TRACE-LLM: Traffic Recognition and Classification of Encrypted LLM Browser Services

Anonymous Author(s)

Abstract

Artificial intelligence companies have seen a rapid growth in the adoption and deployment of LLM services within the past five years, creating widespread availability for querying LLMs over the Internet. Many prior works have examined the privacy implications of such services. However, to the best of our knowledge, no prior work has addressed identifying LLM traffic within the much broader scope of all Internet traffic. Such a capability would aid network researchers to filter and examine LLM traffic from full packet captures and also help network administrators better monitor, troubleshoot, and provision their network based on increasing LLM traffic. In this paper, we propose an approach to reliably identify LLM traffic when it is mixed with non-LLM traffic in enterprise networks, even when payloads are encrypted. We first collect data, generating both LLM and non-LLM traffic using Selenium-based bots, totaling to over 14 million packets and over 10 GB of data. We abstract packets to flowlets, enabling our system to analyze finer-grained metadata features in a 5-tuple flow. We also examine directionality, differentiating between server-to-client (incoming) and client-to-server (outgoing) traffic along with various inter-arrival times. Finally, we train various ML models for three popular browser-based LLM services—ChatGPT, Gemini, and Claude, achieving accuracies ranging from $0.817 - 0.984$. Through in-depth analysis, we share several lessons learned and takeaways, as well as the defining traits of LLM traffic.

1 Introduction

Large Language Models (LLMs) and their applications have become a massively diverse market with many services available to the general public. Browser-based chat applications, like ChatGPT and Google Gemini, are one such service, and they represent a unique opportunity for evaluating LLM client-server communication over the Internet.

There are many reasons to have the ability to identify traffic generated by LLM usage. Just like any traffic classification work, such visibility is helpful for network administrators when managing their networks—monitoring, troubleshooting, traffic engineering, planning, and provisioning. Another prime use case is for educational environments, where there are times, such as exam blocks, where students should not query LLMs. Workplaces, like government contractors with security clearances, handling privileged information would want to restrict LLM usage as LLMs may train on their data. Performing standardized research on network administrator capability to identify LLM traffic flows is also important for understanding user privacy.

In particular, for LLM traffic research, many related works focus on extracting query, agent, or personal information via side-channel attacks that exploit traffic patterns [3, 16, 29, 30, 32, 33, 36]. Yet, an overwhelming body of research on the privacy and security behind browser-based LLM services makes a critical assumption that traffic is already classified as LLM traffic. This guarantee, however, does not exist in the real world—LLM traffic is typically encrypted and streamed alongside all the rest of the traffic on the Internet. Thus, a solution to this problem involves filtering LLM traffic from the *sea of non-LLM traffic*, enabling prior research to bridge from test environments to a broader set of applications on the open Internet.

Browser-based LLM traffic identification is challenging. IP address-based identification is meaningless due to the extensive usage of Content Distribution Network (CDN) for hosting LLM services, as well as the possibility of using VPNs. Destination port-based classification is irrelevant due to the use of HTTPS, and application payload-based identification is also hard, if not impossible, due to TLS encryption. In this case, the standard approach is to perform statistical analysis on flows to identify the defining traits in metadata features such as packet size, timing, and direction [8]. Past research shows machine learning (ML) models are powerful tools for traffic classification and identifying such defining traits [19].

To this end, we present a ML-based LLM traffic identifier system that can reliably filter LLM-generated traffic from a full packet capture that also includes a vast amount of non-LLM traffic. Our best model shows accuracies of 94.4%, 97.4%, and 98.4% for identifying Gemini, ChatGPT, and Claude traffic with F1 scores ranging between 90.2–93.0%.

Building such a reliable system is inherently challenging. First, traffic classification in general is difficult due to the limited availability of high-quality labeled data [10]. Second, identifying and extracting finer-grained features in a 5-tuple flow is nontrivial. In particular, it is unclear how many packets, bytes, or time intervals should be aggregated in a single flow to form a meaningful flow segment that successfully captures the defining characteristics of LLM-generated traffic. Consequently, an open question is what granularity of flow segmentation best serves as input to an ML model for both training and inference.

To address these challenges, we first create two Selenium-based bots, each for generating LLM and non-LLM traffic. Our non-LLM traffic generating bot visits the most visited top-100 websites and performs around 10 actions (e.g., open new page, enter search) per website while adding random artificial delay to mimic realistic human browsing behavior. For our LLM traffic generation, we manually create a *prompt bank* that consists of over 60 prompt chains from various categories. Each chain of prompts contains 1–5 prompts that are sent in the same conversation. Our bot selects random prompt chains from the bank to generate LLM traffic with ChatGPT, Claude, and Gemini.

For flow segmentation, we employ the *flowlet* abstraction to capture a finer-grained feature of a connection. The flowlet abstraction is a powerful notion successfully used in traffic engineering, switching, load balancing, and traffic analysis [15, 23, 27, 31]. We leverage this concept in traffic classification, which allows us to gain granular insight on flowlet-level characteristics in LLM and non-LLM flows.

We train and evaluate three different classification models: Random Forest, XGBoost, and SVM. We deliberately focus on traditional classification models as previous works show such models achieve high accuracy and inference performance while imposing low overhead, to the point where even real-time line-rate inference is possible via in-network ML systems that run on programmable switches [2, 13, 38, 39]. We argue that compute-intensive deep learning-based models, which requires GPUs, are not well justified for the given task and leave this as future work.

Our contributions are as follows:

- We build an automated data collection framework using Selenium to generate and capture LLM traffic across multiple providers and non-LLM traffic against top 100 webistes.
- Through the above pipeline, we introduce a large-scale, balanced dataset of 674,852 cleaned flowlets, consisting of approximately 50% LLM traffic and 50% non-LLM traffic, which will be publicly available after publishing.
- We design and implement an LLM traffic flow classifier capable of distinguishing general LLM-generated traffic from non-LLM traffic.
- Through in-depth analysis, we identify the defining traits of general LLM traffic. In particular, LLM traffic is characterized by a steady steam of *moderately* sized packets, typically a few hundred bytes, bottlenecked by an LLM service provider's generation cadence rather than by network (e.g., bandwidth, congestion control, etc) that the traffic flows on.

2 Background and Related Work

Side-channel attacks on encrypted LLM traffic: Several previous works focus on exposing user queries, the agent in use, or other personal information via side-channel attacks against encrypted LLM traffic. Many of these works exploit comprehensive traffic pattern analysis to extract such sensitive information [3, 16, 29, 30, 32, 33, 36]. However, such works all assume that LLM traffic is already identified, filtered, and given to their to their inference system. In reality, identifying and filtering only LLM traffic from a comprehensive packet capture is a non-trivial task that has not yet been established. A method that reliably filters only LLM traffic would expand the scope of these side-channel attacks to real-world traffic flows rather than just research experiments.

Traffic classification and website fingerprinting: Traffic classification methods arose with the goal of providing network administrators and ISPs with improved information such that they could better partition resources and optimize network performance [25]. Initial methods like port-based protocol identification and IP address tagging/flagging had success in early traffic classification approaches [20]. As more services shift to cloud/CDN providers and become more web-based, however, a shift to ML-based traffic classification methods began [18]. This set of methodologies establish a clear setup to begin identifying specific web pages or applications instead of general traffic patterns and protocols, even in the presence of privacy-preserving technologies.

Website fingerprinting (WF) is a well-researched approach to identifying applications via traffic flow analysis, application signatures, and classification models [25]. Prior research has found success using fingerprinting techniques for traffic classification even with Tor, demonstrating the feasibility of our goal of robust LLM classification regardless of Tor or VPN usage [22]. These methods are often tailored to specific web pages, however, and are thus not entirely applicable to our goal of identifying LLM usage across services—we instead focus on methodologies that approach general process identification (like streaming or browsing).

Another failure of initial WF methods is the overreliance on static statistical features which do not effectively harness the burstiness of TCP traffic. Liu et. al re-frame website traffic as a sequence of states through Markov modeling, which captures the time dependencies regardless of encryption [14]. This demonstrates network traffic as a structured, stochastic set of packets that changes in a temporal pattern related to application activity. We intend to abstract this workflow of remodeling encrypted traffic data and training models to a more generalized application of fingerprinting LLM versus non-LLM traffic, inferring that LLM client-server interactions likely have similarities in their packet patterns.

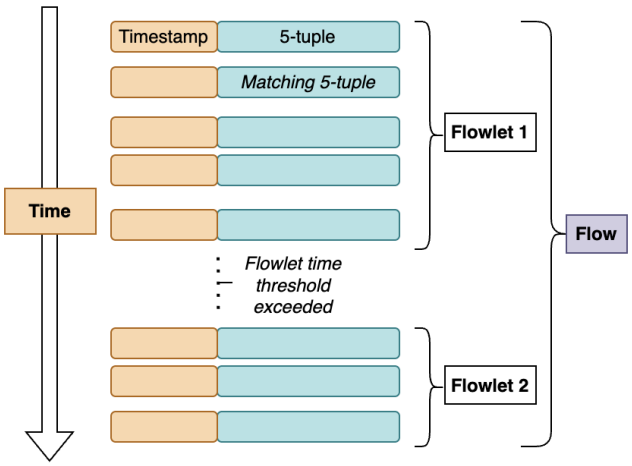


Figure 1: Classification of network traffic into flows and flowlets, used for more fine-grained insight into flow-level behavior.

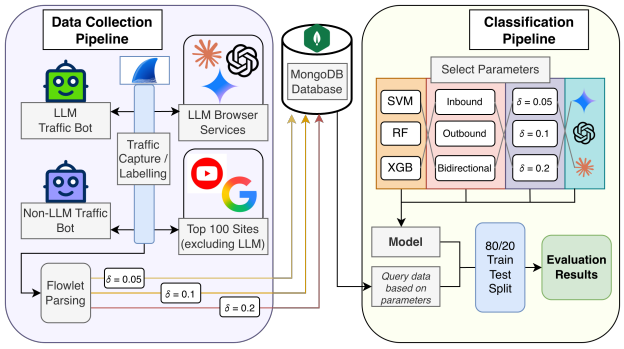


Figure 2: General approach to the LLM versus non-LLM classification problem; two pipelines (data collection and classification) address the separate problems of the lack of data and actual traffic classification.

