Inference-Layer Security: Defending Against Adversarial Inference and Infrastructure Abuse

Keifer Lee
keifer@nymlr.com
New York Machine Learning Research Guild (NYMLR)
New York, NY, USA

Abstract

Operating a large language model (LLM) as a service requires more than inference infrastructure: the provider must also defend against adversarial interactions that seek to exploit the service, including jailbreaking for harmful use, sophisticated denial of service, and distillation attacks. We study this problem at the inference layer, using a hypothetical frontier lab, *Five Elements Inc.*, as a running example. Because no public labelled dataset of adversarial LLM usage exists, we introduce a structural causal model (SCM) that generates a realistically grounded, labelled dataset of user-sessions, with coordinated multi-account campaigns, platform feedback, and three tiers of label observability. On this dataset we train a practical gradient-boosted detector that classifies each user-session as benign or malicious and, if malicious, by attack type. Against oracle labels the detector very nearly solves the binary task (AUPRC 0.993), yet against the operational labels a real Trust & Safety team would hold, the same model scores an AUPRC of only 0.313: the detector is more accurate than the labels used to evaluate it. For attack-type attribution, a naive argmax is dominated by the 98% benign prior (macro-F1 0.295), whereas a simple thresholded decision engine raises macro-F1 to 0.489 without sacrificing accuracy. The dataset is publicly released.

Keywords

adversarial inference, inference security, causal modeling, trust and safety, synthetic data, distillation attacks

1 The Need to Defend AI Systems

A growing share of white-collar workflows now depends on AI in one form or another: roughly 40% of Coinbase's daily code is reportedly AI-generated [21], Cloudflare lets AI agents register their own cloud accounts, subscriptions, and domains end-to-end [17], and Anthropic's Claude has been integrated directly into Excel [3]. Like any engineered system, these deployments carry exploitable vulnerabilities, yet adoption is outpacing security: GenAI usage in organisations rose from 33% in 2023 to 71% in 2024 [12] without a commensurate security stack, including in services where consistency, security, and stability are non-negotiable. Coinbase's CEO has stated that "non-technical teams are now shipping production code" [26] at the exchange, days before its trading engine went down [33]; Cloudflare reports that "85% of IT decision makers report that employees are adopting AI tools faster than their IT teams can assess them" [12]. This haste persists despite METR's randomised controlled trial, which found that experienced developers were 19% *slower* with AI tooling while believing they were 20% faster [9].

Conventional software is already fraught with exploits, known or otherwise. Google's Threat Intelligence Group tracked 90 zero-days exploited in the wild in 2025 alone, a 15% increase year-over-year [10], and these are only the ones we know about; the population of *living* zero-days—unknown to the vendor, exploitable for an extended period, sometimes never patched at all [38]—is by definition uncountable. Software is a formal system layered on other formal systems, and it inherits their incompleteness in the form of edge cases nobody thought to check. The result is a steady stream of high-profile incidents, such as the MOVEit/GoAnywhere/Cleo managed-file-transfer cascade [1] and CVE-2026-32202, a zero-click Windows Shell vulnerability actively exploited by APT28 via weaponised LNK files [15]. Deep generative models expand this attack surface further, introducing vectors that are largely unknown to the public that relies on these services and that are harder to detect and guard against. Notable attack vectors from recent months include:

(1) **Poison 250 documents in a corpus of billions to backdoor any LLM.** A joint Anthropic / UK AI Security Institute / Alan Turing Institute study found that as few as 250 adversarial documents—~0.00016% of a 13B model's training tokens—reliably install a backdoor, regardless of model scale [6, 32].
(2) **Trivial jailbreak via past-tense framing and persona injection.** The OpenAI–Anthropic joint safety evaluation found Claude models were "most vulnerable to the past tense jailbreak" [22], and the Mexican government attacker simply told Claude it was working a bug bounty programme to unlock the full kill chain [13].
(3) **Distillation attacks at industrial scale.** Anthropic disclosed that DeepSeek, Moonshot and MiniMax ran ~24,000 fraudulent accounts and 16M+ exchanges against Claude to extract chain-of-thought training data [7]—MiniMax pivoted to a new Claude model within 24 hours of its release [2].
(4) **Agentic abuse / weaponisation as a multiplier.** Anthropic's report on disrupting AI espionage describes a Chinese state-sponsored actor (GTG-1002) achieving 80–90% autonomy of a cyber-espionage campaign with only 4–6 human touchpoints; their model was making "thousands of requests, often multiple per second" [5].
(5) **AI-authored zero-days in the wild.** Google's GTIG published a case where they could fingerprint a 2FA-bypass zero-day exploit as AI-generated from its hallucinated CVSS score and textbook docstrings [14].

Recent high-profile attacks include:

(1) **The Mexican government breach (Dec 2025 – Feb 2026).** A single operator used Claude Code + GPT-4.1 to compromise 10 government bodies and exfiltrate ∼150 GB / 195 million identities—Claude executed ∼75% of the remote commands [29].
(2) **Linux kernel LPE + Chrome zero-day chain (early 2026).** A high-severity Linux kernel local-privilege-escalation [15] landed alongside four actively exploited Chrome zero-days in the first quarter of 2026 alone [27]—Google was the second-most-targeted vendor of 2025.
(3) **North Korean IT-worker fraud, AI-augmented.** Anthropic disclosed that DPRK operatives were using Claude to fraudulently secure and maintain remote employment at Fortune 500 tech companies [4], funnelling salaries back to the regime's weapons programmes.
(4) **LLMjacking at scale—Operation Bizarre Bazaar.** Pillar Security's honeypots recorded ∼972 attack sessions per day over 40 days targeting exposed LLM endpoints [25], with a 9-minute credential-to-first-exploitation time, sold via the silver.inc marketplace on Telegram/Discord [30].
(5) **Supply-chain compromise of LiteLLM and Trivy.** In March 2026, the TeamPCP / UNC6780 group embedded the SANDCLOCK credential stealer into LiteLLM and Trivy builds [14], exfiltrating AWS and GitHub secrets—and, by extension, the AI environments those keys unlocked.

These examples show that attacks are varied and have been demonstrated to great effect. How, then, can model providers prevent their services from being abused? No single measure suffices; defence requires a suite of measures at each stage of the model lifecycle (Figure 1). In this paper we focus on the serving side, for three reasons:

(1) Model weaknesses are a function of many factors, including the training data, training procedure, infrastructure, and architecture (for example, energy-based models are less prone to hallucination than autoregressive LLMs, while weaker language models lack the coherence needed for fine-grained control). All of these are fixed at training time; once a model is deployed, its priors cannot be changed without a costly retraining run.
(2) The serving side is far easier to control: the serving platform has full visibility and well-tested deterministic levers, such as API rate limiting, harmful-content filtering, IP/device blocklisting, and velocity throttling. The same toolbox that payment networks have used for decades to keep card fraud at ∼7 bps of GMV largely transfers.
(3) Serving-side interventions generalise across model types and attack types. A rate limit is indifferent to whether the model behind it is GPT-5.5 or a fine-tuned Qwen 3.6, and an attacker's clustered account graph has the same shape regardless.

Throughout, we take the perspective of a hypothetical frontier LLM provider, *Five Elements Inc.*, and ask how it can serve its models responsibly while defending against abuse by adversaries. This paper makes the following contributions:

Stage	Attacks at this stage	Defensive levers
baked-in: no runtime lever		
Training data corpus curation	data poisoning (250 docs); sock puppets; PII injection	*none—corpus is frozen*
Training / RLHF pre-train + tune	backdoor via fine-tune; alignment failures; RLHF gaming	*none—weights are frozen*
structural runtime (deploy / infra) — scope of this paper begins here		
Deployment supply chain	LiteLLM/Trivy compromise; SANDCLOCK credential stealer	SBOM; image signing; reproducible builds
Serving infra gateway, isolation	cross-tenant KV leak; credential theft; LLMjacking	tenant isolation; key rotation; TLS pinning; gateway rate limits
live runtime (per-request / session)		
Request layer prompts, filters	prompt injection; past-tense jailbreak; probes	pre-filter: perplexity; post-filter: safety score
User session behaviour, ladder	distillation queries (16M); DoS; agentic abuse	behavioural detection; intervention ladder

Figure 1: The serving stack and where exploits enter at each stage: the model can be poisoned via its training corpus; the infrastructure can be compromised for jailbreaks or data leaks (e.g. leaking system prompts, or other users' prompts); and the user level can be exploited to serve adversarial results, jailbreak, or distill the model.

- a taxonomy, the *Ladder of Abstraction*, for building an inference-layer defence stack rung by rung, from raw event streams to multi-agent games (Section 2);
- a structural causal model (SCM) that simulates coordinated adversarial campaigns against a serving platform, with realistic label delay, account-level label noise, and three tiers of label observability (Sections 3–4.1); the resulting dataset is public [18];
- a practical GBDT detector with a tempered class-weighting recipe, and a family of thresholded decision engines for attack-type attribution under extreme class imbalance (Sections 4–5).

