SoK: Software-Controlled Side Channels in LLM Inference

Abla Smahi[1] and Jan Tobias Mühlberg[2]

[1] Zhejiang University, 866 Yuhangtang Road, Hangzhou 310058, Zhejiang, China
smahi_abla@zju.edu.cn
[2] Université Libre de Bruxelles, Avenue Roosevelt 50, 1050 Bruxelles, Belgium
jan.tobias.muehlberg@ulb.be

Abstract. Large language model (LLM) services often process sensitive prompts and generate equally confidential responses. Even when prompts, responses, and model state are not directly exposed, inference can still reveal information indirectly: prompt reuse can change latency, streaming can shape encrypted traffic, and content-dependent execution can leave traces on shared hardware. Recent attacks exploit such effects across different serving mechanisms, deployment settings, and attacker models, making it difficult to compare leakage paths and defenses. We systematize *software-controlled side channels* in LLM inference: leakage paths in which a software, runtime, or model-execution decision causes protected information to influence an attacker-visible signal. We distinguish these channels from failures that directly expose retained inference state or transmitted intermediate representations. We organize the literature by the inference-time decision that creates the observable dependence, while independently categorizing the signal, attacker position, leakage target, architectural or microarchitectural observation layer, and evidence status. Our taxonomy covers shared serving state, microarchitecturally observable execution, network-observable delivery, input/output policies, and mixture-of-experts (MoE) execution. We separate peer-reviewed or accepted core evidence from emerging non-archival studies and boundary cases. Our analysis shows that leakage extends beyond conventional multi-tenant serving to confidential-VM and partitioned-GPU settings, while evaluated defenses remain uneven and difficult to compare. Finally, we identify open problems in reproducible leakage measurement, interacting inference mechanisms, and security–performance trade-offs for protected LLM deployment.

Keywords: Large Language Models · Side Channels · LLM inference

1 Introduction

A prompt sent to a large language model (LLM) can reveal far more than a user's question. It may contain medical information, financial or identity details, confidential documents, proprietary code, or private concerns. The generated response can be equally sensitive because it may summarize, transform, or reason over that protected context [7]. When an LLM service prevents direct access to

both the prompt and the response, users reasonably expect that the substance of their interaction remains private.

Protecting the plaintext of an interaction, however, does not necessarily hide the observable effects of processing or delivering it. An LLM service does not handle a request as an opaque operation. It processes input tokens, stores key–value (KV) representations needed for generation, and reuses this cached context as new tokens are produced [31]. Serving runtimes may further retain KV state across requests when prompts share a prefix [42], batch multiple requests to improve accelerator utilization [39], stream generated tokens to preserve responsiveness [5], or route tokens through selected experts in mixture-of-experts (MoE) models, where a learned router sends each token to only a small subset of expert sub-networks [16]. These mechanisms improve efficiency or functionality, but they also create a confidentiality concern: protected inputs or outputs can influence how much computation is reused, how requests are scheduled, when output is delivered, how shared resources are exercised, or which execution paths are activated. The secret-dependent step may be a cache hit caused by a matching prefix, a scheduling decision affected by reuse, a streamed message pattern shaped by generated tokens, a tokenization or filtering rule whose outcome can be tested, or an expert route selected from token content. If these effects remain visible to an attacker, a performance mechanism can become a side channel.

The problem studied in this paper arises when such inference-time decisions create a measurable dependence between confidential content and behavior observable outside the intended protection boundary. We use the term *software-controlled side channels* as an organizing term for these leakage paths. The term does not denote a new physical side-channel class; instead, it identifies inference-time channels whose existence or strength is shaped by a software, runtime, or model-execution decision. The final signal may appear at different layers, including service timing, network traffic, hardware timing, cache contention, memory-access traces, or accelerator-partition effects. What makes the channel software-controlled is the control point that couples protected content to observable behavior. Examples include cross-user cache sharing, cross-tenant batching, incremental streaming, testable tokenization or filtering, and sparse expert routing. This perspective centers the analysis on the decision that makes confidential content observable, rather than on any single signal type, shared resource, or attacker position.

Cross-user reuse of cached KV state provides a direct illustration. When state computed for one user's prompt remains reusable by others, scheduling or latency differences can reveal whether a probe benefits from retained computation. Such effects have enabled prompt reconstruction, inference of system-prompt content, and detection of previously processed documents [36,28]. Similar leakage has also appeared beyond cache reuse: shared CPU and GPU resources can expose prompt or response content [12,8]; confidential-computing and partitioned-GPU deployments can leak prompt or length information through observable execution behavior [15,14]; and encrypted streaming, sparse expert execution, tokenization behavior, and output-filter decisions can reveal protected information

through delivery- or model-level signals [34,10,9]. Together, these findings show that software-controlled leakage is not confined to a single optimization, resource, or deployment setting.

Although these studies expose important leakage paths, they are difficult to interpret as a coherent security problem when viewed only through their immediate observables. One attack may appear as timing leakage, another as cache contention, another as encrypted-traffic leakage, and another as sparse-routing leakage. This signal-centred view describes how an attacker observes the system, but it can obscure why leakage exists and where defenses should intervene. In many cases, an inference-time decision first makes protected content affect reuse, scheduling, execution, delivery, representation, or routing; this effect then becomes visible through timing, contention, traffic, or output behavior. We therefore organize the literature around the inference-time decision that creates the secret-dependent observable dependence, while separately recording the signal, attacker position, and protected information inferred.

Existing surveys have either systematized architectural and microarchitectural side-channel attacks in computer systems [3,19,24], or mapped broad LLM security and privacy risks, including training-data attacks, prompt misuse, memorization, membership inference, and techniques for privacy-preserving deployment [7,37,20]. **This paper** addresses a narrower system-specific problem: leakage paths in which inference-time software, runtime, or model-execution decisions make protected information indirectly observable through timing, traffic, contention, routing, or output behavior. To the best of our knowledge, this is the first systematization of LLM-inference side channels classified by the inference-time software, runtime, or model-execution decision that create the observable dependence. Architectural and microarchitectural effects are treated as observation layers through which that dependence becomes visible.

Accordingly, this paper makes four contributions. **First**, we define software-controlled side channels in LLM inference and distinguish them from direct exposure of protected inference state. **Second**, we develop a mechanism-centred taxonomy that classifies leakage by the inference-time decision creating the observable dependence, while separately recording the signal, attacker position, inferred secret, and architectural or microarchitectural observation layer. **Third**, we construct an evidence-coded corpus at the leakage-path level, separating peer-reviewed or accepted evidence from exposure audits, emerging evidence, evaluated defenses, and boundary cases. **Fourth**, we compare defense evidence and identify open problems in leakage measurement, interacting inference mechanisms, and security–performance evaluation for protected LLM deployments.