3 High-level Approach and Design

In this section, we present the high-level approach and design of our system, focusing on how traffic is processed and features are extracted for model usage. Figure 2 illustrates our overall system design with two main components: data collection pipeline and classification pipeline.

3.1 Traffic Organization: Flowlets

Network packets can be classified into 5-tuples with timestamps regardless of encryption, where each 5-tuple contains src-IP, dst-IP, src-port, dst-port, and protocol. A traffic "flow" can be identified as all of the packets with identical 5-tuple characteristics, which represents a one-way connection (and all of the corresponding communication) between two end-hosts. When labeled by timestamp, flows can be partitioned according to a time threshold into "flowlets" that represent more fine-grained aspects of the connection, like individual messages as part of a larger conversation between two hosts, resulting in granular insight on flow-level behaviors [15]. Flowlets have been identified as powerful tools in traffic switching methods, as they more accurately harness the burstiness of TCP-based networking protocols - we thus approach the LLM versus non-LLM classification problem with a focus on flowlet-based models.

We have not found any substantial datasets of general LLM traffic that can be used for fingerprinting, however, so we establish our own data collection methods for automating collection of both LLM and non-LLM traffic data at three different flowlet timeout thresholds (section 4).

3.2 Feature Selection

After collecting an adequate dataset of LLM and non-LLM flowlets, we move on to selecting which features are best for distinguishing LLM from non-LLM flowlets. We began with over 15 features and narrowed down our model to 7 distinct features that were most representative of LLM flowlets. All of these features are related to the characteristics of the flowlet itself, the packets within the flowlet, or inter-packet relationships of the flowlet. The 7 features we selected were:

- Packet Size Standard Deviation
- Packet Size Mean
- Total Bytes
- Packet Count
- Inter Packet Time Standard Deviation
- Duration
- Inter Packet Time Mean

These features were selected because they capture the characteristics of flowlet behavior that are expected to differ between LLM and non-LLM traffic. In particular, packet size statistics (mean and standard deviation) and total bytes reflect the distribution of payload transmission. Features like packet count within a flowlet, inter-packet time statistics, and duration capture the overall volume of a flowlet as well as the burstiness and pattern of packet transmission within a flowlet.

These feature choices are also supported by prior work in encrypted traffic analysis and website fingerprinting, where statistical and temporal properties such as packet size distributions, inter-arrival times, and flow-level aggregation have been shown to be highly discriminative for classification tasks [12, 20, 21]. In particular, time-based features and burst-level characteristics have been widely used to capture behavioral patterns in encrypted traffic without relying on

payload inspection [12]. By focusing on these aggregated, flowlet-level features, we avoid reliance on payload inspection and instead leverage observable traffic patterns that are more robust to encryption and generalize across providers.

3.3 Model Selection

With the traffic structure organized and the feature space defined, we proceed to our classification approach, which splits our data and trains three types of machine learning classifiers commonly used in website fingerprinting methodologies (Section 5). Finally, models are assessed on typical ML classifier metrics like accuracy, recall, and F1 score (Section 6). Our suggestions for flowlet-to-flow aggregation are discussed alongside other future work and limitations in our discussion section (Section 7). Our methods are visualized in Figure 2.

4 LLM and Non-LLM Traffic Dataset

Our data pipeline aims to capture transport-layer packets and parse them into flowlets at three separate thresholds: 0.05 seconds, 0.1 seconds, and 0.2 seconds. We chose these thresholds to prevent packet reordering and preserve burst structure based on the idle gaps commonly implemented at around 50–100 ms in robust traffic machine learning papers [27]. As a result, we create a dataset of flowlets at each threshold and their corresponding statistical information. Additionally, each capture includes a mapping of IP addresses to LLM services, thus flowlets can be directly labeled as LLM or non-LLM. To manually label LLM flowlets, we leverage a temporary user provided `sslkeylogfile` to decrypt the HTTPS payload, allowing us to extract required information to determine ground truth LLMs flowlets. However, because most LLM services are hosted by CDNs such as Cloudflare, using known LLM IP address ranges to identify encrypted LLM flows is generally ineffective.

4.1 Data Collection

Our research artifacts and codebase will be published post-approval, and our dataset will also be publicly available.

Collecting non-LLM traffic: We follow Qian et. al's 2024 methodology for collecting mass amounts of non-LLM website traffic via a Selenium bot, with some adaptations made to better fit our use case [24]. A pre-queried list approximating top-100 websites (omitting LLM services), like *nike.com* and *youtube.com*, is fed to a Selenium bot, which visits these sites and performs around 10 actions on each one. These actions are geared towards generating additional network traffic (opening new pages, making inter-page searches, etc.) while incorporating artificial delay to more closely mimic human activity. An important note is that this activity was static and did not include more complex interaction like video streaming. Thus, we supplement the captures by *also* manually browsing the web with more complex activity at capture time. A Python capture script is run alongside the Selenium bot to capture all traffic on the specified interface using tshark (a CLI version of Wireshark), parsing flowlets and sending them to our privately-hosted MongoDB database. The final result consists of three documents—one each for 0.05, 0.1, and 0.2 second thresholds—containing the parsed flowlets for the traffic recorded on the specified interface during capture time. We also wanted to mix non-LLM data from another source for a robust baseline of representative background traffic, thus we requested access to the CAIDA UCSD Anonymized Internet Traces Dataset - 2019 [4], which was the most recent year available and is guaranteed to not contain any LLM traffic because it is prior to the public release of ChatGPT (2022). After receiving approval, we parsed the raw pcap files into flowlets using the same methodology and incorporated the data into our database. Since directionality is unknown we left this field empty; therefore, the CAIDA data do not play a role in the directionality ratio of Figure 4.

Collecting LLM traffic: First, we *manually* create a prompt bank with LLM assistance, querying an OpenAI *gpt-4o-mini* API endpoint to produce 62 entries. An entry in the prompt bank represents a chain of prompts, and each chain contains 1–5 prompts that are sent in the same conversation. Additionally, the entry has an overall category that describes the prompt chain, like debugging help or productivity planning. A Selenium bot, which operates in either a Chrome or Firefox browser, randomly selects a prompt chain and pastes the first prompt into the designated LLM service - this is done to reduce the impact of the script failing prematurely and thus executing the first prompts disproportionately. In this study, we use ChatGPT, Claude, and Gemini. After a random delay for the response to be generated (injected to more accurately mimic human-LLM interaction, the bot enters the next prompt in the prompt chain. The bot continues until all prompt chains selected and used, an error occurs, or manual halting. The same Python capture script used for non-LLM traffic runs in parallel to the Selenium bot to collect the device's network traffic. However, this time the script additionally identifies and collects the particular LLM service by IP address used for each prompt. Such label information is required for model training purposes later. These scripts are presented as pseudocode in Algorithms 1 and 2 in the appendix.

4.2 Scalable Data Storage and Retrieval

A document database such as MongoDB serves as the central authority for storing parsed network captures and their derived flowlets. During the flowlet parsing stage, the pipeline persists each capture's metadata together with the full set

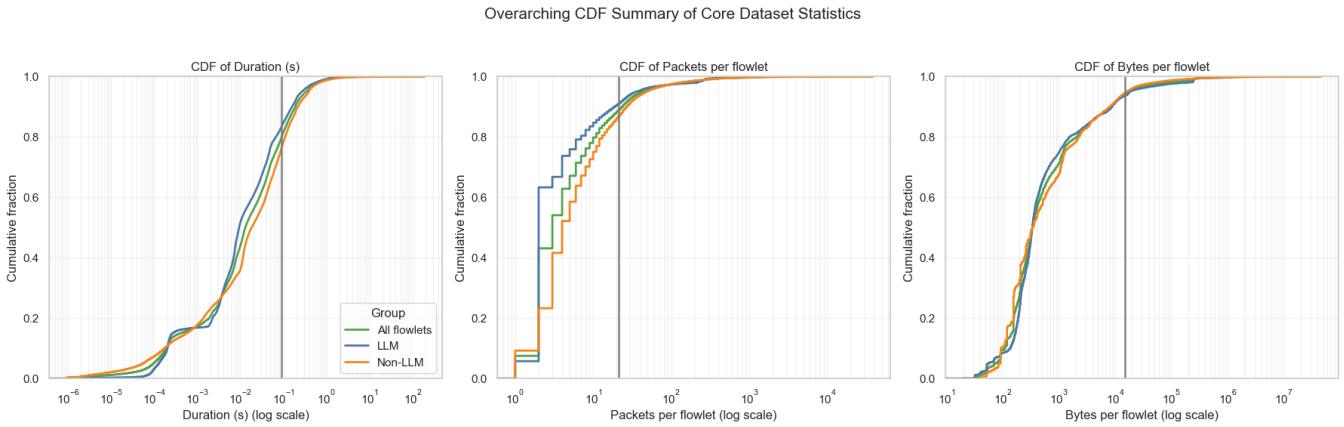


Figure 3: CDFs of key dataset metrics (mean values in gray)

of flowlet-level features into a single MongoDB document, enabling downstream analysis and supervised model work without requiring repeated recomputation. In the codebase, the MongoDB representation is implemented via dataclasses designed to mirror the project's expected in-memory schema, which provides a consistent structure for ingestion and later export, including fields for traffic identity, timing characteristics, statistical summary features, and model-related labels or predictions.

4.3 Summary of Dataset

The MongoDB schema is capture-centric, with each capture stored as a single document keyed by file path. Each document contains metadata (timestamp, status, endpoint-to-LLM mappings) and an embedded array of flowlets, where each flowlet encodes flow identity, annotations (e.g., traffic class, LLM label), directionality, and aggregated timing and size statistics. Optional fields support model predictions and ground-truth labels. To reduce storage overhead, raw per-packet sequences may be omitted in favor of summary statistics, with future work needed to evaluate the impact on the downstream results of this choice.

For model development, subsets of the cloud database are exported as local snapshots. Each snapshot consists of a manifest file describing the export and a directory of per-capture JSON documents that preserve the original nested structure. The export process supports query-based filtering (e.g., by time range or LLM label), enabling controlled dataset construction and reproducible offline training.