2 Defending Five Elements Inc.

Figure 2 summarises our task setup at *Five Elements Inc.* Platform Trust & Safety (T&S), a standard architecture that provides context for the problem statement. The goal is to *give legitimate users a low-friction experience while identifying and blocking adversarial actors.*

Achieving this goal raises several questions:

(1) What interventions should the platform apply, and how should they be enacted?
(2) How can adversarial actors be identified reliably when each *type* presents a different profile and behaviour (e.g. a bot farm vs. a jailbreak)? Is a binary benign/malicious classification sufficient?
(3) How should benign and malicious be defined, given that, for example, a genuine researcher can resemble a distillation attacker?
(4) At what *level* should the platform intervene: user, session, or transaction?

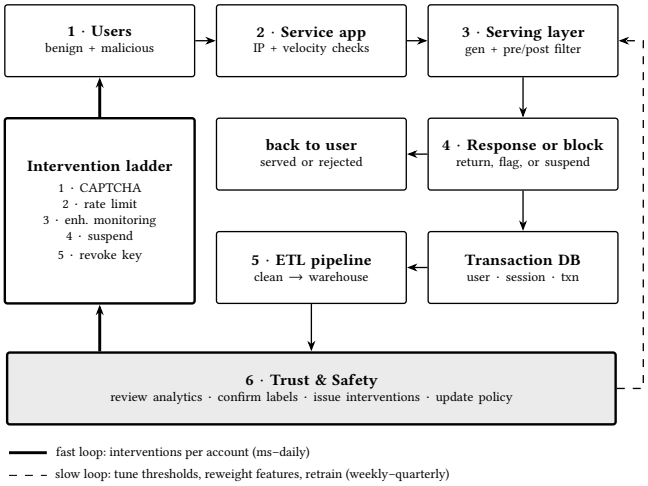


Figure 2: The complete *Five Elements Inc.* serving pipeline, with its intervention (fast) and policy (slow) feedback loops.

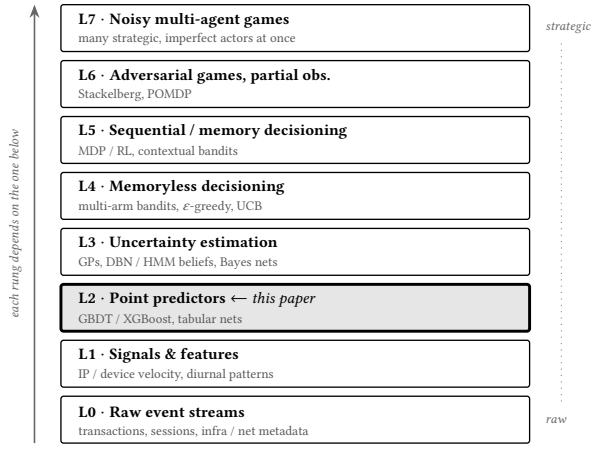


Figure 3: The Ladder of Abstraction: L0 raw event streams → L7 noisy multi-agent games. Build each rung before climbing.

To structure these questions we use the **Ladder of Abstraction** (Figure 3), a solution roadmap that ranges from the most fundamental unit, raw event streams at L0, to noisy, partially observable, multi-agent games at L7.

In this paper we address L2, point-predictor models such as XGBoost [11]. Reliable point predictions are a prerequisite for the higher rungs, since each rung consumes the one below it: a bandit (L4) needs a reward signal, and that reward is only as trustworthy as the point estimate (L2) and the uncertainty around it (L3) that feed it. Likewise, a Stackelberg solver [37] (L6) reasoning about an adversary's best response needs a belief state (L3) to reason over. Decisions built on poor estimates are confidently wrong, which is the most damaging failure mode in an adversarial setting.

2.1 Problem Formulation

Concretely, our task is as follows: given a user-session, determine whether it is benign (0) or adversarial (1), and if it is malicious, what kind of bad actor it is:

$$y_{\text{adv}} \in \{0, 1\}, \qquad y_{\text{type}} \in C, \tag{1}$$

$$C = \left\{ \begin{array}{ll} 0\text{: } \texttt{benign} & 5\text{: } \texttt{dos} \\ 1\text{: } \texttt{distillation} & 6\text{: } \texttt{credential_abuse} \\ 2\text{: } \texttt{jailbreak} & 7\text{: } \texttt{data_extraction} \\ 3\text{: } \texttt{bot_farm} & 8\text{: } \texttt{agentic_misuse} \\ 4\text{: } \texttt{harmful_use} & \end{array} \right\}, \tag{2}$$

where each non-benign class maps to a documented real-world surface (Table 1).

Why aggregate to the user-session? A single transaction is almost information-free in isolation: one prompt that trips the safety filter could be a curious researcher, a typo, or the first probe of a jailbreak. The discriminating signal lives in the *shape* of a session—the trajectory of perplexity, the accumulation of refusals, the diurnal rhythm, the inter-request timing variance (low for bots, high for humans). The user-session preserves these behavioural shapes while remaining granular enough for targeted interventions. A coarser unit (whole-account, all-time) loses the onset of an attack; a finer one (per-transaction) amounts to classifying noise.

The dataset also contains a tenth type at the *campaign level*, `multi_surface`: a coordinated operation spanning several of the above types. Because it is a *group* property requiring cross-account aggregation, it is out of scope for a per-session point predictor, and we defer it to the higher rungs.

Formally, given a feature vector $\mathbf{x} \in \mathbb{R}^d$ for a session, we estimate

$$\hat{y}_{\text{adv}} = f_{\text{adv}}(\mathbf{x}) \in [0, 1], \qquad \hat{\mathbf{y}}_{\text{type}} = f_{\text{mc}}(\mathbf{x}) \in \Delta^8, \tag{3}$$

where Δ^8 is the 8-simplex over the 9 classes (probabilities summing to 1), and f_{adv} collapses to the complement of the benign-class probability in the simplest single-model framing:

$$\hat{y}_{\text{adv}} = 1 - \left[f_{\text{mc}}(\mathbf{x}) \right]_{\texttt{benign}}. \tag{4}$$

2.2 Metrics

For the binary task we report precision–recall curves and, for threshold-agnostic comparison, the area under them. For attack-type attribution, a multi-class task, we report per-class precision and recall and their macro and micro aggregates. For the binary case, with TP, FP, FN the true positives, false positives and false negatives:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \qquad \text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{5}$$

$$\text{AUPRC} = \int_0^1 \text{precision}(\text{recall})\; d(\text{recall}). \tag{6}$$

For the multi-class case we compute these per class k in a one-vs-rest fashion and then aggregate over K classes. With $\text{TP}_k, \text{FP}_k, \text{FN}_k$

Table 1: The attack-type taxonomy C: what each class is, which team at *Five Elements Inc.* a detection is routed to, and what a miss actually costs.

Class	What it is	Routed to	What a miss actually costs
`distillation`	Capability extraction by competitors	Legal & Policy	a competitor ships your exfiltrated capability
`data_extraction`	Training-data / PII / system-prompt exfiltration	Legal & Policy	training-data, PII, or system-prompt leakage
`jailbreak`	Safety bypass (past-tense framing, persona injection)	Trust & Safety	a safety bypass loose in the wild
`harmful_use`	Extortion, malware, ransomware-as-a-service	Trust & Safety	real-world harm—extortion, malware, RaaS
`agentic_misuse`	Tool-chain abuse as an attack platform	Trust & Safety	your own tools turned into an attack platform
`bot_farm`	Coordinated fake-account infrastructure	Platform Integrity	a fake-account fleet scaling unchecked
`dos`	Unbounded consumption / token flooding	Platform Integrity	unbounded consumption, token flooding
`credential_abuse`	Stolen-key / reverse-proxy LLMjacking	Security	stolen-key / reverse-proxy LLMjacking
`multi_surface`[†]	Coordinated multi-vector campaign	Threat Intelligence	a coordinated multi-vector campaign
`benign`	Honest usage	No action	*(the flip side: a good user slapped with needless friction)*

[†] A campaign-level (group) property; out of scope for the per-session predictor.

the counts for class k, precision_k and recall_k follow as above, and

$$\text{macro-P} = \frac{1}{K}\sum_{k=1}^{K}\text{precision}_k \quad \text{(every class counts equally)}, \tag{7}$$

$$\text{micro-P} = \frac{\sum_k \text{TP}_k}{\sum_k(\text{TP}_k + \text{FP}_k)} \quad \text{(every sample counts equally)}. \tag{8}$$

Macro treats `agentic_misuse` (rare) as importantly as `benign` (the majority), whereas micro lets the majority class dominate.

2.3 Data Requirements

Existing LLM-serving datasets, such as BurstGPT [34] and ServeGen [40], capture serving patterns and transactions of generic interactions but do not meet the requirements of this task, which also needs:

(1) **Labels** indicating whether each transaction or user-session is benign or adversarial and, if adversarial, of what *type*. To our knowledge no public labelled dataset of this kind exists, and providers are strongly disincentivised from releasing one: publishing confirmed-adversarial traffic reveals how much got through (reputational damage), creates legal and regulatory exposure, *and* hands adversaries a labelled map of the behavioural signatures that detectors catch.

