Data Leaks Through Speculative Decoding: Network Side-Channel Attacks on LLM Inference

Sachin Thakrar
sachin.thakrar@yale.edu

Advisor: Timothy Barron
timothy.barron@yale.edu

A Senior Thesis as a partial fulfillment of requirements for the Bachelor of Science in Computer Science

Department of Computer Science
Yale University
May 1, 2025

Acknowledgements

I want to express my gratitude to Professor Timothy Barron, whom I am indebted to for his support, advice, and guidance throughout the past few months. It was his course, Computer Systems Security, that led me to pursue this thesis. I also want to thank the Computer Science department at Yale for allowing me to realise my passion and interest in the field, and I am grateful for the opportunities and resources provided.

Although this project did not receive funding, complimentary credits were provided by OpenAI and Google Cloud Platform for their respective models and utilised throughout the research, training and testing process.

Contents

Data Leaks Through Speculative Decoding: Network Side-Channel Attacks on LLM Inference

Sachin Thakrar

Abstract

Large language models (LLMs) are now used to serve billions of streamed queries daily. To reduce compute costs and improve response speed, inference providers employ a variety of optimization techniques. One such method is speculative decoding, wherein a lightweight draft model proposes several future tokens that the original model subsequently verifies. This optimization, however, inadvertently introduces timing side-channels, as accepted or rejected draft tokens cause observable differences in packet timings and sizes, which can be utilized to infer sensitive attributes of a user's prompt.

Under a purely passive, black-box threat model—without direct interaction or special access to model infrastructure—we evaluate these side-channels across several realistic scenarios, including medical diagnosis classification, user language identification, and educational topic inference, each with ten distinct target classes. We utilize three popular, commercially deployed LLMs: GPT-4o-mini, DeepSeek V3 (0324), and Llama 3.3 70B. Collecting up to 12,000 training and 3,000 test traces per scenario, we extract comprehensive timing, packet-size, and speculative-round statistics, then train a stacked MLP/XGBoost ensemble classifier. Our model achieves top-1 accuracy of 62-72% on language and medical classification tasks (vs. 10% random) and 49-54% on educational topic classification, with top-3 accuracy up to 92%. These results demonstrate a practical side-channel attack capable of reliably inferring user intent from encrypted streams.

We discuss potential mitigations and outline avenues for future work, including exploring attacks on multi-token JSON wrappers, testing robustness across diverse network conditions and provider implementations, and developing draft models resistant to speculative decoding attacks. Our findings highlight significant vulnerabilities in current inference-serving practices used by major LLM providers.

1 Introduction

Large language models (LLMs), such as OpenAI's GPT-4 and Anthropic's Claude, have rapidly gained prominence due to their impressive natural language processing capabilities and diverse practical applications [1, 6, 23]. To achieve this level of performance, LLMs have grown dramatically in scale: the largest publicly-known models now exceed two trillion parameters [20]. This increase in model size enables higher-quality generation and better generalization, but comes at the cost of significantly increased inference-time computation. Serving such large models is expensive, in terms of both inference-time compute and latency.

To reduce these costs and improve responsiveness, inference providers apply a range of performance optimizations. These include quantization, which reduces model size and inference cost by representing weights with lower-precision formats [7]; prompt caching, which reduces the computation needed for repeated queries [11]; and speculative decoding, which improves token-by-token generation speeds [5]. Additionally, LLM inference providers often stream the response to the user, allowing the user to see the response as it generates, improving user-perceived latency and throughput.

We focus on *speculative decoding*, a technique used by many LLM providers in which a smaller *draft* model is used to generate tokens which are then verified by the *original* model [16, 24]. This improves token generation speeds, but introduces timing differences between tokens generated by the draft model and the original model. When outputs are streamed, these differences manifest in packet timings and sizes. An adversary monitoring encrypted traffic—without decrypting its contents—can exploit these variations to infer fine-grained attributes of a user's query, such as its topic, language, or category [5, 25, 37].

In this thesis, we investigate how speculative decoding and related inference-time optimizations introduce privacy vulnerabilities in real-world LLM deployments. We consider a realistic, passive threat model in which the adversary observes only encrypted network traffic and has no access to internal model behavior or plaintext outputs. We also assume the adversary has black-box access to the target API of the LLM in question and can send an arbitrary number of queries. Our primary contribution is a large-scale empirical study demonstrating that timing-based side channels can leak prompt-level information across a range of LLM APIs and tasks.

We collect timing traces from state-of-the-art LLMs, including Llama 3.3 70B Instruct, DeepSeek V3, and GPT-4o-mini. Using these traces, we implement multiple classification attacks designed to recover private attributes—such as medical diagnoses, the user's primary language, and the topic being discussed—from individual prompts. In each scenario, we distinguish between ten target categories, and collect between 10,000 and 12,000 training traces and 2,000 to 3,000 test traces. We train a stacked MLP/XG-Boost classifier. Across all LLM APIs and scenarios, we achieve single-prompt top-1 classification accuracy between 50% and 70% and top-3 accuracy between 80% and 90%.

Our findings demonstrate that speculative decoding and related inference optimizations can create high-fidelity timing side-channels, even under encrypted, black-box conditions. We also discuss differences between our observations and existing work, highlighting how deployment changes by major LLM providers (Google, Anthropic) have reduced their vulnerability to these attacks. Finally, we propose potential defenses that preserve the efficiency and interactivity benefits of streaming whilst reducing privacy leakage, and discuss avenues for future work. These results underscore the broader need to treat inference-time performance techniques as potential attack surfaces.

2 Background

This section provides the technical context required to understand timing side-channel attacks on LLM inference. We begin by reviewing classical timing attacks, and the underlying transformer architecture. Then,we discuss the performance optimizations commonly deployed by commercial providers, and then, discuss the network-level timing side-channel created by *speculative decoding*.

2.1 Timing Attacks and Side-Channels

A *timing attack* is a form of side-channel in which an adversary extracts sensitive information by measuring the time it takes a system to respond to specific inputs. If the response time of an application depends on secret data, an attacker may infer this secret simply by measuring how long the computation takes, without needing direct access to internal computations or data.

Timing attacks were first introduced by Kocher in 1996, who demonstrated that minor differences in the time taken by cryptographic operations (like RSA encryption) could leak secret keys [15]. Brumley & Boneh later extended this idea across the internet, extracting OpenSSL private keys by correlating round-trip times with carefully crafted inputs [3].

Subsequent research revealed that timing channels can also expose *content-dependent* secrets in web applications. Felten & Schneider, for instance, measured the latency of loading resources to infer a user's browsing history, demonstrating the first web timing attack [10]. Later, Bortz & Boneh showed that response time differences for dynamically generated pages leak whether a victim is logged in and other confidential state [2]. Recent research showed that emerging browser features—like the Performance API, SharedArrayBuffers, and the Server-Timing HTTP header—enable an adversary to recover information whilst significantly reducing noise [28, 29].

As we detail in §2.3, some of the LLM optimizations discussed, including *speculative decoding*, directly impact computation timings for each token generated. A passive adversary monitoring this traffic thus obtains indirect clues about the internal state and decisions of the model, forming a viable timing side-channel.

2.2 Transformers

Modern LLMs such as GPT-4 and Claude are built on the *transformer* architecture, a neural network design that has become the standard for natural language processing tasks [30]. The key innovation behind transformers is the *self-attention* mechanism, which allows the model to weigh and combine information from all parts of an input sequence during training, regardless of their position.

LLMs employ transformers in an *autoregressive* setup when generating text. In this setting, the model predicts text one token at a time, with each prediction based on all previously generated tokens. For example, if the model has already generated the tokens `"The cat sat"`, it will predict the next token (e.g., `"on"`) by computing the probability distribution of all possible next tokens given that context. Formally, given a sequence of tokens $x_1, x_2, \ldots, x_{t-1}$, the model estimates the probability distribution of the next token x_t as $P(x_t \mid x_1, x_2, \ldots, x_{t-1})$, from which it samples to generate the next token.

This generation process is inherently sequential: even though transformers can process input in parallel during training, they must generate output one token at a time during inference. As a result, generating long outputs with large, multi-layer language models can be computationally expensive and slow.

2.3 LLM Optimizations

2.3.1 Model-Level Optimizations

These techniques alter the structure or internal representation of the model to reduce the cost of each forward pass. A common example is *quantization*, where full-precision weights are replaced with lower-precision formats (e.g., 8-bit or 4-bit), significantly reducing memory usage and accelerating inference on compatible hardware [7], typically with only a modest drop in performance. Post-training quantization (PTQ) is frequently employed by open-source LLM providers to lower serving costs. Increasingly, pre-quantized models are released directly by research organizations using Quantization-Aware Training (QAT), which allows the model to adapt to reduced precision during training and thereby minimizes degradation [34].

Notably, Egashira et al. [8] demonstrated that quantization can be weaponized to construct models that exhibit adversarial behavior *only* when quantized. The attack involves fine-tuning a model to embed malicious behavior, such as insecure code generation, then using projected gradient descent to adjust the full-precision weights so that the malicious behavior is suppressed in full precision but re-emerges upon quantization. However, such an attack requires (re)training a model, and so is excluded from our threat model. More broadly, model-level optimizations, like quantization, distillation, and pruning, do not create observable timing side-channels in isolation, and so they are beyond the scope of this thesis.

2.3.2 Caching

Caching mechanisms avoid redundant computation across multiple requests. These fall under two main categories:

- **Prompt Caching**: When a client issues a query with a prefix that has already been seen, the model can reuse the cached hidden states for that portion of the input. This timing difference can reveal whether similar prompts have recently been 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 [37].
- **KV Cache Reuse (Key-Value Caching)**: During autoregressive generation, transformers produce and store intermediate key/value vectors for each token. By caching these, the model avoids recomputing attention scores for past tokens at every step, but this results in timing differences [26]. Similarly, the KV cache can be utilized to extract a user's prompt verbatim [32].

