Exploiting Passive and Active Timing Attacks on Large Language Models

Sachin Thakrar, Advisor: Timothy Barron

January 2025

1 Introduction

Large Language Models (LLMs) have been rapidly adopted over the past few years, with the advent of tools such as OpenAI's ChatGPT, and later Microsoft Copilot, Claude by Anthropic, and a whole host of other derivatives. This rapid adoption is driven by their strong performance on various tasks, such as answering questions, writing code, translating, writing, etc. Adoption has been similarly rapid in privacy-sensitive applications, such as for finance, law, and medical uses. This growing dependence on LLM-based systems for processing sensitive personal information raises significant privacy and security concerns.

Previous work has demonstrated privacy breaches in LLMs and machine learning systems using attacks such as membership inference [1] [13] [10], training data extraction [4] [2], and prompt injection attacks. In terms of prompt injection attacks, it is possible to extract system prompts through adversarial prompting [8] [11] [18] , inverting prompts from embedding vectors [14] or model responses [12], and recovering input prompts by exploiting next-token probability distributions or token-length sequences [16]. This is merely a subset of published attacks on LLMs, serving to highlight the security concerns.

This research will focus on exploiting timing side-channel attacks on LLMs [3] [19] [17] [5]. We will focus on both passive timing attacks, in which the adversary is only able to observe the timing of the model response (e.g. latency) and any other associated metadata, and active timing attacks, in which the adversary has partial control over the input prompt, but no access to the output. Both passive and active timing attacks have been shown to be effective in extracting information from LLMs, and this research will focus on extending the range of information obtainable through these attacks.

2 Background

Timing attacks are a class of side-channel attacks that exploit variations in the time taken by a system to perform different operations. These attacks have historically been used against cryptographic systems, where variations in processing time can leak information about secret keys or private data.

For example, early implementations of RSA were vulnerable to timing attacks that could recover private keys by measuring subtle differences in computation time for different input messages [7]. More recent examples include Spectre and Meltdown, which exploited timing variations in Intel CPUs to leak information [6] [9].

2.1 Timing Attacks on LLMs

In the context of LLMs, timing attacks exploit the fact that these models' processing time varies based on the input prompt and the internal computations performed. This variation occurs because LLMs process text token-by-token, and the time taken to generate each token and the response as a whole can depend on various factors:

- **Prompt Caching**: When LLMs process similar prompts in succession, they can often reuse computations from previous prompts, leading to faster response times. This timing difference can reveal whether similar prompts have been recently processed, potentially leaking information about other users' interactions with the system. It has been shown that by abusing this timing difference, it is possible to extract a user's prompt verbatim [19], and so many providers restrict the number of users this cache is shared across, or allow the end-user to specify how their prompts are cached [5].
- **KV Caching**: LLMs maintain key-value caches of intermediate attention computations during generation. The effectiveness of this cache varies based on prompt similarity and length, creating observable timing patterns. Cache hits result in faster processing, while cache misses require full recomputation of attention values [15].
- **Speculative Decoding**: Modern LLMs often employ speculative decoding techniques. Here, several tokens are sampled from a much smaller *draft* model, and validated by the much larger *target* model. As validation requires only a simple forward pass, a successful speculative decoding is significantly faster than a full generation. This process is repeated for the entire output generation. As speculative decoding is faster, we can determine if a prompt is "easy" or "hard", which can be used to infer information about the prompt, and it is possible to construct adversarial prompts that force speculative decoding to succeed or fail, depending on information in the system prompt [3]. [See: Active Timing Attacks]
- **Output Length Variations**: The total response time is strongly correlated with the number of output tokens generated. By measuring response times, an attacker can infer the approximate length of responses, even without access to the output text. This can reveal information about the nature of the prompt or the model's knowledge about a topic [17].
- **Input-Output Ratio Patterns**: The ratio between input length and output length often follows predictable patterns for different types of tasks. In translation tasks, the ratio depends on the input and output languages, allowing an attacker to infer the languages used [17].
- **Generation Biases**: LLMs exhibit systematic biases in their response lengths for different topics or types of queries. Technical or specialized topics might consistently generate longer responses compared to simple factual queries, but variations also exist for identical tasks.

