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) now answer billions of streamed queries each day. To keep latency acceptable, inference providers adopt techniques to improve resource efficiency and throughput. Many utilise speculative decoding, wherein a lightweight draft model proposes several future tokens that the original model then verifies. This technique, however, unintentionally introduces timing side-channels, as the acceptance or rejection of draft tokens leads to observable differences in inter-packet timings and sizes, which can be utilised to infer sensitive attributes of a user's prompt.

Under a purely passive, black-box threat model we perform a multi-scenario evaluation across tasks like medical diagnosis classification, user language identification, and educational topic inference, each with ten target classes. We utilise 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 task, we extract timing, packet size, and speculative round statistics, including inter-packet delays and token-per-packet distributions and train a stacked MLP/XGBoost ensemble. Our classifier 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 reaching up to 92%, demonstrating a high-fidelity side-channel capable of reliably inferring user intent in widely deployed inference APIs that undermines TLS-protected streaming.

We discuss potential mitigations and outline avenues for future work: on attacks targeting multi-token JSON wrappers; on robustness across varied network environments, models, and providers; and on developing draft models resistant to this attack. Our results show critical vulnerabilities in model-serving practices utilised by major 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, 5, 21]. To achieve this level of performance, LLMs have grown dramatically in scale: the largest publicly-known models now exceed two trillion parameters [18]. This increase in model size enables higher-quality generation and better generalisation but comes at the cost of significantly increased inference-time computation. Serving such large models is expensive, both in terms of latency (time to first token) and compute resources.

To mitigate this cost and enhance interactivity, LLM inference providers employ a range of performance optimisations for efficient inference. Techniques employed include quantisation, which reduces model size and inference cost by representing weights with lower-precision formats [6]; prompt caching, which reduces the computation needed for repeated queries [9]; and speculative decoding, which improves token-by-token generation speeds [4]. 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 *draft* model [14, 22]. When the responses are streamed to a client, either token-by-token or in small batches, the inter-packet timings and packet sizes allow an adversary to learn which tokens were generated with speculation and which were not. These timing differences, though subtle, create powerful side-channels. An attacker with only passive access to network traffic can exploit them to infer fine-grained attributes of a user's query—such as its topic, language, or category—without ever decrypting the underlying data, as those attributes map to certain patterns of successes and failures within the speculative decoding process [4, 23, 33].

In this thesis, we investigate how speculative decoding and related inference-time optimizations create privacy vulnerabilities in real-world LLM deployments. We consider a realistic threat model in which the adversary is limited to passive network observation of encrypted network traffic, and has no access to model internals or plaintext outputs. We also assume the adversary has black-box access to the LLM in question and can send an arbitrary number of queries. Our primary contribution is a large-scale empirical evaluation of timing-based side-channel attacks against leading LLM APIs.

We collect timing traces from a range of 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 categories, and train a classification model on between 1,000-1,200 traces for each class. Across all models and scenarios, we achieve single-prompt classification accuracy between 55% and 70%, with top-3 accuracy between 80% and 90%.

These results show that speculative decoding and related inference optimisations 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. To conclude we propose practical mitigation strategies that preserve user experience and efficiency while limiting privacy leakage.

2 Background

This section provides the technical context required to understand timing side-channel attacks on LLM inference. We begin with a concise overview of transformer-based LLMs and their inference pipeline, introduce the performance optimisations commonly deployed by commercial providers, and then review classical timing attacks and the network-level timing side-channel created by *speculative decoding*.

2.1 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 [26]. 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—such as `''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 predicts the probability distribution of the next token x_t as $P(x_t|x_1, x_2, \ldots, x_{t-1})$, which we then sample from.

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, especially when high interactivity is required.

2.2 Timing Attacks and Side-Channels

A *timing attack* is a method of extracting secret information from a system by observing the amount of time it takes to respond to certain inputs. The core idea is that if the response time of an application depends on secret data, an attacker may deduce or infer this secret simply by measuring how long the computation takes, without needing direct access to internal computations or data. Timing attacks were first popularized by Kocher in 1996, who demonstrated that minor differences in the time taken by cryptographic operations (like RSA encryption) could leak secret keys [13]. Such attacks exploit the fact that certain computational steps—such as conditional branches, memory accesses, or arithmetic operations—can vary in execution time based on the underlying secret values.

In network-based scenarios, timing attacks typically involve an adversary who observes the timings of encrypted responses over a network, without decrypting the content itself. For example, Brumley and Boneh (2003) successfully extracted cryptographic keys from an OpenSSL server merely by measuring subtle differences in response times over the internet [2].

Some of the LLM optimisations discussed in the next section—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 *timing side-channel*. A side-channel leverages unintended information leaks arising from how computations are performed, and in this case, we exploit the difference in time between token generations.