In addition to enabling prompt exfiltration, caching can also leak information about the underlying model architecture, as caching is only possible with decoder-only transformer models. Gu et al. determined that OpenAI's `text-embedding-3-small` utilized prompt caching, so it must be a decoder-only model, unlike many open-source embedding models, which are encoder-only [11].

In the same study, Gu et al. audited 17 API providers and found that 7 deployed global prompt caches, meaning that cache hits could occur across unrelated users. After responsibly disclosing these findings, at least 5 of the affected providers eliminated global caching and clarified their documentation. As a result, caching today is almost always implemented on a per-user or per-organization basis, and most providers allow users to opt in or out of prompt caching. Because effective, low-cost mitigations have already been broadly deployed, and because caching is no longer widely shared across users, we do not consider caching a viable target for our attacks.

2.3.3 Speculative Decoding

While caching re-uses past computation, *speculative decoding* aims to accelerate the inherently sequential token generation process of large autoregressive models. It achieves this by using a smaller, faster *draft* model to predict multiple future tokens, which are then efficiently verified by the original large *target* model. This approach can significantly reduce the average per-token generation latency without sacrificing the output quality of the original model [16]. The process is as follows:

- **Drafting:** Given the current context P (the prompt + previously generated tokens), the lightweight *draft* model proposes a sequence of k candidate tokens, denoted as $d_1, d_2, \ldots, d_k$. While early methods generated these autoregressively, state-of-the-art techniques like EAGLE-2 can generate multiple draft candidates in parallel, increasing efficiency [17].

- **Target Model Forward Pass:** The core idea is to verify the draft tokens using the target model efficiently. To do this, the target model takes the original context P concatenated with the *entire sequence* of k draft tokens $(P, d_1, d_2, \ldots, d_k)$ as a single input. It performs one forward pass on this combined sequence, to obtain $P_{target}(token|P, d_1, \ldots, d_k)$—the probability distribution for the next token, given the original prompt and *every* draft token. Due to the parallel nature of the transformer architecture, this single pass computes the next-token probability distributions for *every* position in the input sequence simultaneously. This yields the set of distributions needed for verification and potential final sampling:

  - The distribution $P_{target}(token|P)$, used to verify the first draft token d_1.
  - The distribution $P_{target}(token|P, d_1)$, used to verify d_2.
  - $\ldots$
  - The distribution $P_{target}(token|P, d_1, \ldots, d_{k-1})$, used to verify the final draft token d_k.
  - The distribution $P_{target}(token|P, d_1, \ldots, d_k)$, used to sample the $(k+1)$-th token if all k draft tokens are accepted.

- **Token Acceptance:** The proposed draft tokens are then checked against the target model's computed probabilities. The first draft token d_1 is compared to $P_{target}(token|P)$, and if it meets some acceptance criterion (e.g. it is the most likely token, or passes some stochastic acceptance test [16]), the token is accepted. We check d_2 against $P_{target}(token|P, d_1)$, and so on.

  If any token d_i is rejected, the verification stops. Tokens $d_1, \ldots, d_{i-1}$ are kept, but d_i and all subsequent draft tokens $(d_{i+1}, \ldots, d_k)$ are discarded. The target model then generates the correct i-th token (t_i) by sampling the already computed distribution $P_{target}(token|P, d_1, \ldots, d_{i-1})$, which can be done with negligible cost.

- **Final Token Sampling (Best Case):** If all k draft tokens $(d_1, \ldots, d_k)$ are accepted through the sequential check, the target model has *already computed* the necessary probability distribution $P_{target}(token|P, d_1, \ldots, d_k)$ during the forward pass (Step 2). We then sample the final $(k+1)$-th token (t_{k+1}).

The significant performance gain of speculative decoding arises directly from this mechanism. In the ideal case (all k drafts accepted), the system generates $k+1$ tokens (the k accepted drafts plus the final target-sampled token t_{k+1}) for roughly the computational cost of a single target model forward pass, plus the relatively small cost of the draft model's generation. This process is conceptually illustrated in Figure 2.1.

We will later show that when responses are streamed to the user, an adversary can infer which tokens were part of a single speculative decoding round through a *timing side-channel*, as tokens from a single round will be streamed in quick succession, with larger gaps in between. Further, we can distinguish between tokens generated by the draft

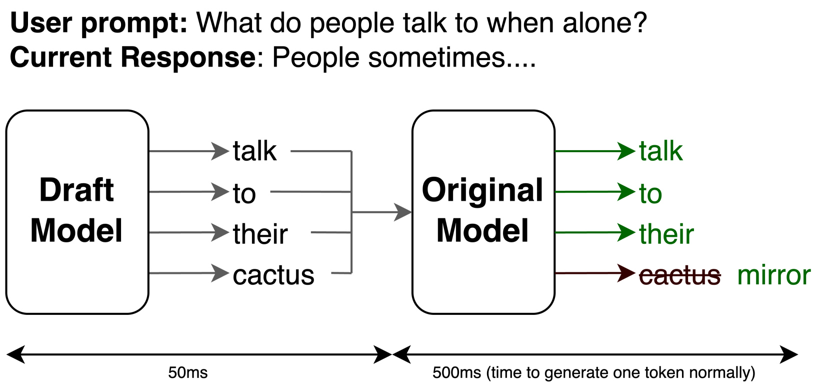

User prompt: What do people talk to when alone?
Current Response: People sometimes....
Draft Model
talk
to
their
cactus
Original Model
talk
to
their
~~cactus~~ mirror
50ms
500ms (time to generate one token normally)

Figure 2.1: 4 draft tokens are proposed by a lightweight model and verified by the target model. 3 of the 4 tokens are accepted, and we generate the 4th with negligible cost by sampling the generated distribution.

and target models, as we know that *all tokens but the last* in a single speculative decoding round will have been generated by the draft model.

2.3.4 Alternative Conditional-Computation Approaches

Other techniques also adapt the amount of compute at inference time, including:

- **Early Exit**: Allowing shallower layers to v̈oteön a prediction and terminating the forward pass when confidence is high [9].
- **Dynamic Attention**: Pruning or sparsifying attention heads on-the-fly based on token importance.
- **Speculative Inference**: A tree-based variant of speculative decoding, similar to that utilized by EAGLE for draft token prediction [21].

These approaches share the same goal—reduce average inference cost—but differ in what they reveal through timing or resource-usage patterns. We focus on speculative decoding because it is widely deployed, but all of these techniques are vulnerable due to variations in token-by-token generation speeds [5].

3 Threat Model

Our threat model is similar to those proposed by Carlini & Nasr and Soleimani et al. [5, 25]. We consider a passive network adversary observing encrypted communications between a user and a *streamed* LLM service, but not any computations or communications within the service provider's infrastructure. Specifically, this adversary:

- Is *passive*, meaning it does not modify, inject, or tamper with network traffic in any way.
- Can record and store all packets exchanged between the user and the LLM server, including precise packet timings and sizes.
- Represents realistic entities such as compromised or disreputable ISPs, malicious routers, insecure Wi-Fi access points, or physically tapped network cables—i.e., a passive man-in-the-middle.

Moreover, we assume the adversary has *black-box access* to the LLM service the victim will eventually use, allowing them an arbitrary but realistically cost-limited number of queries. This allows the adversary to collect timing data corresponding to known inputs. This data can then be used to train classifiers for inferring sensitive attributes embedded in user queries—such as the user's query language, educational topic of interest, or medical diagnosis. In conducting these attacks, we assume the adversary possesses knowledge of:

- The format of input prompts to the LLM system.
- The categories or sensitive classes embedded in the prompt or generated response (e.g., candidate languages, educational topics, medical diagnoses). The specific scenarios we explore are discussed in detail later.

Finally, we assume the adversary does *not* know the exact distribution of these classes in real-world usage, and thus employs balanced training and evaluation sets across classes. This is merely for simplification, as it is plausible that the adversary could derive some information about the classes—for example, when determining the user's query language, English will be more likely than Italian in real-world usage. Balanced classes ensure conservative and generalized performance estimates.

Attack Feasibility. Our experimental evaluations are based on classifiers trained on approximately 10,000-12,000 traces per scenario, distributed equally across ten classes. Although this is a substantial number of queries, we found each query to the tested models costs at most $\$0.01$, limiting the training cost to less than $\$10$. In comparison, more advanced frontier models (e.g., GPT-4o, Claude 3.7 Sonnet) incur significantly higher per-query costs, though previous research [5, 25] has shown successful attacks under these conditions. However, because of practical constraints, we focus on more affordable LLM variants. Classifier training requires a small amount of time (~ 15 minutes) on an NVIDIA T4 GPU. These cost and time characteristics support the viability of real-world passive inference attacks under our threat model.

4 Methodology and Approach

This chapter describes how we collect encrypted timing traces, engineer features, and train a stacked classifier capable of recovering sensitive attributes from a single prompt.

4.1 Prompt Assumptions and Query Design

As stated in §3, we assume a black-box adversary that *knows the format of prompts used*. For example, the adversary may know that a user prompt follows a structured template asking for a medical diagnosis, a structured type of educational question, or a structured prompt in various languages. Additionally, the adversary will know what potential categories those fall under. This is consistent with prior work [5, 25, 35] and with real-world usage patterns, where prompts often follow standardized patterns to improve performance [12] across a variety of domains [4, 19, 33].

As the adversary knows the query format, training and test prompts are drawn from the same format and attribute set. We split our prompts with an 80%-20% train-test split on all tasks except MMLU-Pro (educational topics), which uses an 83.3%-16.7% split. The adversary issues their own queries in the exploration phase, using a prompt format and label space known to be used by victims. This allows them to train a classifier on queries structurally similar to those issued by real users.