Table 9 provides a high-level look at our datasets numbers while Figure 4 visualizes our packet and flowlet count direction ratio for LLM and non-LLM traffic. Our resulting dataset contains 674,852 total flowlets split roughly evenly between LLM (334,428; 49.6%) and non-LLM (340,424; 50.4%) traffic, indicating a balanced binary classification benchmark. The

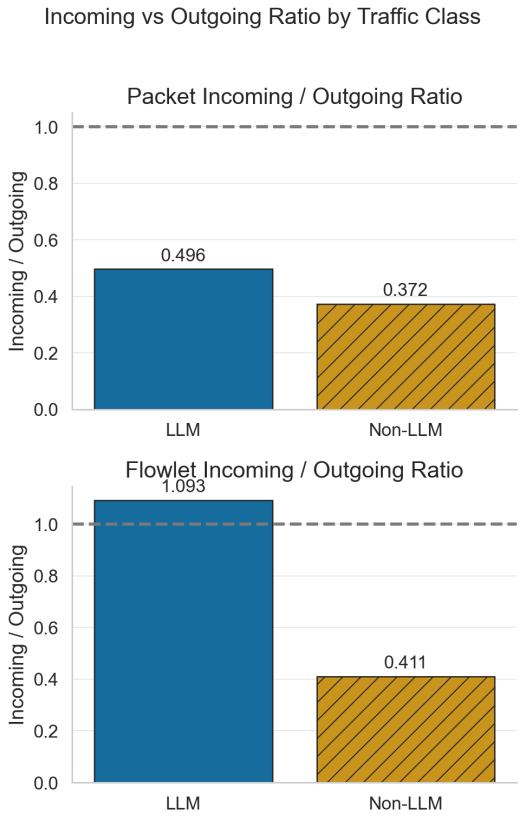


Figure 4: Direction cardinality comparison between LLM and non-LLM traffic classes - the first figure shows the packet count ratio while the second figure shows the flowlet count ratio.

threshold versus flowlet count split followed an expected linear trend, with a 269,419 (41.02%) / 213,001 (32.43%) / 174,400 (26.55%) split for 0.05s, 0.1s, and 0.2s accordingly. Among

Table 1: Core dataset statistics

Traffic class	Flowlets	Dur. mean	Pkts mean	Bytes mean
All flowlets	674,852	0.082	21.072	15,001
LLM	334,428	0.061	16.875	13,260
Non-LLM	340,424	0.102	25.195	16,711

LLM flowlets, Gemini contributes the largest share (129,408), followed by Claude (108,807) and ChatGPT (96,213) - we expect this to be due to Gemini having more spaced-out packets that accrue a higher quantity of total flowlets. Figure 4 shows that directionality of LLM traffic differs between packet counts and flowlet counts; there are more outgoing packets from the user's perspective and roughly symmetric incoming/outgoing flowlets. Non-LLM traffic instead has a similar packet and flowlet incoming/outgoing ratio with more outgoing in both measurements. Across both LLM and non-LLM traffic, the protocol composition is heavily TCP-dominated. UDP, ICMP, and other protocols contributed negligibly due to *all* network traffic to/from the machine being recorded alongside Selenium-driven data during non-LLM captures (capturing artifacts like routing protocols and any other user activity on the machine, leading to more a representative dataset). This protocol distribution is expected, given that modern web services (LLM or otherwise) communicate almost exclusively over HTTPS.

5 Selection of Classification Models

The selected classifiers—Random Forest, Support Vector Machine (SVM), and XGBoost—were chosen to represent and evaluate a diverse set of machine learning models commonly used in network traffic classification. Prior work has proven that these models achieve strong performance in tasks such as traffic classification and anomaly detection in networks. In particular, ensemble methods such as Random Forest provide robust performance on high-dimensional and noisy network data [1], while SVMs are effective at capturing complex, non-linear decision boundaries [7]. Gradient boosting approaches such as XGBoost have further demonstrated state-of-the-art performance on structured datasets due to their ability to model intricate feature interactions [6]. These models are widely adopted in the network security and traffic analysis literature, where they achieve high accuracy across a variety of classification settings. Lashkari et al. (2017) employ a Random Forest classification methodology to characterize Tor traffic flows, Panchenko et. al (2016) use SVM models to greatly improve Tor classification methods, and Sheikhi & Kostakos (2024) fingerprint malicious websites using XGBoost classification models—all model works illustrating how these classifier choices are suited for the task [9, 21, 26].

6 Evaluation

In this section, we evaluate the effectiveness of our proposed framework for detecting LLM traffic in encrypted traffic. We present the classification model's performance, the impact of directional modeling, feature importance, and robustness across different flowlet configurations.

We evaluate our models on the dataset described in Section 3, consisting of both LLM-generated traffic (ChatGPT, Gemini, Claude) and non-LLM background traffic. The dataset is partitioned into training and testing sets using an 80/20 split. For model analysis, we use four different metrics: *Accuracy*, *Precision*, *Recall*, and *F1 score*. We conduct all experiments across three flowlet partition thresholds: $\delta \in \{0.05, 0.1, 0.2\}$ seconds.

6.1 Overall Classification Performance

Table 2 summarizes model performance across LLM providers and types of machine learning models for incoming flowlets at the 0.1s flowlet timeout threshold.

Table 2: Overall incoming-traffic model performance on LLM flowlet classification (flowlet timeout threshold $\delta = 0.1$s). Best metrics by provider are highlighted in green.

Model	Accuracy	Precision	Recall	F1
Gemini Random Forest	0.944	0.964	0.899	0.930
Gemini XGBoost	0.941	0.961	0.896	0.927
Gemini SVM	0.867	0.916	0.749	0.824
ChatGPT Random Forest	0.974	0.951	0.865	0.906
ChatGPT XGBoost	0.970	0.916	0.866	0.890
ChatGPT SVM	0.883	0.835	0.218	0.346
Claude Random Forest	0.984	0.902	0.902	0.902
Claude XGBoost	0.983	0.895	0.902	0.899
Claude SVM	0.950	0.731	0.637	0.681

Takeaway #1: Tree-based models outperform linear models when filtering LLM from non-LLM traffic. Tree-based models (Random Forest and XGBoost) consistently outperform SVM across all metrics. Overall, Random Forest performs the best across all LLM providers. This result is consistent with previous works that found that ensemble methods such as Random Forest and Gradient boosting approaches such as XGBoost provide robust performance on high-dimensional and noisy network data [1, 6].

Takeaway #2: Models show high accuracy and precision, but low recall. Overall, the results for the three providers show that the accuracy of each model is consistently higher than its corresponding precision, followed by a more significant drop in recall. Additionally, the low recall results are

the driving force lowering F1 scores across all of our models, providers, and timeout thresholds. In the use case of identifying LLM traffic, we argue *precision* is more important than *recall*. In other words, it is critical to be correct when a traffic is flagged as LLM traffic, i.e., have a low false positive rate. It is worse to incorrectly drop, throttle, or perform LLM traffic analysis on non-LLM traffic than missing some portion of LLM traffic.

6.2 LLM Provider Comparison

Across LLM providers, there are meaningful differences in how distinguishable LLM traffic is even though they were evaluated under the same feature representation and classification framework.

Takeaway #3: ChatGPT is easiest to identify, followed by Claude and Gemini. ChatGPT traffic is consistently the easiest to identify. It achieves the strongest balance between precision (0.936) and recall (0.732), indicating that its flowlets exhibit more stable and separable patterns. ChatGPT responses appear to follow more regular timing structures, which likely contributes to more deliberate classification. Claude traffic is much more difficult to detect. Both the precision and recall drop to 0.768 and 0.627 respectively, indicating that a large fraction of Claude flowlets are misclassified as non-LLM traffic. Although the accuracy is still as high as 0.944, the lower recall indicates more false negatives for Claude as a provider. Gemini's results show a lower precision than ChatGPT, but a higher recall. This indicates that there may be more false positives, but there are also less false negatives identified in this model. Although this tradeoff is notable a precision score of 0.914 and recall of 0.867 are still strong in this case.

6.3 Classification with Combined Model

Each of the three per-provider models described in the previous sections is a binary classifier answering a narrow question: *does this flowlet look like traffic from my provider's LLM?* Any single expert has only seen exactly one kind of LLM traffic in training, so its recall on the full LLM population is inherently capped. To recover the more general question—*is this flowlet LLM traffic at all?*—we train two mixture-of-experts "overseers" on top of the three existing models and compare them: *max-probability overseer* and *stacked overseer*.

The *max-probability* overseer declares LLM traffic whenever $\max_e p_e(x) \geq 0.5$. The *stacked* overseer instead feeds the expert probabilities plus five aggregate statistics (max, min, mean, std, spread) into an XGBoost meta-learner trained on group-aware out-of-fold predictions (5-fold `GroupKFold` on the flow 5-tuple). As a control, we also train a single *all-LLM* expert that reuses the same feature pipeline but sees every LLM provider at once during training; this expert is evaluated standalone and also added as an optional fourth input to both overseers. The train/test split holds out 20% of flows, yielding 76,499 training and 19,420 test flowlets on our snapshot.

Table 3 summarises every strategy on the shared test split.

Table 3: Test-set comparison of all classification strategies (19,420 flowlets, 6,980 LLM). The three per-provider experts are the level-0 inputs to both overseers. The "4-expert" rows additionally include the all-LLM expert. OS = overseer.

Strategy	Acc.	Prec.	Recall	F1	ROC-AUC
Expert: ChatGPT	0.789	0.920	0.451	0.605	0.916
Expert: Claude	0.768	0.905	0.394	0.549	0.919
Expert: Gemini	0.751	0.900	0.344	0.498	0.876
Single model: all-LLM	0.909	0.852	0.879	0.866	0.966
Max OS (3 experts)	0.897	0.893	0.810	0.850	0.965
Stacked OS (3 experts)	0.897	0.866	0.843	0.855	0.965
Max OS (4 experts)	0.902	0.849	0.886	0.867	0.966
Stacked OS (4 experts)	0.898	0.866	0.846	0.856	0.965

Both overseers lift recall roughly 35–45 absolute points above any individual per-provider expert and reach essentially the same operating point (F1 $\approx$ 0.85, AUC $\approx$ 0.965). Adding the all-LLM expert to the mix does not meaningfully move the overseers either: the 4-expert max rule gains 0.012 F1 and the stacked model gains 0.002 F1 over their 3-expert counterparts. The XGBoost meta-learner's feature importances make the reason explicit: mean p and $p_{\text{all\_llm}}$ jointly absorb 88% of the split weight, while p_{chatgpt}, p_{claude}, and p_{gemini} together account for less than 4%.