(2) **Scale**, on the order of millions of transactions at least, because adversarial interactions are rare relative to benign ones. At a deliberately conservative transaction-level rate of $p = 0.01\%$ (1 in 10,000), each malicious example comes with ∼10,000 benign ones, so collecting 1,000 malicious training rows requires scanning

$$N_{\text{total}} = \frac{1{,}000}{p} = \frac{1{,}000}{10^{-4}} = 10^7 \text{ transactions}. \tag{9}$$

The choice of 0.01% is conservative. Open-web figures are far higher: Arkose Labs found that 73% of web/app traffic in 2023 was bot and fraud-farm traffic [28], and Statista puts 2024 bad-bot traffic at around 37% [31]. These, however, are *traffic*-level figures on the open internet. At the *account* level on a friction-gated platform (payment, KYC, regional restrictions), the rate should be far lower, since these controls filter out most automated and low-effort abuse; we assume on the order of a few percent of accounts at most. Allowing for the fact that active accounts generate more activity, we assume the transaction-level rate for a given platform lies in the 0.01%–5% band, and adopt its lower end. Given these requirements and the absence of a suitable open dataset, we construct a synthetic one.

3 Synthetic Data Generation

The simplest way to synthesise a dataset is to choose a set of features, define a distribution for each, and sample each column independently. This ignores how features correlate and covary. A more faithful generator conditions features on one another, maintains consistent session-to-session behaviour for each user over time, and simulates cross-account coordination in multi-account campaigns.

For example, a user-session associated with a bot farm likely has an outlying velocity footprint, correlated intra-cluster activity cadence, potentially a high degree of device or IP sharing with other bot instances from the same source, and so on. Each of these correlations only makes sense in the context of the *other* sessions in the same cluster—sampling them independently is a category error. Concretely, the value of feature j on session s for user u at time t is not a free-standing random variable; it is conditioned on three axes at once:

$$x^{(j)}_{u,s,t} = f_j\Big(\underbrace{\text{pa}_j(x_{u,s,t})}_{\text{intra-session}},\ \underbrace{\mathbf{h}_{u,<t}}_{\text{trajectory}},\ \underbrace{\mathbf{c}_{C(u),t}}_{\text{campaign}}\Big) + \varepsilon^{(j)}_{u,s,t}, \tag{10}$$

where $\text{pa}_j(\cdot)$ is the set of causal parents of feature j *within* the same session (e.g. `total_tokens` depends on `n_requests`), $\mathbf{h}_{u,<t}$ is a sufficient statistic over user u's prior sessions (escalation, coverage saturation, baseline geo, etc.), $\mathbf{c}_{C(u),t}$ is the shared state of the campaign $C(u)$ that u belongs to (e.g. jailbreak success, coordinated burst tick, domain distillation progress), and $\varepsilon^{(j)}$ is irreducible per-session noise. Each axis is necessary: without the first, features are uncorrelated; without the second, accounts have no memory; without the third, a "bot farm" is just N independent accounts with no shared signature.

We implement this generator as a structural causal model (SCM) [23, 35].[1]

[1]The modern SCM formalism is largely due to Judea Pearl [36], crystallised in *Causality: Models, Reasoning, and Inference* [24] and the do-calculus that earned him the

Rather than sampling each column independently, an SCM represents features as nodes in a directed acyclic graph, with an edge from each cause to its effect, and generates each feature as a function of its parents plus exogenous noise. Sampling in topological order yields synthetic instances whose features co-vary as real ones do: a bot farm's session is not a set of independent draws but the downstream consequence of *being a bot farm* (low cadence variance, shared infrastructure, and so on). Because the structure is causal rather than merely correlational, the model also supports *interventions*: clamping a node to a value, severing its incoming edges, and asking, for example, "what would this session look like if it were not rate-limited?". Such counterfactual queries [16] are impossible with independent sampling and enable counterfactual analysis in future work. Our SCM is organised in three concentric layers, mirroring the three conditioning axes of Eq. (10). A full treatment of the SCM is beyond the scope of this paper; below we give an overview sufficient to interpret the dataset.

3.1 Case Study: A Coordinated Distillation Campaign

We illustrate the generative process end-to-end on a single distillation campaign, in which a coordinated group of actors (or a single operator controlling many automated agents) attempts to extract high-value data from the model. The exact mechanistic components of the SCM are abstracted, since they are one of many possible instantiations of the workflow described here. We further restrict attention to *epistemic* distillation, in which the attacker systematically tiles the model's knowledge across a domain space, accumulating coverage slot by slot. A structurally distinct variant, *orchestration* distillation, feeds the model elaborate multi-tool agentic tasks and captures the resulting action chains for imitation learning; it has a very different signature (deep tool chains, near-zero query entropy, potentially higher tool-use rate, and a preference for strongly agentic models).

The generative process has two components: (1) *definition and instantiation*, which sets the parameters of the simulated world and its actors; and (2) the *rollout*, which produces the tick-by-tick user-sessions of each account in the campaign.

3.1.1 Defining and instantiating the campaign universe. Before any account interacts with the model, the simulator sets up the world in which it will act. This step lays the groundwork for all three axes of Eq. (10) (the intra-session parents pa_j, the account trajectory $\mathbf{h}_{u,<t}$, and the shared campaign context $\mathbf{c}_{C(u),t}$) and is, at its core, a four-layer directed acyclic graph (Figure 4).

At the top are two root draws, the campaign type C and the actor type A, which fix the strategic context: *what* kind of attack is simulated and *who* runs it. From these we draw a *sophistication score* s, a scalar that governs nearly every downstream behavioural parameter. C, A and s are then realised into the *campaign details*, the once-per-campaign quantities such as fleet size, phase schedule and infrastructure pool, which in turn fan out into *per-account realisations*, the leaf draws unique to each account. Once instantiated, an account *acts* on *Five Elements Inc.*'s serving stack, and the platform

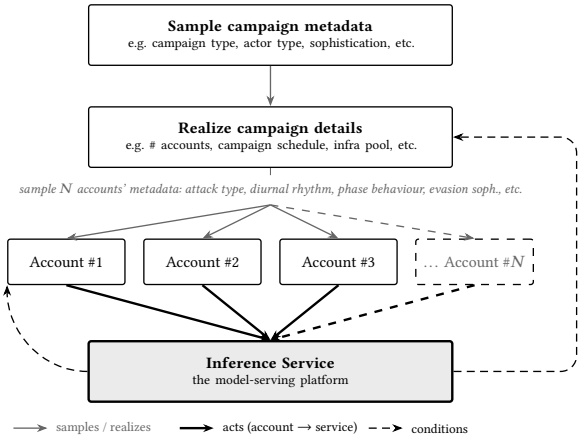


Figure 4: The SCM's generative pipeline. Campaign metadata is sampled, then realized into concrete campaign details, which fan out into N account realizations. Each account acts on the inference service; the service conditions both the accounts and the campaign in return. The "etc." marks a deliberately simplified slice—the full SCM samples far more per box.

Table 2: Campaign-type prior θ.

`distillation`	0.30	`data_extraction`	0.10
`harmful_use`	0.20	`dos`	0.07
`jailbreak`	0.15	`multi_surface`	0.05
`credential_abuse`	0.10	`bot_farm`	0.03

acts back: every rate limit, CAPTCHA, or monitoring flag conditions the account's future behaviour, and sophisticated campaigns can take evasive action to circumvent the platform's interventions. This act/condition loop is the only part of the process that is not a one-shot sample; we describe it tick by tick in the rollout below.

The description that follows is not exhaustive. The parameter values reflect our instantiation of the scenario, set from a combination of domain priors and published estimates, and should be read as illustrative rather than definitive.

Campaign type. $C \sim \mathrm{Cat}(\boldsymbol{\theta})$ draws which attack surface this campaign pursues. Distillation is the modal case at 30% (Table 2), reflecting a world where systematic knowledge extraction is a more common commercial motive than outright harmful use or disruption.

Actor type. $A \sim \mathrm{Cat}(\boldsymbol{\phi})$ identifies who is running the campaign, with $\boldsymbol{\phi} = (0.30, 0.25, 0.25, 0.10, 0.10)$ for (`criminal_org`, `competitor`, `script_kiddie`, `nation_state`, `insider`). Criminal organisations are the modal adversary; nation-states are rare but disproportionately capable.

Sophistication. $s \mid A, C$ is sampled last and is the most consequential of the three—it encodes how capable and disciplined this

2011 Turing Award. The underlying idea—distinguishing structural equations from regression equations—traces back further to Sewall Wright's path analysis (1921) and the Cowles Commission econometricians of the 1940s.

particular operator is:

$$s \mid A, C \sim \text{Beta}\left(\alpha_{A,C}, \beta_{A,C}\right)\Big|_{[s_{\min}, s_{\max}]}. \tag{11}$$

The Beta prior varies by actor type, reflecting our beliefs about each adversary class's capability floor and ceiling. For distillation campaigns:

$$s \mid A \sim \begin{cases} \text{Beta}(8,2) & [\mathbb{E}[s] = 0.80] \quad \texttt{nation_state} \\ \text{Beta}(5,3) & [\mathbb{E}[s] = 0.63] \quad \texttt{competitor} \\ \text{Beta}(4,4) & [\mathbb{E}[s] = 0.50] \quad \texttt{criminal_org} \\ \text{Beta}(3,3) & [\mathbb{E}[s] = 0.50] \quad \texttt{insider} \\ \text{Beta}(1,5) & [\mathbb{E}[s] = 0.17] \quad \texttt{script_kiddie} \end{cases} \tag{12}$$

For distillation campaigns specifically, s is then clamped to $[0.40, 0.95]$—even a script kiddie running a distillation operation needs to be minimally organised, and nobody operates with perfect omniscience. Sophistication propagates to nearly every downstream parameter, including infrastructure reuse, account-creation spread, TLS spoofing, evasion-response complexity, load-redistribution capability, and the replacement speed of burnt accounts.