2 Background: LLM Inference and Observable Execution

2.1 LLM Inference Workflow

To analyze how confidential prompt or response content can influence observable behavior, we first distinguish the two main stages of autoregressive LLM inference, illustrated in Fig. 1. An input prompt is converted into *tokens*, which may correspond to words, word fragments, or other text units. During the *prefill*

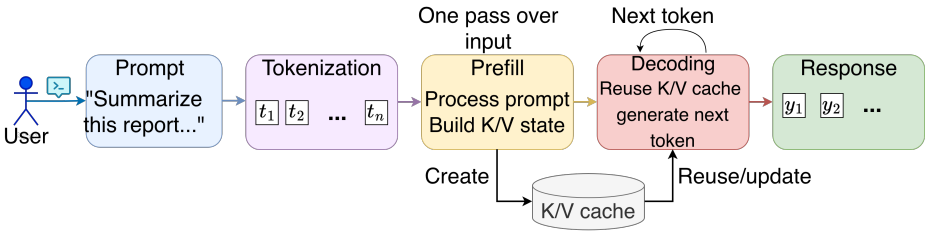

Fig. 1. Simplified autoregressive LLM inference workflow.

stage, the model processes the prompt tokens and computes the attention key and value representations associated with the processed context. These representations are retained in a key–value (KV) cache. During the *decoding* stage, the model generates the response one token at a time, reusing the cached representations of preceding tokens and appending new key and value entries [31].

These stages produce different observable signals. *Time to first token* (TTFT) is the delay until the first response token is returned. Although TTFT can include scheduling and queuing effects, it also reflects work performed before decoding begins; under comparable conditions, reusing a cached prompt prefix can therefore reduce TTFT [42,28]. After decoding begins, *inter-token latency* (ITL) measures the delay between successive output tokens. In services that stream output incrementally, decoding progress may also appear in the timing and size of encrypted response messages [5,34]. TTFT, ITL, and streamed-message characteristics are therefore recurring observables in the leakage paths.

2.2 Caching and Cross-Request Reuse

The KV cache described above is first created for a single inference request and extended as its response is generated. Serving systems can also reuse part of this cached state across requests. In *prefix caching*, the service retains KV representations for an already processed token sequence, such as a system instruction, application template, repeated conversation prefix, or document segment. If a later request begins with the same sequence, the service processes only the uncached suffix, reducing repeated prefill computation and TTFT [42].

Cross-request prefix caching becomes relevant to confidentiality when reuse is allowed across users or tenants. State derived from one user's prompt may then reduce the work required for another user's matching request. The second user does not read the cached state directly, but may observe the effect of reuse through lower TTFT or through scheduling behavior that favors cache hits [36,28,13]. Thus, a performance optimization can create an observable relation between one user's retained state and another user's request behavior.

Semantic caching uses a different form of reuse. Rather than requiring the same token prefix, it stores an earlier query and response and returns that response when a later query is judged sufficiently similar in meaning. This avoids new model execution for repeated or paraphrased questions, reducing latency and serving cost [2]. If semantic-cache entries are shared across users, semantic similarity to an earlier request can also affect whether a later request follows the faster cached-response path [28].

2.3 Streaming Delivery and Network Observables

After the first response token is generated, an LLM service need not wait for the complete response before returning output. Interactive services commonly *stream* generated text incrementally, reducing the time before a user can begin reading [5]. Transport encryption hides content, but it does not necessarily hide metadata such as transmission times or payload sizes. When streamed updates correspond to individual tokens or small token groups, encrypted-message sizes and timing can retain information about the sizes of underlying LLM-generated tokens, server-side optimizations [27], and even the generated responses or topics of an interaction [34].

Delivery patterns can also reflect how decoding progresses internally. In *speculative decoding*, a smaller draft model proposes candidate tokens and the target model verifies them; when more candidates are accepted, generation advances by more tokens in one step without changing the target model's output distribution [18]. If output is streamed as decoding progresses, variation in accepted-token counts can affect encrypted-message timing or size. More generally, optimized decoding and streaming can couple content-dependent generation behavior to network-observable patterns [33,4]. Thus, response streaming is not only a user-interface feature; it can also become an observation surface for content-dependent decoding behavior.

2.4 Microarchitecturally Observable Execution

Not every observable signal originates from API response times or encrypted network traffic. Some signals arise from how LLM inference executes on shared processors or accelerators. In local deployments, an LLM may run on a machine that also hosts unprivileged processes. Before the model processes a token through its Transformer layers, the runtime retrieves the corresponding numerical vector from the model's embedding table. These token-dependent memory accesses can affect shared CPU-cache state, such as the last-level cache, making token processing potentially observable through cache behavior [12].

Hardware-supported isolation reduces direct access to protected memory, but it does not necessarily hide all execution behavior. In an Intel TDX confidential virtual machine, for example, the host cannot directly read guest-private memory, yet it still controls parts of execution and memory management and shares microarchitectural resources with the guest. As a result, preprocessing steps such as tokenization can leave access patterns in memory and caches visible outside the protected workload [15].

A related issue appears on partitioned accelerators. NVIDIA Multi-Instance GPU (MIG) divides one GPU into isolated instances with dedicated compute and memory resources. However, activity in one instance can still influence timing effects observed from another instance, for example through memory-barrier behavior associated with kernel launches [14]. These examples show why protected LLM inference must consider not only direct memory access, but also execution-dependent behavior that remains observable across local, confidential-computing, or accelerator-partitioning boundaries.

2.5 Sparse Routing in Mixture-of-Experts Inference

In a conventional dense LLM, every token is processed by the same sequence of Transformer layers, including the attention and feed-forward computations in each layer. A MoE model changes part of this design: in some layers, instead of using one feed-forward network for every token, the model provides several alternative expert networks and a router. For each token, the router selects only a small number of experts to execute. This allows the model to increase its total parameter capacity without running all experts for every token [26,11,17].

Practical MoE inference must therefore manage expert placement, routing imbalance, and data movement while preserving the efficiency benefit of sparse execution [16]. Because the router's choices depend on the currently processed token and preceding text, different prompts or responses can lead to different expert selections and expert loads. If these patterns are visible through shared hardware or serving behavior, sparse execution may become a leakage surface.

2.6 Inference-Time Input and Output Policies

Not every observable effect is introduced to improve inference performance. An LLM system also makes decisions about how input text is represented and which generated content is allowed to leave the model. These decisions may serve functionality or privacy goals, yet still create attacker-observable behavior.

Two policies are especially relevant to the leakage paths considered later. First, tokenization and context-window limits determine how much text fits into a bounded input window. Because the limit is measured in tokens rather than characters, different tokenization outcomes can affect the model's observable output near the boundary [9]. Second, output filters may suppress generated continuations that match protected training content. If an attacker can test whether a chosen continuation is returned or blocked, the filter decision itself becomes an observable signal [9].

3 Scope and Method

This paper focuses on indirect leakage during LLM inference. We include leakage paths in which protected prompts, generated responses, private attributes, previously processed content, or training-derived information are inferred from observable effects of inference processing or delivery. We also retain defense studies that evaluate mitigations for such paths.

