TrafficChop: On Obfuscation of LLM Traffic Flows for the Prevention of Fingerprinting Attacks

Gavin Crigger
College of Arts and Sciences
University of Virginia

Abstract

Artificial intelligence companies have had a massive boom in the public market within the past 5 years, creating widespread availability for browser-based LLM services. With a new spotlight shining on querying LLMs over the internet, questions on how user privacy may be attacked over this medium have been raised and researched. This paper presents an investigation into obfuscation for improving LLM user privacy, specifically building upon my research with Professor Hyojoon Kim on router-based detection of LLM traffic via flowlet-based fingerprinting. I present a codebase for a simulated LLM-chatbot-to-client architecture modeling cross-network communication and evaluate the effectiveness of multiple obfuscation methods on it. These were: flow reshaping, artificial jitter, dummy packet injection, and a "chaos mix" of all three, achieving an F1-score impact of -0.024, 0.0, 0.0, and -0.064 respectively on a frozen XGBoost model at flowlet threshold 0.05s. Assuming an adaptive adversary results in even less impactful F1-score deltas of -0.016, 0.0, 0.0, and -0.006 respectively. While some model/threshold combinations provided more hopeful results (see appendix), these results suggest that the obfuscation methods put forth are ineffective at boosting the privacy of browser-based LLM users against fingerprinting attacks, primarily due to them only affecting temporal features. Future work should be dedicated towards different obfuscation methods across feature classes and/or more real-world, representative traffic representations for the process as a whole. Artifacts are available at https://github.com/gavinvc/Traffic-Chop [2].

1 Introduction

Large Language Model (LLM) access has skyrocketed globally as more companies invest in browser-based chat services. These browser applications typically allow users to interact with a backend LLM service via a chat interface, taking user queries and streaming back the LLMs response. While there are many different LLM services all with unique data processing and transfer methods, there exists an attack surface in identifying general LLM usage via the shared traffic flow patterns that such services create. Traffic fingerprinting techniques may be applied to specific LLMs by training classification models on encrypted traffic labeled as LLM vs. non-LLM - while this has not yet been extensively researched, this work serves as a proactive investigation into preventing fingerprinting via traffic obfuscation and reshaping.

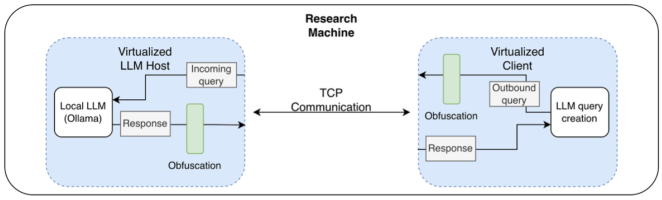

Figure 1: High-level client-server architecture for simulating and capturing LLM queries

This is an extension of my previous work with fingerprinting LLM traffic flows over encrypted enterprise networks. Those methods found success in training different classifiers on manually-collected and labeled LLM/non-LLM traffic flows at the transport layer. Its significance lies in the implication that an adversary may train classifiers on different LLM services and create a potentially service-agnostic fingerprint attack that can bypass user privacy in encrypted enterprise networks. As such, I investigate how obfuscation techniques may reduce the efficacy of fingerprinting attacks, with a goal of not inadvertently create a new traffic shape that can simply be classified via retraining or incur excessive latency costs. Additionally, I present a sandboxed browser-to-LLM environment that virtualizes the client, gateway, and LLM to offer a simulated representation of production/real-word LLM services communicating over the internet.

2 Background

OpenAI broke ground with the release of ChatGPT, bringing language models into the public eye and effectively rewriting daily workflows for countless people. When evaluating the potential privacy risks that LLM services present, we must first consider the architecture of such services. Most browser-based LLM architectures allow the client to send a request to a backend LLM, hosted by services like Cloudflare, which then passes the request through the model and streams back the response. The exact details of these architectures are vastly proprietary for real-word use cases, yet Dartmouth's work in establishing an open-source LLM stack at scale [11] gives a blueprint that can be used to understand the privacy risks of engaging with LLM services.

Website fingerprinting attacks are methods with which attackers create classification models trained on the traffic flows of web pages before feeding recorded traffic through those classifiers and identifying what services users are accessing. This privacy attack works under the assumption of encrypted traffic, simply relying on the patterns that traffic flows (correlated using 5-tuple identification) have with respect to their sources. When it comes to general LLM usage, this privacy concern is especially significant in academic settings (such as student usage of LLMs during sensitive times) and workplaces handling classified information, where LLM usage may not be permitted due to incorporation of queries into model training. Research has been increasingly focused on creating fingerprinting methods that work through obfuscation techniques and noise, thus emphasizing the need for further privacy enhancing methods research. This research is focused on three traffic obfuscation techniques: injecting temporal jitter across streamed packet, injecting dummy packets between real traffic, and comprehensive traffic control through flow reshaping. These methods are visualized in Figure 2.

Flowlet-based traffic switching is a researched practice that governs a significant portion of internet traffic protocols. Many switching methods use dynamic flowlet apportionment that has a time threshold value changing with response to congestion [9]; I use static time thresholds in my experiments due to congestion not being significant in my simulations. In particular, I perform my entire methodology on the same traffic partitioned into flowlets with time thresholds of 0.05, 0.1, and 0.2 seconds. These flowlets and their corresponding statistical information are used for training and testing each classification model to evaluate fingerprinting effectiveness across all obfuscation methods. Relevant flowlet features as discovered in my external LLM research (and backed up through training these models) seem to be packet size mean and total bytes.

This research is performed under the assumption that LLM services operate under a client/server architecture. This architecture implies that a client connects to an LLM service over a browser and is then connected to a backend-hosted LLM,

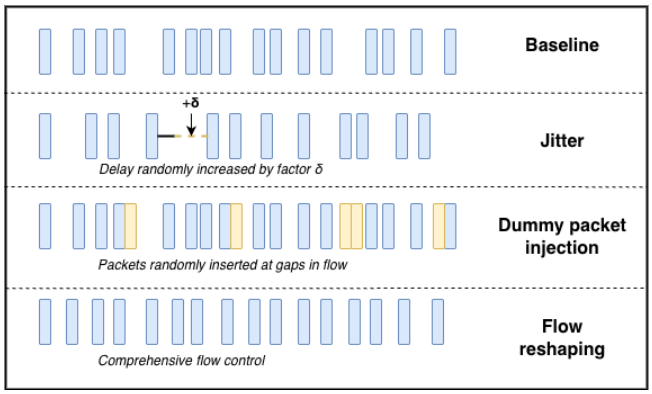

Figure 2: Visual demonstration of baseline traffic flow versus jitter, dummy packet injection, and flow reshaping obfuscation methods

where requests and responses are then exchanged between the two. Handoff from an LLM frontend to a cloud-service backend makes LLM traffic identification non-trivial, as one may not perform simple tag/flag DNS-based privacy attacks. This is the most scalable and realistic architecture for such services given current research, and thus I simulate a similar architecture when collecting data in my following sections [6].

3 Sandbox Environment

In an ideal research scenario, obfuscation experimentation would be done in an environment identical to production LLM chat services. Due to my time and resource constraints, I deferred to creating a sandbox environment that represents the client-server architecture that browser-based LLM chat services follow. Figure 1 depicts the high-level approach that I took; services were isolated through Docker virtualization to simulate different machines speaking to each other over TCP [7]. These services were: a dashboard hosting an LLM client, multiple gateway hops, an LLM model backend, and wrappers around the client/server for obfuscating outbound traffic.

3.1 Network representation

The internet is an excessively complicated structure; as former Google CEO Eric Schmidt said, "the Internet is the first thing that humanity has built that humanity doesn't understand." Thus, I took a more relaxed approach to representing the internet communication between client and LLM service in my sandbox environment. This included multiple queries to LLM copilot services to make changes and inject delays to the structure. The final setup has a gateway with two relays that inject some amount of delay/jitter to routed packets. This pushes the simulated traffic towards a more realistic reflection of traffic traveling across the internet.