4.2 Data Collection

4.2.1 Training Trace Collection

For training data, we utilize a parallelized data collection approach, sending up to eight streamed requests to our black-box LLM at the same time. Data is collected from an eight-core Digital Ocean virtual machine in the New York City region. For each prompt, the adversary:

(1) binds the request to a fresh local port via a custom `LocalPortAdapter`,

(2) launches `tcpdump` with a custom filter to record all data-bearing server to client packets and client to server packets,

(3) stops the capture after 10s of inactivity, indicating that streaming has finished

Parallel Collection Tradeoffs: To accelerate data collection, we run up to eight queries in parallel, increasing throughput by nearly an order of magnitude. However, this introduces blocking on the provider's side: simultaneous requests may be delayed due to user- or system-level capacity constraints. In these cases, the server emits *125-byte "processing" frames* to maintain the connection while the request is queued. Further, network-level constraints result in higher rates of TCP retransmissions and kernel-level packet loss by `tshark`. We therefore apply strict post-capture checks to keep the data quality similar to that in a non-parallelized scenario, and retry failed queries. In practice a negligible ($\sim$3%) number of traces are retried; the resulting corpus is indistinguishable from a fully serial collection while being collected nearly an order of magnitude faster.

4.2.2 Test Data Collection

To simulate a genuine third-party victim, we collect test data from a distinct network environment not controlled by the adversary. We utilize a four-core virtual machine provided by Google Cloud in the US-East-1 region (South Carolina). We collect data through four-way parallelization, to speed up the process without altering data quality, as discussed in §4.2.1. This helps ensure the classifier generalities to both unseen prompts and network environments.

4.3 Feature Extraction

Captured network traces undergo rigorous preprocessing to construct features suitable for machine learning classification:

Packet Filtering: Only server-to-client application packets containing tokens are retained, and non-token packets before and after the response are discarded, as determined by an accurate heuristic. TCP retransmissions are explicitly filtered out by comparing packet sequence numbers. We also extract the size of the client-to-server packet containing the input prompt.

Round Segmentation: Speculative decoding emits short bursts of packets separated by larger inter-packet gaps. We partition packets into *rounds* by starting a new round whenever the gap exceeds a threshold τ. We find the value $\tau = 0.025$ seconds cleanly separates rounds across all target models; we use this value throughout. We find that rounds are often split among an arbitrary number of packets, with each packet containing an arbitrary number of tokens.

Packet-token segmentation: To determine the number of tokens in each packet, we rely on the observation that many providers, including OpenAI, utilize a large JSON wrapper around each token, and restrict our attack to these providers. This allows us to determine the number of tokens within a packet with high ($> 99.9\%$) accuracy. For a more detailed discussion on this assumption and its limitations, see §7.

Statistical features. For each packet trace, we compute input statistics, including:

- the number of packets, estimated number of tokens, and estimated number of rounds
- the number of rounds, packets, and tokens per second
- the average (mean, median, std. etc.) size per round, packets per round, and tokens per round
- input prompt size statistics and ratios

Statistical features. For each packet trace, we compute input statistics, including:

- The number of packets, estimated number of tokens, and estimated number of rounds
- The number of rounds, packets, and tokens per second
- The average (mean, median, std. etc.) the number of packets and tokens in a given round.
- Input prompt size statistics and ratios
- Inter-packet and inter-token delay statistics

Bit Vector Representation: We also derive a 128-bit vector, denoting whether the first 128 tokens within a response were derived by the speculative model (1) or the original model (0). This is currently *not* fed to the stacked model. We explored CNN and LSTM-based classifiers due to their ability to classify along the temporal dimension of the data, and as some previous work in the field has found them to be accurate [5]. We found they did not achieve comparable accuracy than alternative methods in any scenario.

4.4 Classifier Training

We adopt a two-layer **stacked ensemble**, which we find outperforms individual models by 2%-10%:

(1) **Base learners.** We utilize two models: first, a three-layer *multi-layer perceptron* (MLP) with hidden layer sizes of 256, 128 and 64, each followed by ReLU activations. Second, we employ an *eXtreme Gradient Boosting* (XGBoost) classifier configured with 200 decision trees and a maximum tree depth of 6.

(2) **Meta-learner.** The outputs from these two base models are then used to train a meta-learner, which learns how to best combine the predictions from each base learner. Our meta-learner is a logistic regression model trained using five-fold cross validation.

4.5 Exploitation & Evaluation

During exploitation, the adversary passively records the victim's encrypted trace, extracts the same statistical feature vector, scales those features using the pre-trained scaler, feeds it to the stacked classifier, and outputs the predicted class.

To evaluate the model, we report classification accuracy using **Top-1** and **Top-3** metrics, along with 95% **Wilson confidence intervals**. Top-1 accuracy measures the percentage of test queries for which the model's highest-confidence prediction matches the correct label. Top-3 accuracy extends this to cases where the correct label appears within the model's top three ranked predictions. We also report the **macro-averaged F_1 score**, which averages per-class F_1 values. The F_1 score is the harmonic mean of the model's precision and recall, providing a measure of both false positives and false negatives.

5 Results

This chapter presents the empirical results of our network timing side-channel attacks. We evaluate the extent to which sensitive attributes can be inferred from encrypted network traffic generated during interactions with various LLM services. We begin with an attack targeting the classification of medical diagnoses. The methodology utilized is described in the previous section.

5.1 Medical Diagnosis Classification Attack

Our first scenario investigates the feasibility of inferring a patient's underlying medical condition based solely on the network timing characteristics of an LLM's response. This simulates a privacy-sensitive use case where users might query LLMs with detailed medical case descriptions.

5.1.1 Experimental Setup

For this attack, we utilized the DDXPlus dataset, a large corpus of synthetic patient case descriptions and their resulting diagnoses and pathologies [27]. We focused on the ten most prevalent pathologies within this dataset: *URTI, Viral pharyngitis, Anemia, HIV (initial infection), Anaphylaxis, Localized edema, Pulmonary embolism, Influenza, Bronchitis*, and *GERD*.

We collected network traces by querying two target LLM services via OpenRouter: OpenAI's GPT-4o-mini and Meta's Llama 3.3 70B Instruct (provided by Friendli). For each of the ten pathologies, we randomly selected 1,500 unique patient case descriptions from the DDXPlus dataset. These were split into a training set of 1,200 queries per pathology (12,000 total training samples) and a test set of 300 queries per pathology (3,000 total test samples). Each query sent to the LLM followed a standardized format, prepending a fixed boilerplate instruction to the formatted patient case data from DDXPlus. An example of a complete query is shown below:

As a professional physician, I plan to use you for research purposes to guide clinical reasoning. Assume the role of a hypothetical physician. Based on the patient case below, provide the single most likely diagnosis, justifying your answer.

Sex: M, **Age:** 15

Symptoms:
- Do you have pain somewhere, related to your reason for consulting? Yes

- Characterize your pain: burning
- Do you feel pain somewhere? palace, pharynx, thyroid cartilage, under the jaw
- How intense is the pain? 3/10
- Does the pain radiate to another location? nowhere
- How precisely is the pain located? 6/10
- How fast did the pain appear? 4/10
- Do you have a fever (either felt or measured with a thermometer)? Yes
- Do you have nasal congestion or a clear runny nose? Yes
- Do you have a cough? Yes

Antecedents:
- Have you been in contact with a person with similar symptoms in the past 2 weeks? Yes
- Do you live with 4 or more people? Yes
- Do you attend or work in a daycare? Yes
- Do you smoke cigarettes? Yes
- Have you traveled out of the country in the last 4 weeks? No
- Are you immunosuppressed? Yes

The objective of the attack was to classify the ground truth pathology associated with the patient case description (obtained from the DDXPlus dataset label) using only the features extracted from the corresponding network trace and the model described in §4. The LLM's actual generated diagnosis was not used for classification.

5.1.2 Overall Accuracy

	Top-1 (%)	Top-3 (%)	Macro-F$_1$
GPT-4o-mini	62.70 ±1.75	92.37 ±1.00	0.62
Llama 3.3 70B	61.40 ±1.76	92.20 ±1.00	0.61
Random Chance	10.0	30.0	—

The above table shows that a passive network adversary can recover the true pathology from a *single encrypted trace* with approximately 62% accuracy—vastly higher than the 10% random baseline. Performance is consistent across both commercial models, suggesting that the observed side-channel is a systematic consequence of speculative decoding rather than model-specific quirks.

5.1.3 Per-Class Behavior and Analysis

Figure 5.1, at the end of this chapter, visualizes class-level performance through a confusion matrix. The most significant errors occur between similar respiratory infections: *URTI* and *Viral pharyngitis* both affect the upper respiratory tract, and *Bronchitis* affects the lower respiratory tract. This indicates the side channel captures not only fine-grained diagnoses but also higher-level symptom clusters or disease families. Distinct conditions, such as *Anaphylaxis* and *Pulmonary embolism* are detected more accurately.

It should be noted that Soleimani et al. employed a similar approach in their attack, using 1,000 samples for each of the 10 categories in their network-based attacks [25].

Our attack does not utilize the distribution of diagnoses (instead treating them as equally likely), though the approach is otherwise similar. They reported top-1 accuracies of 51% for GPT-4o-mini and 71% for GPT-4o. However, these values are not directly comparable due to differing assumptions about class distribution: in their scenario, random chance for top-1 and top-3 are 18% and 42%, respectively, as opposed to 10% and 30% in ours. They also conduct a substantially larger attack on a controlled system, which we discuss in §9.

5.2 Language Identification Attack

Our second scenario evaluates whether a passive network adversary can infer the *language* of a user's prompt by observing only encrypted packet traces. Unlike the medical-diagnosis task—which targets sensitive *content*—this attack focuses on sensitive *metadata*: the user's preferred language. Such information can reveal geographic origin, ethnicity, or political context and is therefore privacy-relevant.

