Local Modeling and Privacy Analysis of LLMTor

Gavin Crigger, Riley Immel, Matthew Lucio*
University of Virginia
Charlottesville, Virginia

ABSTRACT

LLM-Tor is a recently proposed privacy layer that uses blind signatures and Tor routing to decouple a user's paying identity from their later LLM queries. We audit whether these privacy claims hold up in practice by building a faithful local recreation of the system — desktop client, LLM proxy, Tor routing, and an LM Studio backend, isolated through Docker — and capturing traffic at three TAPs along the path. Using an automated bot that issues sensitive and nonsensitive prompts from a labeled prompt bank, we apply the WhisperLeak side-channel attack against the captured traffic and separately audit the proxy's logging, caching, and pseudonym behavior against LLM-Tor's stated privacy claims. We find that the cryptographic core of the system is sound, but the surrounding metadata is not: shared Phase 2 and Phase 3 logs let a heuristic re-link anonymous queries to the paying user 70.3% of the time, the abuse pseudonym is a stable identifier that compounds any successful linkage across a user's entire history, cache hits return roughly 50,000 times faster than fresh LLM calls, and response size still correlates with prompt length. WhisperLeak itself performs near coin-flip on the client-facing tap (AUPRC 0.62, recall 0.19), but this protection comes largely from full-response buffering, which is an implementation choice rather than a protocol requirement. Our results suggest that LLM-Tor's privacy guarantees rest more on incidental implementation details than its authors' threat model implies, and we outline concrete protocol-level changes to close the gaps.

1 INTRODUCTION

As private LLM usage continues to grow across the world, a significant body of LLM privacy attacks have become prevalent in the literature [1, 5, 8, 9, 11–13]. Many of these attacks derive personal information and make accurate conversation inferences based simply upon the traffic produced by LLM conversations. While most of these attacks remain limited to research environments in their demonstrated impact, they pose the foundational risks of widespread LLM adoption without formal understanding of privacy catching up in the public eye. Users operating on the internet have an inherent expectation of privacy from eavesdroppers, and thus may look for ways to safeguard their privacy while interacting with LLM services. This is where LLM-Tor comes in.

LLM-Tor is claimed to be *"a cryptographically enforced privacy layer that enables unlinkable access to commercial Large Language Model (LLM) APIs"* [2]. This works through routing desktop clients to a proxy that handles blind authorization of LLM queries as a paid service. Creating a service that makes users unlinkable to their queries (at least in the anonymity set of LLM-Tor users) would make existing traffic-based privacy attacks significantly more difficult, even for a global eavesdropper should appropriate timing obfuscation techniques be applied [4]. We set out to investigate whether these privacy claims are held up by LLM-Tor's architecture in this paper.

2 BACKGROUND

LLM privacy attacks generally work from the observation that encrypted traffic does not hide all useful information. Even when an eavesdropper cannot read the contents of a user's message, they can still learn useful information about the interaction from the timing, direction, size, and sequences of packets. This is especially concerning for LLM conversations as prompts often contain private personal, professional, medical, or academic information. A user may naively assume that just because they are using HTTPS or another encrypted transport, their activity is sufficiently hidden. This assumption falls short of the truth as traffic analysis attacks show that privacy can still fail through side channels rather than direct content exposure.

Recent work on LLM traffic analysis demonstrates that these side channels are not just theoretical. WhisperLeak, for example, studies how an attacker can infer sensitive attributes of an LLM conversation from just observable network traffic patterns and without decrypting messages [5]. We discuss this more in Section 4 where we tried applying the WhisperLeak attack directly to LLM-Tor traffic. Other work similarly shows that token streaming, response timing, packet sizes, and request/response structure can leak information about what a user is doing with an LLM service. The common theme across these attacks is that the privacy risk comes from metadata that remains visible despite the actual content itself being encrypted/protected. This makes LLM traffic a useful target for adversaries who can observe network flows but are unable to break encryption.

*Paper authored and research conducted as part of CS 6501 - Network Security and Privacy at the University of Virginia Graduate School of Engineering and Applied Sciences, taught by Yixin Sun

Tor serves as a natural candidate for defense against these risks thanks to its design which focuses on hiding the relationship between a client and the service they are accessing. In Tor, traffic is routed through multiple relays so that no single relay should learn both who the user is and what destination they are contacting [10]. This design provides network-level anonymity, but does not automatically eliminate all traffic analysis risks. If an eavesdropper were to be powerful enough such that they could observe traffic both near the client side and the service side, they may still try to correlate flows using timing and traffic-volume patterns. Therefore, Tor improves unlinkability, but its protection depends heavily on the adversary model and whether the application leaks distinguishable traffic patterns.