Takeaway #4: Mixture of experts is not worth the extra lift. A single all-LLM model matches the best ensemble (F1 = 0.8655 vs. 0.8668 for the 4-expert max rule and 0.8560 for the stacked overseer) while requiring far less memory and computation. Any unique contribution of the three per-provider experts is subsumed by the generalist as soon as it is trained on their pooled data; this is also why the stacked overseer learns to weight $p_{\text{all\_llm}}$ and mean p over the provider-specific probabilities (see Table 4).

A mixture of experts may pay off more in a regime where each provider exposed genuinely distinct, non-overlapping packet-level patterns, and where the dataset was large and varied enough that a single model could not absorb the union of those patterns; with the traffic captured here, ChatGPT, Claude, and Gemini flowlets share enough common structure that a single classifier trained on all three generalizes at least as well as any combination of narrower experts. We therefore recommend the single all-LLM model for downstream use and treat the mixture-of-experts pipeline as a negative result that justifies the simpler design.

Table 4: Feature importances reported by the XGBoost meta-learner in the 4-expert stacked overseer.

Feature	Importance
mean p	0.4547
max p	0.4223
$p_{\text{all\_llm}}$	0.0652
p_{chatgpt}	0.0168
p_{claude}	0.0136
std p	0.0112
p_{gemini}	0.0078
min p	0.0054
max p − min p	0.0030

6.4 Directional Model Analysis

To evaluate the impact of directional modeling, we compare three approaches:

- Incoming-only classifier (LLM → client)
- Outgoing-only classifier (client → LLM)
- Combined classifier using $LLM = I \vee O$

Table 5: Performance comparison of best directional models

Model Type	Accuracy	Precision	Recall	F1
ChatGPT Best Incoming	0.967	0.903	0.689	0.782
ChatGPT Best Outgoing	0.957	0.869	0.577	0.693
Gemini Best Incoming	0.927	0.847	0.852	0.850
Gemini Best Outgoing	0.941	0.833	0.722	0.774
Claude Best Incoming	0.948	0.673	0.421	0.518
Claude Best Outgoing	0.974	0.863	0.496	0.630

Table 5 presents the performance of directional models across providers. Overall, incoming and outgoing traffic exhibit distinct classification characteristics, highlighting the asymmetric nature of LLM communication.

Incoming-only models generally achieve stronger recall, particularly for ChatGPT and Gemini, suggesting that server-to-client response traffic contains more distinctive temporal and structural patterns. In contrast, outgoing-only models often achieve higher overall accuracy but suffer from reduced recall, indicating that client-to-server request patterns are less reliable indicators of LLM usage.

Provider-specific trends further reinforce this observation. For ChatGPT, incoming traffic yields the strongest detection performance, while for Gemini and Claude, outgoing models achieve higher accuracy but continue to struggle with recall. This suggests that while request-side patterns may be consistent enough for general classification, they lack the richness needed to robustly capture LLM behavior.

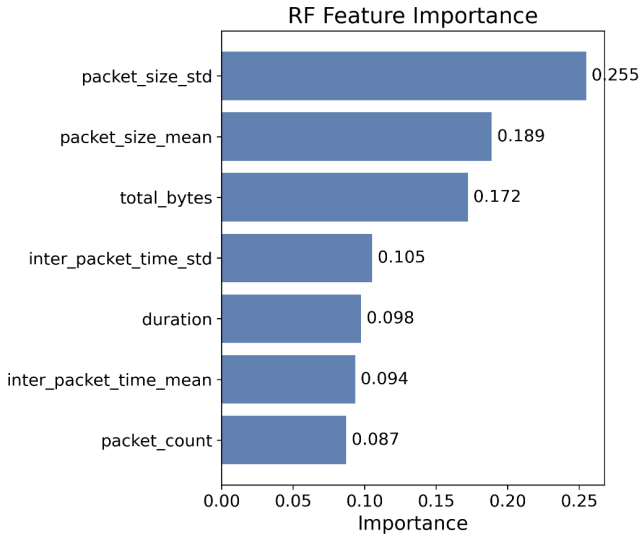


Figure 5: Feature importance ranking for ChatGPT classification model.

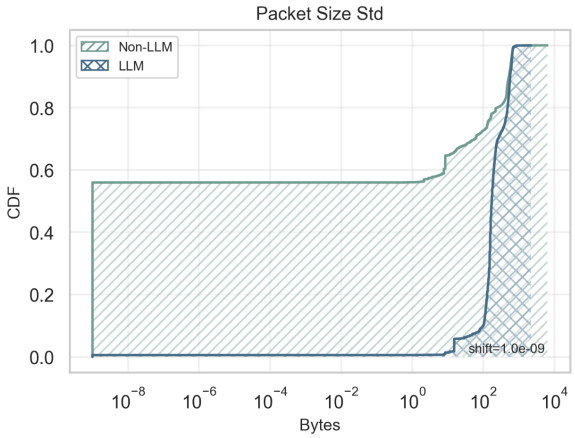
Takeaway #5: Incoming, or server-to-client, traffic has a more distinctive pattern than outgoing: Taken together, these results confirm that LLM traffic is inherently asymmetric. Bidirectional models, which combine both incoming and outgoing information, consistently provide the best balance between precision and recall by capturing a more complete representation of LLM communication patterns.

6.5 Feature Importance

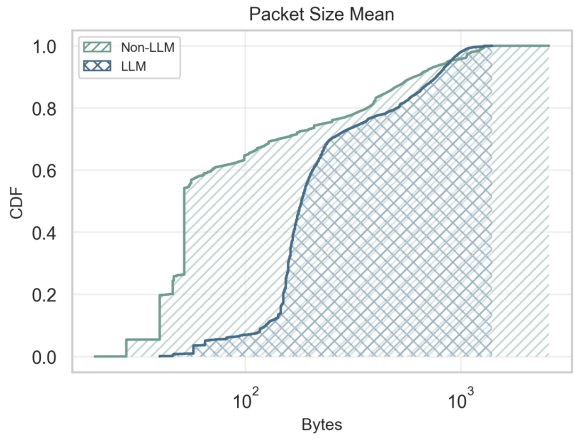
We analyze feature importance to understand which traffic characteristics contribute most to classification performance.

Figure 5 illustrates the relative importance of features. Numerous testing and analysis shows that the most critical feature for LLM traffic identification across LLM providers is the *packet size*—in particular, the standard deviation and mean packet size within a flowlet. This shows that the mean variation in the packet size of LLM flowlet is representative of the traffic when compared to non-LLM traffic.

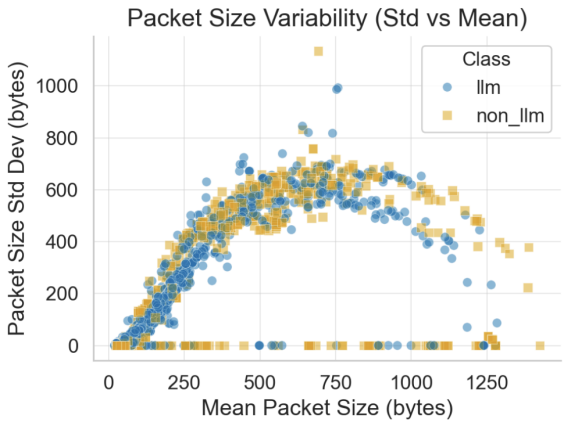
Takeaway #6: Packet size and its distribution are the most defining features of LLM traffic: The distribution of packet size standard deviation and packet size mean within a flowlet are shown in Figure 6. The CDF for standard deviation of the packet size shows that LLM flowlets are tightly concentrated within a narrow range, with over 90 percent of values falling between roughly 100 and 700 bytes. In contrast, non-LLM traffic exhibits a much wider spread, with over half of values concentrated near very low standard deviations (close to 0) and about 40 percent distributed between 50 and



(a) Packet size - standard deviation



(b) Packet size - mean



(c) Relationship between size and size variance of packets for LLM vs. non-LLM traffic

Figure 6: Distribution of packet size standard deviation and mean for ChatGPT flowlets.

200 bytes. This indicates that LLM traffic tends to have more consistent variability in packet sizes within flowlets, whereas non-LLM packet size variability is either near 0 or a value between 50 and 200 bytes.

Figure 6 also indicates that the mean packet size of ChatGPT flowlets is shifted toward higher values compared to non-LLM traffic. The CDF shows that 60 percent of LLM packet means are concentrated in the range of 150 to 250 bytes with the remaining means reaching up to 1000 bytes. Non-LLM packet size means, however, are generally much smaller, with almost 60 percent of means being below 50 bytes. Although non-LLM traffic reaches higher maximum values, the majority of its distribution lies below that of LLM traffic, reinforcing that LLM flowlets generally consist of *larger packet sizes* on average.

6.6 General Results

Model Type Comparison: Random Forest achieves the strongest and most consistent performance, with accuracy exceeding 91% in the combined setting and reaching up to 94% for individual providers. XGBoost performs similarly, with slightly lower but comparable F1 scores. In contrast, SVM exhibits significantly lower recall across all settings, in some cases dropping below 0.30, which results in substantially reduced F1 scores despite moderate precision.

False Positives and False Negatives: A key aspect of our classification scheme is the granular objects under analysis. Since we are using the component flowlets of an overall flow, the cost incurred by false positives or false negatives is not as severe as if we were classifying flows as a whole. A realistic application of this work for identifying flows as being LLM or non-LLM would use a threshold, where flagging a ratio of flowlets above the threshold as LLM flowlets indicates that the entire flow is LLM traffic. Thus, isolated false negatives or positives will not lead to misclassifying an entire flow.