2.3 LLM Optimisations

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 *quantisation*, 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 [6], typically with only a modest drop in performance. Post-training quantisation (PTQ) is frequently employed by open-source LLM providers to lower serving costs. Increasingly, pre-quantised models are released directly by research organisations using Quantisation-Aware Training (QAT), which allows the model to adapt to reduced precision during training and thereby minimises degradation [30].

Although Egashira et al. demonstrated that it is possible to construct models exhibiting adversarial behaviour *only* when quantised, such attacks require white-box access and are not generally exploitable in black-box, encrypted deployment settings [7]. More broadly, model-level optimizations like quantisation and distillation are not known to create observable timing side-channels in isolation, making them unlikely vectors for passive leakage, 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' interaction with the system. It has been shown that by abusing this timing difference, it is possible to extract a user's prompt verbatim [33].

- **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 [24].

As highlighted by Gu et al., most LLM providers now heavily restrict the prompt and KV cache [9]. Caching is often done on a per-user basis or at the organisation level, as opposed to across arbitrary users, and many LLM providers allow the user to specify if and when their prompts are cached. Given this, caching is unlikely to be a vector for leakage.

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 [14]. 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 $d_1, d_2, \dots, d_k$. While early methods generated these autoregressively, state-of-the-art techniques like EAGLE-2 can generate multiple draft candidates in parallel, increasing efficiency [15].

- **Forward Pass:** The *target* model then takes the original context P and the entire sequence of k draft tokens $(d_1, \dots, d_k)$ as input. It performs a single forward pass on this combined sequence. Due to the transformer architecture, we can calculate the necessary probability distributions for the next token at each position in one pass: $P_{target}(token|P)$, $P_{target}(token|P, d_1)$, ..., $P_{target}(token|P, d_1, \dots, d_{k-1})$, and $P_{target}(token|P, d_1, \dots, d_k)$.

- **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 [14]), 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 $d_1, \dots, d_{i-1}$ are kept, but d_i and all subsequent draft tokens $(d_{i+1}, \dots, d_k)$ are discarded. The target model then generates the correct i-th token (t_i) by sampling the already computed distribution $P_{target}(token|P, d_1, \dots, d_{i-1})$, 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 below in Figure 2.1.

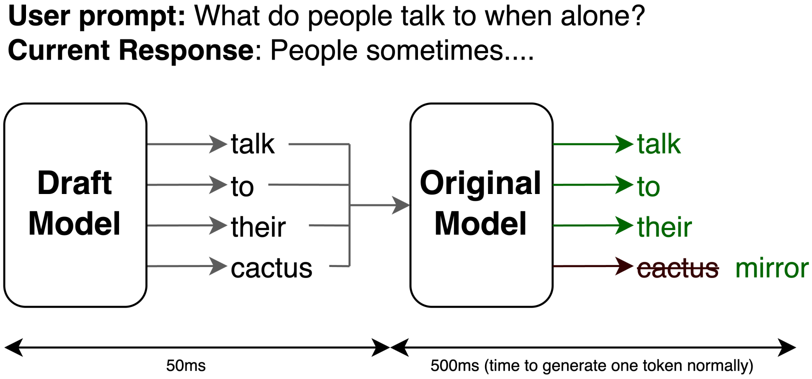

User prompt: What do people talk to when alone?
Current Response: People sometimes....
Draft Model
talk
to
their
cactus
Original Model
talk
to
their
~~cactus~~
mirror
50ms
500ms (time to generate one token normally)

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.

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 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 ”vote” on a prediction and terminating the forward pass when confidence is high [8].
- **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 utilised by EAGLE for draft token prediction [19].

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 [4].

3 Threat Model

Our threat model is similar to those proposed by Carlini & Nasr and Soleimani et al. [4, 23]. We assume there is a network adversary that can view encrypted communications between the 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, precise packet timings and packet sizes exchanged between the user and the LLM server.
- Could realistically be embodied by entities such as compromised ISPs, malicious routers, web browsers, insecure Wi-Fi access points, physically tapped network cables, or passive man-in-the-middle observers.

Throughout the thesis, we assume the adversary has *black-box access* to the system the victim will eventually use. That is, the adversary can make an arbitrary (albeit reasonable and cost-limited) number of queries to the system, which is primarily used during an offline phase to collect timing data from known inputs, allowing the adversary to train the classification models used in the attack. The adversary aims to infer sensitive, private information embedded within or implied by user queries, such as the user's query language, specific educational topics, or medical conditions. We assume the adversary is knowledgeable about:

- The format of input prompts to the LLM system.
- The potential classes or private fields present within the LLM-generated responses (e.g., possible 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 so train and test each class equally. 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.

