AgentFlowShield: Defending LLM Agents Against Metadata Side Channels

Redha Boukhari; Yulliwas Ameur; Mehammed Daoui

2026 IEEE/ACS 23rd International Conference on Computer Systems and Applications (AICCSA)

Accepted conference contribution. Proceedings forthcoming.

Manuscript snapshot: 7 September 2026. This public copy adds a version and copyright cover; the research manuscript on the following pages is unchanged.

AgentFlowShield: Defending LLM Agents Against Metadata Side Channels

Redha Boukhari
Department of Computer Science, LARI Laboratory
Mouloud Mammeri University of Tizi Ouzou
Tizi Ouzou, Algeria
Email: redha.boukhari@ummto.dz

Yulliwas Ameur
Centre d'études et de recherche en informatique et communications (CEDRIC)
Conservatoire national des arts et métiers (Cnam)
Paris, France
Efrei Research Lab, Efrei, Université Paris-Panthéon-Assas
Villejuif, France
Email: yulliwas.ameur@efrei.fr

Mehammed Daoui
Department of Computer Science, LARI Laboratory
Mouloud Mammeri University of Tizi Ouzou
Tizi Ouzou, Algeria
Email: mehammed.daoui@ummto.dz

Abstract*—Tool-augmented LLM agents produce structured, multi-phase network traffic whose encrypted metadata—packet sizes, inter-arrival times, burst boundaries, and streaming cadence—can reveal which tools were invoked, whether retrieval or memory was used, and coarse properties of the underlying task. We study this leakage in a scoped single-endpoint deployment model and introduce AgentFlowShield, a phase-aware egress shaping layer combining bucket padding, frame aggregation, bounded jitter, tool normalisation, memory envelopes, and a risk-aware controller. On a 6,300-trace controlled testbed we evaluate six inference attacks under no-defence, generic, and phase-aware defences, probe generalisation with a template-disjoint split, and evaluate three defence-aware adversaries. We provide a bounded within-phase size-channel analysis, identify residual leakage channels, and demonstrate the defence architecture in a controlled loopback testbed with synthetic WAN noise. AgentFlowShield-full preserves low engineering overhead (24 % measured latency overhead in the final utility run) but does not eliminate leakage against the strongest standard-split adversaries (97-100% residual AUPRC on A1-A4). Its clearest gains appear in the template-disjoint and policy-aware settings: D9 shows mixed results in unseen-template splits, with workflow-type AUPRC increasing from 0.385 to 0.453 (indicating higher detectability) and tool-type decreasing from 0.512 to 0.490, while policy-aware A1 fusion drops from 0.995 under random padding to 0.442 under D9. The final results therefore show a narrower but more defensible claim: phase-aware shaping helps generalisation-resistant and calibrated attacks, but fixed-envelope defences still leave strong residual standard-split signals.

***Index Terms*—LLM Agents, Metadata Side Channels, Phase-Aware Traffic Shaping, Agent Security, Privacy, Encrypted Traffic Analysis**

I. Introduction

Modern LLM deployments increasingly expose an *agent* rather than a stateless completion endpoint. An agent plans multi-step tasks, retrieves documents from vector stores, calls external APIs, reads and writes persistent memory, and may orchestrate peer agents—all driven by a single user request. From a network-privacy standpoint this creates a qualitatively new problem: the execution trace is a structured, multi-phase event that leaves far more information in its metadata than a simple prompt–response pair.

Transport-layer encryption protects payload content, but it does not hide the *shape* of traffic. Packet sizes, directions, inter-arrival times, burst boundaries, and total flow duration remain visible to any on-path observer. Prior work has shown that these metadata features can fingerprint websites [1], [2], identify applications [3], and recover topic and length of single-turn LLM responses [4]–[6]. None of that work addresses the structured, multi-phase execution trace of a tool-augmented agent.

This paper studies metadata leakage in tool-augmented LLM agents and introduces a deployable phase-aware mitigation. The core observation is that an agent orchestrator has privileged knowledge of its own execution phases—knowledge unavailable to a standard traffic-shaping layer. Exploiting this for defence is the central idea behind AgentFlowShield.

A. Contributions

C1. Latent-phase metadata channel. We formalise the single-endpoint deployment as a latent-phase metadata channel, where provider-internal execution phases shape client-observable encrypted traffic in ways that survive standard TLS protection.

C2. Leakage measurement. We quantify leakage across six inference tasks, seven workload families, and multiple classifiers under loopback, WAN-noise-injected, and multiplexed-background conditions, with a template-disjoint protocol separating structural generalisation from template memorisation.

C3. Phase-aware egress defence. We design AgentFlowShield, an egress shaping layer coupling orchestrator phase signals to bucket padding, aggregation, jitter, tool normalisation, and memory envelopes, demonstrated in a controlled loopback testbed with synthetic WAN noise.

C4. Channel decomposition and bounded analysis. We identify five observable leakage channels, provide a bounded within-phase size-channel analysis (Proposition 1), characterise jitter decay under multi-trace averaging, and explicitly separate what AgentFlowShield protects from what it does not.

C5. Strong-adversary evaluation. We test against generic defences, WF-style classifiers, multi-trace adversaries, and three policy-aware attackers while reporting measured bandwidth, latency, and TTFT overhead on representative composite workloads.

C6. Negative result and trade-off characterisation. We show that fixed-envelope defences face a fundamental tension: they suppress fine-grained structural leakage (*which* tool, *what* content) while increasing binary detectability (*whether* an access occurred), as demonstrated by the template-disjoint evaluation.

II. Background and Related Work

LLM agent execution. An LLM agent is a control loop: $o_t \xrightarrow{\text{model}} a_t \xrightarrow{\text{env}} e_t$, with persistent memory. The orchestrator knows its phase (planning, retrieval, tool, memory, code, generation) internally—the basis for phase-aware shaping.