3.2 Obfuscation techniques

Traffic obfuscation is extensively researched in the security and privacy research communities [3]. I employ three specific obfuscation techniques drawing from recent papers: flow reshaping, dummy packet injection, and artificial jitter. Flow reshaping refers to using delays and controlled release of packets to mitigate the identifiable bursts of specific websites. Dummy packet injection involves padding or empty packets and can be either static or adaptive, with the latter being far more successful at reducing identification rates while being easier on latency. Finally, injecting artificial jitter simply adds an artificial delay that staggers throughout time, changing the timing between packets.

The specific tuning of my implementation also extends existing work. I use parameters in line with Holland & Hopper's (2022) RegulaTor obfuscation framework for my flow restructuring approach [4]. The obfuscation wrapper allows an initial surge rate of 240 packets per second and a 0.97 decay rate constant, and it starts additional surges once the buffered packet queue reaches the threshold of 30% the current send rate. As a result, the traffic flow ends up much smoother while maintaining a bounded overhead for performance. While this clearly reshapes the traffic flow (and thus makes it hard to identify as LLM traffic if the attacker lacks white-box knowledge about the obfuscation), it can be easily re-trained upon unless it effectively mimics some other identifiable traffic flow.

Recent works have proven static jitter/packet injection methods as ineffective at preventing packet timing-based fingerprinting attacks [10], thus I defer to a more adaptive approach. My jitter implementation gives each packet a 35% chance to incur an additional delay δ based on a Gaussian distribution with a 3ms standard deviation. Because this is a live forwarder, we may not inject negative delay, and any negative values are treated as zero delay. Dummy packet injection is realized with an adaptive chaff controller monitoring inter-packet time across connections in line with recent IEEE BigDIA work [13]. If an idle interval over 50*ms* occurs, the controller sends a 0.08 probability chaff burst of 1 to 3 packets randomly sized from 512 to 1500 bytes. This method's latency overhead is bounded by a 300 limit on the number of dummy packets, which is reset after a timeout of 2 seconds (indicating a pause in information transfer).

The resulting obfuscation service is implemented as a transport-layer wrapper that implements one or more of the aforementioned obfuscation techniques through interception of outbound traffic. This is done by terminating the client connection and opening a *new* connection, where the asynchronous wrapper may control the packet scheduling and thus inject obfuscation. Enabled techniques are managed through environment variables at build-time, including an approach combining the methods. I found it important to make the obfuscation wrapper agnostic to the client-server services, as it should remain modular enough to implement on other models and dashboards. Furthermore, I decided on using stream timeouts of 360s to prevent hanging during automated data collection, as the model had a tendency to stream for excessively long periods of time.

4 Measurement

There were two questions that I had going into evaluating my obfuscation techniques on LLM traffic. The first was: how successful are different obfuscation techniques at reducing the classifiability of standard LLM traffic? My classification methods build upon my ongoing research into using fingerprinting attack methods to distinguish between LLM and non-LLM traffic flows from an encrypted enterprise router's vantage point. I employ XGBoost, Random Forest, and SVM models each trained and evaluated once for each threshold value. This led to a *lot* of graphs and tables being created, the entirety of which is included in the appendix.

Secondly, how much performance impact do each of these obfuscation techniques have from an end-user perspective? I reasoned and implemented a few benchmarks to ensure that these questions were adequately assessed: per-message latency normalized by response length, flow identification rate via multiple classification models, and the privacy-performance intersection of each obfuscation methods. Since browser-based LLM services are wholly user-facing applications with pressing load around the clock, any performance sacrifices must be significantly supported by privacy improvements. I thus construct a privacy-performance plot for each obfuscation method to get a big-picture idea of how different implementations compare on the two dimensions.

Data was collected via a Selenium bot that was run on the baseline experimental setup as well as all obfuscation methods until a sufficient set of traffic captures had been collected for each traffic methodology (about 100Mb of files) [1]. This resulted in at least 10,000 flowlets per technique - exact numbers for the 0.05 threshold can be seen in Table 1, and the algorithm pseudocode is detailed in Appendix Algorithm 1.

Technique	Test LLM Flowlets	Test Non-LLM
Baseline	37,483	37,483
Flow Reshaping	12,014	37,483
Jitter	37,483	37,483
Dummy Injection	25,978	37,483
Chaos Mix	37,483	37,483

Table 1: Flowlet counts by technique at the 0.05s flowlet timeout threshold

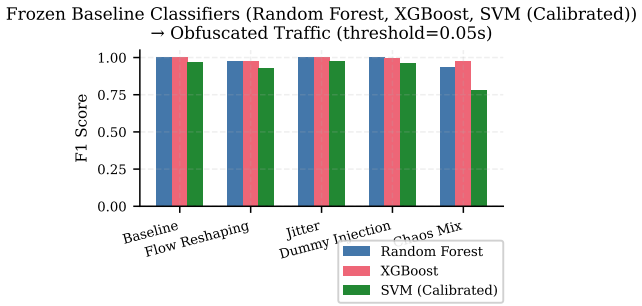

Figure 3: F1 scores of baseline-trained classifiers on obfuscated traffic. Only flow reshaping and chaos mix produce meaningful degradation.

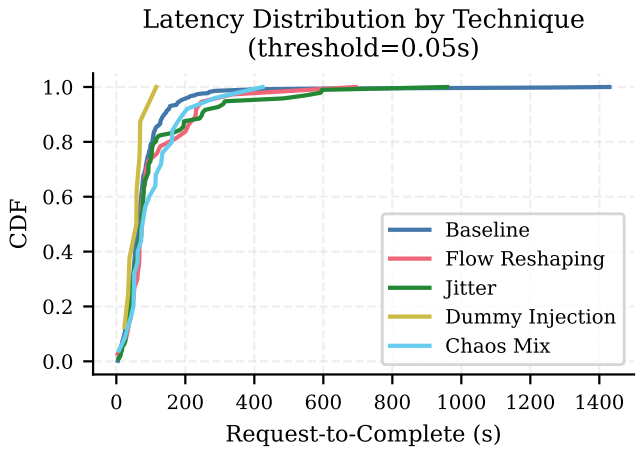

Figure 4: CDF of latency impacts for each obfuscation method relative to baseline latency measurements.

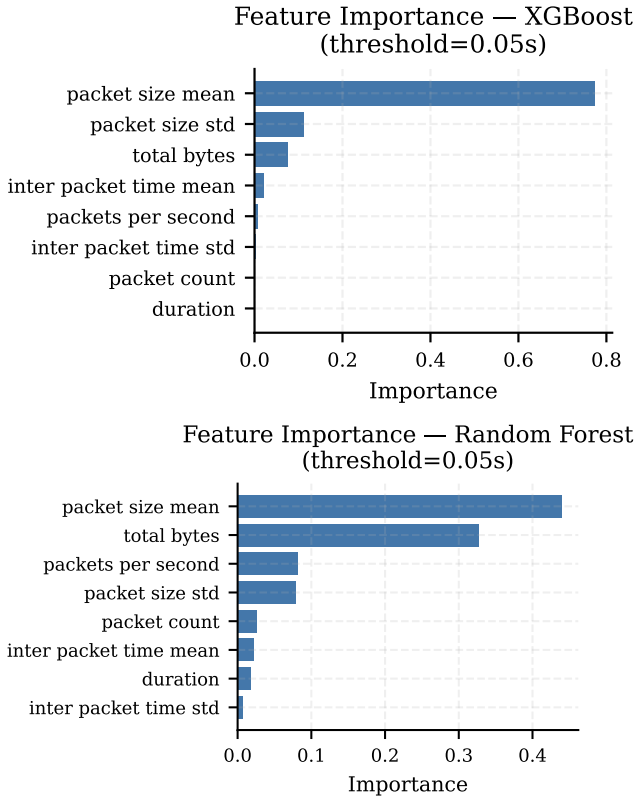

Figure 5: Feature importances for XGBoost (left) and Random Forest (right). Volume-based features account for the majority of discriminative power.

5 Results

All results in this section use a flowlet threshold of 0.05 seconds unless otherwise noted. Random Forest, XGBoost, and SVM classifiers are trained and evaluated on per-flowlet statistical features extracted from transport-layer captures. All experiments were performed using a locally-hosted `phi3:mini` Ollama model [12].