Attack Feasibility. We assume the adversary can make a reasonable and cost-limited number of queries to the LLM system. In this thesis, we train classifiers on 10,000–12,000 traces per task, across ten classes. Though this is a large number of queries, in practice, the cost is at most $\$0.01$ per query for the models utilised, and so the total training cost is under $\$10$. More powerful frontier models (e.g., GPT-4o, Claude 3.7 Sonnet) impose significantly higher per-query costs. Though other research utilises frontier models [4, 23] and show that these attacks do work in those cases, we limit ourselves to cheaper models due to cost constraints. Classifier training requires a small amount of time (~ 15 minutes) on a modest 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* in the target setting. For example, the adversary may know that a user prompt follows a structured template asking for a medical diagnosis, a query in a certain language, or an educational question. Additionally, the adversary will know what potential categories those fall under. This is consistent with prior work [4, 23, 31] and with real-world usage patterns, where prompts often follow standardised patterns to improve performance [10] across a variety of domains [3, 17, 29].

As the adversary knows the query format, both 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 utilise a parallelised 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 → client packets and client → server packets,

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

Parallelism vs. Data Quality. Running eight queries simultaneously increases throughput nearly eight-fold but introduces head-of-line blocking at the provider: requests may stall briefly due to (per-user or provider-wide) capacity constraints, emitting *125-byte "processing" frames* to keep the connection open. 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 that keep the data quality similar to that in a non-parallelised 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 utilise a four-core virtual machine provided by Google Cloud in the US-East-1 region (South Carolina). We collect data through four-way parallelisation, to speed up the process without altering data quality, as discussed in §4.2.1. This helps ensure the classifier generalises 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 $\tau = 0.025\ s$ cleanly separates rounds across all target models; we use this value throughout. We find that rounds are often split amongst 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, utilise 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 (98%) accuracy. For a more detailed discussion on this assumption and its limitations, see §6.

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

- per-round counts, sizes, and durations (mean, median, std. etc.),

- inter-round gap statistics,
- global packet-size, IPD, and token-per-packet summaries.
- second-order metrics such as IPD accelerations and size-gap correlations.
- input prompt size statistics and ratios

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 target 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 [4]. 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.** (i) A four-layer MLP $(512, 128, 64)$ with ReLU activations and an adaptive learning rate; (ii)an XGBoost classifier with 200 trees and $max\_depth = 6$.

(2) **Meta-learner.** A logistic-regression layer fitted on five-fold out-of-fold probabilities from the base models.

4.5 Exploitation Phase

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.

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 utilised 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 diagnosis and pathology [25]. 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 1500 unique patient case descriptions from the DDXPlus dataset. These were split into a training set of 1200 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 ≈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 Behaviour and Analysis

Figure A.1 in the Appendix visualises 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 mirrors overlap between early-stage symptoms and indicates the side-channel captures not only individual pathologies, but coarse-grained disease families. Distinct conditions, such as *Anaphylaxis* and *Pulmonary embolism* are detected more accurately.

It should be noted that Soleimani et. al. utilised a similar approach in their attack, though they utilised 1,000 samples for each of the 10 categories in their network-based attacks [23]. Our attack does not utilise the distribution of diagnoses (instead making them equally likely), though otherwise is similar. They found top-1 accuracies of 51% for GPT-4o-mini and 71% for GPT-4o, though the values are not directly comparable due to the knowledge of a disease distribution—random chance for top-1 and top-3 are 18% and 42% in their scenario, as opposed to 10% and 30% in ours. They also conduct a substantially larger attack on a controlled system, which we discuss in §8.

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 [11]; the remaining nine languages —*English, Arabic, Bengali, German, Spanish, French, Hindi, Indonesian, Japanese*, and *Simplified Chinese*—come from the Multilingual MMLU dataset [20]. The Multilingual MMLU dataset contains the MMLU dataset translated into 14 other languages, and the 9 selected are the most widely spoken of those 14.

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*. And so, our test prompts were not just not seen during training, they also differ across languages, helping to test generalisation more effectively.

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.

5.2.2 Overall Accuracy

	Top-1 (%)	Top-3 (%)	Macro-F_1
GPT-4o-mini	63.30 ±1.74	89.63 ±1.14	0.63
DeepSeek V3 (0324)	72.33 ±1.63	91.20 ±1.07	0.72
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 $\approx 63 - 73\%$ accuracy—vastly higher than the 10% random baseline. DeepSeek V3 (0324) appears significantly more vulnerable than GPT-4o-mini, with a top-1 accuracy nearly 10% higher. This is similar to results seen in prior work by Soleimani et. al., where larger models were more vulnerable [23]. 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 Behaviour

Figure A.2 in the Appendix visualises class-level performance through a confusion matrix. We see that the European languages—German, English, Spanish, and French—are commonly misclassified for 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 a 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 from the *MMLU-Pro* [27] dataset—an extended, harder variant of the original MMLU benchmark that covers a wide range of academic fields. From the 13 disciplines available we select the ten largest: *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 utilise GPT-4.1's 'Structured Output' features for this, which enables consistent responses within a JSON schema. Each question was augmented at most once.

Augmentation involved asking the LLM to reword both the question and answer options whilst preserving all semantic information. We then shuffle the answer options.