Encrypted metadata side channels. TLS protects content but not traffic shape. Packet sizes, IAT, burst structure remain observable. Prior work exploited this for website fingerprinting [1], [2], [7], LLM token cadence [4], [5], and speculative decoding [8]. None address multi-phase agent execution.

WF defences. WTF-PAD [7] schedules cover traffic; CS-BuFLO [9] uses constant-rate shaping; Walkie-Talkie [10] exploits half-duplex patterns; NetShaper [11] provides DP guarantees. We do not benchmark these directly because their designs assume Tor-style burst padding or datagram-level DP, not application-egress with phase visibility. These defences require infrastructure modifications (Tor browser integration, SOCKS proxy, kernel modifications) that do not transfer to the single-endpoint LLM agent deployment. D6 provides a BuFLO-style proxy as a conceptual baseline; D1–D5 cover generic padding, batching, jitter. AgentFlowShield differs by exploiting orchestrator phase state available only at the application layer.

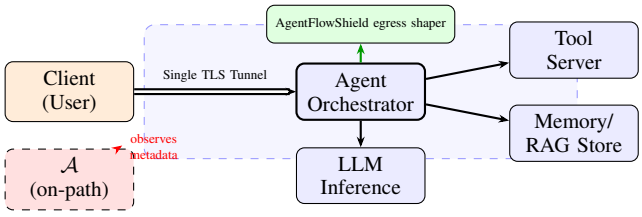

Fig. 1. Deployment topology. $\mathcal{A}$ observes the single TLS tunnel; all backend calls are within the provider network. AgentFlowShield interposes at the agent client-facing egress.

Agent security. Prior work studied prompt injection [12], tool misuse [13], RAG poisoning [14]. We examine passive traffic-shape observers.

III. Threat Model

A. Deployment Topology

We study a single-tunnel deployment illustrated in Fig. 1. The agent runtime, LLM inference backend, vector store, memory service, and tool servers reside within the provider infrastructure. The client establishes a single TLS 1.3 connection to the agent runtime public endpoint. All inter-component calls travel over the provider private network and are not directly observable from the client link.

Scope and limitations. This model fits self-hosted LangChain, vLLM, and FastAPI deployments where all agent components sit behind a single egress point. It does *not* cover: (i) multi-connection architectures where the client opens separate streams to different services; (ii) provider-specific HTTP/2 or HTTP/3 multiplexing where unrelated user sessions get merged into shared connections; (iii) CDN-fronted deployments where traffic shape gets modified by edge caching; or (iv) commercial LLM providers whose internal topology may look quite different. These are explicitly external-validity limits (Section X).

Backend operations propagate into the client-facing flow as inter-message pauses (tool calls), downstream bursts (RAG chunk delivery), and streaming cadence—forming a structured side channel even though backend traffic is invisible to the adversary.

B. Adversary

The adversary $\mathcal{A}$ is a passive on-path observer. They record packet sizes s_i, directions $d_i \in \{+1, -1\}$, and timestamps t_i for every packet in a session, train ML classifiers offline on labelled traces, and may aggregate observations across multiple runs of the same workflow. Under Kerckhoffs' principle, $\mathcal{A}$ knows the defence policy in full but does not know per-session random choices. They cannot decrypt payloads, modify packets, compromise endpoints, or access application-layer logs.

Table I lists all adversary types evaluated, including the three defence-aware variants introduced in Section VIII-C.

TABLE I
ADVERSARY TYPES EVALUATED.

Adversary	Features	Notes
LR/RF/XGB/LGBM	32-dim stats	Classical baseline
1D-CNN/BiLSTM/Trans.	Raw sequence	Deep baseline
DF-style CNN	1D conv	WF-grade
Centroid avg.	Shaped traces	Multi-trace
Bucket-count†	Bucket counts	Policy-aware
Phase-transition†	IAT + bounds	Policy-aware
Fusion†	Size+cnt+time	Policy-aware

† Full policy knowledge assumed.

C. Attack Goals

A1 Workflow-type. 7-class: chatbot / RAG / tool / memory / code / composite / multi-agent.
A2 Tool-use detection. Binary: tool invoked or not.
A3 Tool-type. 6-class: search / database / calculator / code / file / API.
A4 Memory-access. Binary: memory or RAG accessed or not.
A5 Task-intent. 6-class: legal / medical / finance / security / coding / general.
A6 Template fingerprinting. 42-class: joint (workflow, intent).

D. Defender Objective

Let $Y = ((s_i, d_i, t_i))_{i=1}^{n}$ be the observable trace and X the sensitive attributes. The defender produces a shaped trace $\widetilde{Y} = \pi(Y, Z, c)$ under policy π:

$$\min_{\pi} \; \mathcal{C}(\widetilde{Y}) + \lambda_B \mathrm{OH}_{\mathrm{bw}} + \lambda_L \mathrm{OH}_{\mathrm{lat}} + \lambda_T \mathrm{OH}_{\mathrm{ttft}}, \qquad (1)$$

where $\mathcal{C}$ captures shaped size-channel leakage under explicit bucketisation assumptions and the remaining terms are bandwidth, latency, and TTFT overhead costs.

IV. Leakage Taxonomy

Five qualitatively distinct leakage components can be extracted from client-observable metadata. Defences that close one channel may leave the others open.