Flowlet Threshold Sensitivity: We evaluate how the choice of flowlet timeout threshold δ impacts classification performance. Figure 8 illustrates the relationship between the timeout threshold and precision, recall, and F1 score. Results at the 0.1s threshold are the most consistently effective (in line with the rest of our evaluation). Interestingly, the precision and recall results seem to "stabilize" at the 0.1s threshold, whereas the 0.05s and 0.2s thresholds see recall and precision deviate much more from each other in the same models. This analysis highlights the tradeoff between temporal granularity and noise in flowlet construction, with 0.1s seeming to be a "sweet spot."

6.7 Summary of LLM Traffic Characteristics

Across every experiment in this section, the same picture emerges: LLM traffic is consistently identifiable because the

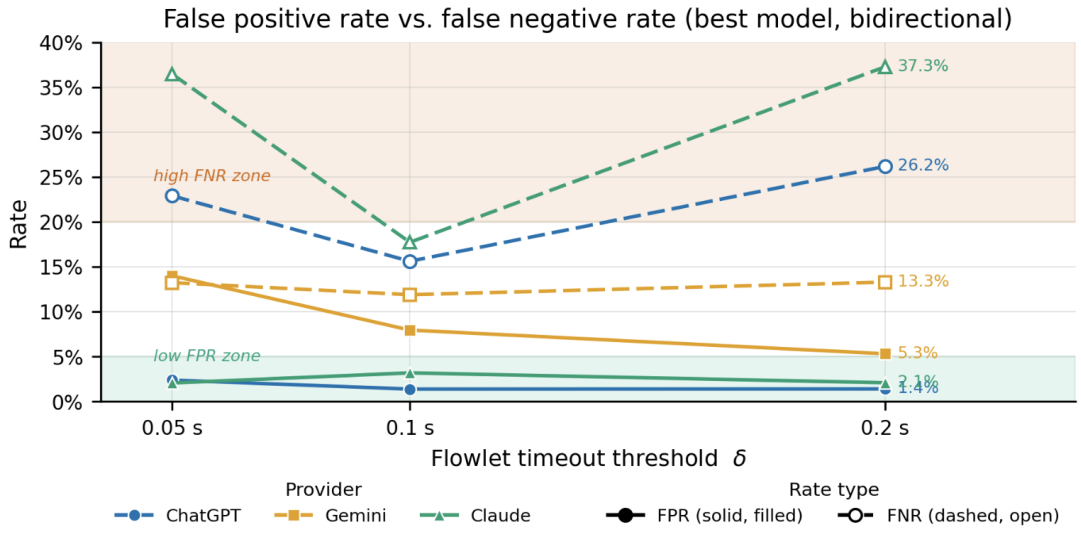


Figure 7: Plot of false positive and false negative rates across thresholds for the best bidirectional classifiers. Results seem to stabilize along the 0.1s threshold similarly to classification metrics.

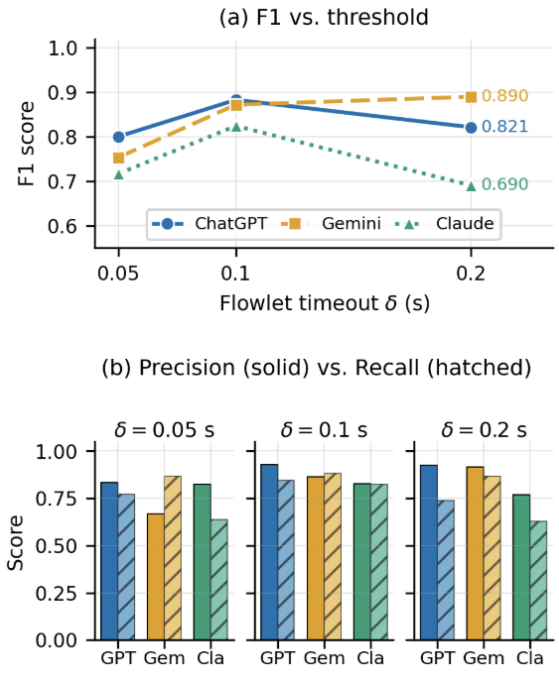


Figure 8: Threshold impact on F1 score (upper), precision, and recall (lower) for best bidirectional models.

way LLMs *deliver* a response to the user is fundamentally different from non-LLM services move data. A normal web flow either ships a payload as quickly as the link allows, producing a small number of large, MTU-sized packets, or it carries short, sporadic control traffic with very small packets and long idle gaps. An LLM response sits in a third regime that neither pattern captures well. Because tokens are generated one at a time and streamed back as they are produced, the server emits a steady train of *moderately* sized packets (typically a few hundred bytes), bottlenecked by the model's generation cadence rather than by the network's congestion control. When compared to other streaming traffic, such as video or music streaming, the relatively slow throughput of LLM responses stands out. This is exactly what our feature importances pick up on: the standard deviation and mean of packet size dominate, with inter-packet-time variability, total bytes, and flowlet duration close behind. Figure 6 shows this; non-LLM flowlets pile up at very small mean packet sizes with low variance (control traffic) or at very large ones (bulk transfer), while LLM flowlets concentrate in the 200–400 byte range with a characteristic, non-zero spread that reflects per-token chunking.

The client side of an LLM session is less characteristic, as prompts are sent instantly, so essentially all of the discriminative structure lives on the incoming, server-to-client direction, which is why our incoming-only models recover most of the bidirectional performance. Crucially, the same fingerprint generalises across providers: ChatGPT, Claude, and Gemini all use streamed, token-by-token delivery over TLS/QUIC, and our combined-classification results show that a single model trained on all three providers matches the best mixture-of-experts ensemble. In other words, what our classifiers are really learning is the shared statistical signature of *interactive token streaming*, which is an architectural property of how today's LLM services are built and why

classification remains consistent across providers, flowlet thresholds, and model families.

7 Discussion

Taken together, the results show two compatible facts about LLM flowlets. First, each per-provider expert is a strong detector of its own provider's traffic: ChatGPT, Claude, and Gemini are all recognized with precision above 0.90 by their dedicated classifiers, confirming that each service leaves a consistent packet-level fingerprint and that our feature pipeline is expressive enough to capture it. Second, when a single classifier is trained on all three providers at once it reaches essentially the same accuracy and F1 as either overseer, which tells us that the three fingerprints, while individually distinctive, overlap heavily in the statistical regime our features measure (burst timing, chunked token streams, small control-packet tails). The net interpretation is that LLM traffic is detectable, and the *provider* within LLM traffic is also detectable, but the two tasks are not additive: once a model has learned the shared "LLM-ness" signal from any one of the providers, adding the other two contributes far more by broadening its coverage than by introducing genuinely new patterns for a mixture-of-experts stage to exploit.

7.1 Flowlet to Flow Aggregation

While the classification models we created operate at the flowlet level, practical deployment requires classification at the level of full network flows. Because individual flowlets capture only partial segments of a connection, single flowlet misclassifications (false positives or false negatives) may not reflect the true behavior of the overall flow. To address this, we aggregate predictions across all flowlets in the same 5-tuple flow to produce a final flow-level classification.

We apply a majority voting scheme, where a flow is labeled as LLM traffic if the majority, i.e., over 50 percent, of its constituent flowlets in a flow are classified as LLM. This approach reduces sensitivity to noise and improves robustness to isolated errors in classification. In particular, our research shows the LLM flowlet classification typically has higher precision compared to recall. This means that when the model predicts a flowlet as LLM-generated, it is usually correct (lower false positive rate). However, the model fails to identify a nontrivial portion of actual LLM flowlets, leading to a higher false negative rate.

This impacts classification by making aggregation *more conservative*, where flows containing LLM traffic may require a sufficient number of correctly identified flowlets to surpass the majority threshold, otherwise they risk being misclassified as non-LLM. However, consistent misclassification across multiple flowlets can still propagate to the flow level, having bigger impact than isolated incorrect classifications.

From a systems perspective, flow-level classification provides a natural interface for integration with cloud providers and network monitoring. Aggregated predictions enable more reliable downstream decisions, such as traffic characterization and anomaly detection. Overall, aggregation improves classification stability and accuracy, serving as a critical step for practical flow classification.

7.2 Extension to Broader Privacy Works

As mentioned earlier, a key goal of this research is to link existing LLM privacy research to a more expansive application space by eliminating the restrictive assumption that traffic has already been classified as LLM traffic. By providing a methodology to determine whether a traffic flow represents LLM traffic or not within encrypted networks, we argue that previously-researched privacy attacks on LLM services may be extended from isolated test settings to real-world mixed-traffic deployments. For example, Weiss et al.'s 2024 work assessing a token-length side-channel attack on ChatGPT-4 was able to infer the topic of 55% of analyzed responses, but they *explicitly* assumed that the classification step was already done. Their threat model positions an eavesdropper between a client and an LLM service, where the eavesdropper already knows that the target is communicating with an LLM [33]. This is an assumption that is fundamentally difficult to satisfy in practice, but one that our methodology directly enables. We propose that most similar network-based LLM privacy works, like McDonald & Bar's 2025 Whisper Leak and Carlini & Nasr's 2024 timing attacks, may have their scope broadened in a similar fashion [3, 16].

7.3 Relevancy as LLMs Evolve

We believe our tools, approaches, and techniques remain relevant even when LLM providers change their protocols or models in the future. Since LLM services are relatively new, it is true that models, protocols, and hardware and software infrastructure would likely change for performance reasons, patches in response to jailbreaks, or releases of their newly trained LLMs [5]. To accurately classify LLM provider traffic over time, one would need to constantly be collecting data on all providers they want to track. We believe our overall pipeline, including data collection scripts, data preprocessing, model training and application can be easily reapplied to updated or new LLM providers. However, it is also interesting to note that changes to application protocols occur less frequently than anticipated—for example, Zoom traffic filtering and analysis tools from Michel et. al from 2022 [17] still work seamlessly in 2026.

7.4 Limitations

Our results should be interpreted in light of several important considerations.