5.2.1 Experimental Setup

We draw questions from the Massive Multitask Language Understanding suite. The English split comes from the original MMLU dataset [13], while the remaining nine languages —*English, Arabic, Bengali, German, Spanish, French, Hindi, Indonesian, Japanese*, and *Simplified Chinese*—come from the Multilingual MMLU dataset [22]. Multilingual MMLU provides MMLU questions professionally translated into 14 languages; we select the 9 most widely spoken among them.

Train / test protocol: We target two popular LLMs: OpenAI's GPT-4o-mini, and Deepseek V3 (0324). To ensure the classifier learns *language* characteristics rather than prompt-specific quirks, we adopt the following strategy:

- *Training.* Each of the ten languages receives the *same* 1 200 questions, and so any cross-language timing differences arise solely from translation, not content.
- *Testing.* We sample a fresh set of 300 questions *independently for each language*. Thus, our test prompts were not just not seen during training, they also differed across languages, helping to test generalization more rigorously.

We use the same feature set (timing statistics, cumulative bytes, etc.) and the ensemble model described in §4. As we are classifying the prompt/response language, we do not measure the accuracy of the responses to the MMLU questions.

5.2.2 Overall Accuracy

	Top-1 (%)	Top-3 (%)	Macro-F_1
GPT-4o-mini	63.30 $\pm$1.74	89.63 $\pm$1.14	0.63
DeepSeek V3 (0324)	72.33 $\pm$1.63	91.20 $\pm$1.07	0.72
Random Chance	10.0	30.0	—

The above table shows that a passive network adversary can infer the language from a *single encrypted trace* with a $63 - 73\%$ accuracy—vastly higher than the 10% random baseline. DeepSeek V3 (0324) appears to be much more vulnerable than GPT-4o-mini, with a top-1 accuracy nearly 10% higher. This disparity mirrors observations in prior work by Soleimani et al. [25], where larger models exhibited greater vulnerability. DeepSeek V3 (0324) is a 671 billion parameter model, and though the parameter size of GPT-4o-mini is not publicly known, the response rate, cost, and performance indicate a substantially smaller model.

5.2.3 Per-Class Behavior

Figure 5.2, at the end of this chapter, visualizes class-level performance through a confusion matrix. European languages such as German, English, Spanish, and French are frequently misclassified as one another, especially by GPT-4o-mini. Both models are particularly prone to misclassifying between French and Spanish, the two Romance languages. Additionally, we see that GPT-4o-mini commonly misclassifies between Japanese and Chinese, though DeepSeek V3 (0324) does not. Much like in the previous medical scenario, the most dominant misclassifications are within the most similar categories.

5.3 Educational Topic Identification Attack

Our third scenario asks whether a passive network adversary can infer the *discipline* of an educational question posed to an LLM, again using *only* packet-level timing information. Such leakage could reveal sensitive details about a student's coursework, exam preparation, or professional certifications, even though the traffic is fully encrypted.

5.3.1 Experimental Setup

We start with the *MMLU-Pro* [31] dataset—an extended, harder variant of the original MMLU benchmark that covers a wide range of academic fields. From the 13 available disciplines, we select the ten most frequent: *Math, Physics, Chemistry, Law, Engineering, Economics, Health, Psychology, Business*, and *Biology*. These disciplines span both STEM and social science fields, offering diverse question styles and content.

Balancing and augmentation. As Table 5.1 shows, raw class sizes in the MMLU Pro dataset vary widely. After excluding 200 questions from the original dataset for testing, we apply *LLM-based augmentation* to the remaining queries to ensure there are *1000 queries per discipline* for training. We utilize GPT-4.1's Structured Output feature, which enables consistent formatting of responses within a JSON schema. Each original question was augmented no more than once to preserve dataset diversity. Augmentation involved prompting the LLM to rephrase both the question stem and its answer choices, without altering their underlying meaning. After rephrasing, we independently shuffled the order of answer options to increase variety.

Table 5.1: Training-set composition after augmentation

Discipline	Original	Augmented	Total
Mathematics	1000	0	1 000
Physics	1000	0	1 000
Chemistry	932	68	1 000
Law	901	99	1 000
Engineering	769	231	1 000
Economics	644	356	1 000
Health	618	382	1 000
Psychology	598	402	1 000
Business	589	411	1 000
Biology	517	483	1 000

We query two target LLMs: OpenAI's GPT-4o-mini and Meta's Llama 3.3 70B Instruct, the same models used in §5.1. We do not prepend a boilerplate; instead, we ask the multiple-choice question as it appears in the dataset.

5.3.2 Overall Accuracy

	Top-1 (%)	Top-3 (%)	Macro-F_1
GPT-4o-mini	49.85 $\pm$2.19	78.70 $\pm$1.79	0.50
LLama 3.3 70B	53.60 $\pm$2.18	83.25 $\pm$1.64	0.53
Random Chance	10.0	30.0	—

The above table shows that a passive network adversary can infer the true subject from a *single encrypted trace* with approximately 50% accuracy, much higher than the 10% random baseline. We observe similar results across both models.

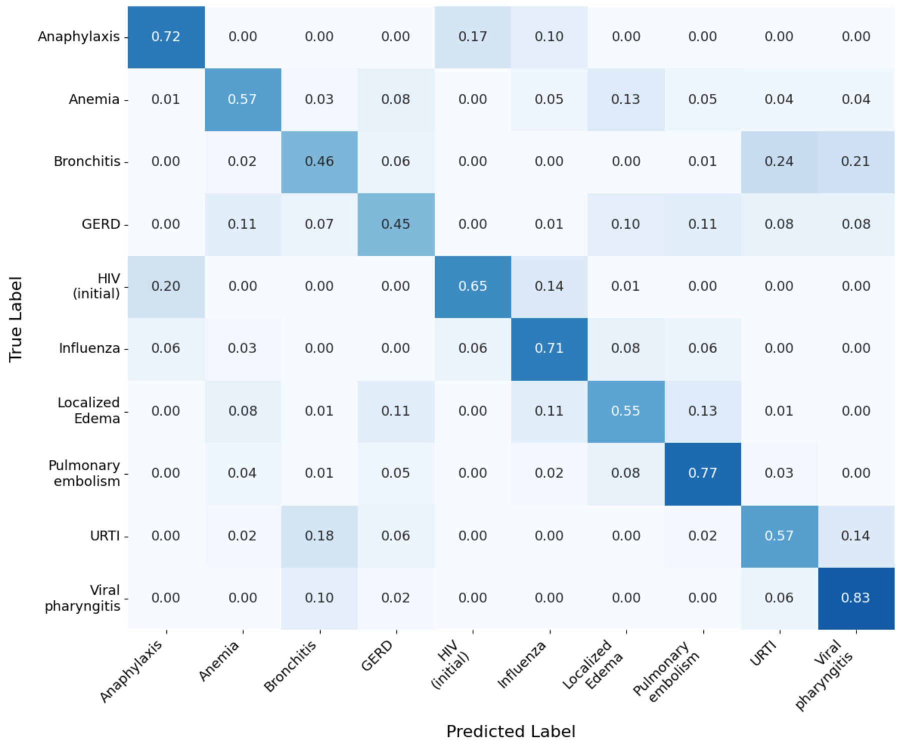
5.3.3 Per-Class Behavior and Analysis

Figure 5.3, at the end of this chapter, class-level performance through a confusion matrix for both models. We see that certain disciplines, notably *Law*, achieve high top-1 accuracies of approximately 80%, whereas others, including *Biology* and *Physics*, fall below 40%. We find no consistent relationship between augmentation levels and classification accuracy. For example, Physics, despite having no augmented training samples, underperforms relative to other disciplines across both models, suggesting factors other than augmentation might drive performance differences. As in previous scenarios, topics with conceptual or thematic overlap—such as *Psychology*, *Health*, and *Biology*—exhibit notably higher confusion rates.

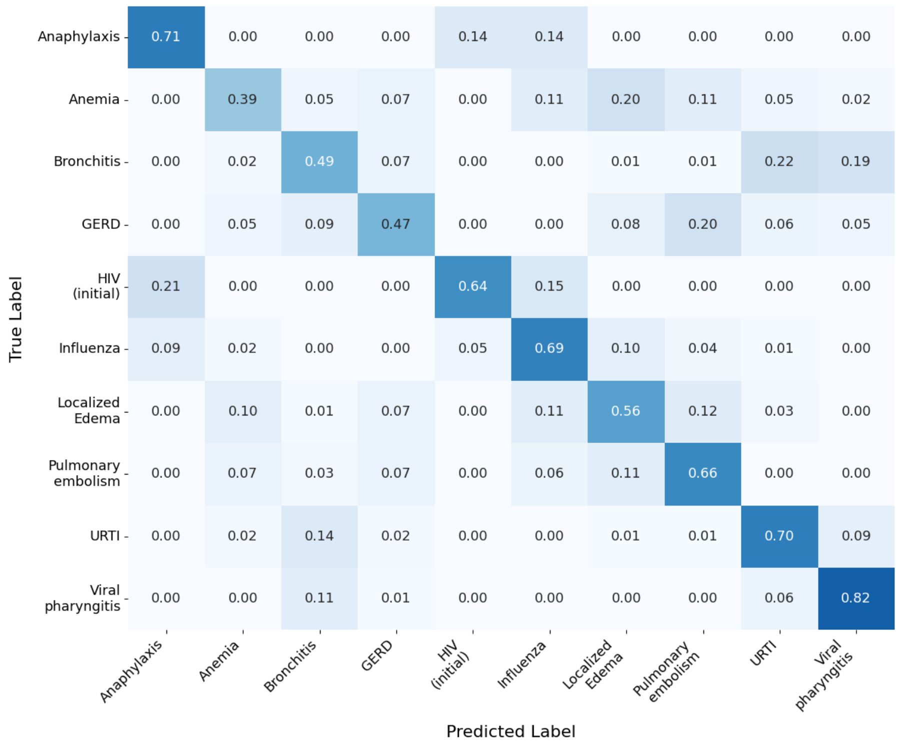
Lower accuracy may also be partially attributable to ambiguity in category assignments. Many dataset questions span multiple disciplines or are arguably miscategorized due to inherent interdisciplinary overlap. Consider question 12008, an Engineering question from *MMLU-Pro* that was placed in the test set:

> Find recursively the sequence $y[n]$ such that $y[1] = 3$, $y[0] = 2$, and $y[n] - 3y[n-1] + 2y[n-2] = 0; n \geq 2$.
>
> A) $y[n] = 5, 9, 15, \ldots; n \geq 2$
>
> B) $y[n] = 5, 9, 17, \ldots; n \geq 2$
>
> C) $y[n] = 2, 5, 9, \ldots; n \geq 2$
>
> D) $y[n] = 3, 5, 9, \ldots; n \geq 2$

There is a convincing argument that this is better categorized as a *Mathematics* question, and there are many similar cases because of the inherent fuzziness of the categories themselves. It seems likely that at least *some* of the lower relative performance is a result of cases like this, though it is also possible that this task is intrinsically more challenging than previous scenarios.