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 utilise a boilerplate, instead asking 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 ±2.19	78.70 ±1.79	0.50
LLama 3.3 70B	53.60 ±2.18	83.25 ±1.64	0.53
Random Chance	10.0	30.0	—

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

5.3.3 Per-Class Behaviour and Analysis

Figure A.3 in the Appendix visualises class-level performance through a confusion matrix for both models. We see that some topics, like *Law*, are predicted with an ∼80% accuracy, whilst others, like *Biology* and *Physics* are below 40%. There does not appear to be any pattern between the proportion of the subject's dataset that was augmented, and the accuracy: Physics is one of two disciplines to use entirely original training data, and performs poorly across both models. Much like in previous cases, we see similar topics are often those with the highest rates of confusion, such as *Psychology*, *Health*, and *Biology*.

Another reason for lower accuracy may be to do with poor labelling. Many questions on the dataset are across multiple categories, or arguably miscategorised. 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 strong argument that this is better categorised as a *Mathematics* question, and there are many similar cases due to the inherent fuzziness of the categories themselves. That being said, the task itself may just also be significantly harder, but it seems likely that at least *some* of the lower performance is a result of cases like this.

5.4 Mitigations

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 the number of tokens in the response.

A simple solution is to use a *non-streamed* LLM service. Though the response size is visible, no round-by-round data is available. We also could delay tokens to stream at the rate of the original model. In both cases, the speed of the model and its interactivity are significantly harmed, and so are unlikely to be adopted.

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 reduced accuracy to be similar to random guessing [23]. It should be noted that the padding required would induce significant network overhead—the longest token in the GPT-4o tokenizer is 80 bytes, and for Claude it is 16 bytes. In that former scenario, generating $k = 5$ tokens via speculative decoding each round would require padding to 480 bytes of content, leading to wrapped packet sizes of around $850 - 900$ bytes, assuming we place all tokens in the same wrapper. Still, it would not alter user experience beyond this.