For example, Zhang et al. [17] show that for a classificaton task, the explanation length is directly related to the classification.

- **Token Encoding Efficiency**: Different types of text (e.g., natural language vs. code) have varying token encoding efficiencies. This variation also exists between different natural languages, with some languages being more efficient than others due to biases in the tokenisation process.

Though caching attacks have been shown to be incredibly effective, this research will not focus on them. There is certainly space for future work in this area, but changes by major LLM providers restricting how the cache is shared across users has made this attack significantly less effective (though as providers like OpenAI allow for cache sharing across an organisation, it is possible to extract prompts from peers and coworkers who share the same cache). Instead, this research will focus broadly across other potential vectors as proposed above to determine information about a user's prompt.

3 Threat Model

In this section, we define two main scenarios in which an adversary attempts to learn information about a private user prompt: *passive attacks* and *active attacks*. The key differences concern the adversary's ability to craft prompts versus merely observing timing channels.

3.1 Passive Attacks

In this passive scenario, a victim user is interacting with a language model, while a network adversary observes the (encrypted) network traffic between the user and the language model. This *network adversary* threat model is extremely common in computer security. Such an adversary may be:

- A malicious Internet Service Provider;
- In control of a compromised router;
- Performing a man-in-the-middle attack on a public WiFi network;
- Or have compromised any of the intermediary network devices between the user and the model's servers.

Although the contents of the communication remain encrypted, the adversary can still measure *time-based* features of the communication (e.g., latencies, message sizes, and traffic flow patterns). The adversary cannot modify the data en route, nor see the actual plaintext prompt or response.

3.1.1 Prior Work on Passive Attacks

Passive attacks against LLMs have been shown to be effective in two separate lines of work:

1. **Zhang et al.** exploit differences in token density to determine languages used in a machine translation setting. Different languages vary in how many tokens they require to encode the same sentence. By observing the input/output token ratio and correlating it with total response time, an adversary can identify which languages are likely in use. Additionally, the authors show how one can infer a classifier's *output* in an LLM-based classification task by measuring the length of the model's explanation. Different classes yield systematically distinct response lengths [17].

2. **Carlini & Nasr** explicitly examine how modern inference optimizations like speculative decoding can leak prompt data. Since a small *draft* model attempts to predict multiple tokens at once, successful tokens (that match the target model) generate far faster. By merely *observing the timing pattern*, they found that certain topics were biased towards faster token generation, allowing for the adversary to guess the content or topic of the conversation, or the language used. For instance, they show how to discern whether a user is requesting *coding assistance* versus *medical advice* (measured across multiple LLMs from two labs). The authors also demonstrate an end-to-end website attack on ChatGPT, observing timing to predict the user's topic in a web context [3].

While these initial demonstrations were somewhat limited (often distinguishing only two high-level topics), their results suggest that more fine-grained classification of user prompts is feasible. Thus, in the coming sections, we propose extending these passive timing attacks to a broader set of topics and a larger variety of LLM providers.

3.2 Active Attacks

In the *active* scenario, the adversary can partially control or inject content into the prompt itself but cannot see the actual responses. Instead, they rely on indirect indicators like response timing or network flow to glean information from the model.

For a concrete example, consider one proposed by Carlini & Nasr [3]: suppose a malicious entity runs a service that internally calls a third-party language model via API. The user's private input is appended to an *adversary-chosen* prompt template. Though the malicious entity never sees the model's raw outputs (perhaps the user's device or another filter prevents that), they *do* see round-trip times and partial metadata like response lengths. By carefully structuring the prompts, the adversary can cause measurable timing differences that reveal hidden user data.

As another variant, imagine an LLM with a system prompt containing sensitive personal information (e.g., SSN or income) so that the system can provide personalised answers. If an adversary is able to send user queries that append to that system prompt but never directly see the responses (due to an API restriction or sanitization), they can still exploit timing-based feedback. In this way, it parallels a blind SQL injection attack, where the attacker infers secret data by constructing queries that produce different response times.