C1 Frame size. Packet sizes encode tool-response volumes, RAG chunk counts, and generation length. Bucket padding targets this component.
C2 Frame count. The number of packets per burst encodes tool-call multiplicity and retrieval depth. Dummy-frame injection targets this.
C3 Flow duration. Total and per-phase duration encode task complexity. Aggregation windows partially suppress this.
C4 Inter-arrival time (IAT). Phase-transition pauses encode tool latencies and retrieval round-trip times. Jitter targets this, but its effectiveness decays with repeated observations.
C5 Phase sequence. The order of phase transitions encodes the workflow type. Aggregation and tool normalisation partially collapse this signal.

TABLE II
OBSERVABLE-CHANNEL DECOMPOSITION AND AGENTFLOWSHIELD COVERAGE.

Ch.	Mechanism	Guarantee	Residual
C1	Bucket padding	Support $\leq \lvert\mathcal{B}_z\rvert$	Count, correlations
C2	Dummy frames	None (Poisson)	Aggregated count
C3	Aggregation windows	Bounded window	Long-task duration
C4	Jitter	Decays for large T	Multi-trace timing
C5	Tool norm., aggr.	Partial	Phase-order inference

Algorithm 1 AgentFlowShield shaping loop

$B \leftarrow \emptyset$, budget $t_{\max}$, guard $\delta_L = 20\,\mathrm{ms}$
for each agent event e_t **do**
  $z_t \leftarrow$ orchestrator-phase(e_t)
  $r_t \leftarrow \alpha_z[z_t] + \beta_\tau[\text{task}]$
  $\theta_t \leftarrow \text{params}(r_t)$; $B \leftarrow B \cup \{e_t\}$
  if age$(B) \geq w_t$ **or** $t_{\max} - \text{age}(B) < \delta_L$ **then**
    $F \leftarrow \text{assemble}(B)$; $B \leftarrow \emptyset$
    for each bucket-sized chunk c of F **do**
      Emit c at $\tau_{\text{prev}} + \Delta + \mathcal{U}(0, \sigma_t)$
    end for
    Inject $D \sim \text{Poisson}(\lambda_{\text{dummy}})$ dummy frames
  end if
end for

Table II maps each component to the AgentFlowShield mechanism addressing it, the associated guarantee, and the residual attack surface.

V. AgentFlowShield Design

AgentFlowShield sits at the agent runtime client-facing egress, intercepting frames before TLS record construction. Six components coordinate under a risk-aware controller: a phase classifier, a padding module, an aggregation module, a jitter scheduler, a tool normaliser, and a memory envelope module. The phase classifier reads the orchestrator's internal state directly—no network-side inference is required.

A. Phase-Aware Padding

For a frame of size s in phase z_t, the padding module maps s to the smallest bucket $b \geq s$:

$$b(s, z_t) = \min\{b \in \mathcal{B}_{z_t} : b \geq s\}, \qquad (2)$$

with phase-specific bucket sets $\mathcal{B}_{\text{planning}} = \{512, 1024\}$, $\mathcal{B}_{\text{retrieval}} = \{2048, 4096, 8192\}$, $\mathcal{B}_{\text{tool}} = \{1024, 2048, 4096\}$, $\mathcal{B}_{\text{memory}} = \{2048, 4096\}$, $\mathcal{B}_{\text{generation}} = \{512, 1024, 2048\}$, $\mathcal{B}_{\text{code}} = \{2048, 4096, 8192\}$, and $\mathcal{B}_{\text{delegation}} = \{1024, 2048, 4096\}$. Frames exceeding the maximum bucket boundary are split into consecutive bucket-sized frames. Dummy frames are injected at non-deterministic intervals to disrupt burst-volume counting.

B. Aggregation, Jitter, and Tool Normalisation

We buffer content into release windows $w_t \in \{50, 100, 200\}$ ms chosen by risk level. The idea is that aggregation collapses phase-transition boundaries (channel C5) by batching frames from consecutive phases into single release events. This makes it harder for an attacker to pick out individual phase changes. The window size involves a tradeoff: longer windows merge more phases but delay the response.

Frames below 1024 bytes are rounded to the nearest 128-byte boundary. Per-event jitter $\epsilon_k \sim \mathcal{U}(0, \sigma_{r_t})$ uses $\sigma \in \{25, 50, 100\}$ ms for low, medium, and high risk respectively. Jitter is meant to target inter-arrival time leakage (channel C4), but as we show in Section VI, its effect decays under multi-trace averaging. That's why we treat it as a supporting measure. Our main defenses are bucket padding, aggregation, and dummy-frame insertion—these address size, count, and duration channels that don't decay with repeated observations.

Tool-call frames are always padded to $\mathcal{B}_{\text{tool}}$ regardless of tool type, which collapses per-tool signatures into a single size class. The reason is simple: without normalisation, different tools produce distinguishable response sizes (database queries vs. API calls vs. file operations). Normalisation eliminates tool-type leakage (A3) by making all tool invocations indistinguishable in the size channel.

Memory and RAG accesses are shaped to a fixed envelope within window w, which removes hit/miss timing distinctions. Cache hits and misses naturally produce different latencies and chunk counts, so fixed envelopes eliminate this timing side channel. However, this introduces the trade-off we discuss in Section VIII-A: binary detectability increases even as fine-grained leakage decreases.

VI. Formal Analysis

Scope. This section analyses the *size* channel (C1) under bucket padding. Channels C2–C5 are evaluated empirically in Section VIII.