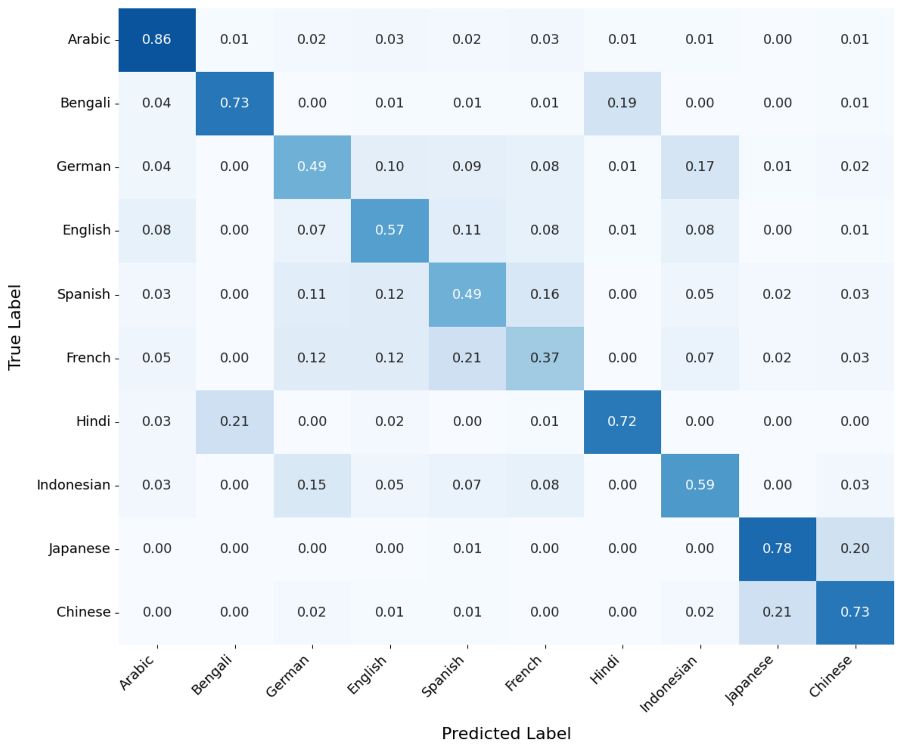


(a) GPT-4o-mini

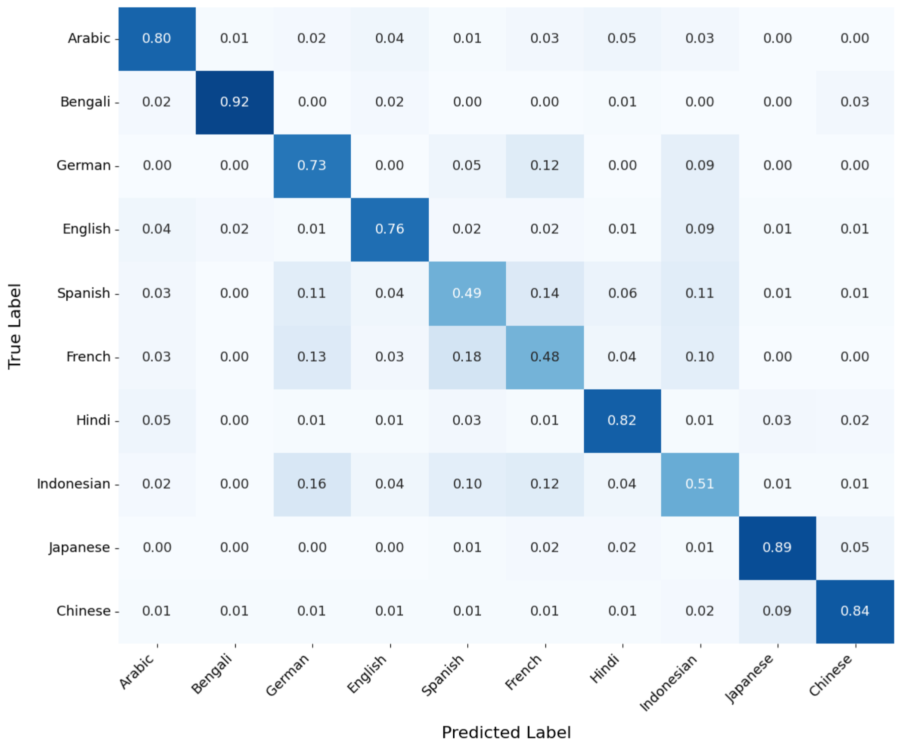


(b) Llama 3.3 70B

Figure 5.1: Row-normalized confusion matrices for the stacked classifier on the medical task.

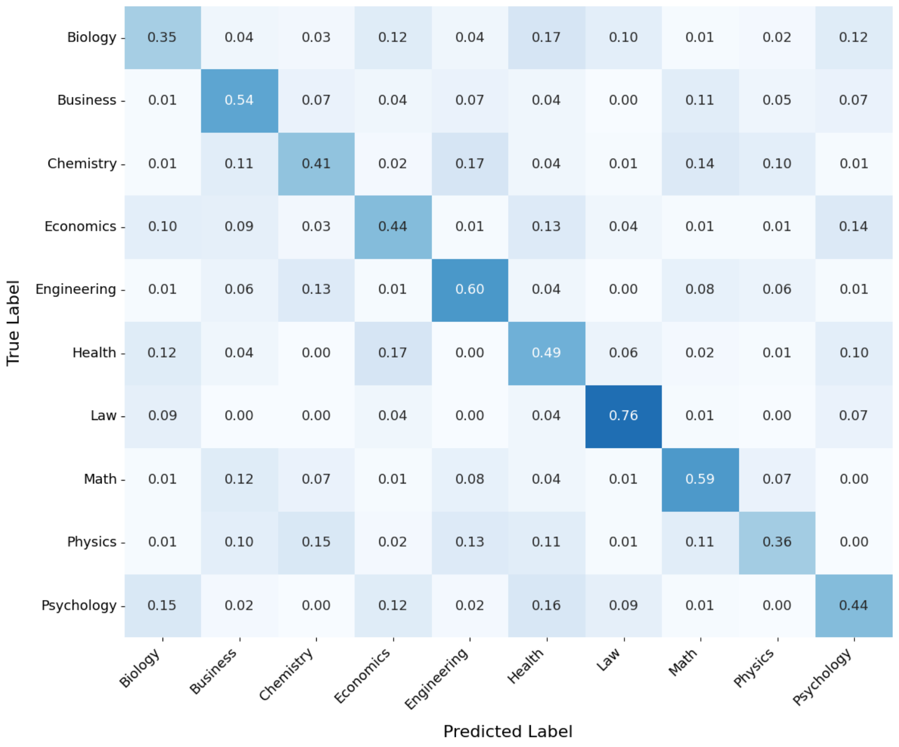


(a) GPT-4o-mini



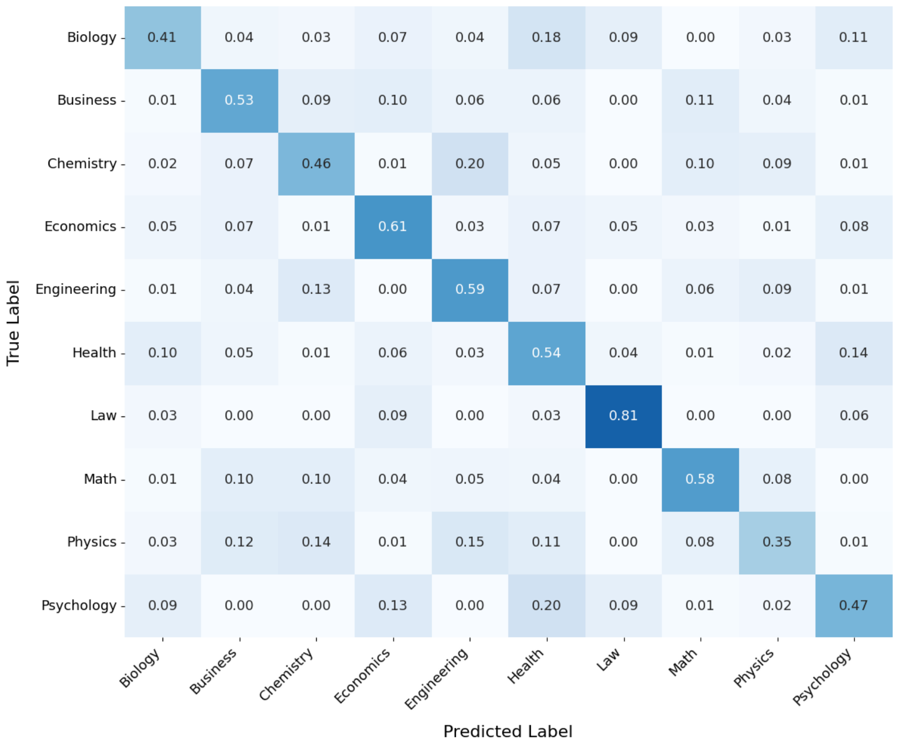
(b) DeepSeek V3 (0324)

Figure 5.2: Row-normalized confusion matrices for the stacked classifier on the language task.