6 Limitations

6.1 JSON Packet Assumption

To reduce the scope of our paper, we focus on providers that utilise *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 6.1 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.

```
{
  "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. As the size of the JSON wrapper substantially exceeds the size of the token payload, an adversary can determine the number of tokens transferred in a single packet with high accuracy (around 98%). This behaviour holds for all OpenAI models and many open-source third-party providers.

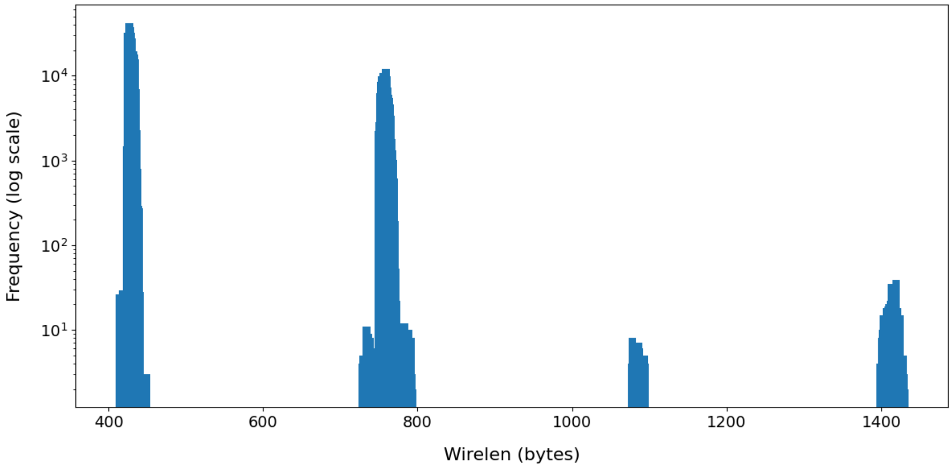


Figure 6.1: Observed packet wire lengths for 1 000 queries to GPT-4o-mini, up to 1500 bytes. Distinct peaks correspond to 1-4 tokens per packet.

6.1.1 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 response format conveys three tokens within a single wrapper. This reduces network overhead whilst maintaining comparability with infrastructure designed to use the per-token wrapper system. However, it significantly complicates token-count estimation. These three tokens are encoded as four characters, but it is easily possible for a single token to be the same length (e.g. `home`) or larger (e.g. `wheel`).

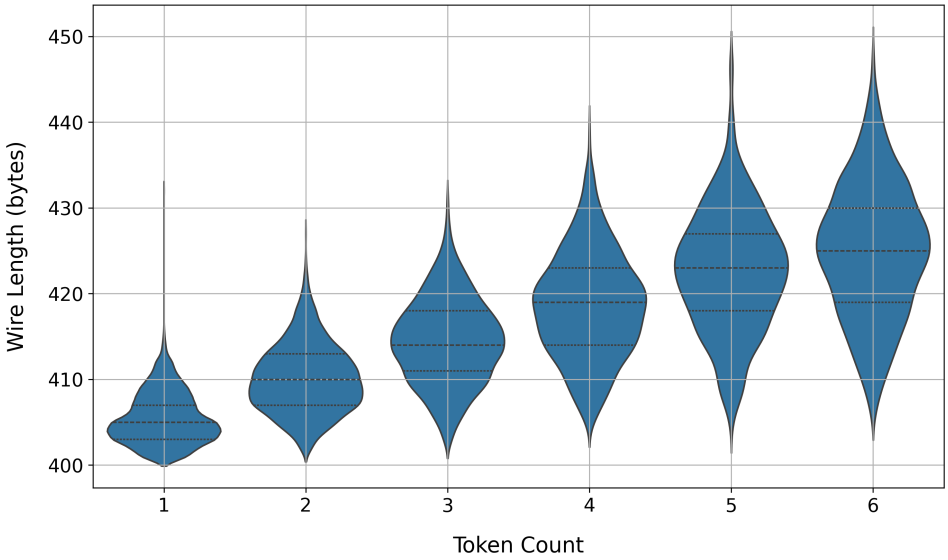


Figure 6.2: Token counts per packet for 100 queries to Llama 3.3 70B Instruct via Parasail (a multi-token wrapper provider). Overlap between token groups blurs boundaries, complicating exact inference.

We also note that some prior work, including studies involving earlier versions of Claude, observed a per-token wrapper behaviour similar to that of OpenAI models. Specifically, Carlini & Nasr [4] state that ”each token is contained within a large JSON object” for 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 for an adversary to determine the number of tokens within a packet at extremely high accuracy, they do not eliminate the underlying side-channel, and so it is likely we would see a slight degradation in attack accuracy in this scenario. Figure 6.2 visualises the resulting overlap for a multi-token provider.

6.1.2 Large Number of Tokens in Single Wrapper

Google’s Gemini utilises a unique system, where up to 64 tokens are included within a single packet. In a long response, the first few packets will contain an increasing number of tokens until the model streams 64 tokens at a time, up until the final packet, which contains an arbitrary number of tokens to finish the response. Much like in the per-token JSON wrapper scenario we restrict to, we know the number of tokens in most packets, and so it is likely this system is also vulnerable. However, given that each response contains 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 behaviour similar to that of OpenAI models. Soleimani et. al. [23] state that Gemini and ChatGPT both utilise a ”per-token JSON encoding” at the time of data collection. Though