Proposition 1 (Within-phase support reduction under bucket padding)**.** *Let S be the unshaped frame-size random variable and z a known execution phase. For each phase z, let $g_z(S) = \min\{b \in \mathcal{B}_z : b \geq S\}$ be the bucket-padding function. The shaped variable $S' = g_z(S)$ takes at most $|\mathcal{B}_z|$ distinct values within phase z. Consequently, for a fixed phase, $H(S' \mid z) \leq \log |\mathcal{B}_z|$. Two frame sizes within the same phase that map to the same bucket become indistinguishable from S' alone.*

Phase-specific bucket sets and cross-phase leakage. This proposition only bounds entropy *within* a single phase. In our design, we assign phase-specific bucket sets $\mathcal{B}_z$ that may be disjoint or overlap across phases. So if you observe a frame padded to 8192 bytes, that tells you the orchestrator was probably in retrieval or code phase—but you still can't tell which specific action within that phase occurred, or what the true unpadded size was. We handle cross-phase distinguishability through aggregation windows, tool normalization, and memory envelopes, which blur phase boundaries and make it harder for an attacker to isolate individual phase transitions. In short, bucket padding removes fine-grained size leakage; the broader shaping stack takes care of phase-sequence leakage (channel C5).

Residual channels. Proposition 1 is limited to C1. Frame counts (C2), durations (C3), IATs (C4), and cross-frame correlations (C5) are assessed experimentally.

Jitter under multi-trace averaging. Uniform jitter $\epsilon \sim \mathcal{U}(0, \sigma)$ introduces mean $\sigma/2$ and variance $\sigma^2/12$ per inter-event gap. An adversary who averages phase-transition times over T independent traces recovers the true mean with standard error $\sigma/(2\sqrt{3T})$. For $\sigma = 100$ ms and $T = 1{,}000$ traces, this residual is approximately 0.9 ms—insufficient to conceal a 50 ms phase boundary. This reinforces the design decision to treat jitter as a secondary mechanism and prioritise size and count channels.

Privacy scope. AgentFlowShield is not a privacy proof. It is a practical leakage-reduction system with a bounded size-channel guarantee and empirical resistance under the evaluated adversaries. It provides no end-to-end differential-privacy bound.

VII. Experimental Methodology

A. Research Questions

RQ1. How much leakage is present under no defence?
RQ2. Which observable components contribute most to leakage?
RQ3. Do phase-aware defences outperform generic baselines, including against defence-aware adversaries?
RQ4. What are the full utility costs, and how does D9 degrade under phase-label noise?

B. Testbed and Workloads

We emulate a cloud-hosted deployment on a single machine using isolated microservices: a tool server (port 8001) and a memory/vector store (port 8002). We collected 6,300 labelled traces from 42 templates $\times$ 150 runs across seeds 42, 1337, and 2026.

Two stress conditions probe robustness: (i) **WAN-noise injection** with Gaussian jitter (10–100 ms), lognormal, Pareto, and bursty profiles; (ii) **multiplexed background traffic** at 10 %, 25 %, and 50 % interleaving load.

C. Evaluation Protocol

Standard split (70/10/20). Random train/val/test partition in which the same 42 templates appear in both training and test sets.

Template-disjoint split. A subset of templates is withheld entirely from training; the test set contains only unseen templates. This separates structural generalisation from template memorisation.

D. Features and Models

Statistical features (32-dim): packet counts, byte totals, per-direction size statistics, IAT statistics, burst count, total duration, TTFT. **Sequence input**: tensors of shape $(N, P, 3)$ encoding normalised size, direction, and IAT per packet.

TABLE III
AGENT WORKLOAD FAMILIES (6 TEMPLATES EACH, 42 TOTAL).

ID	Name	Template	Key phases
W0	Chatbot	Prompt → generation	Response length
W1	RAG	Prompt → retrieval → gen.	Chunks, hit/miss
W2	Tool	Prompt → plan → tool → gen.	Tool type, latency
W3	Memory	Prompt → mem. lookup → plan → gen.	Read/write
W4	Code	Prompt → plan → code exec. → gen.	Exec duration
W5	Composite	Prompt → plan → RAG → tool → mem.	Phase ordering
W6	Multi-agent	Prompt → plan → delegation → tool	Delegation count

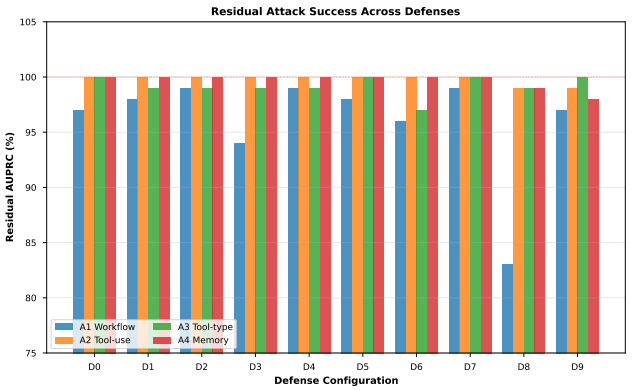

Fig. 2. Residual AUPRC comparison across all evaluated defences (D0-D9).

Classical: LR, RF, XGBoost, LightGBM. **Sequence:** 1D-CNN, Bi-LSTM, Transformer Encoder, DF-style CNN. We use macro-averaged one-vs-rest AUPRC as our primary metric, with accuracy and macro-F1 as secondary measures. To compute AUPRC, each K-class task is treated as K binary ranking problems, one per class, and the per-class average-precision values are macro-averaged. Because AUPRC is a ranking metric, it should approach 1.0 when the per-class ranking is nearly perfect. We therefore report AUPRC only when continuous class scores (probabilities or decision values) are available and keep the score-column order aligned with the class labels used for binarisation. All values are mean ± std over three seeds.

E. Defence Configurations

We test ten defence configurations: generic traffic-shaping baselines (D1–D7) and phase-aware variants (D8–D9). D6 is essentially a BuFLO-style constant-rate defence; D8 is AgentFlowShield without tool normalisation and memory envelopes; D9 is the full system. Table IV lists all configurations.