5.1 Frozen Baseline Classifier

The first experiment performed investigates what privacy improvements could be observed by implementing obfuscation methods to existing browser-based LLM services that adversaries had already trained classifiers on. Each classifier is trained on a 0.8/0.2 train/test split, and the total number of baseline and non-LLM flowlets was sized down across methods to match the highest quantity of flowlets collected for an individual obfuscation method. Dataset and classifier parameter specifics can be found at appendix sections A-C. The most important features across classification methods were those related to packet size; the top three features were packet size mean, total bytes, and packet size standard deviation. This shows that, while timing was still a factor (for the Random Forest model in particular), classification models ended up relying mostly on size-based features, which is a discrepancy from most network traffic side-channel attacks (which typically exploit subtle quirks in timing).

All obfuscation methods ended up being classified by *some* model with an F1-score of at least 0.9, with most of them being far closer to 0.99. Figure 6 demonstrates how the flow reshaping and chaos mix obfuscation methods were the most effective at reducing fingerprinting ability while still remaining identifiable as a whole. Another trend clear in this figure is the discrepancy between classifier models. Random Forest and XGBoost models were similar in their effectiveness across time thresholds, with XGBoost barely edging out RF in the most effective cases, while SVM classifiers were far more variable and generally ineffective.

5.2 Adaptive Adversary

A more realistic scenario of a privacy-busting adversary assumes retraining on obfuscated traffic to maximize the effec-

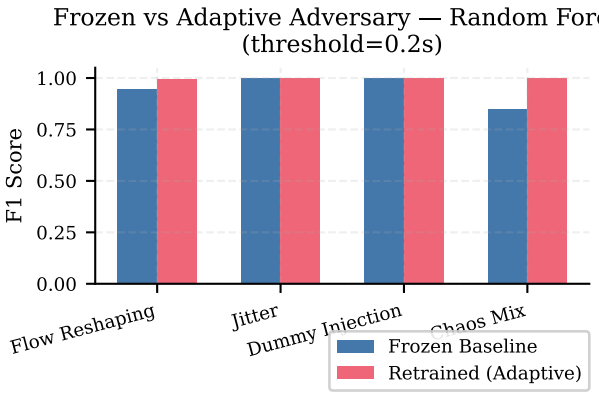

Figure 6: Detection shift between a frozen and adaptive adversary on a Random Forest classification scheme and a 0.2s threshold - this less effective set of results is chosen to make the F1 score increase more discernible.

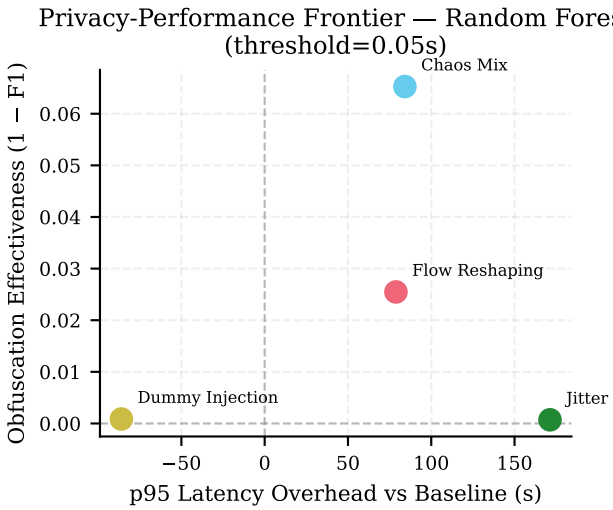

Figure 7: Privacy–performance trade-off (Random Forest). Chaos mix offers the best balance; jitter incurs latency overhead with negligible privacy benefit. Random Forest model selected as a more representative privacy-performance trend across all models/thresholds.

tiveness of classification models. This experiment performs retraining of the baseline models on the newly-obfuscated request/response traffic, which resulted in improved F1 scores across the board. Figure 6 shows the F1 score increase that adaptive retraining has on each obfuscation method for a Random Forest classification method and a 0.2s threshold (chosen to show more exaggerated result). Any initial success in improving LLM user privacy is very quickly squandered with an adaptive adversary attack model, demonstrating that more complex obfuscation methods are in line.

5.3 Performance Evaluation

Latency is a concern for any alterations to networking protocols, and this is twofold for browser LLM services, where users are expecting responses to be streamed back to them relatively quickly. Performance is evaluated in this experiment via a few benchmarks, the most significant of which being the request-to-complete time of each response. Figure 4 offers a CDF illustrating how each obfuscation method impacts the time it takes for the simulated user to receive a complete response. The baseline method has a more standard-appearing curve with the chaos mix following suit, while flow reshaping, jitter, and dummy packet injection have more erratic curves. Interestingly, the dummy packet injection obfuscation method *improved* the overall latency of the requests. As the only obfuscation method that added additional packets to the streamed response, it is possible that the method inadvertently killed the response early on. The rest of the latency results are rather expected, and while there is a definite time cost incurred, it does not seem to be excessive on the whole. Finally, the overall lengthy request-to-complete times can be attributed to the smaller model streaming for a *very* long time in many cases.

6 Discussion

As mentioned in the Results section, many fingerprinting-based privacy attacks follow methodologies that exploit timing patterns that applications inadvertently inject into their network traffic. These results, however, show that the most effective classifier models relied on features related to packet size. This is likely due to the obfuscation methods under test being largely time-based - the classifiers had more success with deciding on raw and derived packet size features due to the injected variability on the time domain. Obfuscation methods that rely on one feature class, like the time-based ones here, thus do not seem effective in improving privacy against fingerprinting attacks. It follows that future work should experiment with cross-class obfuscation that does not neglect any traffic patterns. Additionally, there is potential for adaptive obfuscation that *exploits* feature class isolation by cycling through different, isolated obfuscation methods to effectively "mistrain" adversarial models in the process.

Figure 7 demonstrates the trade-off between privacy and performance across all obfuscation methods, with the baseline request-to-complete p95 latency overhead used as a reference. Ignoring the previously-examined dummy injection time improvement, we see that flow reshaping and our chaos mix have very similar latency impacts while the chaos mix incurs a higher privacy increase. The jitter-based method incurs the most significant latency impact while providing little to no privacy improvement. It follows that an effective mix of multiple obfuscation methods can still have a reasonable impact on latency rather than summing up the performance impacts

of all combined methods. Thus, it is suggested that future work investigates how obfuscation methods can be pieced together to maximize privacy benefit while keeping latency impacts in check. Finally, I assert that this traffic generalizes to other LLM flows, even with the ad hoc simulation structure, as browser-based LLM services generally boil down to the same structure of passing requests from over the internet to a backend LLM API and streaming back the response.

6.1 Challenges and Limitations

With my lack of access to modern/distributed browser-based LLM services, the most pressing challenge to my research was creating an adequate simulation for not only collecting traffic data but also implementing obfuscation techniques. In an ideal setting, I would be able to perform both of these tasks as wrappers around services like ChatGPT on the client and server sides, but my current access is limited to client-side traffic modulation. This segues into the limitations to the findings in this study. Chief among those is the gap between my simulated environment and real-world browser-based LLM service architectures. While there are consistencies underlying principles of LLM-gateway-client and traffic measurement for classification model training, I would be remiss to make a direct jump from my work to the implication that such methods are applicable to LLM services as a whole. Furthermore, the sampled LLM data was limited to simple, text-based prompt exchanges, which is not wholly representative of the diverse user interactions that LLM services support. Finally, my non-LLM data collection could have been tainted by LLM traffic inadvertently - I ran my scripts on my personal machine, and while I was careful to not also run LLM services while collecting data, it is very possible that I *did* end up running both at the same time by mistake.