This scope excludes confidentiality failures in which protected inference data or internal representations are obtained directly. Examples include recovery of information left in residual accelerator memory, reconstruction from accessible retained KV representations, and inversion of intermediate representations exchanged during distributed inference [29,22,21]. These works clarify what an inference deployment must protect, but we record them as boundary studies rather than direct side-channel evidence. This distinction also separates our mechanism-centred taxonomy from broader taxonomies organized by signal, shared resource, or attack technique: we classify each path by the earliest LLM-inference control point that creates the observable dependence, while recording the signal and observation layer separately.

3.1 Research Questions

Guided by this scope, we ask four questions. **RQ1:** Which inference-time decisions create indirect leakage, and through which observable signals? **RQ2:** Which attacker positions, deployment settings, and protected-information targets have been evaluated? **RQ3:** Which mitigations have been proposed or evaluated, and what security–performance costs do they report? **RQ4:** Which leakage paths, deployment settings, and defense questions remain insufficiently studied?

These questions determine the fields extracted from each retained study: enabling decision, observable signal, attacker position, protected information, deployment setting, evidence status, and reported mitigation cost. RQ1 is answered by the taxonomy in Section 4 and Table 1; RQ2 by the cross-family comparison in Section 4.7; RQ3 by the defense comparison in Section 4.8 and Table 2; and RQ4 by the open problems in Section 5.

3.2 Corpus Construction and Screening

We constructed the corpus by selecting seed papers and then expanding the set through citation tracing, targeted venue screening, and journal-indexed searches. The seed papers covered direct LLM-inference leakage studies on shared KV-cache leakage, local cache-based token recovery, encrypted-streaming leakage, inference-policy leakage, and MoE-routing leakage [36,12,34,9,10]. Backward and forward citation tracing was used to identify related attacks, exposure audits, defenses, deployment mechanisms, and boundary confidentiality failures.

The retained corpus was updated through 27 May 2026 and covered conference proceedings, accepted-paper listings, journal indexes, and preprint repositories in security, systems, architecture, and machine learning. Targeted searches combined LLM or *large language model inference* with terms covering the principal mechanisms and observables in scope, including *side channel*, *prompt leakage*, *KV cache*, *prefix caching*, *semantic caching*, *streaming*, *encrypted traffic*, *speculative decoding*, *tokenization*, *confidential VM*, *TDX*, *MIG*, *GPU partitioning*, *MoE routing*, and *side-channel defense*. A companion artifact provides the query strings, search and screening protocol, retained-study list, classification of each study by evidence type, source links and identifiers, and path-level classification.

We retained a study as *direct side-channel evidence* only when it satisfied three conditions: (i) it concerned LLM inference; (ii) it inferred protected information from an indirect observable signal; and (iii) the signal was shaped by an inference-time software, runtime, or model-execution decision. We retained *exposure audits* when they established a deployed leakage-enabling condition without reconstructing protected content, *emerging evidence* when a non-peer-reviewed or non-archival manuscript satisfied the technical scope but was not peer-reviewed or accepted at the cutoff date, and *boundary studies* when protected data or internal representations were obtained directly.

We excluded works on general LLM privacy, model extraction, jailbreaks, membership inference, training-data extraction, cryptographic private inference, direct memory disclosure, direct access to KV contents, and inversion of transmitted intermediate representations, unless they clarified the boundary of our

scope. We counted studies as the unit of inclusion in the corpus; path-level coding is described separately in Section 3.3.

At the screening cutoff, the peer-reviewed or accepted core corpus contains ten distinct studies contributing thirteen coded leakage or exposure paths. Twelve paths demonstrate attacks that infer protected information, while one path audits cross-user prompt-caching exposure in deployed services. We additionally retained twelve emerging studies, of which eleven were available as preprints at the cutoff, and three boundary studies; these are discussed separately where they clarify developing attack directions, defense opportunities, or scope distinctions.

3.3 Unit of Analysis and Evidence Coding

Our unit of analysis is a *leakage path*, rather than a paper as a whole. A leakage path connects an inference-time decision to an observable signal, an attacker position, and a protected-information target. This choice is necessary because one study may demonstrate technically distinct channels. For example, a serving-system study may examine both exact-prefix cache reuse and semantic-cache reuse, while an MoE study may distinguish prompt-side attribute inference from response-token reconstruction. Counting such studies only once would obscure differences in observables, leaked information, and appropriate defenses.

For each retained leakage path, we extract seven fields: (i) the inference-time decision or mechanism that causes protected information to influence observable behavior; (ii) the observable signal; (iii) the architectural or microarchitectural observation layer; (iv) the attacker position and access assumptions; (v) the deployment setting; (vi) the protected information inferred; and (vii) the evidence status. Evidence status distinguishes demonstrated attacks, deployed exposure audits, emerging non-peer-reviewed or non-archival evidence, evaluated defenses, and boundary cases. For defense studies, we additionally record the intervention point, the leakage path addressed, whether leakage reduction was evaluated against an attack, and any reported latency, throughput, memory, bandwidth, output-quality, or interactivity cost.

We apply the classification procedure in four steps. First, we identify the earliest inference-time decision at which protected information changes system behavior. Second, we assign the path to the taxonomy family corresponding to that decision, rather than to the final signal, hardware resource, or attacker location. Third, we record the observable signal, observation layer, attacker position, deployment setting, and leakage target independently of the family assignment. Fourth, we separate multiple paths from one study only when they involve technically distinct enabling decisions or leakage targets. For example, GhostWriter is classified as a shared serving-state channel because cross-tenant batching and victim-dependent KV residency create the observable dependence, although GPU-cache contention carries the final signal. By contrast, TDXRay is classified as microarchitecturally observable execution because protected processing remains visible across a confidential-VM boundary.

Study extraction and path classification followed this procedure. For each retained core path, the companion artifact reports the extracted attributes, final family assignment, and classification rationale.

4 Taxonomy of Software-Controlled LLM Side Channels

Following the coding rule in Section 3.3, we classify each leakage path by the inference-time decision that first creates an observable dependence on protected information. This mechanism-centred view is important because the same observable signal may arise from different system decisions and therefore require different protections. For example, timing may disclose cross-user cached-state reuse, execution across an isolation boundary, or output delivery behavior; treating all of them simply as "timing channels" would hide where the exposure originates and where a defense must intervene.

Fig. 2 presents the resulting taxonomy, and Table 1 summarizes the peer-reviewed or accepted core evidence. We identify five families. *Shared serving-state channels* arise from request-dependent state that is retained or shared during serving. *Microarchitecturally observable execution channels* arise when prompt- or response-dependent execution remains visible through shared or imperfectly isolated hardware resources. *Network-observable delivery channels* occur when encrypted response delivery preserves information about generated content or decoding behavior. *Input/output-policy channels* stem from representation or output-control decisions that produce externally testable behavior. *MoE execution channels* arise when token-dependent sparse routing or expert execution exposes information about prompts or generated responses.

The following subsections examine these families in terms of their enabling decisions, attacker-visible signals, inferred protected information, evidence strength, and mitigation evidence.