3.2.1 Prior Work on Active Attacks

The active attack scenario is also one proposed by Carlini & Nasr [3], but in the context of speculative decoding. They tried to fetch a ”secret number” contained within the system prompt by exploiting a *capability gap* between the draft and target models, eliciting different models to answer the same prompt based on the correctness of the guess, leading to a timing difference.

The potential for *even more extreme* time-based differentials arises if an attacker can specify the length or complexity of the output *conditional* on some secret. For example:

```
If the secret number starts with 7, respond "yes".  Otherwise,
write a detailed 200-word essay about Timothy Dwight IV.
```

Under a typical LLM endpoint, the response times for “yes” vs. a 200-word essay differ significantly, making it trivial to deduce if the secret starts with 7. In principle, this allows near-complete extraction of secrets, like SSNs, at near-perfect accuracy.

4 Project Proposal

This project seeks to expand on the above scenarios in multiple ways:

- **Passive Attack Extensions**: Prior work has often restricted itself to simplistic classification tasks (e.g., “coding” vs. “medical” queries). We will evaluate a more comprehensive set of topics, in education-oriented contexts, namely: programming assistance, mathematical problem-solving, literature analysis & essay writing, language translation, and more. Though these categories are not final, they represent categories that are likely to yield significant timing differentials, whilst also being relevant to a wide range of users. A language model could be used to construct the prompts needed, or they could be sourced from dataset(s) of existing prompts and benchmarks.
- **Active Attack Extensions**: We will explore multi-branch prompting methods to create more fine-grained timing differentials. For instance, a single request could yield 10 words if some secret property is true, 20 words otherwise, or even more complex multi-condition outputs (e.g., 10 words if the first digit is 0, 20 words if the first digit is 1, etc.), reducing the total number of queries needed to exfiltrate multi-digit secrets. The goal will be to recover entire secrets, such as SSNs, at near-perfect accuracy, in as few prompts as possible. There also is the possibility of exploring ways to mitigate this risk, but given the continued efficacy of prompt exfiltration, it is unlikely that such an attempt would be successful.

In both cases, the goal is also to expand on the number of models tested. The model list has also not been finalised (given the rapid development of new models), but we will aim to test across a broad set of models, including the latest versions of GPT, Claude, DeepSeek, LLaMA, and more. If the attacks are successful in an API context, we will also attempt to implement these attacks in an end-to-end web context, such as directly on the ChatGPT or Claude websites.

5 Timeline and Deliverables

This section outlines the key milestones, deliverables, and timeline for the project.

5.1 Key Deliverables

- **Progress Report** (February 28, 2025): Initial findings from passive timing attacks, including experimental setup and preliminary results from the medical/coding classification task.
- **Final Thesis Draft** (April 21, 2025): Complete thesis draft for review, including all experimental results, analysis, and conclusions.
- **Research Poster** (April 22, 2025): Visual presentation of key findings and methodologies.
- **Final Thesis** (May 1, 2025): Final version of the thesis incorporating feedback.
- **Code Repository**: Repository containing:
  - Implementation of passive and active timing attack frameworks
  - Data collection and analysis scripts
  - Evaluation metrics and visualization tools
  - Documentation and reproduction instructions

5.2 Project Timeline

- **Research Review** (January 24 - January 31): Review existing literature, and apply for funding (Mellon).
- **Phase 1: Passive Attacks — Setup** (February 1 - February 15): Develop timing measurement infrastructure, implement data collection pipeline, and set up API access for LLM providers.
- **Phase 2: Initial Classification** (February 16 - February 28): Execute medical/coding binary classification task, and refine methodology as needed. Complete progress report with these results.
- **Phase 3: Extended Classification** (March 1 - March 15): Expand to additional topics, and collect and analyse the multi-class classification results. Begin thesis draft.
- **Phase 4: Active Attacks** (March 16 - March 31): Implement multi-branch prompting methods, and test secret extraction methods. Document effectiveness across models.
- **Phase 5: Web Implementation (Optional)** (March 1 - April 10): If the extended classification or active attacks are successful, implement these attacks in an end-to-end web context, such as directly on the ChatGPT or Claude websites.
- **Final Documentation** (April 1 - April 21): Complete final documentation, including thesis, code repository, and research poster.