7 Future Work

This body of work is restricted to a sandboxed environment which is not entirely representative of end-to-end LLM service pipelines. Another restriction in this work is only using static obfuscation methods, which an adversary may adapt to and circumvent [8]. Thus, future work should investigate partnerships with large-scale LLM services and more robust obfuscation techniques to produce research that is more applicable to the real world. Future attempts at adaptive obfuscation could employ random query-time method selection, ensuring that the method selected obfuscates traffic across all feature classes (time, size, etc.). Additionally, under the assumption of an adaptive adversary, future works could consider how a goal of "mistraining" the adversary's classification model by cycling through intermittent poor obfuscation techniques as part of a generally effective obfuscation scheme. This obfuscation scheme could follow a game theoretic model to minimize additional latency while maximizing the privacy benefit, similar to the DataSentinel approach to detecting prompt injection attacks [5].

8 Ethics and GenAI

The traffic used for classification, training, and analysis was limited to public datasets widely used in the research community (MAWI backbone data) and simulated packets generated as part of my sandbox environment. Additionally, traffic features were only considered from public-facing header and meta-information. Thus, no reasonable privacy expectations have been violated as part of this work. Generative AI was used to write much of the codebase, like shells for the obfuscation wrapper and the general virtual client-server architecture. Additionally, generative AI copiloting directed figure generation and table formatting. **No generative AI was used in the writing of this paper.**

9 Conclusion

As the world shifts to a more LLM-centric lifestyle, our privacy-enhancing methods and research should expand to cover the ever-increasing attack surfaces that come with it. Obfuscation methods tested here are deemed ineffective at present implementation - flow reshaping, artificial jitter, dummy packet injection, and the chaos mix all achieved negligible privacy improvements against an adaptive adversary. Once more, these F1-score deltas were -0.016, 0.0, 0.0, and -0.006 respectively. Furthermore, the size-centric feature importance for Random Forest and XGBoost models suggest that obfuscation against one class of features simply incentivizes models to shift classification schemes to other feature classes. It follows that future work should investigate comprehensive feature class obfuscation techniques and potentially adaptive obfuscation that cycles through feature classes (thus leading adaptive adversaries to "mistrain" their models). Finally, my codebase, evaluation scripts, and any other artifacts are available at https://github.com/gavinvc/Traffic-Chop.

References

[1] Selenium - Broswer automation tool. https://www.selenium.dev/.

[2] CRIGGER, G. Traffic-chop, 2026.

[3] DENG, X., CHEN, J., YU, L., ZHANG, Y., GU, Z., QIU, C., ZHAO, X., XU, K., AND LI, Q. Beyond a single perspective: Towards a realistic evaluation of website fingerprinting attacks. *Tsinghua Science and Technology* (2025).

[4] HOLLAND, J., AND HOPPER, N. Regulator: A straightforward website fingerprinting defense. *Proceedings on Privacy Enhancing Technologies (PoPETs) 2022*, 3 (2022), 259–278.

[5] LIU, Y., JIA, Y., JIA, J., SONG, D., AND GONG, N. Z. Datasentinel: A game-theoretic detection of prompt injection attacks, 2025.

[6] Mechkaroska, D., Domazet, E., Feta, A., and Shikoska, U. R. Architectural scalability of conversational chatbot: The case of chatgpt. In *Advances in Information and Communication*, vol. 919 of *Lecture Notes in Networks and Systems*. 2024, pp. 54–71.

[7] Merkel, D. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal 2014*, 239 (2014), 2.

[8] Singh, K. P., Pilli, E. S., Laxmi, V., Yusuf, K., and Yadav, M. Adeftor: Adaptive adversarial example generation for website fingerprinting defense in tor. *IEEE Transactions on Networking 34* (2026), 843–856.

[9] Sinha, S., Kandula, S., and Katabi, D. Harnessing tcp's burstiness with flowlet switching.

[10] Sirinam, P., Imani, M., Juarez, M., Niksefat, B., and Hopper, N. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)* (2018), pp. 1928–1943.

[11] Stone, S., Crossett, J., Luker, T., Leligdon, L., Cowen, W., and Darabos, C. Darthmouth chat - deploying an open-source llm stack at scale. In *Practice and Experience in Advanced Research Computing 2025: The Power of Collaboration (PEARC '25)* (July 2025), pp. 1–5.

[12] Team, O. Ollama: Get up and running with large language models, 2024. Software available from https://github.com/ollama/ollama.

[13] Zhou, Z., and Jin, W. Website fingerprint defense based on adaptive padding. In *2025 IEEE International Conference on Big Data Intelligence and Computing (BigDIA)* (2025).

A APPENDIX

A.1 Data Collection

Algorithm 1 Automated Selenium LLM Interaction Bot

Require: JSON Prompt Bank P
Ensure: TShark capture is running and outputting to technique-dependent directory

```
Initialize Selenium WebDriver (undetected Chrome)
Load prompt chains from P
Open LLM dashboard
for each prompt chain do
    Refresh browser tab
    Shuffle prompts in chain
    for each prompt do
        Type prompt (optionally with noise)
        Send prompt
        Wait for model response completion
    end for
end for
Close browser session
```

B Supplementary Figures (0.05s Threshold)

The following table and figures describe the data, classification rates, and performance impacts of the obfuscation methods evaluated from a 0.05s flowlet-partitioning threshold.

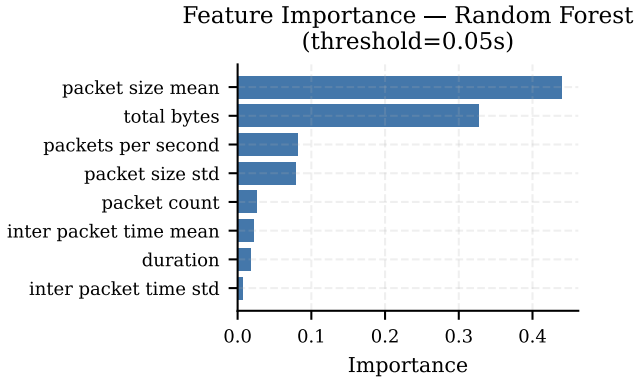

Figure 8: Random forest flowlet feature importance

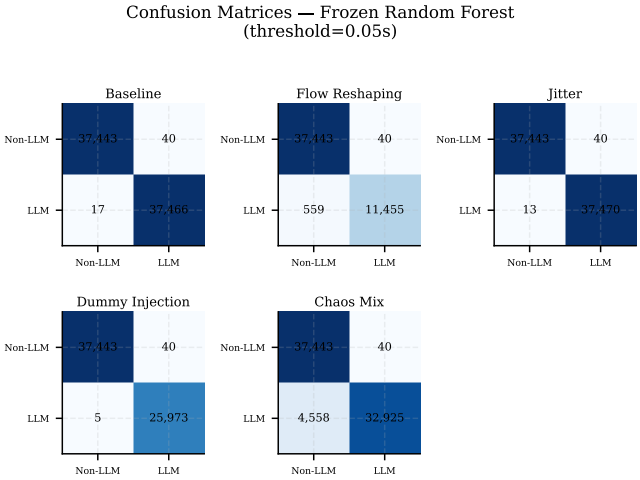

Figure 9: LLM vs. non-LLM classification confusion matrix (Random Forest, frozen adversary)

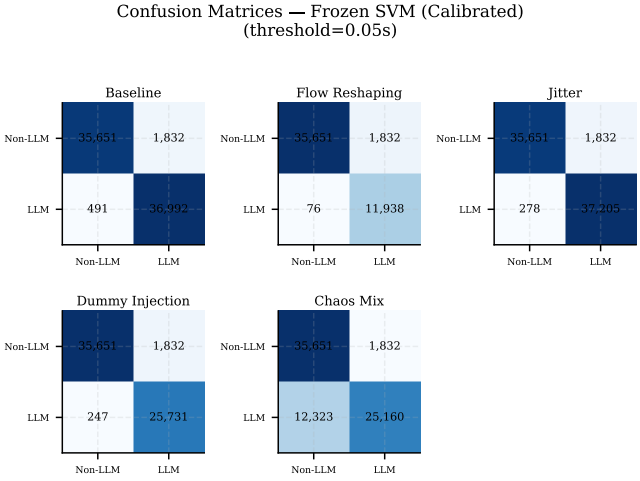

Figure 10: LLM vs. non-LLM classification confusion matrix (SVM, frozen adversary)

Table 2: Classification of obfuscated traffic using a frozen baseline-trained model.