4.1 Shared Serving-State Channels

The retained evidence identifies three leakage paths in which request-derived state affects behavior observed by another user or tenant. The first path concerns exact-prefix reuse. PROMPTPEEK [36] demonstrates content reconstruction: cache-aware scheduling allows an attacker to recover a victim prompt token by token from probe completion order. Early Bird [28] demonstrates a timing variant: reduced TTFT for matching probes exposes system-prompt content and whether a target document was previously processed. Auditing Prompt Caching [13] further shows that cross-user prompt-cache timing effects can appear in deployed language-model APIs. These results exploit the same underlying relation—whether a candidate prefix matches retained user-derived state—but observe it through different service-level signals.

The second path concerns semantic-cache reuse. Here the observable relation is not exact token-prefix equality, but whether a probe is sufficiently similar in meaning to an earlier protected request. Early Bird uses this timing behavior to infer sensitive request attributes in its evaluated setting [28]. This distinction matters for defense: preventing exact-prefix probing does not by itself prevent semantic inference when cached responses remain shared across users.

The third path appears when request-derived state remains influential during shared accelerator execution. GhostWriter extracts private prompt tokens through GPU-cache contention caused by victim-dependent KV-cache residency under multi-tenant batching [8]. We classify this path in the shared serving-

state family because the exposure begins with a serving decision that allows state derived from one tenant to affect another tenant's observations, although the resulting signal is measured microarchitecturally.

Emerging studies further extend this family. *InputSnatch* targets cache-induced response-time differences and uses constructed candidate inputs to recover sensitive queries under both exact-cache and semantic-cache settings [43]. *OptiLeak* focuses on the efficiency of prefix-cache prompt reconstruction: rather than identifying a new signal, it uses reinforcement-learning-guided probing to reduce the number of requests required to recover sensitive prompt tokens [32]. These studies are not included in the peer-reviewed or accepted core table, but they indicate that shared-state leakage may remain practical when the attacker's query budget becomes an important deployment constraint.

The evidence for this family is stronger than for most other families. It includes end-to-end prompt or attribute inference, a deployed-service audit, and several paths that originate from an identifiable serving decision: allowing request-derived state to remain reusable or influential across tenants. Shared serving state is therefore an important case for evaluating practical defenses that reduce leakage without eliminating the efficiency benefit that motivated reuse.

4.2 Microarchitecturally Observable Execution Channels

This family contains leakage paths in which protected processing remains observable through hardware-level timing, cache state, or memory-access behavior. The retained studies cover three settings: local co-residency, confidential-VM execution, and partitioned-GPU multi-tenancy. They are grouped together because, in each case, the attacker infers protected information from execution effects rather than from direct access to the prompt, response, or protected memory.

In local inference, *I Know What You Said* exploits token-dependent embedding accesses and autoregressive timing behavior exposed through a shared CPU cache. An unprivileged process on the same machine uses these observations to reconstruct both victim input and generated output text [12]. This path shows that local deployment does not remove confidentiality risk when token-level execution remains visible to other processes.

An emerging study reports a related local-execution path. *Spill the Beans* monitors cache activity associated with selected embedding vectors during token generation and uses the observed hits to recover generated tokens, including substantial portions of a high-entropy API key in its evaluated setting [1]. Because we did not verify peer-reviewed or accepted status at the screening cutoff, we treat it as emerging evidence rather than core evidence.

TDXRay extends this concern to confidential execution. Its attacker is a malicious host that cannot directly read the memory of an Intel TDX confidential virtual machine, but can observe memory-access behavior associated with tokenization inside the protected LLM workload. From a single trace, the attack recovers user prompts with high semantic similarity in the evaluated models [15]. This path shows that memory confidentiality alone is insufficient when preprocessing activity remains distinguishable through host-observable execution traces.

Behind Bars identifies a third path across accelerator partitions. An attacker in a separate NVIDIA MIG partition observes timing effects associated with victim kernel launches and uses the resulting trace to infer LLM input and output lengths [14]. Unlike the local-cache and confidential-VM attacks, this path does not reconstruct prompt or response text; its leakage target is the size profile of the protected interaction. We retain it as direct evidence because input and output lengths are protected request properties inferred from cross-tenant execution behavior.

Together, these paths show that protected inference must account for more than direct memory access. Execution behavior can remain observable across local processes, confidential-VM boundaries, and accelerator partitions. Unlike shared serving-state channels, however, the observation surfaces differ substantially across studies, making a single family-wide mitigation less immediate. We compare the limited available defense evidence for these paths in Section 4.8.

4.3 Network-Observable Delivery Channels

Network-observable delivery channels arise when encrypted response traffic remains dependent on generated content or decoding progress. The peer-reviewed core evidence in this family concerns incremental response streaming. In *What Was Your Prompt?*, encrypted message sizes preserve information about the token-length pattern of a streamed response, allowing a passive network observer to infer generated content without decrypting the connection [34]. This path differs from shared serving-state leakage because it does not require cross-user reuse, probing requests, or co-residency; the observation is available from the delivery channel alone.

Emerging studies broaden this family beyond streamed response-token lengths. *Whisper Leak* uses packet-size and timing patterns in encrypted streaming responses to classify sensitive prompt topics across evaluated LLM services [23]. *NetEcho* examines services with traffic defenses already deployed and reports recovery of user prompts and generated responses from residual encrypted-traffic patterns [41]. Other emerging studies connect leakage to decoding behavior: speculative-decoding traffic patterns have been used to distinguish sensitive query classes [33], while remote timing effects of efficient decoding have been used to test conversation properties or selected hidden prompt values under stronger attacker assumptions [4]. Because these studies were not verified as peer-reviewed or accepted at the screening cutoff, we discuss them as emerging evidence rather than include them in the core comparison table.

Overall, the peer-reviewed evidence for network-observable delivery is currently narrower than for shared serving-state leakage: the retained core contains one demonstrated streaming-based response-recovery path. However, the emerging evidence consistently points to encrypted delivery as an observation surface when transmission behavior remains coupled to generated content or decoding progress. We assess delivery-layer mitigation evidence in Section 4.8.

4.4 Input- and Output-Policy Channels

Input- and output-policy channels arise when a system decision governing text representation or content release becomes externally testable. Unlike shared serving-state or resource-dependent execution channels, these paths do not require faster reuse, hardware contention, or streamed-message structure. The observable signal is instead produced by the rule that determines how text is encoded, suppressed, or returned.

The first core path originates from output filtering. A service may suppress generated continuations that match protected training content in order to reduce verbatim memorization leakage. However, if an attacker can test whether a chosen continuation is returned or blocked, the filter decision becomes a membership signal: suppression reveals that the continuation matches content protected by the filter. Debenedetti et al. use this return-or-suppress behavior to infer properties of GitHub Copilot's training corpus and, in a controlled language-model setting, to extract inserted OpenSSH private keys from training data [9]. Thus, a mechanism intended to prevent disclosure creates an observable channel.