LLM-Tor builds on this idea by attempting to separate user identity, payment, and LLM usage (actually querying the LLM's API). The central privacy goal is that a user should be able to purchase credits for use with an LLM service and then later submit queries using those credits without their queries being linkable back to the purchasing identity [2]. To accomplish this, LLM-Tor uses blind authorization techniques so that the service can verify that a query is paid for without learning which specific authenticated user originally purchased the token. In this design, the LLM proxy acts as the privacy boundary: it verifies anonymous authorization tokens, strips identifying information, and forwards the query to a commercial LLM API.

However, the simple existence of this cryptographic separation does not by itself prove that the system is private against traffic-based attacks. The cryptographic layer may prevent direct account-to-query linkage at the application level, but a network observer may still attempt to connect users to queries through timing correlation, flow shape, or other metadata. This distinction is important for our paper because LLM-Tor's privacy claims depend on both the authorization design and the surrounding network behavior. Our background assumption is therefore, that LLM-Tor should be evaluated not only as a cryptographic protocol, but also as a networked system exposed to traffic analysis.

3 SETUP

Due to our resource limitations, we cannot perform a full global eavesdropper attack on LLM-Tor as it exists on the internet. We instead create a modified setup consisting of the desktop client, LLM proxy, LM Studio API endpoint, and communication with an ngrok forwarding proxy over Tor [3, 7, 10]. This architecture includes capture "TAPs" (Target Access Points) at three key points - TAP A exists between the desktop client and the Tor network, TAP BC captures incoming traffic forwarded from the ngrok endpoint (mapped to the corresponding Tor exit nodes), and TAP D captures

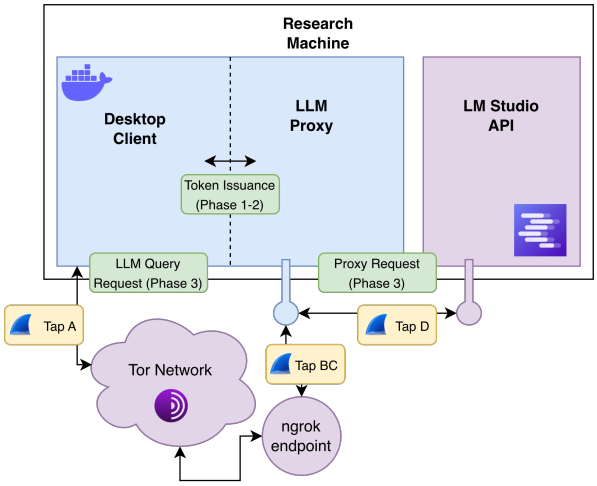

Figure 1: High-level representation of our modified LLM-Tor service, with desktop client, LLM proxy, and LM Studio all running locally.

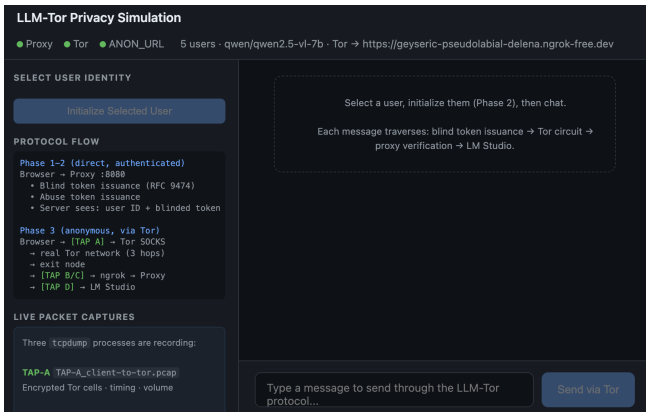

Figure 2: Research machine prototype desktop client

communication between the LLM proxy and the LM Studio API. These TAPs are essential to assessing traffic fingerprinting potential and performing the WhisperLeak attack on the LLM-Tor architecture.

Desktop client: The desktop client serves as the only frontend for LLM-Tor users. For LLM-Tor, this service handles credit purchasing, sign-in, and query issuance over Tor. Interestingly, the LLM-Tor desktop client will not open without Tor running, yet it will not *close* either if you kill the Tor process after having opened the client.