Given these campaign-level parameters, we sample and realise a single distillation campaign; again, we present only a subset of its key states.

Number of accounts.

$$n = \max\left(1,\ \text{NegBin}(r{=}3, p{=}0.3) + 5\right), \qquad \mathbb{E}[n] \approx 12. \tag{13}$$

The negative binomial (not Poisson) is chosen for its over-dispersion: most operations deploy a modest cohort, but the heavy right tail allows occasional large fleets. Here r is the stopping parameter and p the per-trial success probability; the +5 offset enforces a floor of genuine coordination. This is an intrinsic property of the distillation type itself—distillation must tile a large domain space, so it is inherently multi-account—and is independent of A and s.

Phase schedule. A sampled, accumulated timeline from t_{start}:

$$\begin{aligned} \tau_{\texttt{setup}} &\sim U(1,5), & \tau_{\texttt{warmup}} &\sim \text{Exp}(\lambda^{-1}{=}7), \\ \tau_{\texttt{probing}} &\sim U(3,10), & \tau_{\texttt{cooldown}} &\sim U(1,5), \\ \tau_{\texttt{active}} &\sim \text{LogNormal}(3.5, 0.5) & &(\text{median} \approx 33\,\text{d}). \end{aligned} \tag{14}$$

Durations are sampled per phase and accumulated into absolute end-days. Crucially, the setup and warmup phases deliberately emit *benign cover traffic*—a distillation account looks like an ordinary casual user for its entire early life, and only reveals its template-driven, systematic behaviour from probing onward (Figure 5).

Termination. The campaign ends in *success* once aggregate domain coverage reaches 0.80, or *aborts* once the suspended fraction of its fleet reaches 0.70.

Account-creation spread.

$$t_{\text{create}} \sim \mathcal{N}\left(t_{\text{start}},\ \sigma = (1-s)\cdot 14\right), \qquad t_{\text{start}} \sim U(0,\ 0.85\,H). \tag{15}$$

Accounts are spawned clustered around the campaign's start day t_{start} (drawn over the first 85% of the horizon H so it has room to run). The spread narrows with sophistication: $s \to 1$ provisions the whole fleet within a tight window for a tighter campaign cadence—deliberately, often well ahead of activation during the benign setup/warmup cover phase—while $s \to 0$ scatters registrations across ∼2 weeks. This creates a genuine tension: a tight burst *raises* registration velocity (many sibling registrations in one window), so the sophisticated operator pairs the burst with low infra reuse (near-unique IPs per account) and hides it inside the cover-traffic window before any adversarial behaviour invites scrutiny.

Infra pool size and reuse.

$$n_{\text{IP}} \sim \text{Poi}(5),\ \ n_{\text{dev}} \sim \text{Poi}(8),\ \ n_{\text{email}} \sim \text{Poi}(10);\ \ \text{reuse} = 1 - s. \tag{16}$$

The campaign provisions a shared pool of IPs, device fingerprints, and email domains. The reuse factor $1-s$ governs how aggressively accounts share that pool: high $s \to$ low reuse $\to$ near-unique infra per account $\to$ a faint shared-IP / shared-device footprint (hard to catch); low $s \to$ heavy reuse $\to$ a dense, easily clustered footprint. Sophistication ($s > 0.6$) additionally unlocks TLS-fingerprint spoofing.

Load redistribution.

$$\text{mult} = \min\left(3.0,\ \frac{1}{1 - f_{\text{burn}}}\right), \qquad \text{applied only if } s > 0.5. \tag{17}$$

When accounts are suspended (burn fraction f_{burn}), only sophisticated campaigns ($s > 0.5$) compensate by redistributing the lost throughput across surviving siblings, scaling each survivor's activity by up to 3×. Unsophisticated campaigns simply absorb the loss and slow down.

Account realisations. From the campaign definition, individual accounts are spawned to populate its roster.[2] Each then draws its own realized values from the latent distributions laid out above—its personal sophistication, how many sessions it runs per day and how many requests within each session, the average inter-request time, its prompt perplexity and query-structure entropy, the epistemic domain coverage it targets both per session and in coordination with its siblings, its behaviour in each phase, its evasion-response complexity, its diurnal rhythm, and further latent variables.

Label arrival. Label arrival for each user-session is drawn from a log-normal delay distribution, to mimic realistic label discovery and annotation at *Five Elements Inc.*:

$$\Pr(\text{ever labeled}) = 0.10, \quad \text{delay} \sim \text{LogNormal}(\mu{=}4.5, \sigma{=}0.5), \tag{18}$$

i.e. a median delay of $e^{4.5} \approx 90$ days for distillation. Distillation sessions are hard to flag: they overlap substantially with benign user types such as the diligent autodidact, and no single indicator is decisive, so reliable identification requires accumulating and analysing historical sessions over many days. Even then, in our setup only 10% of such sessions are ever identified, with a median delay of 90 days.

Table 3 illustrates the overlap by comparing an active distillation session with a benign power user: on more than half of the listed features, the two profiles are difficult to distinguish.

[2] The `multi_surface` campaign is the most sophisticated in the taxonomy: its roster is a deliberate mixture of roles—e.g. jailbreaker (35%), distillation (30%), agentic misuse (20%), and cover traffic (15%)—and it is restricted to the nation-state sophistication tier ($s \in [0.70, 1.0]$). The variety is intentional: it simulates the realistic operational security of a well-funded adversary who cycles attack surfaces to frustrate pattern-matching defences.

Table 3: Configuration of an active distillation session versus a benign power user.

Feature	Distillation (active)	Benign power user	Separation
`sessions_per_day`	∼4/day	∼5/day	overlap
`n_requests_per_session`	∼33/session, heavy	∼25/session, heavy	overlap
`is_api`	95% API	25% API	**separable**
`avg_inter_request_time_sec`	∼3 s, regular	∼12 s, varied	partial
`std_inter_request_time_sec`	∼8 s jitter, robotic	∼45 s jitter, bursty	**key signal**
`avg_prompt_perplexity`	∼0.0, tight	∼0.2, looser	overlap
`std_prompt_perplexity`	∼0.15, template-driven	∼0.8, varied phrasing	**separable**
`query_structure_entropy`	∼0.25, narrow/repetitive	∼0.43, varied	**separable**
`avg_safety_score`	∼0.03, benign-looking	∼0.04, benign-looking	overlap
`domain_coverage_delta`	∼0.50, aggressive new coverage	∼0.20, narrow focus	**separable**
`total_tokens`	∼1.8k/session, high	∼3.0k/session, high	overlap

The table captures a single moment: C = `distillation`, ϕ_t = `active`, an account mid-campaign. In practice these distributions shift over time: the same account behaves very differently on the first day of warmup than forty days into active extraction, and differently again when rate-limited. Formally, for each account u on day t:

$$P(\mathbf{x}_{u,t} \mid \mathcal{Z}) = \prod_{j \in \text{topo}(\mathcal{G})} P\Big(x_{u,t}^{(j)} \,\Big|\, \text{pa}_j,\ \theta^{(j)}(\mathcal{Z})\Big), \tag{19}$$

$$\mathcal{Z} = \big(C,\ \phi_t,\ \mathbf{h}_{u,<t},\ \mathbf{c}_{C(u),t},\ \text{Interv}_{u,t}\big). \tag{20}$$

Here $\mathcal{Z}$ is a context bundle assembled each day from the campaign configuration, the account's accumulated history, the campaign's shared coordination state, and the interventions currently applied to the account. The per-node parameters $\theta^{(j)}$ start from a base that depends on (C, ϕ_t) *alone*, which is then passed through a three-stage modifier stack, each stage incorporating one further component of $\mathcal{Z}$ (written after the semicolon):

$$\theta^{(j)}(\mathcal{Z}) = \mathcal{T}_{\text{traj}}\Big(\mathcal{T}_{\text{camp}}\big(\mathcal{T}_{\text{interv}}\big(\theta^{(j)}_{\text{base}}(C, \phi_t);\ \text{Interv}_{u,t}\big);\ \mathbf{c}_{C(u),t}\big);\ \mathbf{h}_{u,<t}\Big). \tag{21}$$

The base carries (C, ϕ_t), and the three stages incorporate the remaining components of $\mathcal{Z}$ in turn: platform pressure ($\text{Interv}_{u,t}$), then campaign coordination ($\mathbf{c}_{C(u),t}$), then the account's own trajectory ($\mathbf{h}_{u,<t}$). Together they account for all of $\mathcal{Z}$. The state of an account on a given tick is thus a function of the base parameters fixed at instantiation, the platform's interventions, the directives imposed by the campaign, and the account's history up to that point.

3.1.2 The rollout. With the campaign instantiated, the simulator rolls it out tick by tick, emitting user-session records. The rollout is driven by the phase schedule (Figure 5).