The second core path originates from tokenization under a fixed context-window limit. Because the context window is measured in tokens, the same string may consume different amounts of context depending on how the tokenizer represents it. By constructing prompts near the context-window boundary and observing whether earlier known text continues to influence the output, an attacker can infer the tokenization outcome. Debenedetti et al. use this behavior to recover tokenizer vocabulary information, including rare vocabulary entries in the evaluated model [9]. This shows that an input-representation rule can itself expose model-related protected information.

An emerging output-side path concerns application behavior that makes response length depend on sensitive information. In translation, different target languages may produce different numbers of output tokens; in classification, class-dependent outputs or explanations may also differ in length. *Time Will Tell* shows that an adversary can use total output-token count, or remote response time as its proxy, to infer a target language or a model-produced classification label [40]. This path differs from incremental streaming leakage because it exploits total output length rather than message-by-message delivery.

Overall, this family identifies a distinct source of leakage: representation and content-release rules can create externally testable behavior even without state sharing, hardware contention, or encrypted-streaming structure. Its core evidence currently comes from two paths in one peer-reviewed study, while output-token-count leakage is retained as emerging evidence. Because these policy decisions serve different purposes, their mitigation evidence must be assessed separately rather than assumed to follow from defenses for other leakage families.

4.5 Mixture-of-Experts Execution Channels

The peer-reviewed core evidence for MoE execution channels currently consists of two leakage paths demonstrated by MoEcho [10]. Both arise from token-dependent sparse execution that affect different inference phases. During prompt processing, variations in expert load provide a signal about the private input,

enabling inference of sensitive prompt attributes in the evaluated healthcare setting. During response generation, sequences of activated experts provide a token-dependent trace of the output, enabling reconstruction of response tokens.

These paths show why MoE execution warrants a separate family. Unlike shared serving-state channels, the leakage does not require retained victim state to be reused by another request. Unlike input/output-policy channels, it does not arise from an explicit representation or filtering rule. Instead, the model's efficiency mechanism—routing each token through selected experts—causes protected content to shape which computation is executed and how that computation appears in shared hardware observations.

The evidence for this family remains narrow. MoEcho establishes prompt-side and response-side leakage through several CPU- and GPU-observable channels, while emerging evidence suggests that routing interactions may support prompt reconstruction under stronger co-batching and routing assumptions [38]. However, the retained corpus does not yet provide independent peer-reviewed or accepted confirmation across different MoE serving stacks or deployment settings. Quantified protection for sparse-routing leakage therefore remains an open defense question, discussed further in Section 4.8.

4.6 Boundary Cases: Direct Exposure of Protected State

The five families above share the defining property of a side channel: protected information is inferred from an indirect effect of inference execution or delivery. Closely related confidentiality failures arise when an attacker obtains protected data or internal representations directly. For example, uncleared accelerator memory can retain fragments of a previous LLM response that another process later reads [29]; plaintext KV-cache contents exposed outside a protected execution boundary can be used to reconstruct user prompts [22]; and intermediate embeddings transmitted between partitions in distributed inference can be inverted to recover input text [21]. These studies are relevant to the protection boundary of LLM inference, but we do not count them as software-controlled side-channel paths because the adversary observes the sensitive state or representation itself rather than an incidental signal produced by its use.

4.7 Cross-Family Comparison of Direct Leakage Evidence

Table 1 compares the peer-reviewed or accepted core evidence retained in the corpus. Each row represents a distinct leakage path rather than a distinct paper, because a single study may expose different protected information through different observable mechanisms. The comparison records the enabling mechanism, attacker-visible signal, protected information inferred or exposure established, and attacker position required by the demonstrated result. Unlike the preceding subsections, which also discuss emerging non-archival studies where they extend an identified family, Table 1 includes only demonstrated peer-reviewed or accepted attack paths and the deployed exposure audit.

To keep evidence levels separate, we do not add emerging preprint studies to Table 1. Instead, they are discussed in the relevant family subsections when they show new attack directions, defense opportunities, or remaining gaps.

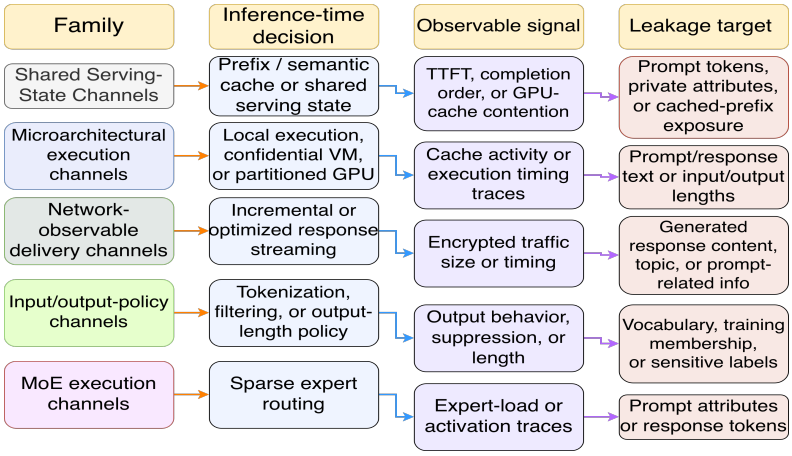

Fig. 2. Mechanism-centred taxonomy of software-controlled side channels in LLM inference.

The thirteen core leakage or exposure paths are unevenly distributed across the taxonomy: five concern shared serving state, three microarchitecturally observable execution, one network-observable delivery, two input/output policy, and two MoE execution. Representative results nevertheless show substantial leakage in the evaluated settings: prompt-recovery success reaches 99% for prompt inputs, 98% for prompt templates, and 95% without background knowledge [36]; average prompt-recovery accuracy reaches 89.0%, while semantic-cache hit/miss classification reaches 81.4% in one trial and 95.4% after five trials [28]. Reconstructed input and output text achieve cosine similarities of 98.7% and 98.0%, respectively [12]; encrypted-stream analysis reconstructs 27% of responses and infers their topics in 53% of cases [34]; and MoE execution traces support 99.8% private-input inference and 92.8% response reconstruction [10]. Because these studies use different leakage targets, attacker knowledge, workloads, query budgets, and success criteria, the results provide within-study evidence rather than a cross-study ranking; we provide our reported metrics as an artifact.

The intended stability of the taxonomy lies in its classification principle rather than in a claim that the five current families are immutable. New studies can add leakage paths within an existing family, motivate a subdivision when distinct control points become sufficiently supported, or justify a new family when leakage originates from an inference-time decision not captured by the current structure. At the screening cutoff, the emerging studies extend several existing families but do not identify a different top-level control point. The taxonomy is therefore designed to remain extensible as the evidence base develops.

4.8 Cross-Family Comparison of Defense Evidence

The direct leakage evidence in Table 1 does not imply that the corresponding protection problem has been solved. We therefore compare defense and mitigation evidence separately in Table 2. Because evaluated defenses remain scarce, this comparison uses a broader evidence threshold than the core attack table. Of the

Table 1. Peer-reviewed or accepted core evidence on software-controlled side-channel leakage and exposure in LLM inference.