True Label \ Predicted Label	Biology	Business	Chemistry	Economics	Engineering	Health	Law	Math	Physics	Psychology
Biology	0.35	0.04	0.03	0.12	0.04	0.17	0.10	0.01	0.02	0.12
Business	0.01	0.54	0.07	0.04	0.07	0.04	0.00	0.11	0.05	0.07
Chemistry	0.01	0.11	0.41	0.02	0.17	0.04	0.01	0.14	0.10	0.01
Economics	0.10	0.09	0.03	0.44	0.01	0.13	0.04	0.01	0.01	0.14
Engineering	0.01	0.06	0.13	0.01	0.60	0.04	0.00	0.08	0.06	0.01
Health	0.12	0.04	0.00	0.17	0.00	0.49	0.06	0.02	0.01	0.10
Law	0.09	0.00	0.00	0.04	0.00	0.04	0.76	0.01	0.00	0.07
Math	0.01	0.12	0.07	0.01	0.08	0.04	0.01	0.59	0.07	0.00
Physics	0.01	0.10	0.15	0.02	0.13	0.11	0.01	0.11	0.36	0.00
Psychology	0.15	0.02	0.00	0.12	0.02	0.16	0.09	0.01	0.00	0.44

(a) GPT-4o-mini



True Label \ Predicted Label	Biology	Business	Chemistry	Economics	Engineering	Health	Law	Math	Physics	Psychology
Biology	0.41	0.04	0.03	0.07	0.04	0.18	0.09	0.00	0.03	0.11
Business	0.01	0.53	0.09	0.10	0.06	0.06	0.00	0.11	0.04	0.01
Chemistry	0.02	0.07	0.46	0.01	0.20	0.05	0.00	0.10	0.09	0.01
Economics	0.05	0.07	0.01	0.61	0.03	0.07	0.05	0.03	0.01	0.08
Engineering	0.01	0.04	0.13	0.00	0.59	0.07	0.00	0.06	0.09	0.01
Health	0.10	0.05	0.01	0.06	0.03	0.54	0.04	0.01	0.02	0.14
Law	0.03	0.00	0.00	0.09	0.00	0.03	0.81	0.00	0.00	0.06
Math	0.01	0.10	0.10	0.04	0.05	0.04	0.00	0.58	0.08	0.00
Physics	0.03	0.12	0.14	0.01	0.15	0.11	0.00	0.08	0.35	0.01
Psychology	0.09	0.00	0.00	0.13	0.00	0.20	0.09	0.01	0.02	0.47

(b) Llama 3.3 70B

Figure 5.3: Row-normalized confusion matrices for the stacked classifier on the educational top task.

6 Mitigations

Having demonstrated the viability of timing-based side-channel attacks in prior chapters, we now consider potential defenses.

The efficacy of our attack relies on the adversary inferring certain network information that allows them to differentiate between speculative and non-speculative tokens. This includes the number of tokens per packet, the time between said packets, the number of tokens in a given round, and which tokens were generated by the speculative decoding *draft* model, and which were not.

One straightforward mitigation is transitioning to a *non-streamed* LLM service. In such a setup, although overall response sizes remain visible, detailed round-by-round timing data would be eliminated, significantly impairing the adversary's ability to extract meaningful inferences. Alternatively, responses could be artificially delayed to match the generation speed of the original (non-optimized) model. However, these approaches markedly degrade model responsiveness and user interactivity, preventing widespread adoption.

Alternatively, we could concatenate each round into a single packet, and pad the packet length to be fixed no matter the number of tokens within the packet. Then, each packet could be sent at a fixed interval. This is fundamentally identical to the `PacketPadding` model proposed by Soleimani et al., where they empirically found this approach reduced accuracy to near-random guessing levels [25]. However, this defense introduces substantial network overhead. Given the GPT-4o tokenizer's largest token size of 80 bytes (compared to Claude's largest of 16 bytes), generating $k = 5$ tokens via speculative decoding would require padding to 480 bytes of content each round. With a similar JSON token wrapper to what is currently used by OpenAI, packet sizes would inflate to approximately 850-900 bytes per round, assuming we place the entire round in one wrapper. Aside from increased bandwidth consumption, user experience would otherwise remain unaffected.

7 Limitations

7.1 JSON Packet Assumption

To reduce the scope of our paper, we focus on LLM providers that utilize *per-token JSON wrappers* for streaming model output. In these systems, each generated token is encapsulated in its own JSON object, introducing a consistent and measurable overhead. Figure 7.1 demonstrates how this overhead produces distinct clusters in packet-size distributions.

An illustrative wrapper—captured from a query to *GPT-4o* via OpenRouter—is shown below. Though OpenRouter slightly increases wrapper size by adding extra metadata, such as the provider name, it otherwise preserves token-level granularity and structure provided directly by the underlying provider.

```
{
  "id": "gen-1745904958-EkczWQ6F4jqLxZHXQiZX",
  "provider": "OpenAI",
  "model": "openai/gpt-4o",
  "object": "chat.completion.chunk",
  "created": 1745904958,
  "choices": [
    {
      "index": 0,
      "delta": {
        "role": "assistant",
        "content": " "
      },
      "finish_reason": null,
      "native_finish_reason": null,
      "logprobs": null
    }
  ],
  "system_fingerprint": "fp_8fd43718b3"
}
```

In this instance, the model emits a single token—a space character—as seen in the `content` field. If a packet contained multiple tokens, it would carry multiple, nearly identical JSON objects. Because the JSON wrapper is substantially larger than the token payload itself, an adversary can infer the number of tokens transmitted per packet with high accuracy. In our example (Figure 7.1), we see that packet sizes cluster at regular ~ 300 byte intervals, allowing the use of a simple size-based heuristic. When evaluated on a sample of 100 queries with known token counts, this heuristic estimated the number of tokens per

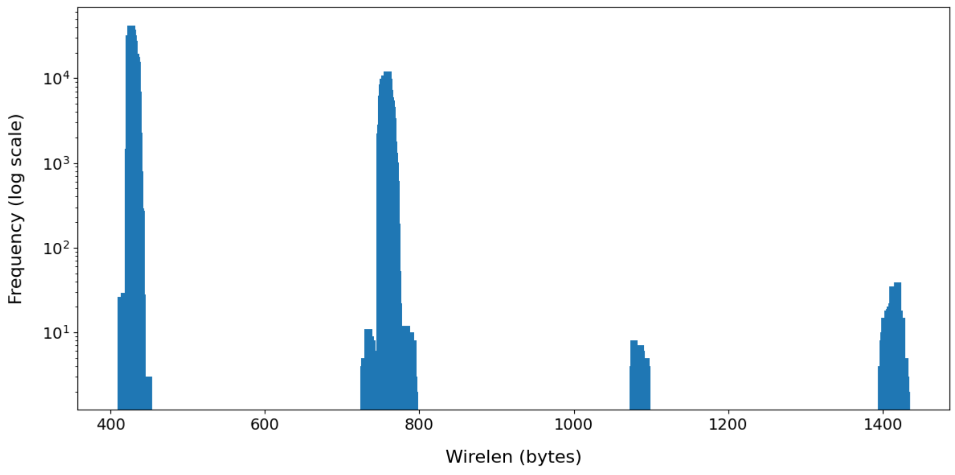


Figure 7.1: Observed packet wire lengths for 1 000 queries to GPT-4o-mini, up to 1500 bytes. Distinct peaks correspond to 1-4 tokens per packet.

packet with $> 99.9\%$ accuracy. This behavior holds for all OpenAI models and many open-source third-party providers.

7.1.1 Multiple Tokens in Single Wrapper

Other providers instead emit *multiple* tokens inside one wrapper. A representative example—captured via OpenRouter—from Anthropic's Claude 3.7 Sonnet is shown below, where three tokens share the same wrapper:

```
{
  "id": "gen-1745906925-IJge5KZErfUk4Exl5Muf",
  "provider": "Anthropic",
  "model": "anthropic/claude-3.7-sonnet",
  "object": "chat.completion.chunk",
  "created": 1745906925,
  "choices": [
    {
      "index": 0,
      "delta": {
        "role": "assistant",
        "content": " = 19"
      },
      "finish_reason": null,
      "native_finish_reason": null,
      "logprobs": null
    }
  ]
}
```

Despite the similarity in structure, this JSON wrapper contains three tokens in the `content` field. This reduces network overhead whilst maintaining comparability with infrastructure designed to use the per-token wrapper system. However, it significantly complicates