Technique	Random Forest			XGBoost			SVM (Calibrated)		
	F1	AUC	Acc	F1	AUC	Acc	F1	AUC	Acc
Baseline	0.999	1.000	0.999	0.999	1.000	0.999	0.970	0.954	0.969
Flow Reshaping	0.975	0.977	0.988	0.975	0.997	0.988	0.926	0.957	0.961
Jitter	0.999	1.000	0.999	0.999	1.000	0.999	0.972	0.956	0.972
Dummy Injection	0.999	1.000	0.999	0.999	1.000	0.999	0.961	0.955	0.967
Chaos Mix	0.935	0.981	0.939	0.975	0.997	0.976	0.780	0.809	0.811

Table 3: Classification of obfuscated traffic using a frozen baseline-trained model.

Technique	Frozen F1	Retrained F1	Δ
Flow Reshaping	0.975	0.984	+0.008
Jitter	0.999	0.999	-0.000
Dummy Injection	0.999	0.999	-0.000
Chaos Mix	0.975	0.993	+0.018

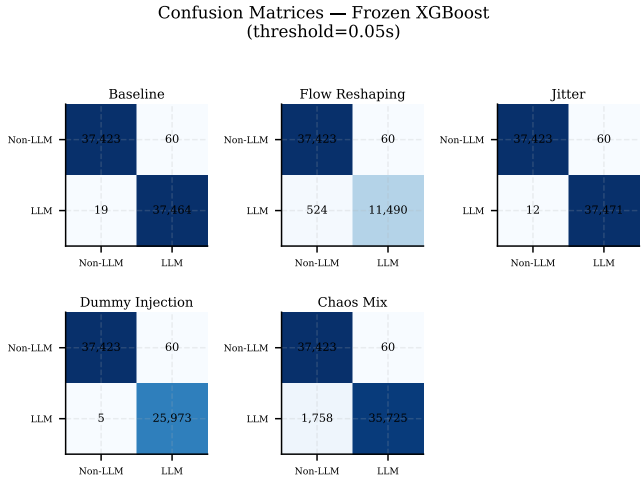

Figure 11: LLM vs. non-LLM classification confusion matrix (XGBoost, frozen adversary)

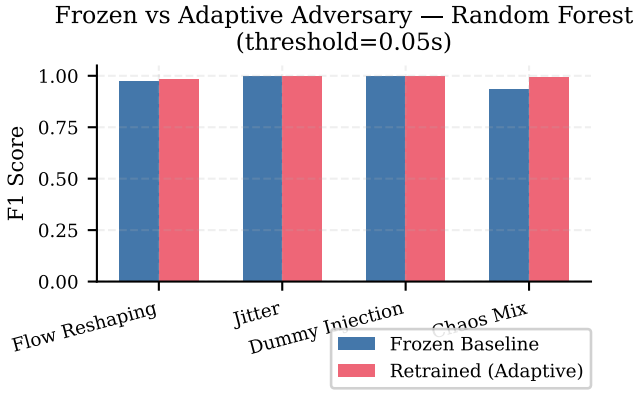

Figure 12: Detection drop for random forest model (frozen adversary versus adaptive adversary)

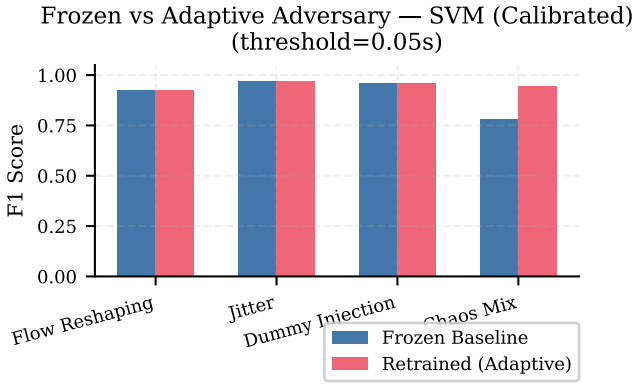

Figure 13: Detection drop for SVM model (frozen adversary versus adaptive adversary)

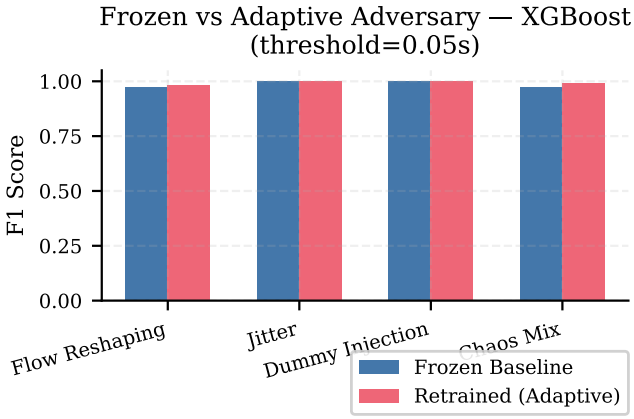

Figure 14: Detection drop for XGBoost model (frozen adversary versus adaptive adversary)

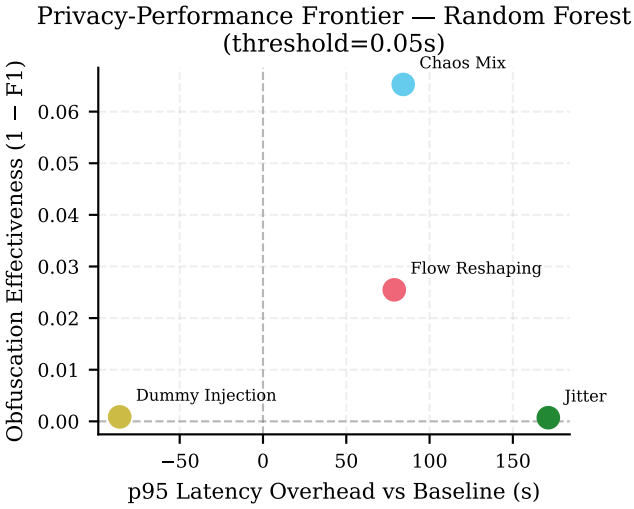

Figure 15: Privacy-performance tradeoff (Random Forest)

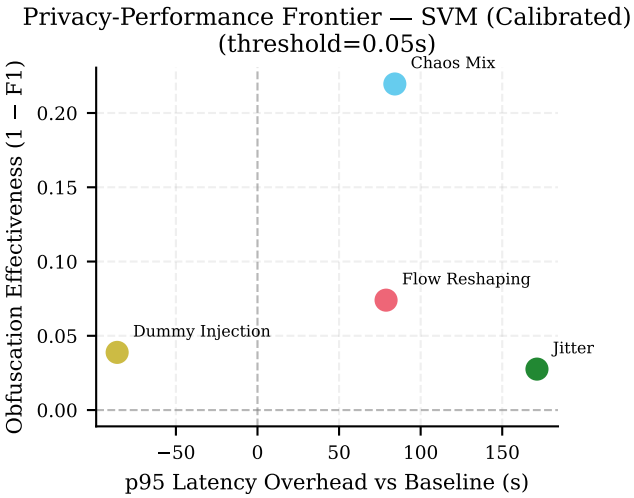

Figure 16: Privacy-performance tradeoff (SVM)

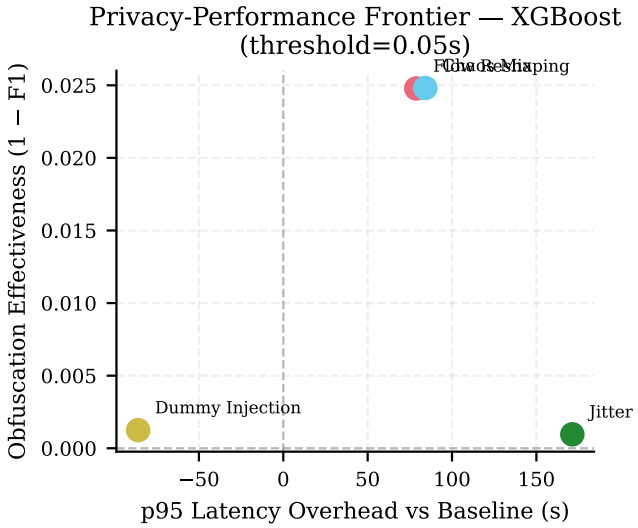

Figure 17: Privacy-performance tradeoff (XGBoost)

C Supplementary Figures (0.1s Threshold)

The following table and figures describe the data, classification rates, and performance impacts of the obfuscation methods evaluated from a 0.1s flowlet-partitioning threshold.