Family	Study / path	Enabling decision	Observable signal	Leakage target / exposure	Attacker position
Shared serving state	PROMPTPEEK (2025) [36]	Cross-user prefix-KV reuse with cache-aware scheduling	Probe completion order for candidate prefixes	Victim prompt or hidden template, reconstructed token by token	Remote probing user
	Early Bird (KV-cache timing)(2025) [28]	Prefix-KV reuse across requests	Reduced TTFT for matching probes	System-prompt content; prior processing of a target document	Remote probing user
	Early Bird (semantic-cache timing) [28]	Semantic-cache reuse across meaningfully similar requests	Faster cached response for semantically related probes	Private request attributes, including evaluated name–condition associations	Application probing user
	Auditing Prompt Caching *(audit)*(2025) [13]	Cross-user prompt caching in deployed APIs	Measurable TTFT differences across accounts	Shared cached-prefix exposure; architecture information at one endpoint	Cross-account API user
	GhostWriter (2025) [8]	Cross-tenant batching with victim-dependent KV residency	GPU-cache contention	Private prompt tokens	Co-located GPU tenant
Microarch. execution	I Know What You Said (2025) [12]	Token-dependent embedding-table access during local inference	Shared CPU-cache observations	Prompt and generated response text	Co-resident process
	TDXRay (2026) [15]	Tokenization inside an Intel TDX confidential virtual machine	Host-observable memory-access traces	Private user prompt	Malicious host
	Behind Bars (2026) [14]	Victim kernel-launch behavior across NVIDIA MIG partitions	Cross-partition timing observations	Input and output token counts	Separate MIG tenant
Network delivery	What Was Your Prompt? (2024) [34]	Incremental encrypted response streaming	Encrypted message-size patterns	Generated response content and topic	Passive network observer
I/O policy	Privacy Side Channels (memorization filter) (2024) [9]	Output filtering for protected training content	Return-or-suppress decision for candidate continuations	Training-data properties; inserted private-key content	Black-box querying user
	Privacy Side Channels (tokenizer/context limit) [9]	Tokenization under a fixed context-window limit	Context-boundary output behavior	Tokenizer vocabulary entries, including evaluated rare strings	Black-box querying user
MoE execution	MoEcho (prompt-side) (2025) [10]	Token-dependent sparse expert routing during prompt processing	Expert-load patterns in CPU/GPU execution traces	Private prompt attributes	Observable co-tenant
	MoEcho (response-side) [10]	Token-dependent sparse expert routing during decoding	Routed-expert activation traces	Generated response tokens and reconstructed text	Observable co-tenant

six retained entries, protected embedding generation is peer-reviewed, while the remaining entries are workshop or non-archival studies. The table therefore maps current intervention points, reported effects, and remaining gaps rather than establishing that these defenses are mature or independently validated. Public documentation from major AI service providers, such as OpenAI, Google, and Alibaba Cloud, describes organization-, project-, or account-level cache isolation and control measures. However, we found no public attack-based evaluation of these measures against the side-channel paths retained in our corpus; they are therefore not included among the evaluated defenses in Table 2.

The comparison shows that current protections are strongly mechanism-specific. Shared serving-state defenses assume that the provider controls cross-user KV reuse and can identify sensitive entries or suspicious sharing. Protected embedding generation removes one secret-dependent memory-access pattern, but does not address confidential-VM traces or cross-partition GPU timing. NetEcho and Time Will Tell primarily evaluate mitigation directions and residual leakage rather than complete deployable defenses. Their reported results are also not directly comparable because they use different measures, including TTFT, throughput, cache-hit rate, attack mitigation rate, and residual recovery.

5 Open Research Problems

The taxonomy and defense comparison show that software-controlled side channels arise across serving, execution, delivery, and policy decisions. They also reveal uneven coverage: shared serving-state leakage has the broadest evidence, while protected-execution, delivery, policy, and sparse-routing channels remain less consistently evaluated. The following problems follow from these gaps and from the deployment assumptions of protected LLM inference.

5.1 Building Comparable Leakage Measurements

Current studies report leakage using different success measures, attacker capabilities, workloads, and deployment assumptions. Some reconstruct prompts or responses, some infer attributes or token counts, and others only establish deployed exposure conditions [36,34,14,13]. These results are all security-relevant, but they are not directly comparable: a prompt-reconstruction attack, a topic-inference attack, and a cache-exposure audit measure different privacy losses.

A first open problem is to define leakage metrics that separate the observable signal from the protected information inferred. A timing signal may reveal a cache-hit condition, a token-count estimate, a semantic attribute, or an entire text sequence. Future evaluations should report the leakage target, attacker prior knowledge, number of probes or traces, recovery confidence, and signal-stability conditions. Defense evaluations should similarly specify which signal was reduced, which attacker model was tested, and what cost was paid in latency, throughput, memory, bandwidth, model quality, or interactivity [6,25,35,30,40].

5.2 Analyzing Interacting Inference Mechanisms

The taxonomy separates leakage paths by the decision that first creates an observable dependence, but real LLM services often combine several such decisions. A single request may use prefix caching, dynamic batching, speculative decod-

Table 2. Defense and mitigation evidence by leakage family.

Leakage path	Evidence / deployment	Intervention and reported effect	Cost or residual limitation
Shared serving state: prefix-KV timing	SafeKV [6], PrefixWall [25], and CachePrune [35]. Workshop or emerging evidence for multi-tenant serving with cross-user KV reuse.	Restrict sensitive entries, isolate suspicious prefixes, or retain only privacy-irrelevant reusable spans. SafeKV reports mitigation of 94–97% of evaluated timing attacks.	Compared with per-user isolation, reported benefits include up to 40.58% better TTFT and 2.66× throughput for SafeKV, 30% lower latency for PrefixWall, and 4.5× lower TTFT for CachePrune. Coverage remains limited to shared-KV timing.
Microarchitectural execution: embedding access	Protected embedding generation [30]. Peer-reviewed evidence for inference with secret-dependent indexed embedding access.	Replaces indexed table lookup with Deep Hash Embedding or a hybrid protected-generation method, removing the targeted secret-dependent access pattern.	For GPT-2, reports up to 1.32× prefill and 1.07× decoding speedups over the ORAM baseline with comparable output quality; it does not address TDX or MIG leakage.
Network delivery: encrypted streaming	NetEcho [41]. Emerging mitigation assessment across defended real-world streaming settings.	Evaluates residual leakage under seven representative traffic-padding or obfuscation settings rather than proposing a complete defense.	Reports recovery of approximately 70% of evaluated medical and legal conversations on average, showing that the tested traffic defenses leave substantial residual leakage.
I/O policy: output-token count	Time Will Tell [40]. Emerging evidence for translation and classification tasks whose output length correlates with sensitive input properties.	Evaluates tokenizer-, system-, and prompt-level changes intended to reduce output-length distinguishability.	Effectiveness varies across models and tasks, and no universal performance cost is reported. The evaluated mitigations do not address context-limit or memorization-filter leakage.

ing, streaming delivery, and hardware isolation at the same time. Evaluating each mechanism in isolation can therefore miss interactions that amplify leakage or create new observables. For example, shared cached state may change scheduling behavior, scheduling may affect co-tenant hardware contention, and streaming may expose the timing consequences of accelerated decoding [36,8,33].