ChatGPT still uses this system, it is clear that Gemini does not.

6.2 Other Limitations

Model & Provider Coverage: We utilise GPT-4o-mini, Llama 3.3 70B Instruct (Meta, served via Friendli), and DeepSeek V3 (0324). Different models, and different providers may expose different speculation pipelines and processes. Examples of alternative processes include: (i) multi-draft cascades (e.g., EAGLE-3), (ii) variations in early-exit, or (iii) token batching (similar to Gemini) could leak weaker—or stronger—timing signatures. Different models and providers may result in variations in accuracy, too.

Network Conditions: All traces were captured on two U.S. data-centre VMs. The adversary trained on an NYC-based Digital Ocean VM, and our victim a Google Cloud VM in the US-East-1 region (South Carolina). We do not explore cross-client generalisation across a wider variety of systems and network settings, but we posit 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. [23], in their speculative decoding attacks, find that 100ms of network jitter results in a misclassification rate of 4%, and 50ms results in 0% (i.e., no change).

Attack Pre-training Requirements: Our passive stage is zero-interaction, but the attacker still needs a supervised corpus to train the stacked classifier. We used 1 000–1 200 traces per class ($\sim$12 k–13 k total) to reach the reported 55–72% top-1 accuracies. We do not analyse the impact of significantly increasing or decreasing the number of traces, but existing research by Soleimani et. al. [23] shows that there are diminishing returns as we increase the number of training prompts, but returns nonetheless.

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

7 Future Work

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

JSON Packet Variation: As discussed extensively in §6.1, we restrict our attack to models utilising per-token JSON wrappers, and all existing research appears to utilise models using this setup. However, modern inference providers utilise both multi-token wrappers, where it is far harder to precisely determine the number of tokens in a packet. The impact of this change has not been quantified. This also applies to Google's Gemini, whose 64 token batching has not been analysed. Future work should determine the accuracy of attacks in these increasingly common scenarios.