Our recreation of the desktop client maintains the key cryptographic/privacy properties of the LLM-Tor client. This implementation supports client authentication, blind RSA, and anonymous querying in line with LLM-Tor. Additionally,

our desktop client includes multi-user simulations and is open to the "TAP A" capture, and it is virtualized through Docker [6].

LLM proxy: LLM-Tor performs most of its privacy guaranteeing operations via the LLM proxy. This is made up of a Go server which issues purchased tokens and handles anonymous LLM queries. This is done by verifying blind signatures / abuse tokens before forwarding queries to the desired LLM API (stripped of any identifiable information).

Our proxy recreation remains faithful to these privacy techniques by using a Go server with the same authentication flow, blind signing, and abuse token system. The key distinction that our resource limitations led us to, is that this proxy is hosted on the local research machine, though the service is notably isolated from the desktop client as they are both containerized via Docker.

Traffic forwarding: Communication over the Tor network is the cornerstone of LLM-Tor's privacy guarantees. Since both the desktop client and the LLM proxy are run locally on the research machine, we require a forwarding proxy to use Tor in our architecture. This is where ngrok comes in: we deploy an ngrok forwarding service that receives traffic and sends it back to the research machine. To be precise, the desktop client sends traffic to the ngrok proxy *over the Tor network*, and the ngrok proxy forwards the traffic back to the research machine's LLM proxy container, maintaining service isolation on the machine. This provides a faithful reconstruction of the LLM-Tor networking scheme that we assert is sufficient for the purposes of this privacy analysis.

4 WHISPERLEAK ATTACK

The WhisperLeak side channel attack [5] works on several pieces within LLM architectures. The first requirement for the attack is the ability to listen on the targeted user's network or path to the backend LLM. Next, the attacker must be able to correlate the traffic under study with the actual user to render the discovered conversation attributes useful. Finally, the attacker may perform the attack by collecting data to recognize the traffic patterns of sensitive data. This gives us a clear roadmap. We first collect traffic data on benign and sensitive prompts at each of our capture TAPs along with network traffic that is not using LLM-Tor, then we use that data to train models to apply the side-channel attack.

4.1 Data Collection

Algorithm 1 Labeled prompt collection

Require: Labeled prompt bank P_ℓ, where $\ell \in \{sensitive, nonsensitive\}$, simulated LLM-Tor research system up via Docker compose

Load prompt bank P_ℓ from `prompts_ℓ.json`
Initialize simulated users per client architecture
for each prompt $p \in P_\ell$ **do**
    $t_0 \leftarrow$ now()
    Submit prompt p as user u via web client (Phase 2 blind-token issuance, then Phase 3 anonymous proxy via Tor)
    Await response completion; record Phase 2 / Phase 3 latency
    Log prompt and response event
    Sleep for base delay w/ jitter
end for
Close log writers; capture sidecars continue until container shutdown

We employ an automated methodology to collect mass amounts of simulated LLMTor traffic data for the WhisperLeak attack analysis. We chose to evaluate the Qwen2.5 VL provider for its free API. Running `docker compose up` our architecture with an environment variable `WHISPER_MODE=<sensitive/nonsensitive>` and `–profile whisperleak` sets up a bot that queries our LLM proxy through several users. This is done via one of two prompt banks – one with sensitive prompts and one with nonsensitive prompts in line with WhisperLeak's predictive attack. Capture TAP results and event logs are named according to the timestamp and sensitivity label. We collected about 3,000 flowlets worth of data (mostly at TAP A) before proceeding with the LLM-Tor WhisperLeak attack analysis.

As a baseline to measure the effectiveness of LLM-Tor against WhisperLeak, we also developed a Python script to collect device network traffic (using tshark a CLI version of wireshark) and send data to our privately hosted MongoDB database. The user must provide an SSL key file to decrypt their traffic in order to filter out packets that are not talking to LLMs. We split their packets into flowlets: 5-tuples (Src IP, Dst IP, Src Port, Dst Port, Protocol: TCP/UDP) based on a time threshold (0.1 seconds). We also developed a Python script to spawn Selenium Google Chrome instances that visit either ChatGPT, Gemini, or Claude (in the browser to not accumulate API token costs) and sends multiple prompts (either sensitive or non-sensitive) iteratively to simulate human conversations. We use the undetected-chromedriver library to bypass Cloudflare bot detection. We collected about 10,000 flowlets (roughly evenly distributed between the LLM

providers but more for ChatGPT) worth of data before proceeding with the non LLM-Tor WhisperLeak attack analysis.