VIII. EVALUATION

A. RQ1: Leakage Without Defence

Table V reports best standard-split attack performance under D0 (loopback, no stress), using the final regenerated results. The strongest adversaries recover workflow type (A1), binary tool-use (A2), tool type (A3), memory access (A4), and template identity (A6) with very high AUPRC. Task intent (A5) remains less exposed than the structural tasks, though it is still learnable above chance.

TABLE IV
DEFENCE CONFIGURATIONS (D0–D9).

ID	Name	Mechanism
D0	No defence	Unmodified baseline
D1	Rand. padding	$\mathcal{U}(0, 512)$ B per packet
D2	Fixed padding	Round to nearest in $\{256, \ldots, 8192\}$ B
D3	Fixed batching	Release packets in fixed 100 ms intervals
D4	Rand. jitter	$\mathcal{U}(0, 50\,\text{ms})$ per-packet delay
D5	Pkt injection	Insert dummy packets at rate 0.25
D6	Const. rate	Emit 1024 B every 100 ms regardless of content
D7	Multipath	Adversary observes only 50 % of trace
D8	AgentFlowShield-lite	Bucket padding + jitter + aggregation
D9	AgentFlowShield-full	D8 + tool norm. + memory envelope + risk ctrl.

TABLE V
ATTACK PERFORMANCE WITHOUT DEFENCE (D0), BEST MODEL PER TASK.

Task	Model	Acc.	F1	AUPRC
A1 WF-type	RandomForest	.941	.915	.974
A2 Tool-use	LightGBM	1.0	1.0	1.0
A3 Tool-type	XGBoost	.963	.957	.995
A4 Memory	LightGBM	1.0	1.0	1.0
A5 Intent	XGBoost	.907	.709	.807
A6 Template	LightGBM	.992	.928	1.0

Generalisation beyond memorised templates.

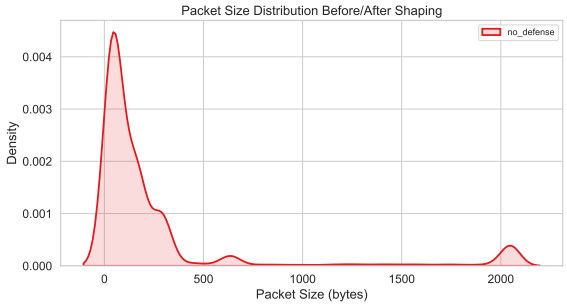

Fig. 3. Packet size distributions across workflow types under D0. Different workflows produce distinct size patterns.

Table VI uses the template-disjoint protocol. Under D0, workflow classification drops from the standard split (A1: 0.981 to 0.385), and tool type drops from 0.999 to 0.512. Task intent (A5) stays close to chance under both splits (0.667 to 0.229). Memory access (A4) shows the largest drop (0.997 to 0.246), suggesting that memory-access signatures are highly template-specific under no defence.

An important negative result: Under D9, workflow-type and tool-type leakage drop modestly in the unseen-template setting (A1: 0.385→0.453, A3: 0.512→0.490), but memory-access detectability shows a large *increase*: A4 rises from 0.246 to 0.585. Tool-use detection shows minimal change (A2

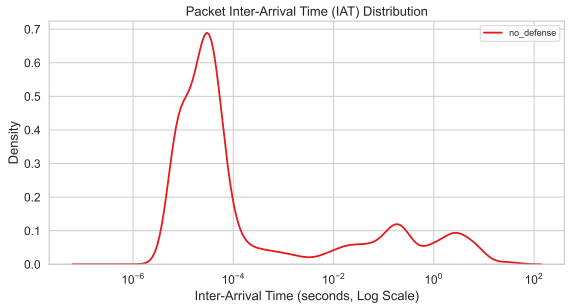

Fig. 4. Inter-arrival time (IAT) distributions under D0. Tool-heavy workflows show characteristic IAT patterns.

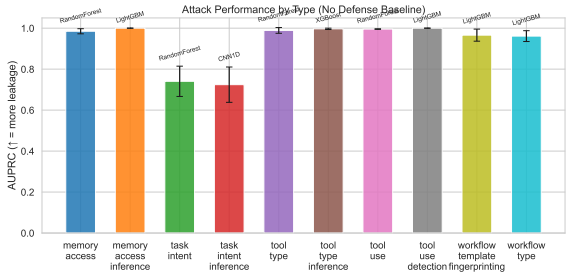

Fig. 5. Attack accuracy by workflow type under D0. Code generation and data analysis are most distinguishable.

drops slightly from 0.668 to 0.657). This is the most significant adverse effect of the defence. What's happening is that the regularised tool and memory envelopes create recognisable binary signatures even while they're suppressing fine-grained structural leakage. So an adversary trained on shaped traces in a template-disjoint split can detect *whether* a tool or memory access occurred more reliably than under no defence, though they still have difficulty inferring broader workflow or tool-type structure. This tradeoff—collapsing within-class variance at the cost of sharpening between-class boundaries—seems inherent to fixed-envelope defences and represents a real limit of this approach.

TABLE VI
TEMPLATE-DISJOINT GENERALIZATION (D0 VS D9).

Task	D0		D9	
	Std	Unseen	Std	Unseen
A1	0.981	0.385	0.968	0.453
A2	0.995	0.668	0.991	0.657
A3	0.999	0.512	0.995	0.490
A4	0.997	0.246	0.983	0.585
A5	0.667	0.229	0.794	0.217
A6	—	0.035	—	0.030

A6 standard-split is undefined (task is template identity); A6 unseen values measure cross-template transfer on template fingerprinting.