Broader Model and Provider Survey: As stated earlier, different models and providers may utilise different speculation pipelines and processes. Future work should explore the impact of utilising different providers and models, and their effect on accuracy. Soleimani et. al. [4] found that larger models yielded higher accuracy, a result we replicated in the Educational Topic attack (see §5.3), but there are no results analysing different speculation pipelines.

Robustness Under Real-World Networks: An analysis of cross-client generalisation across different network architectures would help show the attacks robustness in different 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 analyse the rates of speculative decoding successes and failures by topic or category. Future work could analyse both the rates of success and failure, and determine if certain prompts or topics result in significantly higher rates. It is likely that the size and strengths of the draft model determine success rates in a given scenario, and it may be possible to design a draft model that minimises these variations whilst remaining useful, thus reducing vulnerability.

8 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.

8.1 Remote Timing Attacks on Speculative Decoding

Carlini and Nasr [4] 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. [23] 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 utilising the same DDXPlus dataset [25]. 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.

8.2 Output-length, framing, and other network channels

Zhang *et al.* [31] 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 utilise 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.

8.3 Cache-driven prompt leakage

Speculative decoding is not the only optimisation 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 [24].
- Gu *et al.* replicate the phenomenon and show a 95 % cache hit-rate detector with ten minutes of training data [9] against a variety of closed-source LLM APIs, for both prompts sent by the user and prompts sent by *other users*.
- PromptPeek proposes an end-to-end attack that iteratively brute-forces a victim's hidden system prompt one token at a time [28]. Similarly, InputSnatch uses a similar idea, combining cache-hit timings with replay to steal a user's input under realistic load [32].

Nowadays, most LLM providers heavily restrict caching [9]. Caching is generally done on a per-user basis or at the organisation level, and often needs to be explicitly enabled. This means that any exfiltration attack can only exfiltrate from the user or their organisation, not from arbitrary victim users.

8.4 Classical Timing and Side-Channel Attacks

Timing side-channels have long been recognised 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 [13]. Building on this, Brumley and Boneh [2] 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 [12, 16].