4.2 Attack Methodology

After collecting a reasonable amount of data (although we suspect that collecting more would only improve our performance) for both the non LLM-Tor baseline and LLM-TOR datasets, we train 3 binary classifiers (Random Forest, XGBoost, SVM, etc.) with sklearn to predict if LLM conversation contains sensitive content (e.g., Money Laundering). As mentioned earlier, the raw packets are parsed into flowlets, with important metadata features such as packet size and timing mean and standard deviation that cannot be obfuscated (although recent patches to LLM providers have tried to batch responses together and add random delay to mitigate attacks like theses). For both situations, Random Forest tended to perform the best overall so we focused on those results in our results in Table 1.

4.3 Attack Results

Table 1: Comparison of WhisperLeak Attack Effectiveness: Standard TLS vs. LLM-Tor (Qwen 2.5 VL) with Random Forest

Environment	LLM Model	AUPRC	ROC-AUC	Precision	Recall	F1-Score
Standard TLS	Gemini	**0.9753**	**0.9153**	**0.8956**	**0.8935**	**0.8946**
Standard TLS	Claude	0.5548	0.8250	0.5103	0.5564	0.5324
Standard TLS	ChatGPT	0.5435	0.6611	0.5393	0.4991	0.5185
LLM-Tor	Qwen 2.5 VL (TAP A)	0.6196	0.5285	0.6821	0.1932	0.3011
LLM-Tor	Qwen 2.5 VL (TAP D)	0.6964	0.6096	0.7547	0.2963	0.4255

Table 1 compares the overall results (most importantly AUPRC, Precision, and Recall) for the standard TLS implementation of WhisperLeak (without LLM-Tor) across 3 LLM providers and the LLM-Tor results at TAPs AB and D with the Qwen 2.5 VL provider. Notably, AUPRC, precision and especially recall, drop significantly when using LLM-Tor. For example, without LLM-Tor our model achieves a 0.97 AUPRC and 0.89 recall for Gemini to 0.62 AUPRC and 0.19 recall for Qwen 2.5 (with LLm-Tor). However, the difference in LLM providers, amount of data, and capture methodology mean these results should be taken with a grain of salt. We leave our interpretation of these results to the Discussion section.

5 EVALUATION

5.1 Audit of LLM-Tor's Privacy Claims

For our audit, we looked at our own faithful recreation of LLM-Tor rather than the original binary. This means that findings about the upstream LLM-Tor source require the caveat that we cannot rule out differences between our implementation and theirs. However, our recreation was built directly from the published LLM-Tor specification and repository, and the holes we found are structural — they follow from the protocol design and log schema rather than from implementation bugs — so the findings apply to any implementation that follows the same design. The tap capture infrastructure (pcap sidecars and structured event logs) is our addition and not present in the original code; it exists purely to enable measurement and does not affect the protocol behavior under audit.

We audited the proxy server (Go), desktop and web clients (TypeScript/Node), and Tor configuration against LLM-Tor's four stated privacy claims: (C1) the proxy cannot link a paying user to a later anonymous query, (C2) Tor hides the user's IP from the proxy, (C3) content is confidential in transit, and (C4) abuse pseudonyms allow rate-limiting without identifying users. We label holes in these claims H1–H6; D5 is a standalone factual check that does not correspond to a hole.

D5 — Response delivery over Tor. Clients issue Phase 3 requests through a SOCKS5 agent and read the response on the same connection, so the reply rides the same Tor circuit. We confirmed this empirically: all return traffic on tap A was between the client and the Tor guard, and filtering tap A for proxy IP returned zero packets. C3 holds at the network layer.

H1 — Log-level user linkage. The proxy's existing event log records the authenticated user's real IP at Phase 2 token issuance (`main.go:368-375`) and the Tor exit IP plus permanent pseudonym at Phase 3 (`handler.go:111`) in the same file. Any party with access to that log such as the proxy operator or an attacker who obtains it, can time-correlate the two streams without breaking any cryptography: a token issued for a user shortly before an anonymous Phase 3 request arrives is likely that user's request. We measured this on a 3-user, 75-query run: a heuristic that picks the most recent Phase 2 issuance within 60 seconds guessed the correct user 70.3% of the time overall, and was correct every single time it had any candidate in the window. This breaks C1.