Time advances in discrete daily ticks. On each tick, before any session is drawn, the campaign performs five bookkeeping steps in a fixed order, each reading the state written by the previous one.

(1) Decay. Interventions imposed earlier (a rate limit, a CAPTCHA gate, a period of enhanced monitoring) carry a countdown; each timer is decremented and expired interventions are lifted, allowing a penalised account to recover full throughput gradually.

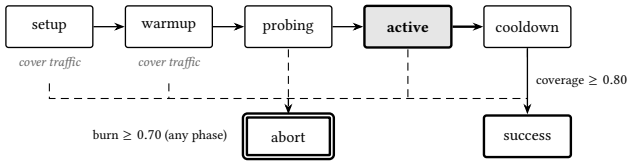


Figure 5: Distillation campaign lifecycle. Setup and warmup emit benign cover traffic, making the account indistinguishable from a casual user; the systematic extraction pattern only emerges from probing onward. Abort can fire from any phase once 70% of the fleet is burned.

(2) Fallout. Every account suspended since the last tick is marked burned and removed from the active roster, and the campaign checks its survival condition: if the cumulative burn fraction reaches 0.70, it aborts. Otherwise, sophisticated campaigns ($s > 0.5$) redistribute the load of burned accounts across the survivors per Eq. (17), holding total throughput roughly constant as the fleet shrinks; the multiplier is capped at 3× so that survivors do not reveal themselves through implausible activity spikes. Unsophisticated campaigns skip this step and simply lose throughput.

(3) Phase transition. The campaign consults its schedule and coverage progress to decide whether to advance, e.g. from probing to active extraction.

(4) Shared-state update. The campaign aggregates the domain coverage achieved by its accounts into a single progress measure:

$$\text{cov} = \frac{\sum_{u \in \text{active distill.}} \text{covered_slots}_u}{\text{total_domain_sectors}}. \tag{22}$$

This quantity is read by the next tick's success check (≥ 0.80) and also signals to each account how much of the domain remains uncovered, which shapes its activity.

(5) Activity allocation. The campaign issues a daily directive to each surviving account. For a distillation account in the active phase, the key field is the number of new domain slots to target that day:

$$\text{slots_today} = \min\big(\text{slots remaining in block},\ \text{Poi}(2) + 1\big). \tag{23}$$

Only after this bookkeeping do accounts interact with the model. For each active account, the rollout assembles a sampling context, which summarises everything the generative model needs about

Table 4: The three influences stitched into each day's sampling context.

Influence	Captures	What it carries for distillation
Intervention state	what the platform is currently doing to the account	whether it is rate-limited; its sophistication (which governs how it evades)
Campaign context	the cross-account coordination signals	today's slot allocation; remaining coverage, $1 - \text{cov}$
Trajectory	the account's own accumulated history	its personal saturation, covered/assigned $\in [0, 1]$

that account on that day, and then draws sessions one at a time. The context combines three sources, each carrying a different aspect of the campaign mechanics (Table 4).

Crucially, sessions within a day are not drawn independently: they are generated sequentially, and the account's cumulative statistics are updated between draws. After each session, the newly covered slots are recorded,

$$\text{covered_slots} \mathrel{+}= \max\left(1,\ \lfloor \delta \cdot \text{assigned_slots} \rfloor\right), \tag{24}$$

so the next session on the same day already sees a slightly more saturated trajectory. This ensures that an account does not cover the same territory twice and that its progress tapers naturally as its block fills. Because these records are exactly what the campaign aggregates at the start of the next tick, the loop is closed: each day's sessions determine the next day's progress measure and directives, and the campaign advances until it either covers 80% of the domain or is aborted (Figure 6).

3.2 The Dataset

The public dataset [18] is *Five Elements Inc.*'s raw service-usage log for its frontier models (model endpoints are named after the Claude family for simplicity). A stratified sample of rows, one per attack type, is given in Appendix A. Campaign types are distinct from the attack type of each individual account within a campaign. Several columns come in two variants (e.g. `noisy_avg_prompt_perplexity` and `avg_prompt_perplexity`): the former is the realistic noise-injected signal and the latter its oracle value. The dataset has three notable simplifications:

(1) **Fixed roles.** An adversarial account is locked to a single campaign and attack type for its entire life; no `harmful_use` account later turns to `distillation`, or vice versa. Real operators do pivot surfaces mid-campaign; we fix roles for tractability.
(2) **Cold start.** The fleet spawns directly into `setup` with no burn-in, so the first days of any run are a transient rather than a steady state. A more faithful simulation would run a longer horizon and discard the initial period; we handle this at training time instead (Section 4.2).
(3) **Label delay is a prior, not a mechanism.** Each surface draws its arrival lag from its own log-normal delay distribution (`dos` resolves in under a day, `distillation` has a ∼90-day median), so at the type level delay tracks how hard a surface is to catch. The medians were set by hand to encode the intuition, familiar to T&S teams, that conspicuous attacks resolve quickly while quiet, patient ones resolve slowly. There is, however, no account-level causality beneath this prior: two `distillation` accounts draw from the same distribution regardless of their `sophistication` or how convincingly either impersonated a power user.

These simplifications make the synthetic task easier than its real counterpart, but the dataset remains sufficient for the questions studied here.

4 Training the Detection Model

Recall from Section 2 that, given a session feature vector $\mathbf{x} \in \mathbb{R}^d$, we want to predict two targets: the binary $y_{\text{adv}} \in \{0, 1\}$ (benign vs. adversarial) and the multi-class $y_{\text{type}} \in C$ (which kind of adversarial account). Here C corresponds to `attack_type` (not `campaign_type`); that is, we detect the type of adversariality of a *given account*, not of the campaign *the account belongs to*. The former is the simpler task and a natural starting point; the dataset accordingly focuses on single-attack campaigns, with the exception of `multi_surface` campaigns, which are an ensemble of attack types. Since y_{type} is strictly more descriptive than y_{adv}, we focus on the multi-class case.

4.1 Labels and Observability

The raw data contains several label columns, such as the `campaign_type`, `campaign_label`, `noisy_campaign_label` triplet. These labels vary along **two independent axes**: their **purpose**, i.e. *what* a label describes, and their **observability**, i.e. how faithfully the channel that produced the label reflects the ground truth.

Axis 1, purpose. Three label families, nested from coarse to fine:

(1) `is_adversarial`, *the binary verdict*: 1 for an adversarial account of any kind, 0 for an honest one. This is the target of the binary task, but it carries no information beyond the type label: it is a collapse of `attack_type`, with everything outside the `benign.*` archetypes mapped to 1.
(2) `attack_type`, *the per-account type*: the role of the account in a campaign: `benign.{casual|power_user|developer|enterprise|researcher}`, or one of the adversarial types, plus `cover_traffic`. Each adversarial account serves exactly one vector (the fixed-role assumption), and this is the target of the multi-class model. During `warmup`/`setup` an adversarial account behaves as a benign archetype, since it has not yet begun misbehaving, and within a `multi_surface` ring an account can play a `cover_traffic` role that differs from the ring's actual objective.
(3) `campaign_type`, *the per-ring type*: the surface of the coordinated operation the account belongs to, absent for organic benign accounts. Because several accounts map to one campaign, this is a *group* property; it is out of scope for the per-session predictor but retained so that a higher-rung model can aggregate across accounts and recover the ring.

Axis 2, observability. Here synthetic data offers a unique advantage. In practice a label is never observed directly or perfectly, but through an evidentiary channel that is delayed, lossy, or wrong. We model three levels, M3, M2 and M1, in *decreasing* order of fidelity and, correspondingly, *increasing* order of realism:

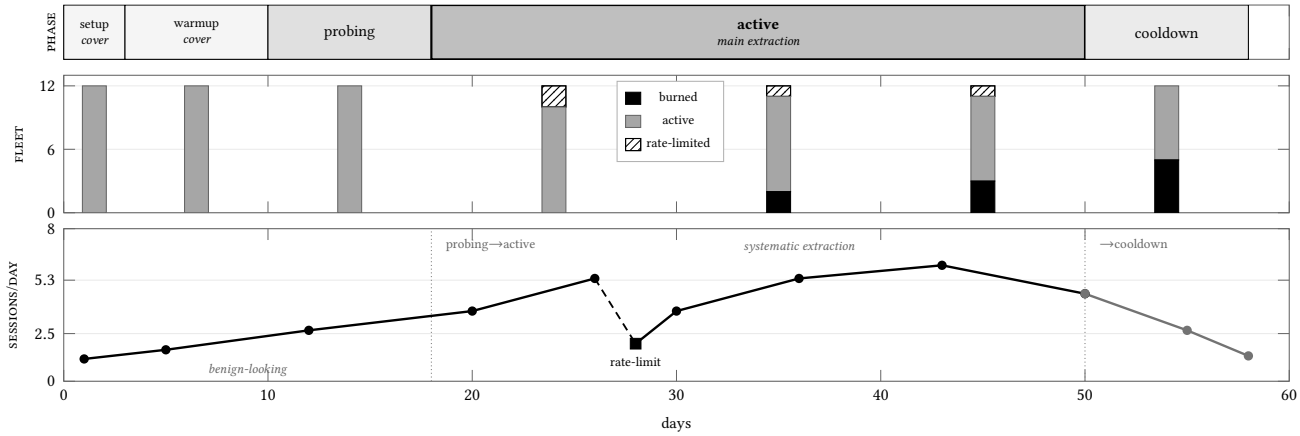


Figure 6: A 60-day distillation campaign rollout. Top: campaign phase bands. Middle: fleet composition per tick—active, rate-limited, and burned accounts. Bottom: a single account's session-density trace, with a rate-limit event mid-campaign and the characteristic ramp from a benign flatline to active extraction.