[t]

Technique	Test LLM Flowlets	Test Non-LLM
Baseline	28,190	28,190
Flow Reshaping	8,202	28,190
Jitter	28,190	28,190
Dummy Injection	13,832	28,190
Chaos Mix	28,190	28,190

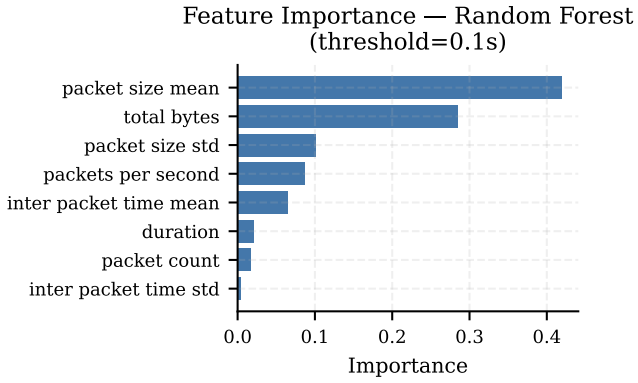

Figure 18: Random forest flowlet feature importance

Table 4: Classification of obfuscated traffic using a frozen baseline-trained model.

Technique	Random Forest			XGBoost			SVM (Calibrated)		
	F1	AUC	Acc	F1	AUC	Acc	F1	AUC	Acc
Baseline	0.999	1.000	0.999	0.999	1.000	0.999	0.974	0.968	0.974
Flow Reshaping	0.964	0.966	0.984	0.967	0.998	0.986	0.930	0.963	0.966
Jitter	0.999	1.000	0.999	0.999	1.000	0.999	0.976	0.970	0.976
Dummy Injection	0.999	1.000	1.000	0.999	1.000	0.999	0.956	0.969	0.970
Chaos Mix	0.876	0.976	0.890	0.897	0.998	0.907	0.769	0.817	0.805

Table 5: Classification of obfuscated traffic using a frozen baseline-trained model.

Technique	Frozen F1	Retrained F1	Δ
Flow Reshaping	0.967	0.987	+0.019
Jitter	0.999	0.999	+0.000
Dummy Injection	0.999	0.999	-0.000
Chaos Mix	0.897	0.995	+0.098

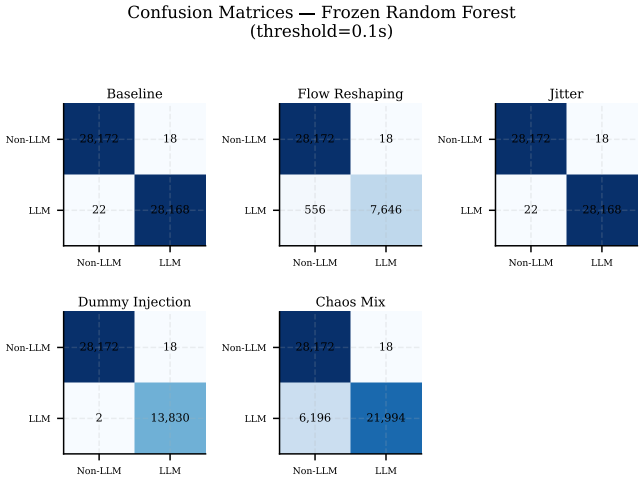

Figure 19: LLM vs. non-LLM classification confusion matrix (Random Forest, frozen adversary)

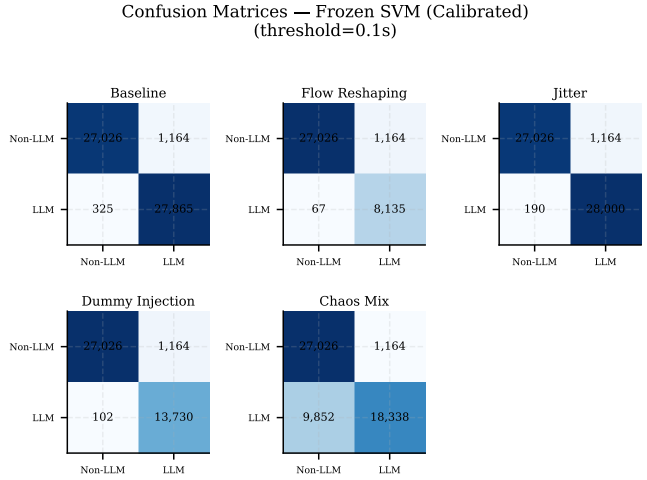

Figure 20: LLM vs. non-LLM classification confusion matrix (SVM, frozen adversary)

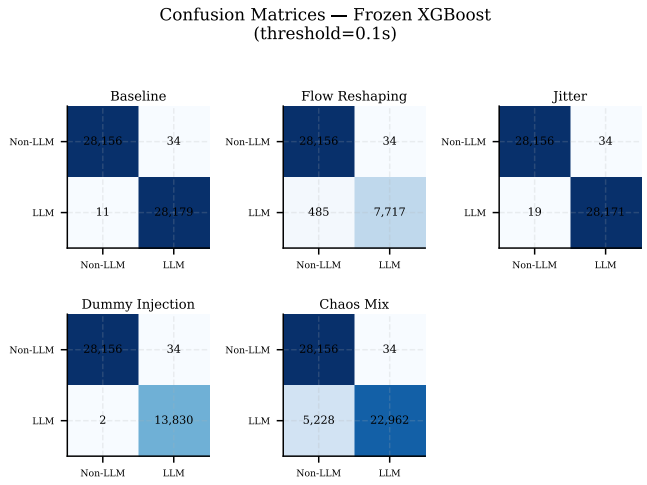

Figure 21: LLM vs. non-LLM classification confusion matrix (XGBoost, frozen adversary)

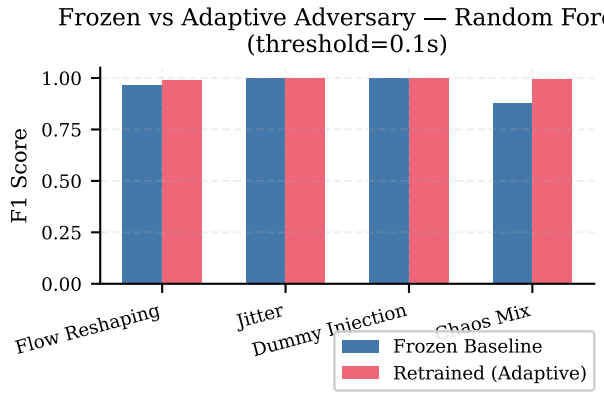

Figure 22: Detection drop for random forest model (frozen adversary versus adaptive adversary)

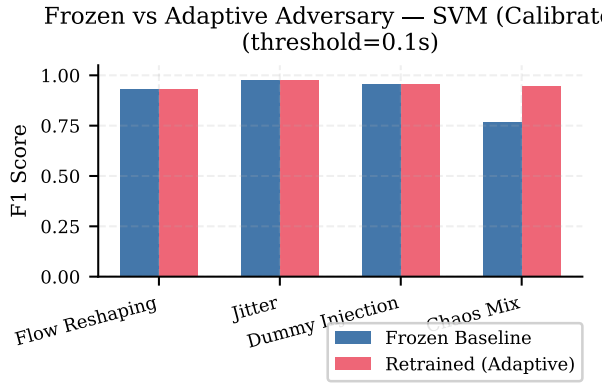

Figure 23: Detection drop for SVM model (frozen adversary versus adaptive adversary)