Limited dataset and LLM providers: The LLM traffic consists of only a small set of providers and collection settings. The non-LLM traffic, while broader, still does not capture full-range, large-scale traffic. Our work should be considered as the first step towards a universally reliable LLM traffic detector, which must be evaluated against a larger and more diverse set of LLM and non-LLM traffic datasets.

Information loss via flowlet abstraction: Our analysis operates on a flowlet abstraction rather than raw packet sequences. This was a practical choice that substantially reduces data volume and makes large-scale feature extraction and model training tractable. However, it necessarily compresses the temporal structure and may discard information that a packet-level model could exploit. Exploring the tradeoff between efficiency and model performance would be an interesting future work.

Possible change in protocol and how users interact with web-based LLM services: The "chat" interface paradigm has been well-established, thus, we anticipate that large provider-side protocol changes are not likely in the short term. However, our models implicitly assume a degree of stability in how services stream responses. Although unlikely in the short term, substantial changes to chunking, timing, or transport behavior could weaken our performance.

Active traffic manipulation: A user or provider familiar with our approach and model could obscure LLM traffic and hinder identification through padding, timing perturbation, tunneling, or traffic morphing. However, this is true for any traffic classification or identification work. We justify this limitation by noting that our system is not intended as a detector for sophisticated adversaries or users who want to hide their traffic due to privacy concerns. We strive for use cases where fast and accurate LLM traffic identification is beneficial for both network administrators and users.

7.5 Future Work

There is substantial room to extend this work. The most immediate extensions fall into three categories: enlarging and diversifying the dataset, moving beyond per-flowlet classifiers, and real-world deployment on an operational network.

Dataset extension. The majority of our dataset consists traffic exchange with US-based website and services. Additionally, this study is limited to three LLM services: ChatGPT, Claude, and Gemini. An interesting follow-up study could explore adding more diverse websites, applications, and more LLM service providers.

Beyond traditional classification models. More systematic hyperparameter search and calibration would tighten our precision/recall trade-off, but larger gains may come from sequence models (e.g., RNNs, temporal convolutional networks, or small transformer encoders) that consume raw packet data over time rather than aggregated Markov features. However, such models have higher overhead and incur longer inference latency compared to traditional classification models. It would be interesting to examine the tradeoff.

Real-world deployment and line-rate classification. Our work has not yet been tested on a large enterprise network, and our pipeline is not optimized for ingesting and processing traffic in real time. We believe the components in our pipeline could run as XDP/eBPF or a DPDK program at the edge. Recent advancement in SmartNICs are also promising. Previous work on in-network ML systems enables running classification models at line-rate on P4-programmable switches [2, 11, 13, 28, 34, 35, 37–40],

8 Conclusion

This research addresses the fundamental challenge of identifying browser-based LLM traffic within encrypted enterprise networks. While existing side-channel research often assumes traffic has already been classified as containing an LLM, we propose a methodology to perform the initial identification amidst the noise of general Internet traffic. We developed a comprehensive pipeline for data collection using tshark and Selenium bots to generate human-like traffic by querying ChatGPT, Gemini, and Claude with diverse prompt chains to be sent to a MongoDB database. To evaluate these services against regular Internet activity, non-LLM traffic was collected from the top 100 websites to create a balanced dataset of 674,852 flowlets. Classification is performed using seven metadata-based features, including packet size mean and standard deviation, inter-packet time mean and standard deviation, total bytes, duration, and packet count using Random Forest, XGBoost, and SVM ML models, achieving accuracies between 81% and 98% with reasonable recall and precision. We experimentally determined that 0.1s flowlet threshold was optimal and that incoming traffic from the LLM achieves higher recall. We finally trained single models and MoE approaches for general LLM classifiers, achieving similar results. Overall, our research lays the groundwork for network administrators to monitor LLM usage in restricted environments without the need for invasive packet inspection and enables further LLM website fingerprinting and side-channel attack research.

References

[1] Leo Breiman. 2001. Random Forests. *Mach. Learn.* 45, 1 (Oct. 2001), 5–32.

[2] Coralie Busse-Grawitz, Roland Meier, Alexander Dietmüller, Tobias Bühler, and Laurent Vanbever. 2019. pforest: In-network inference with random forests. *arXiv preprint arXiv:1909.05680* (2019).

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

[4] Center for Applied Internet Data Analysis (CAIDA). 2019. The CAIDA UCSD Anonymized Internet Traces Dataset - 2019. http://www.caida.org/data/passive/passive_dataset.xml.

[5] Timothée Chauvin, Erwan Le Merrer, François Taïani, and Gilles Tredan. 2026. Log Probability Tracking of LLM APIs. arXiv:2512.03816 [cs.LG]

[6] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 785–794.

[7] Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. 20, 3 (Sept. 1995), 273–297.

[8] Alessandro Finamore, Chao Wang, Jonatan Krolikowski, Jose M. Navarro, Fuxing Chen, and Dario Rossi. 2023. Replication: Contrastive Learning and Data Augmentation in Traffic Classification Using a Flowpic Input Representation. In *Proceedings of the 2023 ACM on Internet Measurement Conference* (Montreal QC, Canada) *(IMC '23)*. Association for Computing Machinery, New York, NY, USA, 36–51.

[9] Arash Habibi Lashkari, Gerard Draper Gil, Mohammad Mamun, and Ali Ghorbani. 2017. Characterization of Tor Traffic using Time based Features. 253–262.

[10] Eyal Horowicz, Tal Shapira, and Yuval Shavitt. 2022. A few shots traffic classification with mini-FlowPic augmentations. In *Proceedings of the 22nd ACM Internet Measurement Conference* (Nice, France) *(IMC '22)*. Association for Computing Machinery, New York, NY, USA, 647–654.

[11] Syed Usman Jafri, Sanjay Rao, Vishal Shrivastav, and Mohit Tawarmalani. 2024. Leo: Online {ML-based} Traffic Classification at {Multi-Terabit} Line Rate. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 1573–1591.

[12] Arash Habibi Lashkari, Gerard Draper-Gil, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. 2017. Characterization of Tor Traffic using Time based Features. In *International Conference on Information Systems Security and Privacy*.

[13] Jong-Hyouk Lee and Kamal Singh. 2020. Switchtree: in-network computing and traffic analyses with random forests. *Neural Computing and Applications* (2020), 1–12.

[14] Chang Liu, Zigang Cao, Gang Xiong, Gaopeng Gou, Siu-Ming Yiu, and Longtao He. 2018. MaMPF: Encrypted Traffic Classification Based on Multi-Attribute Markov Probability Fingerprints. *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)* (2018), 1–10.

[15] Liming Liu, Ruoyu Li, Qing Li, Meija Hou, Yong Jiang, and Mingwei Xu. 2025. FlowletFormer: Network Behavioral Semantic Aware Pretraining Model for Traffic Classification. (2025).

[16] Geoff McDonald and Jonathan Bar Or. 2025. Whisper Leak: a side-channel attack on Large Language Models. arXiv:2511.03675 [cs.CR]

[17] Oliver Michel, Satadal Sengupta, Hyojoon Kim, Ravi Netravali, and Jennifer Rexford. 2022. Enabling passive measurement of zoom performance in production networks. *IMC '22: Proceedings of the 22nd ACM Internet Measurement Conference* (2022), 244–260.

[18] Andrew Moore and Denis Zuev. 2005. Internet traffic classification using Bayesian analysis techniques. *Sigmetrics Performance Evaluation Review - SIGMETRICS* 33, 50–60.

[19] Thuy TT Nguyen and Grenville Armitage. 2009. A survey of techniques for internet traffic classification using machine learning. *IEEE communications surveys & tutorials* 10, 4 (2009), 56–76.

[20] Thuy T. T. Nguyen and Grenville Armitage. 2008. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials* 10, 4 (2008), 56–76.

[21] Andriy Panchenko, Fabian Lanze, Andreas Zinnen, Martin Henze, Jan Pennekamp, Klaus Wehrle, and Thomas Engel. 2016. Website Fingerprinting at Internet Scale.

[22] Eva Papadogiannaki and Sotiris Ioannidis. 2021. A Survey on Encrypted Network Traffic Analysis Applications, Techniques, and Countermeasures. *ACM Computing Surveys (CSUR), Volume 54, Issue 6* (2021), 1–35.

[23] Jonathan Perry, Hari Balakrishnan, and Devavrat Shah. 2017. Flowtune: Flowlet control for datacenter networks. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 421–435.

[24] Yuwen Qian, Guodong Huang, Chuan Ma, Ming Ding, Long Yuan, Zi Chen, and Kai Wang. 2024. Enhancing Resilience in Website Fingerprinting: Novel Adversary Strategies for Noisy Traffic Environments. *IEEE Transactions on Information Forensics and Security* 19 (2024), 7216–7231.

[25] Muhammad Sameer Sheikh and Yinqiao Peng. 2022. Procedures, Criteria, and Machine Learning Techniques for Network Traffic Classification: A Survey. *IEEE Access* 10 (2022), 61135–61158.

[26] Saeid Sheikhi and Panos Kostakos. 2024. Safeguarding cyberspace: Enhancing malicious website detection with PSOoptimized XGBoost and firefly-based feature selection. *Computers & Security* 142 (2024), 103885.

[27] Shan Sinha, Srikanth Kandula, and Dina Katabi. 2004. Harnessing TCP's Burstiness with Flowlet Switching. In *ACM Workshop on Hot Topics in Networks*.

[28] Giuseppe Siracusano and Roberto Bifulco. 2018. In-network neural networks. *arXiv preprint arXiv:1801.05731* (2018).

[29] Mahdi Soleimani, Grace Jia, In Gim, Seung seob Lee, and Anurag Khandelwal. 2025. Wiretapping LLMs: Network Side-Channel Attacks on Interactive LLM Services. Cryptology ePrint Archive, Paper 2025/167.

[30] Linke Song, Zixuan Pang, Wenhao Wang, Zihao Wang, Xiaofeng Wang, Hongbo Chen, Wei Song, Yier Jin, Dan Meng, and Rui Hou. 2024. The Early Bird Catches the Leak: Unveiling Timing Side Channels in LLM Serving Systems. *IEEE Transactions on Information Forensics and Security* 20 (2024), 11431–11446.