Table 5: The labelling scheme: purpose × observability.

Channel	Binary	Attack type *(per account)*	Campaign type *(per ring)*
M3 oracle	`is_adversarial`	`attack_type`	`campaign_type`
M2 golden	*(derived)*	`label`	`campaign_label`
M1 operational	*(derived)*	`noisy_label`	`noisy_campaign_label`

M3: no delay, no noise; M2: + investigation delay; M1: + delay + noise.

(1) **M3, oracle.** The ground truth known to the simulator by construction: `attack_type`, `campaign_type`, `is_adversarial`, the exact `attack_phase`, and the latent `sophistication`, available the instant a session is sampled, with no noise, delay, or ambiguity. M3 does not exist in a real T&S setting and is used for evaluation only.

(2) **M2, investigation-confirmed ("golden").** The latent fields are removed and only the *coarse* labels are kept, subject to a realistic **investigation delay** $\tau \sim \text{LogNormal}(\mu_d, \sigma_d)$, typically 14–90 days between a session and an analyst's confirmed verdict. M2 is given by the `label` / `campaign_label` columns. It corresponds to the golden dataset a mature T&S team accumulates over years: clean and trustworthy, but late. It is also used for evaluation only.

(3) **M1, operational.** The investigation channel is removed as well, leaving only the *policy outputs of the system itself* (whether an account was suspended) plus a label-noise model: false positives from over-eager policies, false negatives from attackers who exfiltrate and leave before any rule fires, misattribution, and the same delay. M1 is given by `noisy_label` / `noisy_campaign_label`. It is what a T&S team holds most of the time, and the only channel that scales without consuming analyst time. **We train on M1.**

Table 5 summarises the full labelling scheme.

The information-loss hierarchy. Each row is a *stochastic function of the row above it*: M2 coarsens and delays M3; M1 corrupts and delays M2. Having all three lets us evaluate the model in different settings. Evaluation against M3 gives the **ceiling**, the irreducible Bayes error once the world is fully observed. Evaluation against M2 indicates what a **well-resourced** T&S team could plausibly achieve. Evaluation against M1 indicates what can be **deployed today** with the labels actually available, and, by subtraction, how much performance is lost by not investing in a labelling pipeline. We report results against all three.

How the noise and delay are applied. Generating M1 by flipping labels i.i.d. per session would be unrealistic. Once a single session of an account is annotated, that verdict *recontextualises the entire account*: an analyst does not conclude that one session was a jailbreak and the next was benign, but that *the account* is malicious, and that judgement applies to its entire history. Per-session corruption would destroy exactly the cross-session consistency that a real labelling process imposes. We therefore corrupt labels with an **account-level confusion draw**: *one* draw per account, not per session, from a confusion distribution (hand-tuned per surface, in the same spirit as the log-normal delay), deciding whether and how that account's labels are corrupted: benign flagged as adversarial (FP), adversarial missed entirely (FN), or one attack type misread as another (misattribution). The outcome of the draw is applied *consistently* across every session of the account.

The counterpart of consistent noise is **consistent backfilling**. When an account is confirmed malicious, a T&S team does not only tag the session that triggered the alert; it reviews and relabels the account's history. *How far* back it relabels is not uniform, and handling this correctly keeps the dataset both realistic and free of temporal leakage. The deciding factor is whether an attack type is *always-adversarial*, malicious from its first session (e.g. `dos` and `distillation`), or can plausibly appear innocent early on (e.g. a `jailbreak` prober who initially resembles a curious researcher). The generator follows the rules in Table 6.

The purpose of this split is **temporal integrity**. An always-adversarial account was malicious *by construction* from its first

Table 6: Backfilling rules by account fate and attack type.

Account fate	Type	What gets labelled
Suspended / revoked	always-adv.	**All** sessions backfilled—intentional from day one
Suspended / revoked	not always-adv.	**Only** the session that triggered suspension—earlier ones were genuinely ambiguous
Active, label lands late	always-adv.	**All** sessions backfilled once the investigation retroactively confirms them
Active, label lands late	not always-adv.	Only sessions from `label_arrival_day` onward—preserving temporal honesty
Never confirmed	any	**Unlabelled**—the adversaries that simply slip through

day, so backfilling its entire history leaks nothing. A not-always-adversarial account, however, genuinely *was* ambiguous early on, so it is labelled only from the confirmation day forward; labelling it from day one would leak information from the future and inflate every offline metric.

4.2 Feature Engineering

The raw data is a session-level log with dozens of columns, oracle and noisy signals side by side, and benign and adversarial traffic interleaved. We transform it into a clean numeric matrix $\mathbf{X}$ suitable for gradient-boosted trees in four steps (Table 7).

Step 1: clip the horizon at both ends. Every account spawns directly into `setup` with no burn-in, so the opening ticks are a cold-start transient rather than a steady state. The tail has the opposite defect: for a session near the final day, the 14-to-90-day investigation lag has not yet elapsed, so its M1/M2 labels read `benign` only because the verdict has not arrived, a right-censoring artefact of the delay. We therefore discard a burn-in head and a burn-out tail and keep the steady-state band $n_{\text{burn_in}} \le \text{day} \le n_{\text{burn_out}}$, with $n_{\text{burn_in}} = 40$ and $n_{\text{burn_out}} = 320$. The head clip removes the warm-up transient; the tail clip acts as a *label-delay backstop*, preventing training on sessions whose `benign` label reflects only data immaturity.

Step 2: materialise the label grid. The M3→M2→M1 hierarchy becomes nine concrete columns, one triplet (`is_adversarial`, `attack_type`, `campaign_type`) per channel. The construction rule is uniform: a null in the source label means "no confirmed adversariality on this channel", so `is_adversarial` is an `is_not_null` cast to an integer, while the type strings get their nulls filled with the literal `benign`.

Step 3: compute per-account rolling history features. A single session carries little information in isolation; the discriminating signal lies in the *trajectory*. A bot farm's refusal count rises day over day, a distillation ring's domain coverage grows steadily, and a jailbreak prober's safety scores drift as it warms up. To expose this temporal structure to the trees without a recurrent model, we summarise each account's recent history over a trailing seven-day window,

$$W_{u,t}^{(7)} = \left\{ s : u_s = u, \;\; t - 7 < d_s \le t \right\}, \tag{25}$$

the set of sessions of account u in the seven days ending on (and inclusive of) day t. Each rolling feature is a simple aggregation over that window,

$$\text{mean}_{u,t}(x) = \frac{1}{|W_{u,t}^{(7)}|} \sum_{s \in W_{u,t}^{(7)}} x_s, \qquad \text{sum}_{u,t}(x) = \sum_{s \in W_{u,t}^{(7)}} x_s, \tag{26}$$

and likewise for running maxima and counts, applied to a handful of base signals: session volume, average request and token counts, the noisy safety / timing / coverage measurements, accumulated refusals, and intervention histories. The key constraint is $d_s \le t$: the window is *strictly trailing* and never includes a session from the future.

Step 4: select the allowed feature set. The exclusions here matter as much as the inclusions. We group the observable columns into tiers according to how much processing separates the raw event from the value:

(1) **Tier 1, fully observable.** Counts and facts the platform logs directly: `n_requests`, `session_duration_min`, `total_tokens`, `output_input_ratio`, inter-request timing mean/std, the `is_api` / `is_datacenter_ip` / `is_vpn` flags, and `n_refusals`.
(2) **Tier 2, model-derived.** Signals that are themselves outputs of an upstream model: prompt perplexity, safety scores, query-structure entropy, tool-use rate, and domain-coverage delta. We use the **noisy** variants, because in production the defender only sees the noised measurement.
(3) **Tier 3, derived / aggregated.** Cross-account graph signals such as `noisy_shared_ip_count`, `noisy_shared_device_count`, `registration_velocity`, and `concurrent_sessions`, again in their noisy variants.

The family of surface flags (`template_query_pattern`, `divergence_probe_pattern`, `token_flood_pattern`, and others) is excluded entirely: in our generator these are close to oracle signals and are unlikely to be available in practice without substantial preprocessing, although they remain useful for diagnostics. A real T&S team holds the *raw* perplexity and entropy signals and must *learn* the boundary that a surface flag hard-codes; we hold our model to the same standard.

4.3 Training Setup

After cleaning, the adversarial prevalence across the three splits is as given in Table 8.[3] The oracle channel M3 records the most adversarial activity and the operational M1 somewhat less, since much of it is missed or backfilled late; M2 is lowest of all, as its investigation channel confirms only a fraction of accounts, and late. Whichever channel is used, however, fewer than one session in sixty is adversarial: the dataset is highly imbalanced.