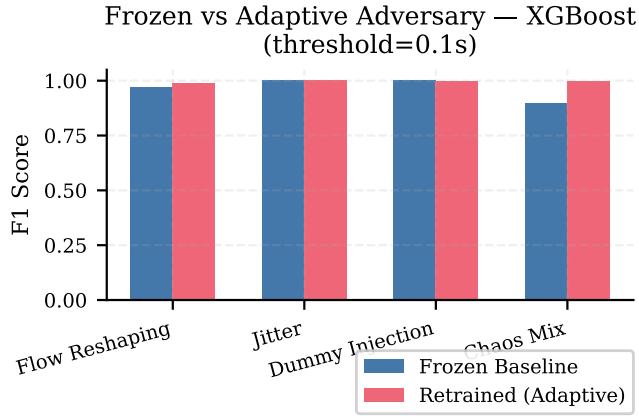

Figure 24: Detection drop for XGBoost model (frozen adversary versus adaptive adversary)

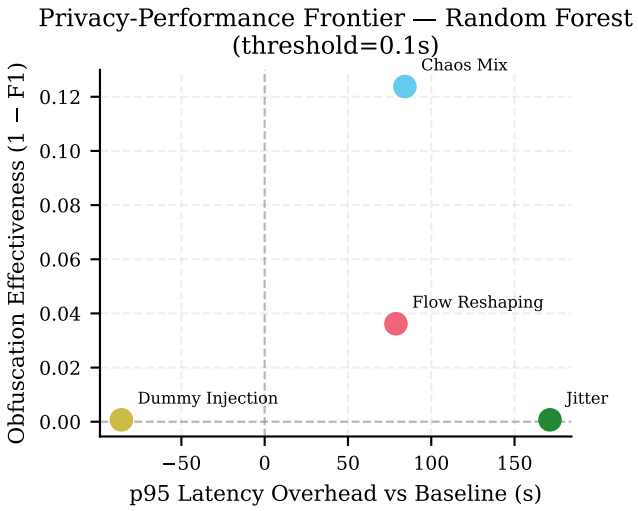

Figure 25: Privacy-performance tradeoff (Random Forest)

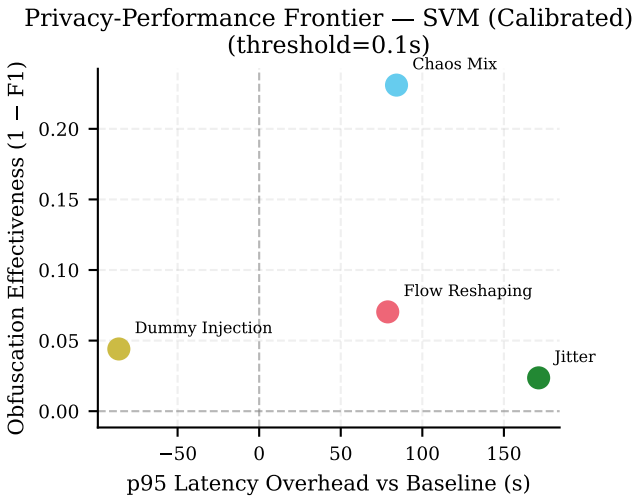

Figure 26: Privacy-performance tradeoff (SVM)

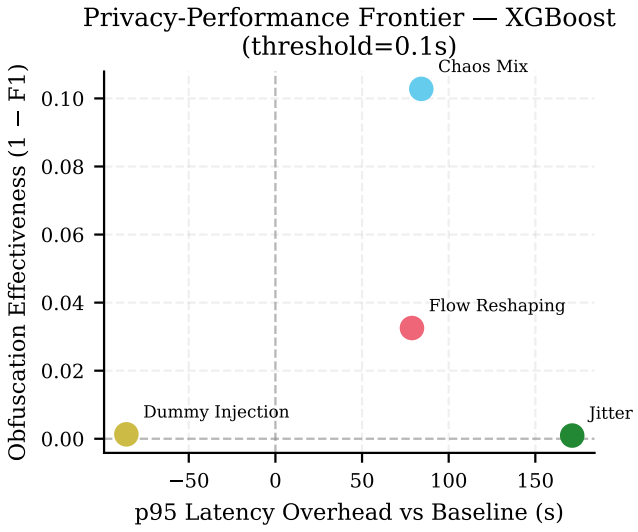

Figure 27: Privacy-performance tradeoff (XGBoost)

D Supplementary Figures (0.2s Threshold)

The following table and figures describe the data, classification rates, and performance impacts of the obfuscation methods evaluated from a 0.2s flowlet-partitioning threshold.

[t]

Technique	Test LLM Flowlets	Test Non-LLM
Baseline	13,592	13,592
Flow Reshaping	4,889	13,592
Jitter	13,592	13,592
Dummy Injection	1,969	13,592
Chaos Mix	12,421	13,592

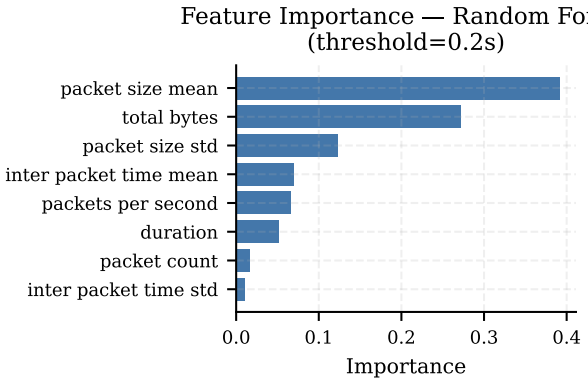

Figure 28: Random forest flowlet feature importance

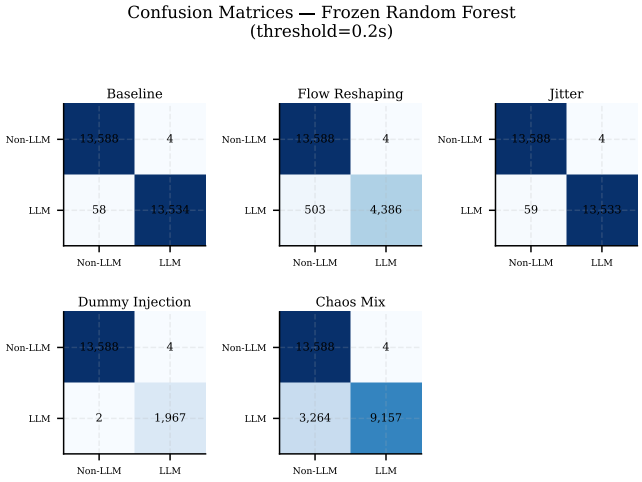

Figure 29: LLM vs. non-LLM classification confusion matrix (Random Forest, frozen adversary)

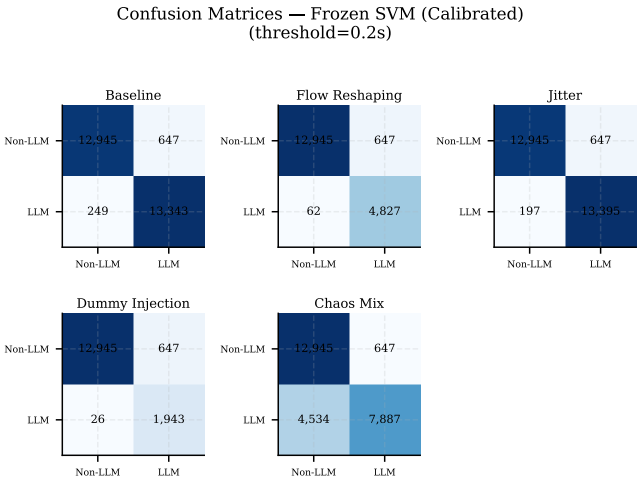

Figure 30: LLM vs. non-LLM classification confusion matrix (SVM, frozen adversary)