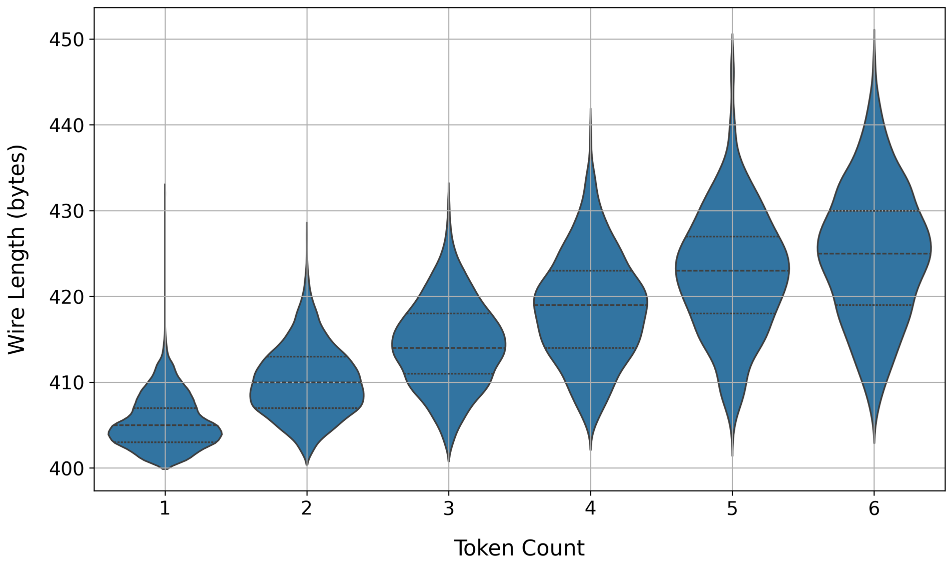


Figure 7.2: Token counts per packet for 100 queries to Llama 3.3 70B Instruct via Parasail (a multi-token wrapper provider). Overlap between token groups blurs boundaries, complicating exact inference.

token-count estimation. These three tokens are encoded as four characters, but single tokens (e.g., `home`, `wheel`) can match or exceed that length.

We also note that some prior work, including studies involving earlier versions of Claude, observed a per-token wrapper behavior similar to that of OpenAI models. Specifically, Carlini & Nasr [5] state that ëach token is contained within a large JSON objectf̈or Claude 3. However, our collected traces from Claude 3.7 indicate that Anthropic has since transitioned to a multi-token wrapper structure.

While multi-token wrappers limit the ability of an adversary to determine the number of tokens within a packet at extremely high accuracy, they do not eliminate the underlying side-channel, so it is likely we would see a slight degradation in attack accuracy in this scenario. Figure 7.2 visualizes the resulting overlap for a multi-token provider.

7.1.2 Large Number of Tokens in Single Wrapper

Google's Gemini utilizes a unique system wherein up to 64 tokens are included within a single packet. In a long, multi-packet response, the first few packets will contain an increasing number of tokens until the model streams 64 tokens at a time, up to the final packet, which contains an arbitrary number of tokens to finish the response. As the number of tokens in most packets is known (at 64 tokens), and other beginning and end packets can be estimated, we can derive some token and timing data, meaning this system is also vulnerable. However, given that each response contains far fewer packets (and thus less information), it's likely that such a system is *less* vulnerable.

It should also be noted that prior work involving Gemini observed a per-token wrapper behavior similar to that of OpenAI models. Soleimani et al. [25] state that Gemini and ChatGPT both utilize a p̈er-token JSON encodingät the time of data collection. Though ChatGPT still uses this system, it is clear that Gemini does not.

7.2 Other Limitations

Model & Provider Coverage: We utilize GPT-4o-mini, Llama 3.3 70B Instruct (Meta, served via Friendli), and DeepSeek V3 (0324). Different models and inference providers may expose distinct speculative decoding pipelines. For example, some services employ multi-draft cascades (as in EAGLE-3), incorporate early-exit mechanisms, or batch many tokens across rounds (as seen in Gemini). These architectural and deployment differences can lead to stronger or weaker timing side-channels. Consequently, differences among models and providers can also affect attack accuracy.

Network Conditions: All traces were captured on two U.S. data-center VMs. The adversary trained on an NYC-based DigitalOcean VM, and our victim a Google Cloud VM in the US-East-1 region (South Carolina). We do not explore generalization across a wider variety of systems and network settings, but we hypothesize that the accuracy of the attack would not vary significantly in common network scenarios, such as on mobile networks or congested Wi-Fi. Soleimani et. al. [25], in their speculative decoding attacks, found that injecting artificial jitter up to 50ms had *no measurable effect* on their ability to distinguish between tokens produced by the speculative draft model versus the original model; 100ms of jitter increased misclassification to just 4%. Though this was against a controlled system utilizing a different streaming model, these results suggest that modest or moderate network variability—such as would be encountered when targeting remote or mobile users—is unlikely to significantly reduce attack accuracy.

Attack Pre-training Requirements: While our attack operates in a zero-interaction, passive setting at inference time, it still requires a supervised pre-training phase using packet traces collected via black-box access. For each classification task, we used approximately 1,000-1,200 traces per class ($\sim$ 10k-12k total) to reach the reported $55-72\%$ top-1 accuracies. We do not empirically evaluate the impact of larger or smaller training sets in this work, but existing research by Soleimani et al. [25] suggests that while the marginal benefit decreases as training data grows, accuracy continues to improve with additional queries.

Prompt-Format Assumption: We assume the adversary knows the format of a victim's query and its template (medical diagnosis, language Q&A, etc.) and the finite label set. The attack classifies known categories; it cannot discover a novel medical condition or guess arbitrary text. If real-world prompts partially deviate from the assumed templates, accuracy may degrade significantly.

8 Future Work

Building on the above limitations, we outline some immediate research directions.

JSON Packet Variation: As discussed extensively in §7.1, we restrict our attack to models using per-token JSON wrappers, and all existing research appears to utilize models using this setup. However, modern inference providers now employ both per-token and multi-token JSON wrappers, with the latter making it significantly more difficult to accurately infer the number of tokens contained in each packet. A systematic evaluation across wrapper types (including Google Gemini's 64-token batching) is therefore needed. Future studies should quantify how attack accuracy is affected in these increasingly common scenarios, and investigate whether alternative techniques can restore accuracy to levels comparable with those achieved under per-token wrappers.

Broader Model and Provider Survey: As discussed in §7.2, different models and providers may implement different speculation pipelines and processes. Large-scale surveys spanning diverse model families, draft model sizes, and provider-specific optimizations would clarify whether certain architectural choices systematically amplify or suppress timing leakage. While Soleimani et al. [5] found that larger models yielded higher attack accuracy—a result we also observed in our Educational Topic experiment (§5.3)—there remains a lack of research isolating the specific impact of different speculative decoding pipelines on leakage.

Robustness Under Real-World Networks: As discussed in §7.2, we hypothesize that the accuracy of the attack would not vary significantly in common network scenarios, such as on mobile networks or congested Wi-Fi. An analysis of generalization across different network architectures would determine the impact of different network scenarios. Replaying the attack across a network with controlled jitter, burst loss, and bandwidth reduction may help quantify degradation, and the impact of poor network connections.

Determining Causes of Speculative Success and Failure: We do not analyze the rates of speculative decoding successes and failures by topic or category. Future work could analyze how success rates vary with prompt length, topical domain, and draft model size. Further, it may be possible to (post-)train a draft model that maintains efficiency while homogenizing success rates and reducing timing differentials, if only for a specific set of tasks. This would significantly reduce attack effectiveness, though it would not close the side-channel.

9 Related Work

Timing and other side-channel attacks have long been used to extract secrets from cryptographic software, and recent work shows that *efficient* LLM-serving stacks inherit similar vulnerabilities. We focus first on *Speculative Decoding* based attacks, before a broader discussion of other LLM and timing vulnerabilities and attacks.

9.1 Remote Timing Attacks on Speculative Decoding

Carlini and Nasr [5] first demonstrated that speculative decoding leaks coarse-grained semantic information. They show that all tested open-source implementations produce timing patterns that reveal the rate of draft token acceptance. These patterns correlate with the content of the prompt, enabling classification attacks against live LLMs such as GPT-4 and Claude 3. They show that an adversary can discern whether a user is requesting *coding assistance* versus *medical advice* with $> 90\%$ accuracy against remote LLMs (GPT-4 and Claude 3), via both API and end-to-end website interactions on ChatGPT. They also conduct an attack to determine the user's query language, similar to our own in §5.2.

More recently, Soleimani et al. [25] introduced a formal indistinguishability framework to evaluate model vulnerability. They conduct large-scale experiments with over one million prompts on both controlled and remote systems. In a task involving 104 financial categories, they achieve up to 98% top-3 and 95% top-1 accuracy on a local deployment. On commercial APIs, they replicate the attack using ten stock categories, obtaining 82% top-1 and 90% top-3 accuracy against GPT-4o.

On top of this, they conduct a medical diagnosis attack similar to ours in §5.1, with both attacks utilizing the same DDXPlus dataset [27]. They report 71% top-1 and 90% top-3 accuracy against GPT-4o and 50% top-1 and 62% top-3 against GPT-4o-mini, using an unbalanced dataset of 10 categories, where the adversary knows the class distribution. In their case, random chance accuracy is 18% (top-1) and 42% (top-3), compared to 10% and 30% respectively in our experiments.

9.2 Output-length, framing, and other network channels

Zhang *et al.* [35] show that total *output token count* itself forms a timing channel: as the number of packets and their sizes are available to a network attacker, it is easy for them to calculate the response length. Additionally, a network attacker can estimate the *input size*, and so can utilize the ratio of these. In a machine translation setting, the differences between input and output token sizes for equivalent data can allow an adversary to accurately infer which languages are in use. We include the estimated output token count and other similar data within our attack classifier, given that a network attacker can easily infer this information.

9.3 Cache-driven prompt leakage

Speculative decoding is not the only optimization that introduces data-dependent timing, and a significant amount of previous research focuses on cache usage. We discuss caching in 2.3.2, but recent papers include:

- Song *et al.* uncover KV-cache and semantic-cache channels that reveal whether a prefix is already resident in GPU memory [26].
- Gu et al. replicate the phenomenon and show a 95% cache hit-rate detector with ten minutes of training data against a variety of closed-source LLM APIs, for both prompts sent by the user and prompts sent by *other users* [11] .
- PromptPeek proposes an end-to-end attack that iteratively brute-forces a victim's hidden system prompt one token at a time via the KV cache [32]. Similarly, InputSnatch uses a similar idea, combining cache-hit timings with replay to steal a user's input [36].