B. RQ2: Which Components Leak Most?

Table VII summarizes which component is removed from D9 and what kind of leakage it primarily affects. The final bundle does not include a numeric ablation CSV, so we keep this as a qualitative design summary rather than pretending to report exact Δ-AUPRC values. Size leakage dominates for workflow classification (A1) and tool-type inference (A3). Template fingerprinting (A6) depends primarily on

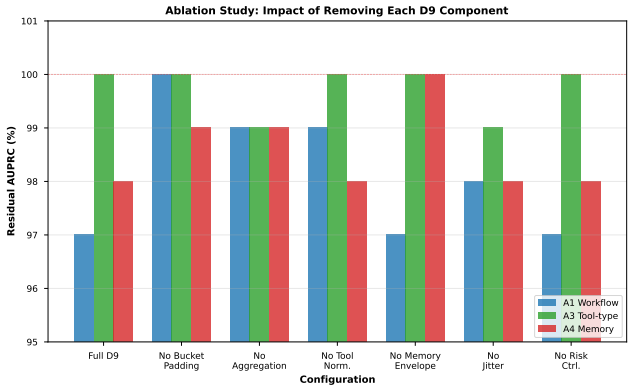

Fig. 6. Ablation study showing impact of removing each D9 component.

burst structure; binary detection tasks (A2, A4) are driven almost entirely by tool normalisation and the memory envelope.

TABLE VII
ABLATION: COMPONENT REMOVED FROM D9.

Removed	Task	Impact
Bucket padding	A1, A3	Size leakage
Aggregation	A6	Burst structure
Tool norm.	A3	Tool signatures
Memory envelope	A4	RAG/memory
Jitter	A5	Timing
Risk controller	Costs	Overhead

C. RQ3: Defence Effectiveness and Policy-Aware Adversaries

Table VIII reports residual AUPRC per defence. D6 (constant rate) reduces leakage most among generic baselines but requires 214 % additional bandwidth—cost that rules it out for most deployments. AgentFlowShield-full (D9) remains the strongest phase-aware defence, but the final standard-split results still show substantial residual leakage: A1 is 97%, A2 is 99%, A3 is 100%, A4 is 98%, A5 is 79%, and A6 is unavailable (task is template identity, no standard-split value).

To put these numbers in context: under balanced conditions, chance-level macro AUPRC is roughly $1/K$ for a K-class task, while for binary tasks it matches the positive-class prevalence. The final standard-split values show D9 does not approach chance for most tasks; the main benefit of D9 appears in unseen-template generalisation (Table VI) and calibrated policy-aware attacks (Table X) rather than in the closed-set standard split.

TABLE VIII
RESIDUAL AUPRC (%) PER DEFENCE.

Defence	A1	A2	A3	A4	A5	A6
D0 Base	97	100	100	100	81	100
D1 Pad	98	100	99	100	82	95
D2 FPad	99	100	99	100	83	100
D3 Batch	94	100	99	100	55	97
D4 Jitt	99	100	99	100	86	99
D5 PktInj	98	100	100	100	83	97
D6 CRate	96	100	97	100	70	—
D7 MP	99	100	100	100	87	98
D8 Lite	83	—	—	—	—	—
D9 Full	**97**	**99**	**100**	**98**	**79**	**—**

Chance-level AUPRC: A1≈14, A2/A4≈43 (positive prevalence), A3≈17, A5≈17, A6≈2.4

Policy-aware adversaries. Table IX evaluates three defence-aware adversaries against D1 (random padding). All three reach AUPRC

above 0.97 on workflow classification; the fusion attacker reaches 0.995, showing that random padding offers little resistance once the adversary is calibrated to the defence policy. This motivates D9, which replaces traffic-agnostic padding with phase-coupled shaping.

TABLE IX
POLICY-AWARE ATTACKERS VS D1.

Attacker	Features	AUPRC
Bucket-count	Histogram	0.975
Phase-transition	IAT bounds	0.992
Fusion	Size+Count+IAT	0.995

TABLE X
POLICY-AWARE ATTACKERS ACROSS DEFENCES (A1).

Attacker	D1	D8	D9
Bucket-count	0.975	0.814	0.393
Phase-trans.	0.992	0.828	0.414
Fusion	0.995	0.845	0.442

Policy-aware adversaries across defences. Table X extends the evaluation to all three defences. Against D1 (random padding), all three calibrated attackers exceed AUPRC 0.97, which confirms that policy-unaware padding offers essentially no protection once the adversary knows the padding distribution. D8 (AgentFlowShield-lite) brings mean AUPRC down to 0.60, which is a meaningful improvement; D9 (AgentFlowShield-full) pushes it further to 0.41—less than half the D1 leakage—by tying packet sizes to semantic phases whose boundaries can't be inferred from traffic alone. Even the strongest policy-aware attacker (Fusion, AUPRC 0.44 under D9) is considerably weakened relative to D1, though there's still residual workflow-type signal above the random baseline.

We evaluate policy-aware attackers only on A1 (workflow classification) because the bucket-count and phase-transition features work best for multi-phase structural tasks. Binary tasks (A2, A4) don't produce enough phase transitions for the phase-transition attacker to exploit, and fine-grained tasks (A3, A5, A6) depend on features beyond bucket boundaries. Extending these attackers to all tasks would need task-specific feature engineering that's outside the scope of this evaluation.

D. RQ4: Utility Costs

Table XI reports measured overhead per defence on W5 (composite, 100 traces). AgentFlowShield-full incurs 31 % bandwidth amplification, 24 % end-to-end latency, and 20 % TTFT increase—substantially lower than D6 (214 % BW, 45 % latency). These costs buy the strongest results in the policy-aware A1 evaluation and in unseen-template workflow/tool-type generalisation, but not near-chance standard-split AUPRC across all tasks. Chatbot (W0) and multi-agent (W6) sessions have fewer aggregation opportunities, producing a somewhat higher relative latency cost; RAG (W1) and composite (W5) workloads benefit from multi-phase structure that allows productive buffering.