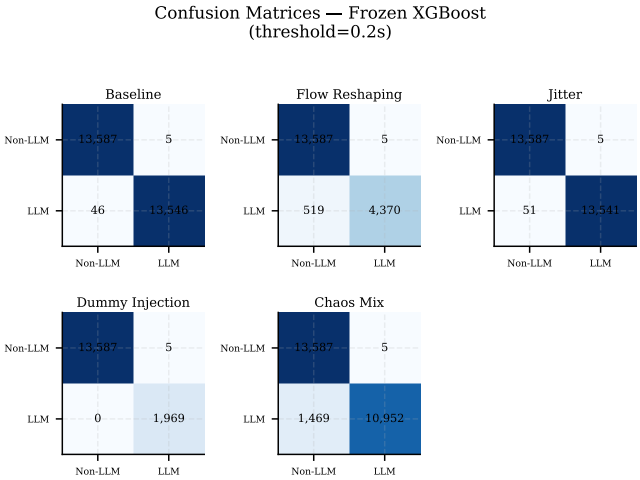

Figure 31: LLM vs. non-LLM classification confusion matrix (XGBoost, frozen adversary)

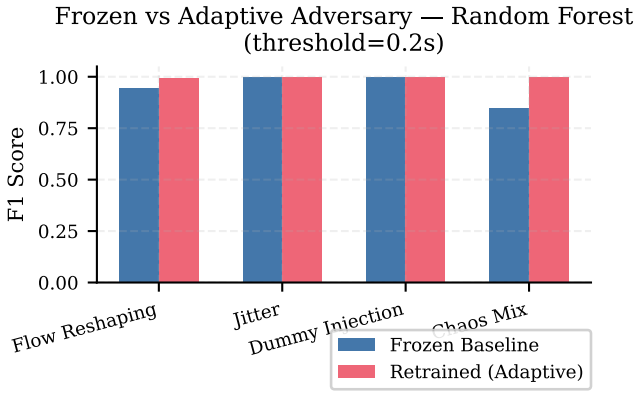

Figure 32: Detection drop for random forest model (frozen adversary versus adaptive adversary)

Table 6: Classification of obfuscated traffic using a frozen baseline-trained model.

Technique	Random Forest			XGBoost			SVM (Calibrated)		
	F1	AUC	Acc	F1	AUC	Acc	F1	AUC	Acc
Baseline	0.998	1.000	0.998	0.998	1.000	0.998	0.968	0.971	0.967
Flow Reshaping	0.945	0.951	0.973	0.943	0.996	0.972	0.932	0.955	0.962
Jitter	0.998	1.000	0.998	0.998	1.000	0.998	0.969	0.965	0.969
Dummy Injection	0.998	1.000	1.000	0.999	1.000	1.000	0.852	0.973	0.957
Chaos Mix	0.849	0.949	0.874	0.937	0.996	0.943	0.753	0.864	0.801

Table 7: Classification of obfuscated traffic using a frozen baseline-trained model.

Technique	Frozen F1	Retrained F1	Δ
Flow Reshaping	0.943	0.992	+0.049
Jitter	0.998	1.000	+0.002
Dummy Injection	0.999	0.998	-0.001
Chaos Mix	0.937	0.995	+0.058

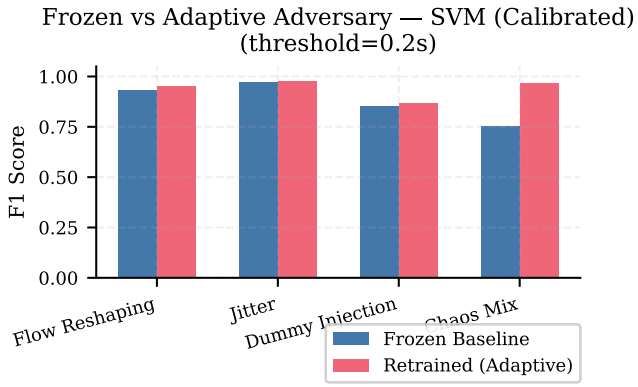

Figure 33: Detection drop for SVM model (frozen adversary versus adaptive adversary)

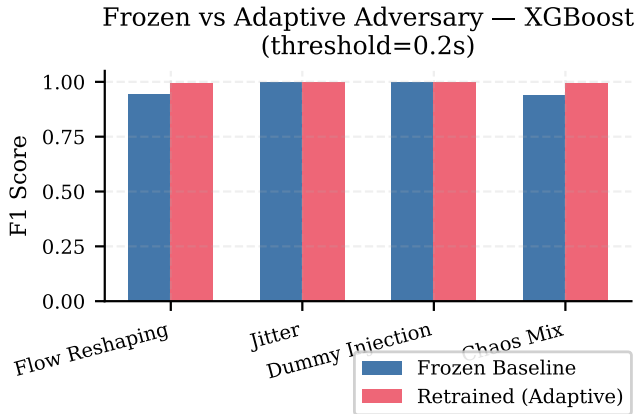

Figure 34: Detection drop for XGBoost model (frozen adversary versus adaptive adversary)

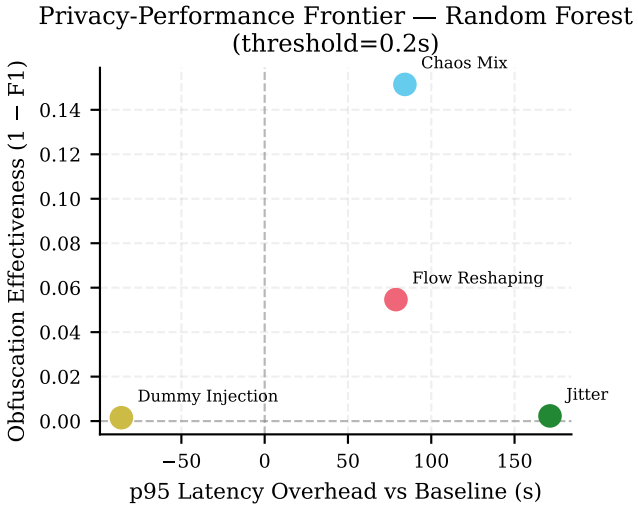

Figure 35: Privacy-performance tradeoff (Random Forest)

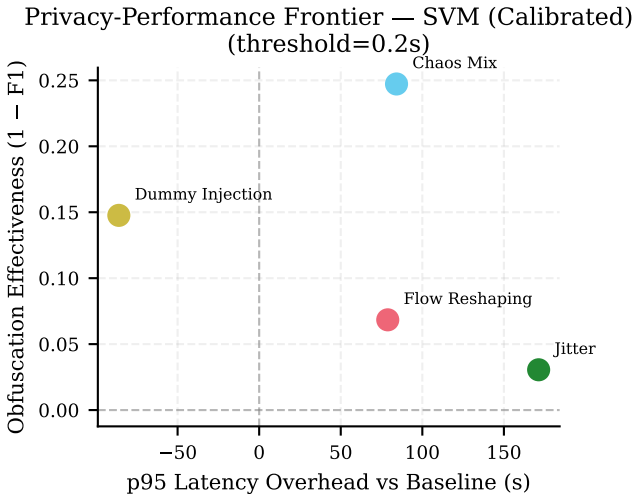

Figure 36: Privacy-performance tradeoff (SVM)

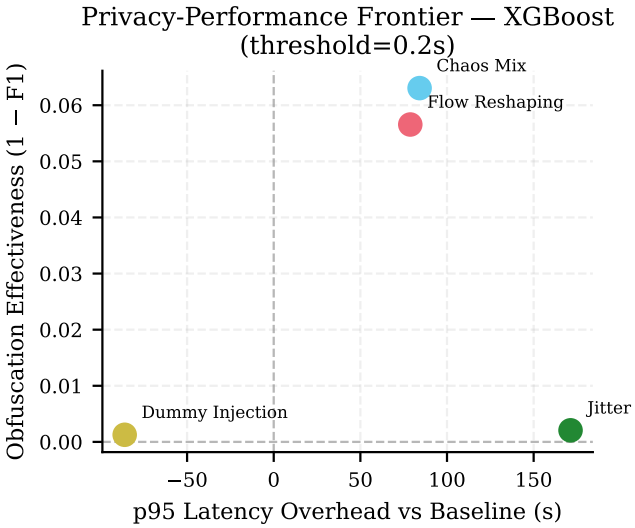

Figure 37: Privacy-performance tradeoff (XGBoost)

Notes

This research is performed as part of *CS 6501 - Security of AI Systems: Attacks and Defenses* at the University of Virginia. It has not gone through any review processes and is limited to course resources.