H2 — Permanent pseudonym is a stable identifier. The permanent abuse pseudonym is a deterministic hash of a long-lived token (`handler.go:113-115`). LLM-Tor acknowledges this stability and argues it is safe because the pseudonym cannot be tied back to a purchasing identity. That argument holds in isolation, but it compounds with H1: if the timing attack links even a single Phase 3 request to a real user, every subsequent request that user ever makes, past or future, carrying the same pseudonym is also linked. A 3-user, 9-message run confirmed the 1-to-1 mapping holds by construction. In a real deployment the exposure grows with usage.

H3 — Cache timing oracle. When a spent token is replayed, the proxy returns a cached response rather than re-querying the LLM. We forced a cache hit by extracting

a spent POST body from a tap BC capture and replaying it against the proxy. In a single measurement, the fresh LLM call took 5078 ms; the cache hit returned in under 1 ms, a ~50,000× gap. While we only measured one instance of each, the gap is structural rather than incidental: the cache hit path skips LLM inference entirely, so its latency is bounded by local handler overhead regardless of the request. This makes replays trivially detectable to a passive observer on tap A. Side-channel against C1.

H4 — Response size leakage. Response size is logged at `handler.go:245-246` and is also approximately observable on tap A as Tor cell count. On 74 matched pairs from a WhisperLeak-profile run we measured Pearson $r = +0.325$ between prompt length and response size. This is a weak correlation, but the WhisperLeak prompt bank is intentionally length-balanced to avoid giving classifiers an easy size signal. So +0.325 is likely a lower bound on the leakage in a natural usage setting where prompts vary more widely in length and topic. The channel exists regardless; its practical strength depends on what the attacker knows about the user's prompt distribution.

H5 — Full-response buffering as an unintentional defense. The proxy buffers the full LLM response before sending (`handler.go:220`), removing inter-token timing from the attacker's input. We believe this is the primary reason the WhisperLeak attack fails on tap A (Section 4). This should be an explicit protocol requirement, not an incidental implementation detail.

H6 — Per-user circuit isolation. `IsolateSOCKSAuth` gives each user their own circuit, which is correct for exit unlinkability. A concern is that a guard could count circuits to count users, but comparing tap A TCP tables between a 1-user and 3-user run showed the same 2 guard connections in both cases—Tor multiplexes circuits over the same TCP sessions. A circuit-timing intersection attack is theoretically possible but requires Tor-cell-level analysis, well beyond a passive adversary. Minor concern against C2.

Overall, the cryptographic core (RFC 9474 blind RSA, abuse token verification, request stripping) appears sound. The practical holes are in the surrounding metadata: H1, H3, and H4 together give an adversary multiple non-cryptographic paths to attack C1, and H2 compounds with H1 to mean that a single successful linkage exposes a user's entire query history.

6 DISCUSSION

The WhisperLeak results suggest LLM-Tor does dampen the side channel on the client-facing side. At tap A, AUPRC is 0.6196 and recall is only 0.1932, which is roughly coin-flip territory on the class that actually matters for a real user. Tap D does a bit better at 0.6964 since that segment never crosses Tor and isn't shaped by the proxy's response buffer. We want to be careful with the Gemini comparison since the underlying model is different (Qwen 2.5 VL vs. Gemini), but the Claude and ChatGPT TLS rows sit at 0.54–0.55 AUPRC, so the model is probably doing some of the work. The cleaner takeaway is that H5 from Section 5.1 lines up with what we see at tap A: the proxy buffers the full response before forwarding it, which kills the inter-token timing WhisperLeak relies on. Tor helps with unlinkability, but buffering is what is really blocking this particular attack.

We found LLM-Tor's privacy claims to be cryptographically solid but leaky everywhere around the cryptography. The blind signatures, abuse-token checks, and request stripping all do what the spec says. The problems lie in the metadata. H1's log timing correlation re-linked Phase 3 queries to the right Phase 2 user 70.3% of the time without breaking anything, and H2 makes that worse since the abuse pseudonym is a deterministic hash of a long-lived token; one successful linkage exposes every query that user will ever make. Add H3's ~50,000× cache-hit timing gap and H4's residual prompt-to-response size correlation, and there are several ways to chip at C1 without touching the blind RSA. C2, C3, and C4 mostly hold; C1 is where the real exposure is.