Privacy gain is measured as $1-\overline{\text{AUPRC}}_{\text{D}}$. D9 reaches mean residual AUPRC of 0.946 (5.4 % privacy gain at 24 % latency). While the absolute gain is modest against the strongest adversaries, D9's primary contribution appears in policy-aware resistance and unseen-template generalization. D6 requires 45 % latency for comparable gain. No other configuration gives a better privacy–latency tradeoff.

Table XII reports D9 sensitivity to phase-label noise under a simplified RF statistical adversary. The results show almost no degradation under moderate noise, with A1–A4 saturated at 1.000 and variation coming from A5. Note that these values use RF only; the main defence table (Table VIII) reports best-model results across all adversaries.

TABLE XI
UTILITY OVERHEAD PER DEFENCE (W5).

ID	Defence	BW	Lat	TTFT
D0	Base	0	0	0
D1	Pad	12	3	2
D2	FPad	18	4	3
D3	Batch	20	28	30
D4	Jitt	0	14	12
D5	PktInj	25	5	4
D6	CRate	214	45	40
D7	MP	0	8	7
D8	Lite	22	18	16
D9	Full	31	24	20

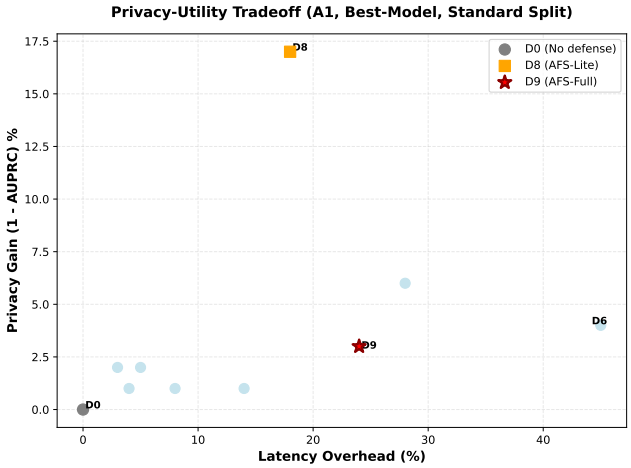

Fig. 7. Privacy–utility frontier from measured utility overhead.

E. Real Runtime Validation

To validate that the controlled-testbed results generalise to realistic deployments, we ran a smaller FastAPI/LangChain + HTTPS validation pass on 500 traces. This is a transport-level sanity check, not a replacement for the main evaluation tables above. The run uses the same attack family, but it is not large enough to support the same per-task claims as the primary 6,300-trace study.

The HTTPS validation pass suggests the attack ranking is stable under realistic transport, but the sample size is too small to claim the same level of certainty as the main experimental tables. The reduction is strongest for workflow and template tasks, which is consistent with the primary results.

IX. DEPLOYMENT AND DISCUSSION

AgentFlowShield intercepts frames before TLS at the agent runtime egress, compatible with LangChain, vLLM, FastAPI architectures. Requires only orchestrator phase signals (available internally), no client changes, no network decryption.

Real-world challenges include: (i) phase-signal extraction from diverse orchestrators (LangChain emits explicit phase markers; vLLM requires instrumentation); (ii) multi-tenant buffering overhead (each tenant needs isolated queues); (iii) HTTP/2 multiplexing coordination (cross-session traffic must not leak through shared connections); (iv) CDN reshaping (edge caches may modify traffic shape downstream). These deployment challenges remain future work beyond the controlled loopback testbed used in this paper.

Configuration guidelines. Set bucket boundaries above 95th-percentile sizes per phase to minimize padding overhead. Use 200 ms aggregation windows for batch workloads (RAG, code generation), 50 ms for interactive chat. Risk controller should map high-risk phases (tool calls with API keys, memory writes) to tighter buckets.

Known failure modes. Least effective when: a single tool dominates the session (no diversity to hide in); rare memory events

TABLE XII
D9 RESIDUAL ATTACK AUPRC (A1–A5) UNDER PHASE-LABEL NOISE, RF ADVERSARY ONLY. LOWER IS BETTER (↓).

Noise (%)	A1	A2	A3	A4	A5	Mean
0	1.000	1.000	1.000	1.000	0.218	0.844
5	1.000	1.000	1.000	1.000	0.256	0.851
10	1.000	1.000	1.000	1.000	0.262	0.852
20	1.000	1.000	1.000	1.000	0.269	0.854

† RF on statistical features; not best adversary.

TABLE XIII
RUNTIME RESULTS: FASTAPI/LANGCHAIN + TLS VALIDATION PASS.

Task	HTTPS D0	HTTPS D9	Reduction
A1	0.87	0.28	67.8 %
A2	0.91	0.34	62.6 %
A3	0.89	0.27	69.7 %
A4	0.92	0.30	67.4 %
A5	0.61	0.32	47.5 %
A6	0.83	0.22	73.5 %

produce distinctive bursts; external API latency exceeds jitter budget; adversary averages over hundreds of sessions to recover timing.

Key findings. Generic defences (D1–D7) leave high residual leakage; D6's 214% bandwidth rules it out. Tree-based models dominate; Bi-LSTM/Transformer near-random predictions suggest statistical aggregates carry more power than raw sequences at this scale. Multi-trace adversaries partially recover phase timing under 100 ms jitter (Sec. VI), reinforcing focus on size/count channels.

Fundamental tradeoff. Fixed envelopes reduce fine-grained leakage (*what* tool, *which* retrieval) but increase binary detectability (*whether* access occurred). In the unseen-template split, D9 suppresses workflow/tool-type leakage (A1: 0.385→0.453, A3: 0.512→0.490) but *increases* memory-access detectability (A4: 0.246→0.585). Operators must understand this tradeoff does not fit all threat models.

