Tchango, Rishab Goel, Zhi Wen, Julien Martel, and Joumana Ghosn. Ddxplus: A new dataset for automatic medical diagnosis, 2022.
- [60] scikit learn. *DecisionTreeClassifier*, 2024. <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>.
- [61] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective attacks and provable defenses for website fingerprinting. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 143–157, San Diego, CA, August 2014. USENIX Association.
- [62] Amir Sabzi, Rut Vora, Swati Goswami, Margo Seltzer, Mathias Lécuyer, and Aastha Mehta. NetShaper: A differentially private network Side-Channel mitigation system. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 3385–3402, Philadelphia, PA, August 2024. USENIX Association.
- [63] David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.
- [64] Andrew M. White, Austin R. Matthews, Kevin Z. Snow, and Fabian Monrose. Phonotactic reconstruction of encrypted voip conversations: Hookt on fon-iks. In *2011 IEEE Symposium on Security and Privacy*, pages 3–18, 2011.
- [65] Ki Suh Lee, Han Wang, and Hakim Weatherspoon. PHY covert channels: Can you see the idles? In *NSDI '14*, pages 173–185, 2014.
- [66] Steven Gianvecchio and Haining Wang. Detecting covert timing channels: an entropy-based approach. In *Proceedings of the 14th ACM Conference on Computer and Communications Security, CCS '07*, page 307–316, 2007.
- [67] Jamie Hayes and George Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 1187–1203, Austin, TX, August 2016. USENIX Association.
- [68] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18*, page 1928–1943, New York, NY, USA, 2018. Association for Computing Machinery.
- [69] Sanjit Bhat, David Lu, Albert Kwon, and Srinivas Devadas. Var-cnn: A data-efficient website fingerprinting attack based on deep learning. *Proceedings on Privacy Enhancing Technologies*, 2019(4):292–310, July 2019.

- [70] Xiaokuan Zhang, Xueqiang Wang, Xiaolong Bai, Yinqian Zhang, and Xiaofeng Wang. OS-level side channels without procs: Exploring cross-app information leakage on iOS. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018.
- [71] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 148–179, 2019.
- [72] Annelie Heuser and Michael Zohner. Intelligent machine homicide - breaking cryptographic devices using support vector machines. In Werner Schindler and Sorin A. Huss, editors, *Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings*, volume 7275 of *Lecture Notes in Computer Science*, pages 249–264. Springer, 2012.
- [73] Neil Zhenqiang Gong and Bin Liu. You are who you know and how you behave: Attribute inference attacks via users’ social friends and behaviors. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 979–995, Austin, TX, August 2016. USENIX Association.
- [74] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2017.
- [75] Shuyu Jiang, Xingshu Chen, and Rui Tang. Prompt packer: Deceiving llms through compositional instruction with hidden attacks. *arXiv preprint arXiv:2310.10077*, 2023.
- [76] Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. Jailbroken: How does llm safety training fail? *Advances in Neural Information Processing Systems*, 36, 2024.
- [77] Haoran Li, Dadi Guo, Wei Fan, Mingshi Xu, Jie Huang, Fanpu Meng, and Yangqiu Song. Multi-step jail-breaking privacy attacks on chatgpt. *arXiv preprint arXiv:2304.05197*, 2023.
- [78] Edoardo DeBenedetti, Giorgio Severi, Nicholas Carlini, Christopher A. Choquette-Choo, Matthew Jagielski, Milad Nasr, Eric Wallace, and Florian Tramèr. Privacy side channels in machine learning systems. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 6861–6848, Philadelphia, PA, August 2024. USENIX Association.
- [79] OpenAI. Prompt caching guide. <https://platform.openai.com/docs/guides/prompt-caching>. Accessed: 2025-01-16.
- [80] Anthropic. Prompt caching with claude. <https://www.anthropic.com/news/prompt-caching>. Accessed: 2025-01-16.
- [81] Xinyao Zheng, Husheng Han, Shangyi Shi, Qiyang Fang, Zidong Du, Qi Guo, and Xing Hu. Inputsntatch: Stealing input in llm services via timing side-channel attacks. *arXiv preprint arXiv:2411.18191*, 2024.
- [82] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Proceeding 2000 IEEE symposium on security and privacy. S&P 2000*, pages 44–55. IEEE, 2000.
- [83] Neeraj Varshney, Agneet Chatterjee, Mihir Parmar, and Chitta Baral. Accelerating llm inference by enabling intermediate layer decoding. *arXiv preprint arXiv:2310.18581*, 2023.
- [84] Sangmin Bae, Jongwoo Ko, Hwanjun Song, and Se-Young Yun. Fast and robust early-exiting framework for autoregressive language models with synchronized parallel decoding. In *EMNLP*, pages 5910–5924. Association for Computational Linguistics, 2023.
- [85] Yanxi Chen, Xuchen Pan, Yaliang Li, Bolin Ding, and Jingren Zhou. EE-LLM: large-scale training and inference of early-exit large language models with 3d parallelism. In *ICML*. OpenReview.net, 2024.
- [86] In Gim, Seung-seob Lee, and Lin Zhong. Asynchronous llm function calling. *arXiv preprint arXiv:2412.07017*, 2024.
- [87] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. 2024.
- [88] Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. Toolace: Winning the points of LLM function calling. *CoRR*, abs/2409.00920, 2024.
- [89] OpenAI. Function calling guide, 2025.
- [90] Model Context Protocol. Introduction to the model context protocol, 2025.