These attacks rely on the observations that performance optimisations 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.

9 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 remains cheap and scalable: $<$ \$10 total training cost per task for ≈12 k traces, < 15 minutes of GPU time, and zero victim interaction beyond passive packet capture (§4).
- Cross-model consistency suggests the leakage stems from *fundamental timing asymmetries* in speculative verification, *not* provider-specific quirks (§5).

Observed accuracies far exceed random baselines (10% top top-1 / 30% top-3). As speculative decoding is utilised 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 avenues for future work, such as: expanding the attack to multi-token JSON wrappers which providers such as Anthropic utilise; a broader model and provider survey; and determining the specific causes of draft model success and failure to potentially design more resistant draft models.

In conclusion, performance shortcuts like speculative decoding in LLM serving stacks are not free: they trade latency for *latent leakage*. As language models continue to permeate privacy-sensitive workflows, closing this timing channel is vital to preserving confidentiality that TLS 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] 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.

[3] 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.

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

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

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

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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/.

[29] 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.

[30] 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.

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

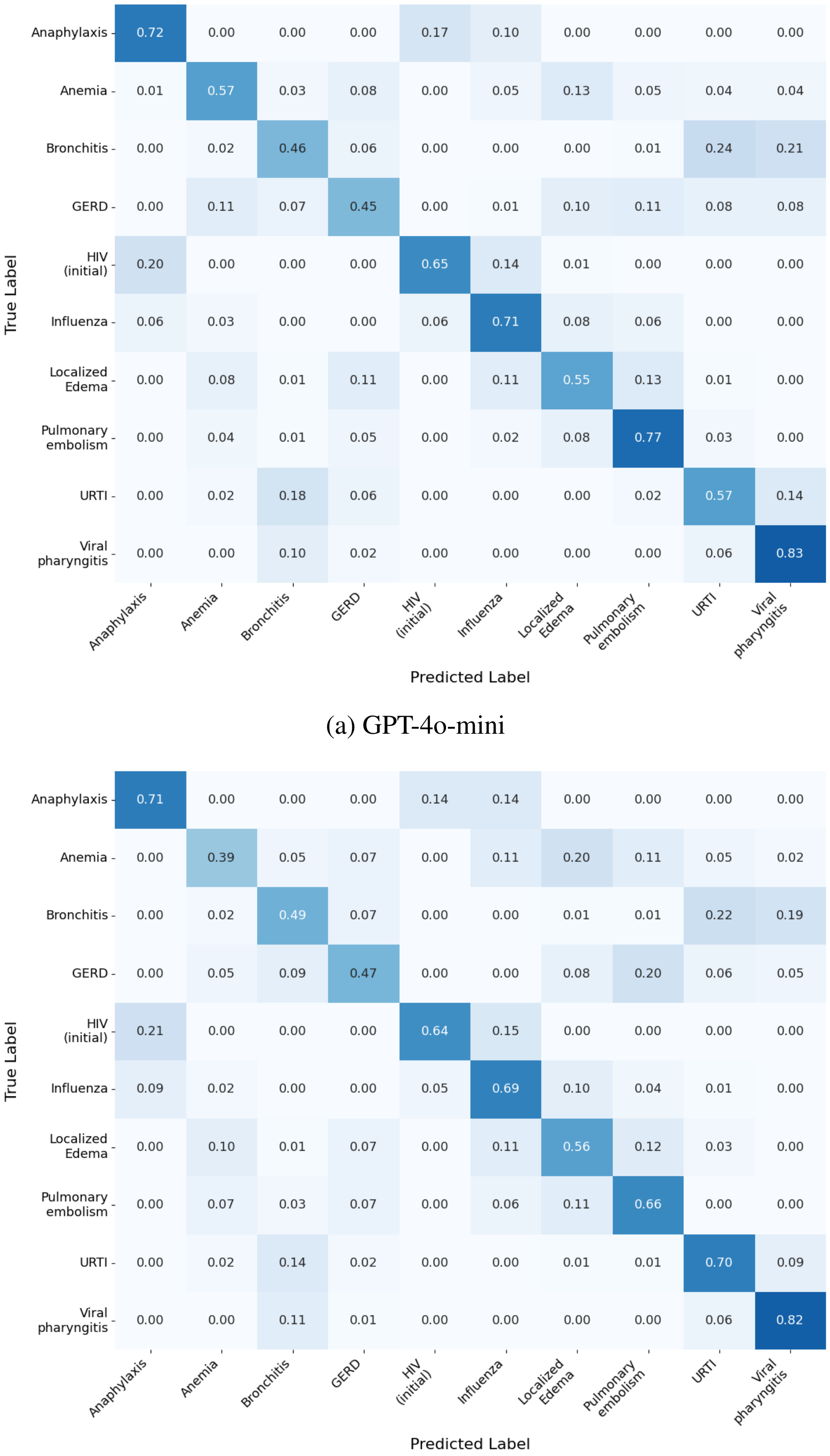
[32] Yiming Zhang, Nicholas Carlini, and Daphne Ippolito. Effective prompt extraction from language models. In *First Conference on Language Modeling*, 2024.

[33] 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.

A Appendix

A.1 Confusion Matrices

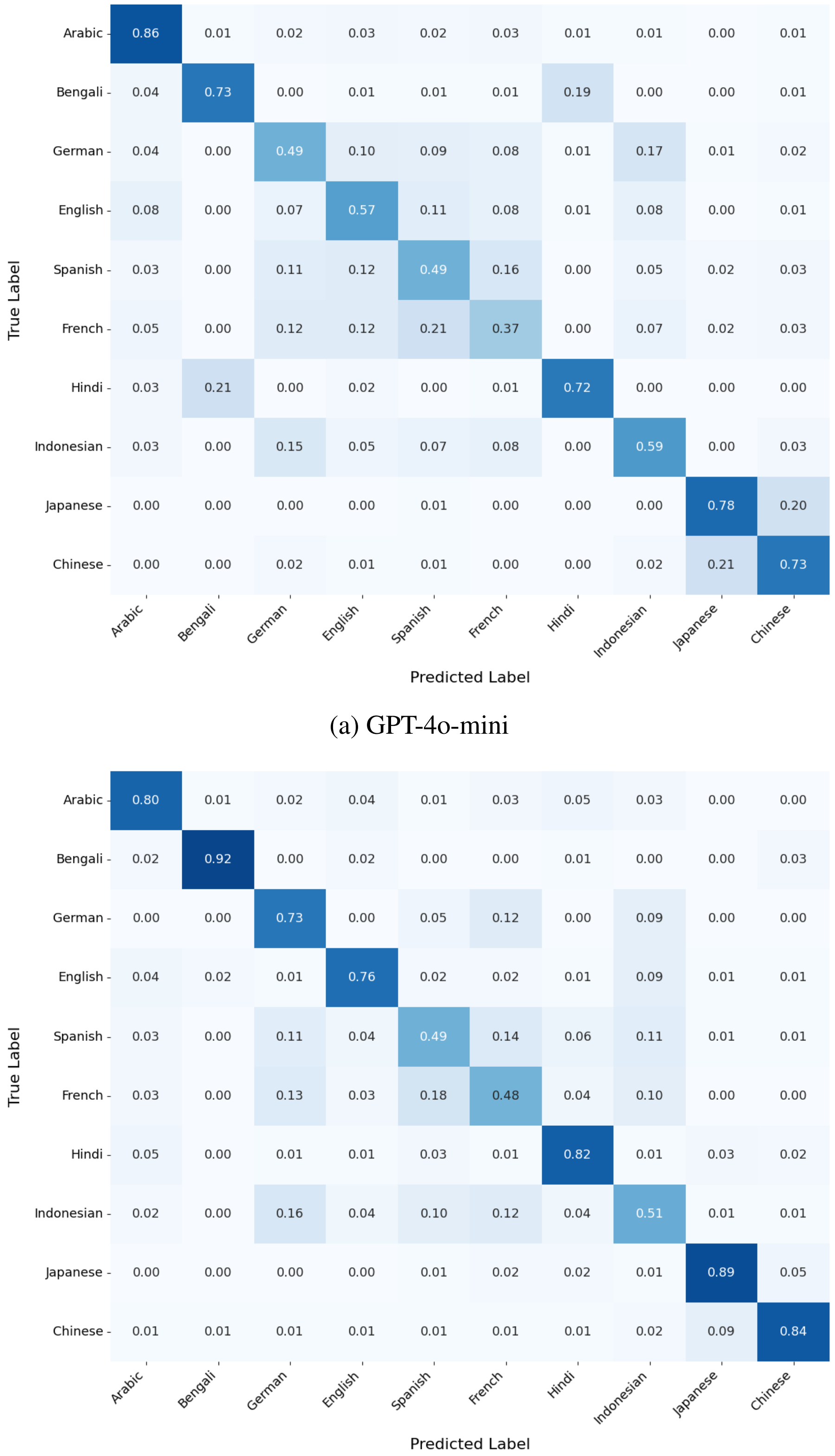
Figures on next pages.



(a) GPT-4o-mini

(b) Llama 3.3 70B

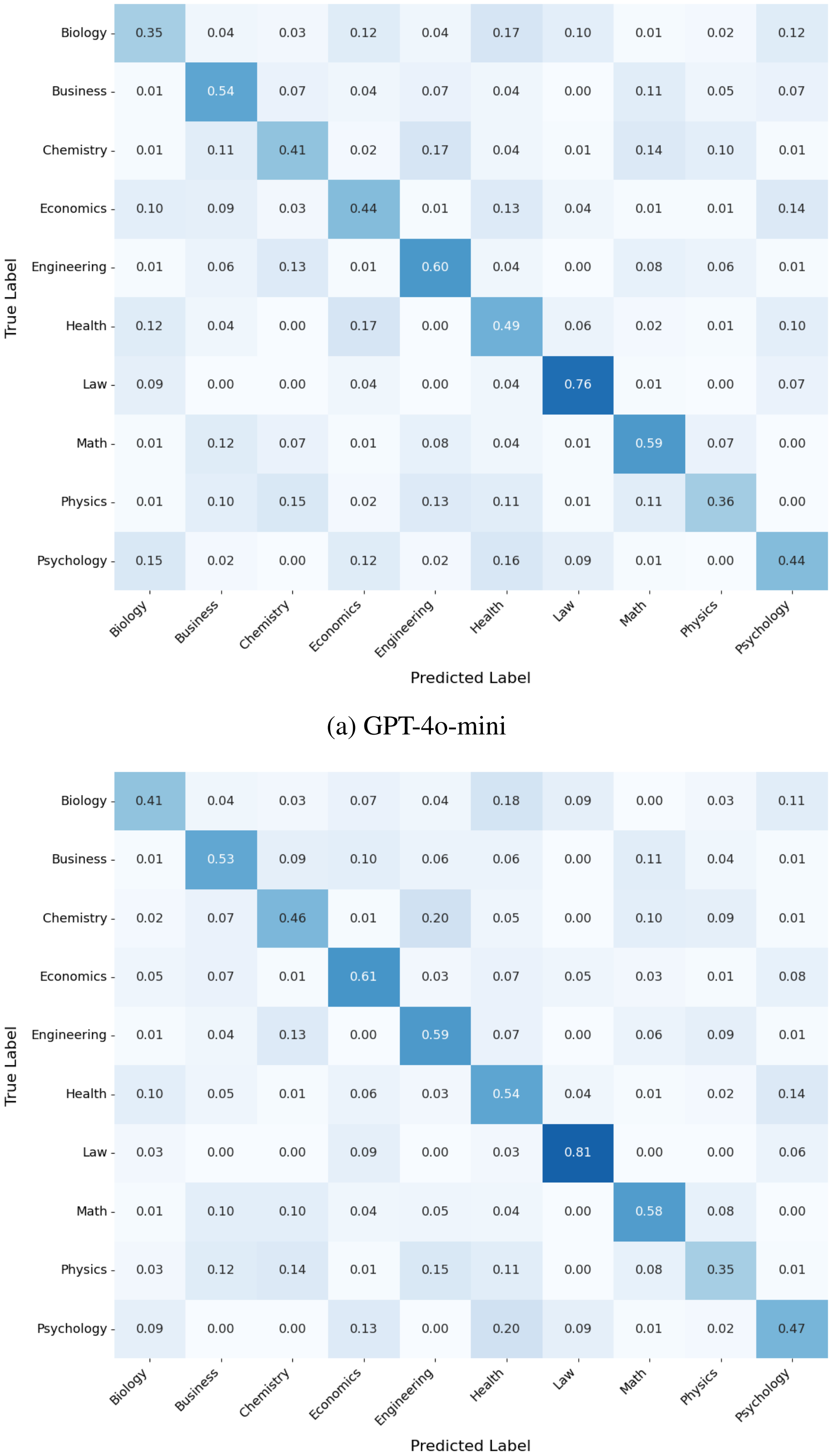
Figure A.1: Row-normalised confusion matrices for the stacked classifier on the medical task.



(a) GPT-4o-mini

(b) DeepSeek V3 (0324)

Figure A.2: Row-normalised confusion matrices for the stacked classifier on the language task.



(a) GPT-4o-mini

(b) Llama 3.3 70B

Figure A.3: Row-normalised confusion matrices for the stacked classifier on the educational top task.