[31] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. 2017. Let it flow: Resilient asymmetric load balancing with flowlet switching. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 407–420.

[32] Jiankun Wei, Abdulrahman Abdulrazzag, Tianchen Zhang, Adel Muursepp, and Gururaj Saileshwar. 2026. When Speculation Spills Secrets: Side Channels via Speculative Decoding In LLMs. arXiv:2411.01076 [cs.CL]

[33] Roy Weiss, Daniel Ayzenshteyn, Guy Amit, and Yisroel Misky. 2024. What Was Your Prompt? A Remote Keylogging Attack on AI Assistants. *SEC '24: Proceedings of the 33rd USENIX Conference on Security Symposium* (2024), 3367 – 3384.

[34] Zhaoqi Xiong and Noa Zilberman. 2019. Do switches dream of machine learning? toward in-network classification. In *Proceedings of the 18th ACM workshop on hot topics in networks*. 25–33.

[35] Jinzhu Yan, Haotian Xu, Zhuotao Liu, Qi Li, Ke Xu, Mingwei Xu, and Jianping Wu. 2024. {Brain-on-Switch}: Towards advanced intelligent network data plane via {NN-Driven} traffic analysis at {Line-Speed}. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 419–440.

[36] Yixiang Zhang, Xinhao Deng, Zhongyi Gu, Yihao Chen, Ke Xu, Qi Li, and Jianping Wu. 2025. Exposing LLM User Privacy via Traffic Fingerprint Analysis: A Study of Privacy Risks in LLM Agent Interactions. arXiv:2510.07176 [cs.CR]

[37] Yinchao Zhang, Su Yao, Yong Feng, Kang Chen, Tong Li, Zhuotao Liu, Yi Zhao, Lexuan Zhang, Xiangyu Gao, Feng Xiong, et al. 2025. Pegasus: A Universal Framework for Scalable Deep Learning Inference on the Dataplane. In *Proceedings of the ACM SIGCOMM 2025 Conference*. 692–706.

[38] Changgang Zheng, Haoyue Tang, Mingyuan Zang, Xinpeng Hong, Aosong Feng, Leandros Tassiulas, and Noa Zilberman. 2023. DINC: Toward distributed in-network computing. *Proceedings of the ACM on Networking* 1, CoNEXT3 (2023), 1–25.

[39] Changgang Zheng, Zhaoqi Xiong, Thanh T Bui, Siim Kaupmees, Riyad Bensoussane, Antoine Bernabeu, Shay Vargaftik, Yaniv Ben-Itzhak, and Noa Zilberman. 2024. IIsy: Hybrid in-network classification using programmable switches. *IEEE/ACM Transactions on Networking* (2024).

[40] Changgang Zheng, Mingyuan Zang, Xinpeng Hong, Liam Perreault, Riyad Bensoussane, Shay Vargaftik, Yaniv Ben-Itzhak, and Noa Zilberman. 2024. Planter: Rapid prototyping of in-network machine learning inference. *ACM SIGCOMM Computer Communication Review* 54, 1 (2024), 2–21.

APPENDIX

A Ethics

This work does not raise any ethical issues. All training data comes from authors' own network packet captures collected from personal devices. Network traffic is generated via automated Selenium-based scripts that visit and interact with LLM and non-LLM websites. There was no sniffing of non-consenting party traffic.

B Threshold Results

The following subsections provide complete results for all models that were trained and evaluated as part of this research.

B.1 0.05s Threshold

The 0.05s threshold yielded the weakest results of the three. This was an unexpected finding, as we believed that more fine-grained flowlets would enable better training. It is possible that the lower timeout instead accentuated the effects of packet reordering on the flowlet features. Results are illustrated in Table 6.

Direction	Model	Accuracy	Precision	Recall	F1
Gemini					
Bidirectional	Random Forest	0.862	0.666	0.868	0.754
Bidirectional	XGBoost	0.848	0.644	0.841	0.729
Bidirectional	SVM	0.817	0.662	0.508	0.575
Incoming	Random Forest	0.922	0.848	0.827	0.837
Incoming	XGBoost	0.919	0.850	0.808	0.828
Incoming	SVM	0.838	0.781	0.459	0.578
Outgoing	Random Forest	0.923	0.849	0.769	0.807
Outgoing	XGBoost	0.914	0.840	0.726	0.779
Outgoing	SVM	0.846	0.792	0.351	0.486
ChatGPT					
Bidirectional	Random Forest	0.949	0.832	0.771	0.800
Bidirectional	XGBoost	0.939	0.783	0.746	0.764
Bidirectional	SVM	0.898	0.702	0.407	0.515
Incoming	Random Forest	0.953	0.932	0.530	0.676
Incoming	XGBoost	0.949	0.882	0.520	0.654
Incoming	SVM	0.907	0.875	0.003	0.005
Outgoing	Random Forest	0.968	0.888	0.730	0.801
Outgoing	XGBoost	0.964	0.859	0.710	0.778
Outgoing	SVM	0.928	0.669	0.356	0.465
Claude					
Bidirectional	Random Forest	0.934	0.823	0.635	0.717
Bidirectional	XGBoost	0.927	0.779	0.623	0.692
Bidirectional	SVM	0.879	0.834	0.092	0.166
Incoming	Random Forest	0.957	0.764	0.542	0.634
Incoming	XGBoost	0.956	0.745	0.542	0.628
Incoming	SVM	0.935	0.813	0.072	0.132
Outgoing	Random Forest	0.966	0.847	0.450	0.588
Outgoing	XGBoost	0.963	0.789	0.428	0.555
Outgoing	SVM	0.946	0.692	0.009	0.017

Table 6: 0.05s threshold model performance across providers and traffic directions. Best values per section are highlighted in green.

B.2 0.1s Threshold

Classification results (illustrated in Table 7) at the 0.1s threshold had a very wide range of performance, with F1 scores from $0.025 - 0.930$. Incoming traffic had the highest classifiability scores across all providers, where Gemini, ChatGPT, and Claude models had peak scores of 0.930, 0.906, and 0.902 respectively. Unlike at the 0.2s threshold, Claude models had high classification scores that matched other providers, while Gemini remained the most consistently classifiable overall. Random Forest and XGBoost were the most effective across providers with SVM consistently trailing behind.

Direction	Model	Accuracy	Precision	Recall	F1
Gemini					
Bidirectional	Random Forest	0.906	0.863	0.881	0.872
Bidirectional	XGBoost	0.903	0.858	0.877	0.867
Bidirectional	SVM	0.847	0.859	0.690	0.766
Incoming	Random Forest	0.944	0.964	0.899	0.930
Incoming	XGBoost	0.941	0.961	0.896	0.927
Incoming	SVM	0.867	0.916	0.749	0.824
Outgoing	Random Forest	0.923	0.812	0.789	0.801
Outgoing	XGBoost	0.927	0.818	0.801	0.810
Outgoing	SVM	0.884	0.762	0.591	0.665
ChatGPT					
Bidirectional	Random Forest	0.962	0.926	0.844	0.883
Bidirectional	XGBoost	0.958	0.921	0.827	0.871
Bidirectional	SVM	0.915	0.786	0.688	0.734
Incoming	Random Forest	0.974	0.951	0.865	0.906
Incoming	XGBoost	0.970	0.916	0.866	0.890
Incoming	SVM	0.883	0.835	0.218	0.346
Outgoing	Random Forest	0.955	0.919	0.397	0.554
Outgoing	XGBoost	0.954	0.884	0.400	0.551
Outgoing	SVM	0.930	0.808	0.013	0.025
Claude					
Bidirectional	Random Forest	0.943	0.819	0.814	0.817
Bidirectional	XGBoost	0.945	0.826	0.823	0.824
Bidirectional	SVM	0.920	0.790	0.666	0.723
Incoming	Random Forest	0.984	0.902	0.902	0.902
Incoming	XGBoost	0.983	0.895	0.902	0.899
Incoming	SVM	0.950	0.731	0.637	0.681
Outgoing	Random Forest	0.947	0.724	0.675	0.699
Outgoing	XGBoost	0.951	0.744	0.701	0.722
Outgoing	SVM	0.936	0.677	0.573	0.620

Table 7: 0.1s threshold model performance across providers and traffic directions. Best values per section are highlighted in green.

Direction	Model	Accuracy	Precision	Recall	F1
Gemini					
Bidirectional	Random Forest	0.915	0.914	0.867	0.890
Bidirectional	XGBoost	0.909	0.905	0.861	0.882
Bidirectional	SVM	0.801	0.852	0.599	0.703
Incoming	Random Forest	0.909	0.847	0.759	0.801
Incoming	XGBoost	0.927	0.847	0.852	0.850
Incoming	SVM	0.843	0.886	0.403	0.554
Outgoing	Random Forest	0.941	0.833	0.722	0.774
Outgoing	XGBoost	0.931	0.791	0.696	0.741
Outgoing	SVM	0.896	0.711	0.439	0.543
ChatGPT					
Bidirectional	Random Forest	0.939	0.936	0.732	0.821
Bidirectional	XGBoost	0.938	0.926	0.738	0.821
Bidirectional	SVM	0.871	0.829	0.412	0.551
Incoming	Random Forest	0.967	0.903	0.689	0.782
Incoming	XGBoost	0.964	0.868	0.687	0.767
Incoming	SVM	0.937	0.983	0.267	0.421
Outgoing	Random Forest	0.957	0.894	0.561	0.689
Outgoing	XGBoost	0.957	0.869	0.577	0.693
Outgoing	SVM	0.926	0.629	0.303	0.409
Claude					
Bidirectional	Random Forest	0.944	0.768	0.627	0.690
Bidirectional	XGBoost	0.943	0.773	0.611	0.683
Bidirectional	SVM	0.912	0.771	0.158	0.263
Incoming	Random Forest	0.948	0.673	0.421	0.518
Incoming	XGBoost	0.944	0.618	0.408	0.491
Incoming	SVM	0.931	0.358	0.042	0.075
Outgoing	Random Forest	0.974	0.863	0.496	0.630
Outgoing	XGBoost	0.970	0.761	0.481	0.590
Outgoing	SVM	0.960	0.762	0.134	0.227

Table 8: 0.2s threshold model performance across providers and traffic directions. Best values per section are highlighted in green.