This degree of imbalance is familiar from fraud, payments, and intrusion detection. In the *binary* setting the standard remedies are well established: prevalence-sensitive metrics such as AUPRC rather than accuracy [20], class reweighting via XGBoost's `scale_pos_weight` [39], or losses that down-weight easy majority

[3] **Split protocol.** Rows are not split at random, which would let the same account (and, worse, the same *campaign*) span train, validation and test, leaking the answer. Instead we use a **time-population split**: we partition by account so that no user appears in two splits, *and* cut along the time axis so that training data lies strictly in the past relative to validation and test. Disjoint users imply disjoint campaigns.

Table 7: The feature-engineering pipeline at a glance.

Step	Transformation	What it does	Why
1	Clip horizon	Keep the burn-in/burn-out band ($n_{\text{burn_in}}$=40 to $n_{\text{burn_out}}$=320)	Drop the cold-start transient (head) and the censored tail whose labels have not arrived
2	Materialise label grid	Fold the `noisy_*` / `label` / `attack_type` channels into M1/M2/M3 × (`is_adversarial`, `attack_type`, `campaign_type`)	Turn the M3→M2→M1 hierarchy into concrete train/eval columns
3	Rolling features	Trailing 7-day per-account aggregates ($W_{u,t}^{(7)}$, strictly $d_s \le t$)	Expose behavioural *trajectory* to the model without leaking the future
4	Select allowed features	Tiers 1–3 (noisy) + geo + cumulative, plus the rolling block	Restrict to production-observable signals; **exclude oracle-like surface flags**

Table 8: Adversarial prevalence per split and label channel.

Split	M1 (operational)	M2 (golden)	M3 (oracle)
Train	1.53%	0.74%	2.77%
Val	1.35%	0.57%	2.61%
Test	1.47%	0.70%	3.22%

examples, such as focal loss [8]. All share the aim of preventing the abundant negative class from overwhelming the rare positive one.

Our task, however, is **multi-class**, so this aim must be generalised from two classes to nine. We fit an XGBoost [11] model with the `multi:softprob` objective, which outputs a full probability simplex $\hat{\mathbf{y}} \in \Delta^8$ over the nine classes, and use `mlogloss` (multi-class cross-entropy) for early stopping [39]. Since the scalar `scale_pos_weight` does not extend beyond two classes, we instead supply a **per-sample weight vector** that reshapes the loss so the model does not under-value the rare surfaces.

Weights are applied at training time only. Sample weights change the loss, and therefore the gradient, so that during fitting the optimiser cannot minimise its objective by ignoring the rare classes. At *evaluation* time the opposite is required: a calibrated measure of performance on data as it arrives in production, where the rare classes really are rare. The **training** stream therefore passes through a *weighted* iterator, while the **validation** and **test** streams pass through an *unweighted* iterator, so that `mlogloss` is estimated, and early stopping triggered, on the true distribution. We derive the weights in three steps.

Step 1: inverse-frequency weighting. Each class k receives a weight inversely proportional to its frequency, so that a class seen a thousand times less often is weighted a thousand times more heavily per example. With a smoothing exponent γ,

$$w_k = \left(\frac{N}{K \cdot n_k}\right)^{\gamma}, \qquad N = \textstyle\sum_k n_k, \quad K = 9, \tag{27}$$

where n_k is the count of class k. The weight is attached to each row by indexing this length-K vector with the row's label, w = `class_weights`$[y]$, a single gather that is cheap enough to perform on the fly inside the streaming iterator.

Step 2: square-root smoothing, $\gamma = \frac{1}{2}$. Setting $\gamma = 1$ recovers *raw* inverse frequency, which is unstable under our skew: a surface 10,000× rarer than benign would receive a 10,000× weight, allowing a handful of examples from a single rare class to dominate every gradient update and causing the loss to oscillate. The square root compresses this dynamic range, so the same 10,000× rarity yields a weight of roughly 100×. This trades a small amount of minority emphasis for substantially more stable training.

Table 9: Per-class weights under the three weighting stages, for an illustrative class distribution consistent with the ∼1.5% adversarial train prevalence. Spread = max/min weight.

Class	n_k	Inv. freq.	+ clamp	+ clamp + $\sqrt{\cdot}$
`benign`	985,000	0.11	158.7	12.6
`bot_farm`	4,100	27.1	27.1	5.2
`distillation`	3,200	34.7	34.7	5.9
`jailbreak`	2,600	42.7	42.7	6.5
`harmful_use`	1,400	79.4	79.4	8.9
`dos`	1,100	101.0	101.0	10.1
`agentic_misuse`	1,000	111.1	111.1	10.5
`credential_abuse`	900	123.5	123.5	11.1
`data_extraction`	700	158.7	158.7	12.6
Spread		1407×	5.9×	2.4×

Step 3: the benign clamp. Plain inverse frequency has an undesirable side effect: because benign is the *majority* class, it receives the *smallest* weight, and once every rare surface has been up-weighted, the benign class can be effectively removed from the gradient. A model trained this way no longer learns where the benign boundary lies and produces false positives on any unusual behaviour, which is costly when benign traffic exceeds 98% and every false positive adds friction for a paying customer. We therefore floor the benign weight at that of the heaviest non-benign class,

$$w_{\texttt{benign}} \leftarrow \max\left(w_{\texttt{benign}}, \max_{k \neq \texttt{benign}} w_k\right), \tag{28}$$

ensuring that the correction toward rare types never goes so far that the model loses track of *normal* behaviour.

Why not set $\gamma = 1$ and balance the classes *perfectly*? Full inverse frequency would weight each session of the rarest surface as heavily as over a thousand benign ones, and the training labels are the noisy, operational M1 channel, containing false positives, missed attacks, and misattributions by design. Fitting so aggressively on so few rows leads the model to *memorise* rather than *learn* the rare surface, including its sampling artefacts and mislabels, amplifying label noise. The $\sqrt{\cdot}$ smoothing and the clamp therefore serve a single purpose: not to *equalise* the classes, but to make the rare ones *learnable* without allowing a handful of noisy examples, or one over-weighted class, to dominate the fit.

Table 10: Binary `is_malicious` evaluation on the test split, by label channel.

Channel	Prev.	AUROC	AUPRC	P@1%FPR	R@1%FPR
M1 operational	1.47%	0.842	0.313	0.360	0.379
M2 golden	0.70%	0.994	0.609	0.360	0.801
M3 oracle	3.22%	0.997	0.993	0.767	0.993

Table 11: Multi-class attack-type evaluation under the naive argmax policy.

Channel	Top-1	Top-3	Macro-acc	Log-loss	Routing
M1 operational	0.988	0.997	0.194	0.067	0.989
M2 golden	0.997	1.000	0.244	0.021	0.997
M3 oracle	0.983	0.998	0.247	0.058	0.983

Table 9 compares the three stages on one fixed, imbalanced class distribution. **Plain inverse frequency** starves benign: its weight is near zero while the rarest surfaces receive very large weights, a spread of more than a thousandfold. **Adding the clamp** restores benign, raising its weight to match the heaviest surface. This is a heuristic design choice; it is intuitively motivated and performs well empirically, and we leave its theoretical grounding to future work. Finally, **adding $\sqrt{\cdot}$ smoothing** flattens the distribution further, so that no single surface dominates the gradient and training remains stable.

5 Results and Evaluation

We now evaluate whether the trained model is accurate and whether it is useful to the T&S team of *Five Elements Inc.*.

5.1 Binary Detection

Table 10 reports binary detection performance on the test split for each level of label observability. Against the oracle M3, the task is very nearly solved: an AUPRC of 0.99 and a recall of 0.99 at a 1% false-positive rate, meaning the model detects essentially every adversary while rarely flagging a legitimate user.[4] Against M1, the label type a real T&S team most commonly holds, the *same model* appears mediocre, with an AUPRC of 0.31. However, M1 is M3 observed through delayed, noisy, and partially missing verdicts, so evaluation against M1 measures the quality of the *labels* as much as that of the model. The model is, in effect, more accurate than the operational labels used to grade it, and practitioners evaluating against such labels risk substantially underestimating their detectors.

The gap between the M1, M2 and M3 results also quantifies the value of a golden dataset, costly as it is to build, and confirms that gradient-boosted trees remain a strong baseline for tabular detection tasks of this kind.

5.2 Attack-Type Attribution with a Naive Policy

Beyond detection, identifying the *attack type* is key to applying the correct intervention and routing each case to the appropriate team (Table 1). Table 11 gives multi-class results when the argmax of the class-probability vector is used as a naive decision policy.

The high Top-N metrics contrast sharply with the low macro-accuracy. Top-1 and routing accuracy of 0.98 appear strong, but the argmax, while free of tunable thresholds, is *prior-sensitive*: when benign accounts for 98% of traffic, the majority class wins every close call, and a model can achieve high top-1 accuracy by predicting `benign` almost uniformly. The prevalence-agnostic macro-accuracy is therefore the more meaningful measure, and it reveals weak performance, although the M1–M2–M3 observability gap remains visible.

To understand this, we break the aggregate down by class (Table 13), comparing each class's *ranking* with its *decision*. **PR-AUC** is computed one-vs-rest and measures whether the model ranks a class's true members above all others; **F1** is measured on the class predicted by the naive argmax policy. The classes fall into three groups.