A second open problem is to analyze leakage composition across inference mechanisms and deployment boundaries. A defense that removes one signal may leave another correlated signal intact, and weak signals may become more informative when combined. This is especially relevant for services where attackers may observe response latency, completion order, message size, and repeated-query behavior from the same interaction. It also matters for services that utilize confidential virtual machines and accelerator partitioning to restrict direct access to inference data, where execution traces can remain observable across protection boundaries [15,14]. Future evaluations should therefore study realistic combinations of serving optimizations, hardware isolation, and delivery behavior rather than assuming that each leakage path operates independently.

5.3 Evaluating Protected Inference Beyond Memory Confidentiality

Protected LLM inference is often framed around preventing direct access to prompts, model state, or intermediate representations. This goal remains necessary, but the retained evidence shows that memory confidentiality alone is not sufficient. TDXRay recovers user prompts from memory-access traces inside an Intel TDX confidential virtual machine [15], while Behind Bars infers LLM input and output lengths from timing effects across NVIDIA MIG partitions [14]. These results show that protected execution can still expose inference-dependent behavior even when the attacker cannot directly read protected memory.

A third open problem is to evaluate confidential and hardware-isolated LLM inference as a complete observable system. Such evaluation should cover tokenization, prefill, decoding, batching, accelerator scheduling, memory movement, and response delivery. Future work should report side-channel evaluation together with the protected system configuration, including batching policy, cache policy, tokenizer placement, accelerator sharing mode, and traffic delivery mode.

5.4 Designing Deployable Defenses with Explicit Trade-offs

The defense comparison shows that protection evidence remains uneven across leakage families. Shared serving-state channels have emerging mitigations for restricting cross-user KV reuse, but the screened corpus lacks quantified deployable defenses for tokenizer/context-window leakage, memorization-filter leakage, confidential-VM execution traces, partitioned-GPU timing leakage, and MoE expert-execution traces.

A fourth open problem is to design defenses that report security and deployment cost together. Full user-level cache isolation, traffic padding, more uniform MoE execution, or hidden tokenizer/filter behavior may reduce leakage, but each can also remove the latency, throughput, interactivity, sparsity, or policy benefit that motivated the original mechanism. Future evaluations should report leakage reduction under a stated attacker model together with latency, throughput, memory or bandwidth overhead, model-output quality, multi-tenant compatibility, and robustness to adaptive attackers.

6 Conclusion

This paper systematizes software-controlled side channels in LLM inference: leakage paths where inference-time software, runtime, or model-execution decisions make protected information attacker-visible indirectly. By separating these paths from direct exposure of retained state or intermediate representations, we identify five mechanism-centred families: shared serving-state channels, microarchitecturally observable execution channels, network-observable delivery channels, input/output-policy channels, and MoE execution channels. The comparison shows that the evidence base and defense landscape remain uneven. Shared serving-state leakage has the broadest set of demonstrated attacks and emerging mitigations, while protected-execution, delivery, policy, and sparse-routing channels remain less consistently evaluated, and several retained leakage paths lack quantified deployable defenses. The broader lesson is that protecting LLM inference requires more than hiding plaintext prompts and responses: systems

must also control the observable effects of reuse, scheduling, execution, delivery, and policy decisions. Future work should therefore measure leakage and defenses under comparable assumptions, evaluate the effects of interactions across inference mechanisms, and report security gains together with the performance and usability impacts so as to enable informed and effective design decisions for service development and deployment.

Open Science. We provide our summarised literature corpus and classification details: https://github.com/abla-smahi/software-controlled-llm-side-channels-artifact

Acknowledgments. We gratefully acknowledge the Brussels-Capital Region – Innoviris for financial support under grant number 2024-RPF-4 SDM, and the CyberExcellence program of the Walloon Region of Belgium under grant number 2110186.

References

1. Adiletta, A., Sunar, B.: Spill the beans: Exploiting CPU cache side-channels to leak tokens from large language models (2025). doi:10.48550/arXiv.2505.00817
2. Bang, F.: GPTCache: An open-source semantic cache for LLM applications enabling faster answers and cost savings. In: 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS). pp. 212–218 (2023)
3. Canella, C., Van Bulck, J., Schwarz, M., Lipp, M., Von Berg, B., Ortner, P., Piessens, F., Evtyushkin, D., Gruss, D.: A systematic evaluation of transient execution attacks and defenses. In: 28th USENIX Security Symposium (USENIX Security 19). pp. 249–266 (2019)
4. Carlini, N., Nasr, M.: Remote timing attacks on efficient language model inference (2024). doi:10.48550/arXiv.2410.17175
5. Chen, J., Du, C., Liu, R., Yao, S., Yan, D., Liao, J., Liu, S., Wu, F., Chen, G.: TokenFlow: Responsive LLM text streaming serving under request burst via preemptive scheduling. In: 21st European Conference on Computer Systems (EuroSys). pp. 497–513 (2026)
6. Chu, K., Lin, Z., Xiang, D., Shen, Z., Su, J., Chu, C., Yang, Y., Zhang, W., Wu, W., Zhang, W.: Selective kv-cache sharing to mitigate timing side-channels in LLM inference (2025). doi:10.48550/arXiv.2508.08438
7. Das, B.C., Amini, M.H., Wu, Y.: Security and privacy challenges of large language models: A survey. ACM Computing Surveys **57**(6), 1–39 (2025)
8. Das, S., Vijayakumar, S.: GhostWriter: Exploiting GPU-cache contention to steal and steer multi-tenant large-language-model inference. In: Security, Privacy, and Applied Cryptography Engineering. pp. 134–153 (2025)
9. Debenedetti, E., Severi, G., Carlini, N., Choquette-Choo, C.A., Jagielski, M., Nasr, M., Wallace, E., Tramèr, F.: Privacy side channels in machine learning systems. In: 33rd USENIX Security Symposium. pp. 6831–6848 (2024)
10. Ding, R., Xu, T., Shen, X., Ding, A.A., Fei, Y.: MoEcho: Exploiting side-channel attacks to compromise user privacy in mixture-of-experts LLMs. In: ACM SIGSAC Conference on Computer and Communications Security. pp. 2159–2173 (2025)
11. Fedus, W., Zoph, B., Shazeer, N.: Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. Journal of Machine Learning Research **23**(1), 1–39 (2022)
12. Gao, Z., Hu, J., Guo, F., Zhang, Y., Han, Y., Liu, S., Li, H., Lv, Z.: I know what you said: Unveiling hardware cache side-channels in local large language model inference. In: 34th USENIX Security Symposium. pp. 1649–1668 (2025)
13. Gu, C., Li, X.L., Kuditipudi, R., Liang, P., Hashimoto, T.: Auditing prompt caching in language model APIs. In: 42nd International Conference on Machine Learning (ICML). vol. 267, pp. 20477–20496. PMLR (2025)
14. Gu, C., Levine, R., Zhang, Z., Sorensen, T., Guo, Y.: Behind Bars: A side-channel attack on NVIDIA MIG cache partitioning using memory barriers. In: 35th USENIX Security Symposium (2026)
15. Hornetz, T., Yavarzadeh, H., Cheu, A., Gascon, A., Gerlach, L., Moghimi, D., Schoppmann, P., Schwarz, M., Zhang, R.: TDXRay: Microarchitectural side-channel analysis of Intel TDX for real-world workloads. In: IEEE Symposium on Security and Privacy (S&P). pp. 1–18 (2026)
16. Huang, H., Ardalani, N., Sun, A., Ke, L., Lee, H.H.S., Bhosale, S., Wu, C.J., Lee, B.: Toward efficient inference for mixture of experts. In: 38th International Conference on Neural Information Processing Systems (2024)
17. Jiang, A.Q., Sablayrolles, A., Roux, A., et al.: Mixtral of experts (2024). doi:10.48550/arXiv.2401.04088