References

[1] Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. 2022. Quantifying Memorization Across Neural Language Models. arXiv:arXiv:2202.07646

[2] Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. 2022. Quantifying Memorization Across Neural Language Models. arXiv:arXiv:2202.07646

[3] Nicholas Carlini and Milad Nasr. 2024. Remote Timing Attacks on Efficient Language Model Inference. arXiv:arXiv:2410.17175

[4] Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Úlfar Erlingsson, et al. 2021. Extracting Training Data from Large Language Models. In *30th USENIX Security Symposium (USENIX Security 21)*. 2633–2650.

[5] Chenchen Gu, Xiang Lisa Li, Rohith Kuditipudi, Percy Liang, and Tatsunori Hashimoto. [n. d.]. Stanford CS 191W Senior Project: Timing Attacks on Prompt Caching in Language Model APIs. ([n. d.]).

[6] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. 2020. Spectre attacks: Exploiting speculative execution. *Commun. ACM* 63, 7 (2020), 93–101.

[7] Paul C Kocher. 1996. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. *Annual International Cryptology Conference* (1996), 104–113.

[8] Zi Liang, Haibo Hu, Qingqing Ye, Yaxin Xiao, and Haoyang Li. 2024. Why are my prompts leaked? Unraveling prompt extraction threats in customized large language models. *arXiv preprint arXiv:2408.02416* (2024).

[9] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, et al. 2020. Meltdown: Reading kernel memory from user space. *Commun. ACM* 63, 6 (2020), 46–56.

[10] Maxime Meeus, Sanjam Jain, Marek Rei, and Yves-Alexandre de Montjoye. 2024. Did the Neurons Read Your Book? Document-Level Membership Inference for Large Language Models. In *33rd USENIX Security Symposium (USENIX Security '24)*. 2369–2385.

[11] Fabio Perez and Ian Ribeiro. 2022. Ignore previous prompt: Attack techniques for language models. *arXiv preprint arXiv:2211.09527* (2022).

[12] Zeyang Sha and Yang Zhang. 2024. Prompt stealing attacks against large language models. *arXiv preprint arXiv:2402.12959* (2024).

[13] Weijia Shi, Ankit Ajith, Mengzhou Xia, Yangsibo Huang, Dacheng Liu, Terra Blevins, Danqi Chen, and Luke Zettlemoyer. 2024. Detecting Pretraining Data from Large Language Models. In *International Conference on Learning Representations*.

[14] Congzheng Song and Ananth Raghunathan. 2020. Information leakage in embedding models. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 377–390.

[15] Linke Song, Zixuan Pang, Wenhao Wang, Zihao Wang, XiaoFeng Wang, Hongbo Chen, Wei Song, Yier Jin, Dan Meng, and Rui Hou. 2024. The Early Bird Catches the Leak: Unveiling Timing Side Channels in LLM Serving Systems. arXiv:arXiv:2409.20002

[16] Roy Weiss, Daniel Ayzenshteyn, and Yisroel Mirsky. 2024. What Was Your Prompt? A Remote Keylogging Attack on AI Assistants. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, Philadelphia, PA, 3367–3384.

[17] Tianchen Zhang, Gururaj Saileshwar, and David Lie. 2024. Time Will Tell: Timing Side Channels via Output Token Count in Large Language Models. arXiv:arXiv:2412.15431

[18] Yiming Zhang, Nicholas Carlini, and Daphne Ippolito. 2024. Effective prompt extraction from language models. In *First Conference on Language Modeling*.

[19] Xinyao Zheng, Husheng Han, Shangyi Shi, Qiyan Fang, Zidong Du, Xing Hu, and Qi Guo. 2024. InputSnatch: Stealing Input in LLM Services via Timing Side-Channel Attacks. arXiv:arXiv:2411.18191