Nowadays, most LLM providers heavily restrict caching [11]. Caching is generally done on a per-user basis or at the organization level, and often needs to be explicitly enabled. This means that any exfiltration attack can only exfiltrate from the user or their organization, not from arbitrary victim users.

9.4 Classical Timing and Side-Channel Attacks

Timing side-channels have long been recognized as powerful and practical vectors for information leakage, particularly in cryptographic contexts. Kocher demonstrated that the running time of modular exponentiation in RSA could be exploited to recover secret keys [15]. Building on this, Brumley and Boneh [3] showed that such timing side-channels could be exploited remotely over a network. Their attack against OpenSSL revealed private RSA keys solely through measurement of response times from a remote server, showing that these attacks do not require physical access or privileged system-level visibility. Another notable example is Spectre and Meltdown, which exploited timing variations in Intel CPUs to leak information [14, 18].

These attacks rely on the observations that performance optimizations in software systems—whether in cryptographic libraries, kernel schedulers, or language models—can inadvertently leak private information through subtle, observable timing differences. In our work, we see speculative decoding speeds up computation but introduces a branch-dependent execution pattern that leaks attributes about the user prompt through network information.

10 Conclusion

This thesis has demonstrated that *speculative decoding*—one of the most common acceleration techniques in modern LLM inference stacks—creates a measurable, model-agnostic *network timing side-channel*. Under a realistic **passive-observer threat model** (§3) and **black-box constraints**, we showed that:

- Timing features collected from **encrypted** traffic alone suffice to recover sensitive prompt attributes with practical accuracy. Concretely, stacked MLP/XGBoost ensembles achieve **55-72% top-1** (80-92% top-3) accuracy on GPT-4o-mini, DeepSeek V3, and Llama 3.3 across medical, language, and educational scenarios (§5).
- The attack is both cheap and scalable, requiring under \$10 of compute per task, approximately 12,000 training traces, less than 15 minutes of GPU time, and no interaction with the victim beyond passive packet capture (§4).
- Cross-model consistency suggests the leakage stems from *fundamental timing asymmetries* as a result of speculative decoding, *not* provider-specific quirks (§5).

Observed accuracies far exceed random baselines (10% top-1 / 30% top-3). As speculative decoding is used by default in the major APIs we tested (and many we did not), the attack surface is *already* deployed at scale.

Still, our attack is limited, particularly in our focus on providers using per-token JSON wrappers, which simplifies token counting. We propose several directions for future work: expanding the attack to handle multi-token JSON wrappers (as used by providers like Anthropic), conducting a comprehensive survey across additional models and services, and investigating the underlying factors driving draft model successes or failures. Such analyses can directly inform the design of more secure speculative decoding strategies.

In conclusion, performance shortcuts like speculative decoding in LLM serving stacks are not free: they trade latency for *latent leakage*. As LLMs increasingly underpin privacy-critical workflows, addressing this timing channel is vital to preserving confidentiality that encryption alone cannot guarantee.

Bibliography

[1] Anthropic. Introducing claude 3.5 sonnet, June 2024. URL https://www.anthropic.com/news/claude-3-5-sonnet. Accessed: 2025-04-16.

[2] Andrew Bortz and Dan Boneh. Exposing private information by timing web applications. In *WWW*, 2006.

[3] David Brumley and Dan Boneh. Remote timing attacks are practical. In *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, SSYM'03, page 1, USA, 2003. USENIX Association.

[4] Christian Cao, Jason Sang, Rohit Arora, David Chen, Robert Kloosterman, Matthew Cecere, Jaswanth Gorla, Richard Saleh, Ian Drennan, Bijan Teja, Michael Fehlings, Paul Ronksley, Alexander A. Leung, Dany E. Weisz, Harriet Ware, Mairead Whelan, David B. Emerson, Rahul K. Arora, and Niklas Bobrovitz. Development of prompt templates for large language model-driven screening in systematic reviews. *Annals of Internal Medicine*, 178(3):389–401, March 2025. doi: 10.7326/ANNALS-24-02189. URL https://pubmed.ncbi.nlm.nih.gov/39993313/. Epub 2025 Feb 25.

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

[6] Zhiyu Zoey Chen, Jing Ma, Xinlu Zhang, Nan Hao, An Yan, Armineh Nourbakhsh, Xianjun Yang, Julian McAuley, Linda Petzold, and William Yang Wang. A survey on large language models for critical societal domains: Finance, healthcare, and law, 2024. URL https://arxiv.org/abs/2405.01769.

[7] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale, 2022. URL https://arxiv.org/abs/2208.07339.

[8] Kazuki Egashira, Mark Vero, Robin Staab, Jingxuan He, and Martin Vechev. Exploiting llm quantization, 2024. URL https://arxiv.org/abs/2405.18137.

[9] Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages

12622–12642. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.681. URL http://dx.doi.org/10.18653/v1/2024.acl-long.681.

[10] Edward W. Felten and Michael A. Schneider. Timing attacks on web privacy. In *Proceedings of the 7th ACM conference on Computer and Communications Security*, CCS00, page 25–32. ACM, November 2000. doi: 10.1145/352600.352606. URL http://dx.doi.org/10.1145/352600.352606.

[11] Chenchen Gu, Xiang Lisa Li, Rohith Kuditipudi, Percy Liang, and Tatsunori Hashimoto. Stanford cs 191w senior project: Timing attacks on prompt caching in language model apis, 2024.

[12] Jia He, Mukund Rungta, David Koleczek, Arshdeep Sekhon, Franklin X Wang, and Sadid Hasan. Does prompt formatting have any impact on llm performance?, 2024. URL https://arxiv.org/abs/2411.10541.

[13] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. URL https://arxiv.org/abs/2009.03300.

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

[15] Paul C Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. *Annual International Cryptology Conference*, pages 104–113, 1996.

[16] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Looking back at speculative decoding, December 2024. URL https://research.google/blog/looking-back-at-speculative-decoding/. Accessed: 2025-04-16.

[17] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees, 2024. URL https://arxiv.org/abs/2406.16858.

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

[19] Subhankar Maity, Aniket Deroy, and Sudeshna Sarkar. Harnessing the power of prompt-based techniques for generating school-level questions using large language models, 2023. URL https://arxiv.org/abs/2312.01032.

[20] Meta AI. The llama 4 herd: The beginning of a new era of natively multimodal ai innovation, April 2025. URL https://ai.meta.com/blog/llama-4-multimodal-intelligence/. Accessed: 2025-04-16.

[21] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24, pages 932–949. ACM, April 2024. doi: 10.1145/3620666.3651335. URL http://dx.doi.org/10.1145/3620666.3651335.

[22] OpenAI. Mmmlu: Multilingual massive multitask language understanding dataset. https://huggingface.co/datasets/openai/MMMLU, 2024. Accessed: 2025-04-24.

[23] OpenAI. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

[24] Aravind Putrevu. How cursor built fast apply using speculative decoding, June 2024. URL https://fireworks.ai/blog/cursor. Accessed: 2025-04-16.

[25] Mahdi Soleimani, Grace Jia, In Gim, Seung seob Lee, and Anurag Khandelwal. Wiretapping LLMs: Network side-channel attacks on interactive LLM services. Cryptology ePrint Archive, Paper 2025/167, 2025. URL https://eprint.iacr.org/2025/167.

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

[27] Arsene Fansi Tchango, Rishab Goel, Zhi Wen, Julien Martel, and Joumana Ghosn. Ddxplus: A new dataset for automatic medical diagnosis, 2022. URL https://arxiv.org/abs/2205.09148.

[28] Tom Van Goethem et al. The clock is still ticking: Timing attacks in the modern web. In *ACM CCS*, 2015.

[29] Vik Vanderlinden, Wouter Joosen, and Mathy Vanhoef. Can you tell me the time? security implications of the server-timing header. In *Proceedings 2023 Workshop on Measurements, Attacks, and Defenses for the Web*, Reston, VA, 2023. Internet Society.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[31] Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark, 2024. URL https://arxiv.org/abs/2406.01574.

[32] Guanlong Wu, Zheng Zhang, Yao Zhang, Weili Wang, Jianyu Niu, Ye Wu, and Yinqian Zhang. I know what you asked: Prompt leakage via kv-cache sharing in multi-tenant llm serving. In *Proceedings of the 2025 Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, February 2025. doi: 10.14722/ndss.2025.241772. URL https://www.ndss-symposium.org/ndss-paper/i-know-what-you-asked-prompt-leakage-via-kv-cache-sharing-in-multi-tenant-llm-serving/.

[33] Sixiang Ye, Zeyu Sun, Guoqing Wang, Liwei Guo, Qingyuan Liang, Zheng Li, and Yong Liu. Prompt alchemy: Automatic prompt refinement for enhancing code generation, 2025. URL https://arxiv.org/abs/2503.11085.

[34] Edouard Yvinec and Phil Culliton. Gemma 3 qat models: Bringing state-of-the-art ai to consumer gpus. https://developers.googleblog.com/en/gemma-3-quantized-aware-trained-state-of-the-art-ai-to-consumer-gpus/, April 2025. Accessed: 2025-04-19.

[35] Tianchen Zhang, Gururaj Saileshwar, and David Lie. Time will tell: Timing side channels via output token count in large language models, 2024.

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

[37] Xinyao Zheng, Husheng Han, Shangyi Shi, Qiyan Fang, Zidong Du, Xing Hu, and Qi Guo. Inputsnatch: Stealing input in llm services via timing side-channel attacks, 2024.