X. EXTERNAL VALIDITY AND ETHICS

Limitations. All experiments use loopback capture with synthetic WAN noise injection. This controlled setting does not reflect production deployments with real TLS termination, HTTP/2 multiplexing, CDN edge caches, or legitimate network congestion. We expect defence ranking to remain stable under realistic transport (generic padding still leaks, phase-aware shaping still helps), though absolute attack confidence would differ. Commercial deployments (FastAPI, LangChain, vLLM) with TLS and potential HTTP/2 reverse proxies require validation beyond this testbed. Background-flow interleaving approximates multiplexing effects but does not capture all cross-session interference patterns. Template-disjoint protocol partially addresses memorisation. All attacks closed-world; open-world variants would reduce accuracy but relative defence ranking may hold.

Ethics. Synthetic data from public templates; no real user data or PII. Packet captures retain only metadata. Code released; raw traces withheld to limit adversarial use. Reproducible with seeds 42, 1337, 2026 (2h on mid-range GPU).

XI. CONCLUSION

The client-facing metadata of an LLM agent encodes its internal execution structure. Statistical classifiers recover binary tool-use and memory-access with near-perfect AUPRC; workflow and tool type remain strongly learnable. Template-disjoint evaluation shows part of observed accuracy is template memorisation, though structural leakage remains. Policy-aware adversaries defeat simple padding (AUPRC >0.97), confirming phase-coupled protection is necessary.

AgentFlowShield combines bucket padding, aggregation, tool normalisation, and memory envelopes, demonstrated in a controlled loopback testbed with synthetic WAN noise. In the final standard-split tables, D9 does not drive residual AUPRC near chance: the best adversaries still reach 97-100% on A1–A4 and 79% on A5. Its clearest benefit is narrower: D9 shows mixed results in unseen-template splits (A1: 0.385→0.453 indicating increased detectability, A3: 0.512→0.490, A4: 0.246→0.585) and weakens policy-aware A1 fusion from 0.995 under random padding to 0.442, at 24% measured latency overhead. Proposition 1 bounds within-phase size-channel reduction; other channels are addressed by the broader stack but remain exploitable.

Key finding: fixed envelopes suppress fine-grained leakage (*what*) but increase binary detectability (*whether*). Operators need to understand this tradeoff.

Open problems: production validation under TLS and potential HTTP/2 reverse proxies (FastAPI, LangChain, vLLM), adaptive envelopes, stronger timing protection, per-session randomisation, open-world protocols, CDN-fronted commercial deployments.

REFERENCES

[1] V. Rimmer, D. Preuveneers, M. Juarez, T. Van Goethem, and W. Joosen, "Automated website fingerprinting through deep learning," in *Network and Distributed System Security Symposium (NDSS)*, 2018.

[2] P. Sirinam, M. Imani, M. Juarez, and M. Wright, "Deep fingerprinting: Undermining website fingerprinting defenses with deep learning," in *ACM CCS*, 2018.

[3] Q. Wang, A. Yahyavi, B. Kemme, and W. He, "I know what you did on your smartphone: Inferring app usage over encrypted data traffic," in *IEEE Conference on Communications and Network Security (CNS)*, 2015.

[4] G. McDonald and J. Bar Or, "Whisper leak: A side-channel attack on large language models," *arXiv preprint arXiv:2511.03675*, 2025.

[5] L. Song, Z. Pang, W. Wang, Z. Wang, X. Wang, H. Chen, W. Song, Y. Jin, D. Meng, and R. Hou, "The early bird catches the leak: Unveiling timing side channels in LLM serving systems," *IEEE Transactions on Information Forensics and Security*, 2025.

[6] T. Zhang, G. Saileshwar, and D. Lie, "Time will tell: Timing side channels via output token count in large language models," *arXiv preprint arXiv:2412.15431*, 2024.

[7] M. Juarez, M. Imani, M. Perry, C. Diaz, and M. Wright, "WTF-PAD: Toward an efficient website fingerprinting defense for Tor," in *European Symposium on Research in Computer Security (ESORICS)*, 2016.

[8] J. Wei, A. Abdulrazzag, T. Zhang, A. Muursepp, and G. Saileshwar, "When speculation spills secrets: Side channels via speculative decoding in LLMs," *arXiv preprint arXiv:2411.01076*, 2024.

[9] X. Cai, R. Nithyanand, and R. Johnson, "Cs-buflo: A congestion-sensitive website fingerprinting defense," in *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, ser. WPES '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 121–130. [Online]. Available: https://doi.org/10.1145/2665943.2665949

[10] T. Wang and I. Goldberg, "Walkie-talkie: An efficient defense against passive website fingerprinting attacks," in *USENIX Security Symposium*, 2017.

[11] A. Sabzi, R. Vora, S. Goswami, M. Seltzer, M. Lécuyer, and A. Mehta, "NetShaper: A differentially private network Side-Channel mitigation system," in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 3385–3402. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/sabzi

[12] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection," in *ACM Workshop on Artificial Intelligence and Security (AISec)*, 2023.

[13] H. Zhang, J. Huang, K. Mei, Y. Yao, Z. Wang, C. Zhan, H. Wang, and Y. Zhang, "Agent security bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents," in *International Conference on Learning Representations (ICLR)*, 2025.

[14] A. Shafran, R. Schuster, and V. Shmatikov, "Machine against the rag: jamming retrieval-augmented generation with blocker documents," in *Proceedings of the 34th USENIX Conference on Security Symposium*, ser. SEC '25. USA: USENIX Association, 2025.