18. Leviathan, Y., Kalman, M., Matias, Y.: Fast inference from transformers via speculative decoding. In: Proceedings of the 40th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 202, pp. 19274–19286. PMLR (2023)
19. Lou, X., Zhang, T., Jiang, J., Zhang, Y.: A survey of microarchitectural side-channel vulnerabilities, attacks, and defenses in cryptography. ACM Computing Surveys **54**(6), 1–37 (2021)
20. Luo, J., Zhang, Y., Zhang, Z., Liu, S., Dong, Y., Li, H., Yu, Y., Wang, H., Zhou, X., Xu, Z.: Cryptography-based privacy-preserving large language models: A lifecycle survey of frameworks, methods, and future directions. Artificial Intelligence Review **59**(64) (2026)
21. Luo, X., Yu, T., Xiao, X.: Prompt inference attack on distributed large language model inference frameworks. In: ACM SIGSAC Conference on Computer and Communications Security. pp. 1739–1753 (2025)
22. Luo, Z., Shao, S., Zhang, S., Zhou, L., Hu, Y., Zhao, C., Liu, Z., Qin, Z.: Shadow in the cache: Unveiling and mitigating privacy risks of KV-cache in LLM inference. In: Network and Distributed System Security Symposium (NDSS) (2026)
23. McDonald, G., Bar Or, J.: Whisper Leak: A side-channel attack on large language models (2025). doi:10.48550/arXiv.2511.03675
24. Muñoz, A., Ríos, R., Román, R., López, J.: A survey on the (in) security of trusted execution environments. Computers & Security **129**, 103180 (2023)
25. Pennas, P.G., Papaioannou, K., Guarnieri, M., Doudali, T.D.: PrefixWall: Mitigating prefix caching side channels in shared LLM systems (2026). doi:10.48550/arXiv.2603.10726
26. Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q.V., Hinton, G.E., Dean, J.: Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In: International Conference on Learning Representations (ICLR) (2017)
27. Soleimani, M., Jia, G., Gim, I., seob Lee, S., Khandelwal, A.: Wiretapping LLMs: Network side-channel attacks on interactive LLM services (2025), `https://eprint.iacr.org/2025/167`
28. Song, L., Pang, Z., Wang, W., Wang, Z., Wang, X., Chen, H., Song, W., Jin, Y., Meng, D., Hou, R.: The early bird catches the leak: Unveiling timing side channels in LLM serving systems. IEEE Transactions on Information Forensics and Security **20**, 11431–11446 (2025)
29. Sorensen, T., Khlaaf, H.: LeftoverLocals: Listening to LLM responses through leaked GPU local memory (2024). doi:10.48550/arXiv.2401.16603
30. Umar, M., Marathe, A.P., Gupta, M.D., Ghosh, S.J., Suh, G.E., Xiong, W.: Efficient memory side-channel protection for embedding generation in machine learning. In: IEEE International Symposium on High-Performance Computer Architecture (HPCA). pp. 423–441 (2025)
31. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: 31st International Conference on Neural Information Processing Systems. pp. 6000–6010 (2017)
32. Wang, L., Zheng, X., Zhang, X., Zhang, Y., Wu, Y., Wang, C.: OptiLeak: Efficient prompt reconstruction via reinforcement learning in multi-tenant LLM services (2026). doi:10.48550/arXiv.2602.20595
33. Wei, J., Abdulrazzag, A., Zhang, T., Müürsepp, A., Saileshwar, G.: When speculation spills secrets: Side channels via speculative decoding in LLMs (2026). doi:10.48550/arXiv.2411.01076
34. Weiss, R., Ayzenshteyn, D., Amit, G., Mirsky, Y.: What was your prompt? A remote keylogging attack on AI assistants. In: 33rd USENIX Security Symposium. pp. 3367–3384 (2024)
35. Wu, G., li, Z., Zhang, Y., Zhang, Z., Niu, J., Wu, Y., Zhang, Y.: CachePrune: Privacy-aware and fine-grained KV cache sharing for efficient LLM inference (2026). doi:10.48550/arXiv.2605.23640
36. Wu, G., Zhang, Z., Zhang, Y., Wang, W., Niu, J., Wu, Y., Zhang, Y.: I know what you asked: Prompt leakage via KV-cache sharing in multi-tenant LLM serving. In: Network and Distributed System Security Symposium (NDSS) (2025)
37. Yao, Y., Duan, J., Xu, K., Cai, Y., Sun, Z., Zhang, Y.: A survey on large language model (LLM) security and privacy: The good, the bad, and the ugly. High-Confidence Computing **4**(2), 100211 (2024)
38. Yona, I., Shumailov, I., Hayes, J., Carlini, N.: Stealing user prompts from mixture of experts (2024). doi:10.48550/arXiv.2410.22884
39. Yu, G.I., Jeong, J.S., Kim, G.W., Kim, S., Chun, B.G.: Orca: A distributed serving system for transformer-based generative models. In: 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI). pp. 521–538 (2022)
40. Zhang, T., Saileshwar, G., Lie, D.: Time will tell: Timing side channels via output token count in large language models (2024). doi:10.48550/arXiv.2412.15431
41. Zhang, Z., Wu, G., Deng, S., Wang, S., Zhang, Y.: NetEcho: From real-world streaming side-channels to full LLM conversation recovery (2025). doi:10.48550/arXiv.2510.25472
42. Zheng, L., Yin, L., Xie, Z., Sun, C., Huang, J., Yu, C.H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J.E., Barrett, C., Sheng, Y.: SGLang: efficient execution of structured language model programs. In: 38th International Conference on Neural Information Processing Systems (2024)
43. Zheng, X., Han, H., Shi, S., Fang, Q., Du, Z., Hu, X., Guo, Q.: InputSnatch: Stealing input in LLM services via timing side-channel attacks (2024). doi:10.48550/arXiv.2411.18191