Future work should focus on the metadata side since that is where the cryptography doesn't help. Phase 2 and Phase 3 events shouldn't share a log with correlatable timestamps, and the abuse pseudonym should rotate so that a single linkage doesn't blow up a user's full history (though this trades off against the abuse-tracking the pseudonym is there to provide). Padding cache hits to a fixed latency would close H3, and bucketing response sizes would limit H4. We'd also like to see full-response buffering written into the spec rather than left as an implementation choice, since dropping it for a streaming UX would silently undo the main thing protecting tap A. Our setup is also just one research machine, so a bigger study with actual Tor exit observation and more varied prompts would give a much better read on how H4 holds up in real use.

7 CONCLUSION

In this report, we discuss our privacy audit of LLM-Tor, a service dedicated to decoupling personal identity from LLM queries on a payment-based model. Payment is over Stripe and not Tor, thus linking users to the LLM-Tor service as an anonymity class smaller in scope from general LLM usage. We establish a modified LLM-Tor simulation that can run on an individual research machine, with a desktop client, LLM proxy, and Tor routing service isolated through Docker containers plus communication with an LM Studio backend LLM. We automate sensitive/nonsensitive data collection through a bot that sends queries from a prompt bank through the

architecture. Our privacy audit told us that LLM-Tor's blind signature core is sound, but the surrounding metadata is not — shared logging, a stable abuse pseudonym, a cache-hit timing oracle, and residual response-size leakage give an adversary several non-cryptographic paths to re-link users to their queries, and the system's strongest in-practice defense (full-response buffering) is an implementation accident rather than a protocol requirement. Finally, the WhisperLeak attack showed LLM-Tor meaningfully mitigated the AUPRC and especially the recall at two TAPs. Future work should further investigate the privacy holes of LLM-Tor, ideally on the application itself, and with more training data to strengthen the argument that it defends against attacks like WhisperLeak.

8 ETHICS

Privacy is an integral part of networks research, especially in this paper. Our research upholds user privacy assumptions by not decrypting any packets and operating solely on visible attributes of their traffic flow. All training data came from simulated network packet captures limited to our personal devices (no sniffing of non-consenting party traffic). Additionally, we assert that this audit of user privacy on LLM-Tor is of benefit to the broader public user set of LLM services.

9 ARTIFACTS

Our codebase and evaluation scripts can be found at https://github.com/gavinvc/netsec-project

10 ROLES

Gavin implemented the modified setup of LLM-Tor that is adapted to our available resources, which includes a locally-hosted desktop client (containerized), locally-hosted LLM proxy (containerized), local LLM API (LM Studio), Tor network routing, routing service via ngrok, and captures at TAPs for training data.

Riley conducted the privacy audit and related literature survey. This included reviewing the LLM-Tor protocol design and source code to identify structural privacy holes (H1–H6), collecting data for use in auditing, writing the audit demo scripts and analysis tools that measured each finding against captured traffic, and documenting the results. Riley also surveyed the existing literature and datasets relevant to Tor traffic analysis and LLM privacy attacks.

Matthew implemented the WhisperLeak data collection and analysis pipeline. This included creating a Python script to collect device network traffic (using tshark) and the Python script to spawn Selenium Google Chrome instances that visit either ChatGPT, Gemini, or Claude to simulate human conversations without LLM-Tor. Matthew also trained binary classifiers with sklearn to predict if LLM conversation contains sensitive content and evaluated the results for both non LLM-Tor and with LLM-Tor.

REFERENCES

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

[2] Prince Gupta. 2026. LLM-Tor: Unlinkable Access to Commercial LLM APIs via Blind Signatures and Tor. (2026).

[3] LM Studio. 2026. LM Studio. (2026). https://lmstudio.ai Local inference desktop application for running large language models.

[4] Federico Mason, Federico Chiariotti, Pietro Talli, and Andrea Zanella. 2026. Secure Goal-Oriented Communication: Defending Against Eavesdropping Timing Attacks. *IEEE Journal on Selected Areas in Communications* 44 (2026), 3782–3797. https://doi.org/10.1109/JSAC.2026.3669875

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

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

[7] ngrok Inc. 2026. ngrok. (2026). https://ngrok.com Secure tunneling, reverse proxy, and public URL exposure for local servers.

[8] 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. (2025).

[9] 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.

[10] The Tor Project. 2026. Tor. (2026). https://www.torproject.org/ Free and open-source software for enabling anonymous communication.

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

[12] 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.

[13] 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. (2025). arXiv:cs.CR/2510.07176