Appendix

A Prompt Syntax for Evaluated Use Cases

In §2, we introduced ChatCAD [5, 6] and AWS stock forecasting [34] as privacy-sensitive use cases of interactive LLM services. This section details the exact structure of the prompts for these use cases, which the adversary uses to curate prompts in §7 and §6.

Medical prompts. Each medical prompt describes a patient randomly sampled (without replacement) from the DDx-Plus [59] dataset. The input prompt begins by describing the patient’s age and gender (e.g., “Patient is a 21-year-old Male”) and continues with a list of the patient’s initial survey based on the disease-related questionnaire they filled out. The survey is formatted as a concatenation of questions and answers (e.g., “Characterize your pain: a cramp.”). At the end of the survey section, we include the universe of all possible diagnoses in the DDxPlus dataset (“Given the list of possible pathologies: URTI, HIV, ...”), and ask the LLM to predict the diagnosis from the provided list (“what is the one most likely differential diagnosis?”). We do not control the syntax of the LLM’s medical output and extract the predicted diagnosis using text processing techniques.

An example of the medical attack prompt

- **User prompt:** Patient is a 21-year-old M. Initial evidence: Have you recently had stools that were black (like coal)? Additional symptoms: Characterize your pain: a cramp; Do you feel pain somewhere? ... Given the list of possible pathologies: URTI, HIV (initial infection), ..., Ebola, Bronchiolitis, what is the one most likely differential diagnosis?
- **Output of LLM:** The most likely differential diagnosis for this patient is anemia. The patient’s symptoms of fatigue, dizziness, and black stools (indicative of gastrointestinal bleeding) align with this condition. The absence of fever, cough, or recent travel reduces the likelihood of infections like URTI, pneumonia, or tuberculosis. The lack of chest pain or shortness of breath makes cardiac issues like atrial fibrillation, unstable angina, or myocarditis less likely.

Financial prompts. We sample symbol, date, and stock history without replacement from the Nasdaq-100 dataset [53]. Each financial prompt begins with the field “Date of prediction: {date}” to provide the LLM with time context for the prediction. The prompt continues with an **Instruction** field, specifying the task for the LLM: predicting the next day’s stock return for a specific symbol by analyzing

the company’s profile, a list of positive and negative factors affecting the stock price, recent returns, and a piece of news about the company. This section also describes the trend categorization scheme across bins ranging from D5+ to U5+ for the LLM. Each trend category indicates an increase (U) or decrease (D) in the stock price by a range between x and $x+1\%$, while D5+ and U5+ indicate more than 5% decrease and more than 5% increase in the price, respectively.

The next field in the input prompt is the symbol’s profile, detailing the company and its positive and negative factors. These descriptions are generated using the GPT-4o API. A list of 15 days of stock return history is then provided as a comma-separated list under the “Recent Returns” field. The prompt concludes with a news item formatted as “Recent {news-quality} News: {news}”. Each news is sampled from a set of 144 news items collected for each symbol, categorized into 12 different news-quality categories. A sample of our curated prompts follows.

An example of the financial attack prompt.

- **User prompt:** Date of prediction: 2016-11-02
Instruction: Forecast next day stock return (price change) for ALXN, given the company profile, and the company’s stock returns for the last 15 days. The trend is represented by bins "D5+", "D5", "D4", "D3", "D2", "D1", "U1", "U2", "U3", "U4", "U5", "U5+", where "D5+" means price dropping more than 0.5%, ...
Company Profile: Description: ALXN is a biopharmaceutical company specializing in developing and commercializing treatments for rare diseases.
Positive Factors: ...
Negative factors: ...
Recent Returns: D1, D1, D2, D2, U3, U1, D2, D1, U1, D1, D2, U4, U1, U2, D3
Recent Neutral news: ALXN receives conditional approval for its new product, pending further medical data.
- **Output of LLM: Stock return: U2**
Rationale: ALXN’s recent returns show a pattern of minor fluctuations with a slight upward trend, indicating resilience in the face of negative factors. The company’s strong R&D pipeline and successful medical trials suggest potential for positive developments. Although there are concerns about competition and regulatory hurdles, the growing demand for rare disease treatments and potential M&A activity could drive the stock up. The recent neutral news about Adobe is unlikely to impact ALXN’s performance directly.