B.3 0.2s Threshold

The 0.2s flowlet partition threshold (illustrated in Table 8) gave us moderately successful results, with F1 scores ranging from $0.075 - 0.890$. There is a noticeable discrepancy across providers, with Claude classification models performing significantly worse than ChatGPT and Gemini models. Our results demonstrate Gemini as the easiest to classify at the 0.2s threshold and Random Forest as generally the most effective model across providers.

C Dataset

We present our resulting dataset in three separate divisions for the three thresholds. All data collected is represented in all three flowlet thresholds - if a flow was collected that represented streaming from a ChatGPT query, then it would have flowlet representations for all three thresholds. High-level information on our training dataset is represented in Table 9 alongside derived and per-threshold/provider information in the following tables.

C.1 Core Statistics

Traffic class	Flowlets	Dur. med.	Dur. mean	Pkts med.	Pkts mean	Bytes mean
All flowlets	674,852	0.011	0.082	3	21.072	15,001
LLM	334,428	0.008	0.061	2	16.875	13,260
Non-LLM	340,424	0.013	0.102	4	25.195	16,711

Table 9: Core dataset statistics

C.2 Derived Statistics

Traffic class	Mean pkt size	Pkt size std	Mean IPT	IPT std
All flowlets	223.1	160.3	0.011	0.009
LLM	236.0	188.2	0.010	0.006
Non-LLM	210.4	132.9	0.011	0.011

Table 10: Derived dataset statistics

C.3 Threshold Aggregates

Threshold	Flowlets	Mean dur.	Median dur.	Mean pkts	Mean bytes
0.05	269,419	0.043	0.009	17.4	12,344
0.10	213,001	0.073	0.011	21.7	15,501
0.20	174,400	0.156	0.025	26.3	18,618

Table 11: Per-threshold aggregates

C.4 Per-Provider Aggregates (LLM Only)

Provider	Flowlets	Captures	Median Dur. (s)	Median Pkts	Median Bytes
Gemini	129,408	20	0.007	2.0	356
Claude	108,807	17	0.007	2.0	211
ChatGPT	96,213	15	0.017	2.0	423

Table 12: Per-provider aggregate statistics for LLM-generated flowlets.

C.5 Figures

The following figures illustrate the patterns mentioned in the paper, such as threshold/flowlet count relationships (Figure 13) and provider-to-flowlet sensitivity relationships (Figure 12). These figures were generated using a Jupyter notebook that took in a snapshot of our dataset, and they are thus reproducible should the dataset change or be developed upon in the future.

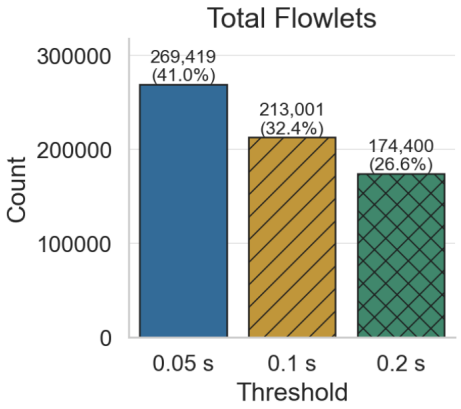


Figure 9: Flowlet count by threshold

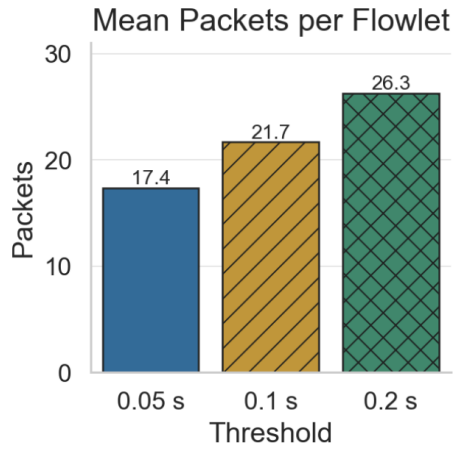


Figure 10: Mean packets per flowlet by threshold

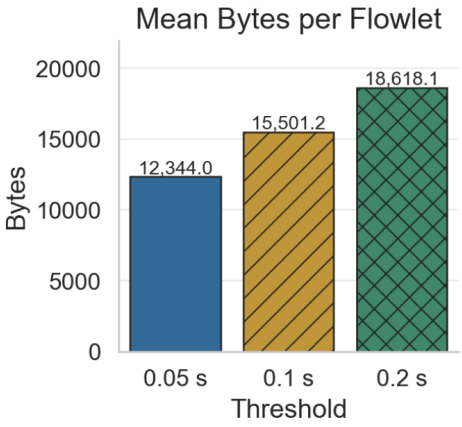
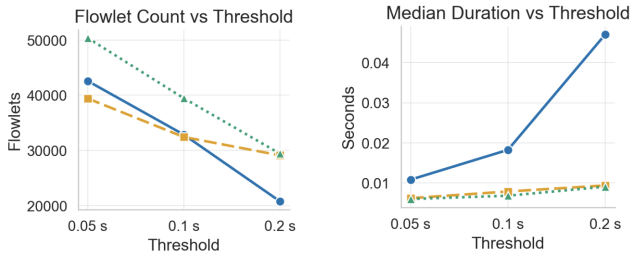
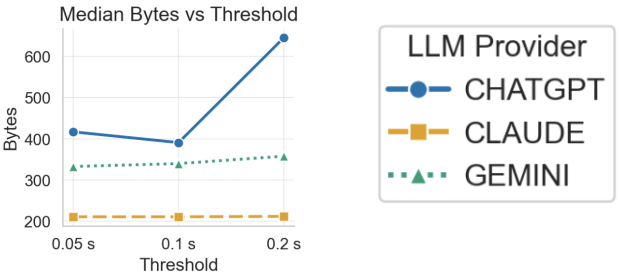


Figure 11: Mean flowlet byte count by threshold (total bytes)



(a) Flowlet count per provider by threshold

(b) Median flowlet duration per provider by threshold



(c) Median flowlet byte count per provider by threshold

(d) Threshold by provider legend

Figure 12: Threshold sensitivity by LLM provider.

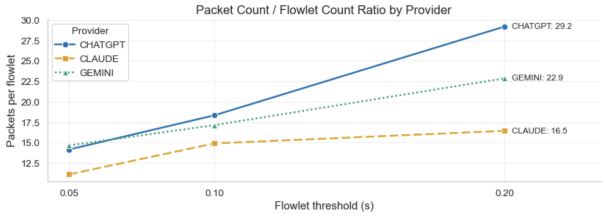


Figure 13: Ratio of packet count to flowlet count for each provider as threshold varies

Metric	0.05	0.1	0.2
Captures	43	43	43
Flowlets	276,739	219,069	179,044
Unique Flows	65,834	63,086	66,097
Flowlets/Flow	4.20	3.47	2.71
Total Packets	4,791,114	4,730,024	4,699,385
Total Bytes	3,408,671,517	3,384,733,294	3,329,970,604
LLM Flowlets (%)	139,687 (50.5)	110,804 (50.6)	83,937 (46.9)
Avg Duration (s)	0.0423	0.0723	0.1539

Table 13: Overview of training flowlet dataset, generated via DB snapshot and Python script

D Collection Methods

Data was manually collected via two main scripts. We ran the `selenium_bot.py` script alongside our capture script to issue human-like prompts to the three browser-based LLM services under test. This generates the packets that end up in our database flowlets, mimicking the chain-of-though prompting that human-LLM interaction generally follows. It is key that the user turns off the "Secure DNS" option in the undetected chromedriver browser and that the browser use the same `SSLKeyLogFile.txt` that the capture script refers to for LLM-to-IP mappings. The pseudocode description for this script is in Algorithm 1.

Algorithm 1 Automated Selenium LLM Interaction Bot

Require: target ∈ {ChatGPT, Gemini, Claude}, SSLKeyLogFile S, JSON Prompt Bank P
Ensure: Collected query–response traces

```
Set environment variable SSLKEYLOGFILE ← S
Initialize Selenium WebDriver (undetected Chrome)
Load prompt chains from P
Open target LLM website
Wait for manual login
for each prompt chain do
    Open new browser tab (new conversation)
    Shuffle prompts in chain
    for each prompt do
        if prompt has PDF attachment then
            Select random PDF
            Upload file to LLM UI
        end if
        Type prompt (optionally with noise)
        Send prompt
        Wait for model response completion
    end for
end for
Close browser session
```

The `live_capture_to_db.py` script executes a tshark command based on the IP range and interface passed, ingests and assigns packets to flows based on 5-tuple keys, partitions flows into flowlets based on our three threshold values, and pushes flowlet objects with derived measurements to the MongoDB document corresponding to the capture and threshold. While all of this is happening, the script also checks for any LLM-to-IP mapping visible through DNS or SNI queries. This script was used for both LLM and non-LLM traffic capture, and Algorithm 2 describes the script in pseudocode.

Algorithm 2 Live Multi-Threshold Flowlet Capture

Require: ip_range, interface, timeout T, mongodbConnection M, sslKeyLogFile S
Ensure: Database exists and contains captures

Initialize database session using M
Resolve capture network from ip_range
Build tshark command w/ interface and sslKeyLogFile S
Initialize map_{LLM} = {}
Spawn separate thread for $threshold \in \{0.05, 0.1, 0.2\}$
Start packet capture process
while process is running **do**
    if $timeout$ is set AND $timeout$ exceeded **then**
        Terminate capture process
        break
    end if
    Read next packet from tshark output
    Parse packet fields (IP, ports, DNS, timestamp, etc.)
    if packet contains LLM indicator **then**
        add LLM-to-IP mapping to map_{LLM}
    end if
    Append packet to flow buffer by flow key
    if inter-arrival time > threshold **then**
        Compute features:
            duration, packet count, etc.
        Insert flowlet into MongoDB via M
        if document exceeds size limit **then**
            Rotate capture collection
        end if
    end if
end while
Flush remaining flow buffers
Commit final capture metadata (map_{LLM}, status)
Terminate tshark process