Good ranking, compatible with argmax. For `benign` (F1 0.99), `dos` (0.62), `bot_farm` (0.48) and `credential_abuse` (0.45), ranking and decision agree: the model separates these surfaces well *and* the argmax selects them reasonably often. These classes have conspicuous, distinctive signatures and could be acted on immediately.

Good ranking, incompatible with argmax. For `distillation` (PR-AUC 0.79, argmax-F1 0.00) and `jailbreak` (PR-AUC 0.88, F1 0.01), the model ranks positives well, yet the argmax decision almost never selects them. The model separates these classes, but their probabilities cannot overcome the large benign prior to win the argmax.

Weak ranking. `harmful_use` (PR-AUC 0.25), `data_extraction` (0.14) and `agentic_misuse` (0.01) are poorly separated. Either the dataset contains too few samples of these classes, or the engineered features fail to capture their signal. These classes require more data or methods from higher rungs of the ladder, not a different decision rule. Per-class precision–recall curves are given in Figure 7.

5.3 Thresholded Decision Policies

We replace the argmax with per-class decision thresholds, since each attack type has a different prevalence and hence a different *sensitivity*. This raises a new question: when several classes clear their thresholds, which should be chosen? We address this with a simple decision engine with two levers, which together generate a family of policies.

The first lever is the selection rule among contenders: the largest probability (an argmax over a shortlist), or, among the classes that cleared their own threshold, the one with the widest margin, measured either in absolute terms (the *absolute lift*, $p_k - t_k$) or relative to the threshold (the *relative lift*, p_k/t_k, which is fairer when thresholds differ greatly in magnitude). The second lever is the treatment of `benign`: it either competes like any other class or, as the overwhelming default, is excluded unless it clears a markedly higher **floor**, so that suspect sessions receive closer scrutiny.

As the best configuration is not known a priori, we evaluate six policies in addition to the naive one. All thresholds are tuned on the validation set using the noisy M1 labels. Let $p = (p_0, \ldots, p_8)$ be the softmax vector over C, t_k the tuned threshold for class k, and $b = 0$ the index of `benign`. All policies depend on the **contender**

[4] The synthetic setting is cleaner than a real one, so the M3 results should be read as an optimistic ceiling.

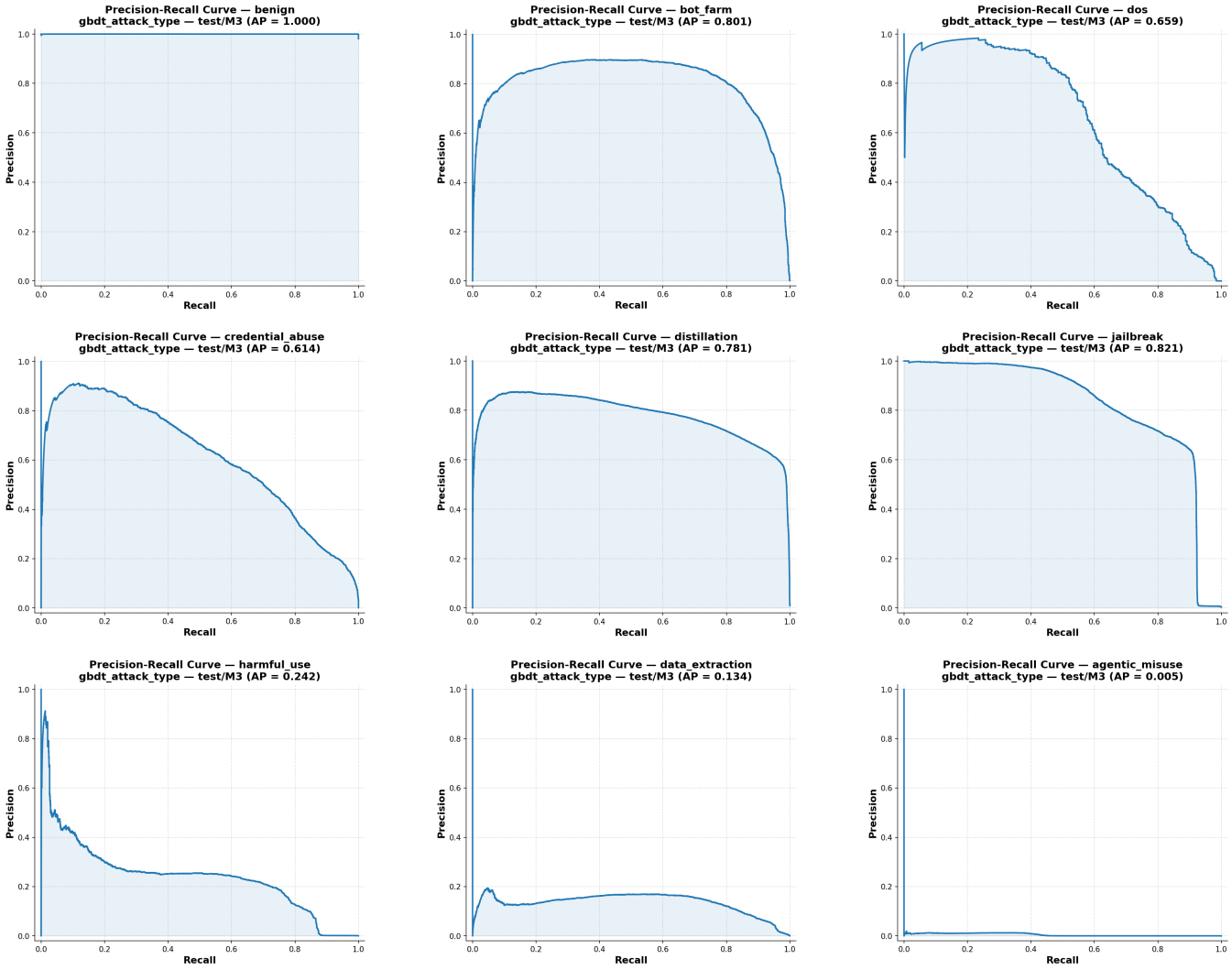


Figure 7: Per-class precision–recall curves for the attack-type GBDT (test split, M3), ordered roughly best to worst. A curve that bows toward the top-right corner is separable; note how `distillation` and `jailbreak` bow high even though their argmax F1 sits at zero.

set, the classes that clear their own threshold:

$$\mathcal{K}(p) = \{\, k \in C : p_k \geq t_k \,\}. \tag{29}$$

Whenever a policy is left with an empty contender set, it abstains to `benign`.

1. Absolute lift. Every class competes, and the winner is the class that exceeds its own threshold by the widest absolute margin: $\hat{y} = \arg\max_{k \in \mathcal{K}(p)} (p_k - t_k)$. Because the margin is measured in probability mass, a class with a high threshold needs a correspondingly high probability to win, which implicitly favours classes the model is already confident about.

2. Relative lift. As above, but the margin is scaled by the threshold, so clearing a low threshold twice over beats clearing a high one narrowly: $\hat{y} = \arg\max_{k \in \mathcal{K}(p)} p_k / t_k$. This comparison is fairer when thresholds differ widely, and gives the rare classes, whose thresholds are necessarily low, a realistic chance of selection.

3. Benign floor → argmax. This policy uses the second lever: `benign` is no longer an ordinary contender and must clear a high floor $\tau_b = 0.65$. Otherwise it is excluded, and the argmax is taken over the remaining classes:

$$\hat{y} = \begin{cases} b, & p_b \geq \tau_b, \\ \arg\max_{k \in C \setminus \{b\}} p_k, & \text{otherwise.} \end{cases} \tag{30}$$

This encodes an explicit prior in the form of a *hard decision boundary* that shifts decisions away from the majority class.

4. Benign floor → absolute lift. As policy 3, except that once `benign` is excluded the remaining classes are ranked by absolute

Table 12: Decision engines on the test split (M3); thresholds tuned on validation (M1).

Decision rule	Acc.	Mac.-F1	Mac.-rec.	Routing	ΔMac.-F1
argmax (baseline)	0.983	0.295	0.247	0.983	—
absolute lift	0.984	0.324	0.267	0.984	+0.029
relative lift	0.987	0.422	0.407	0.989	+0.127
benign floor → argmax	0.984	0.358	0.316	0.985	+0.063
benign floor → abs. lift	0.984	0.321	0.271	0.984	+0.026
benign floor → rel. lift	**0.992**	**0.489**	**0.520**	**0.996**	+0.194
meta-stacker	0.988	0.435	0.445	0.991	+0.140

margin, $\arg\max_{k \in \mathcal{K}(p) \setminus \{b\}} (p_k - t_k)$, with the floor tuned to $\tau_b = 0.60$.

5. Benign floor → relative lift. The most aggressive configuration: `benign` must be near-certain ($\tau_b = 0.95$) to be selected, and the remaining contenders are ranked by relative margin, $\arg\max_{k \in \mathcal{K}(p) \setminus \{b\}} p_k / t_k$. Both levers are at their extremes: the majority class is held to a strict standard, while the minority classes are compared on terms that do not penalise their rarity.

6. Meta-stacker. The only learned policy: a smaller meta-model, here a simpler XGBoost instance, learns a higher-order decision boundary, $\hat{y} = \arg\max_{k \in C} g_k(p,\ p - t)$. It recovers most of what the tuned rules find, but it is a second model to train, calibrate, and maintain, which matters when the rarest classes have only a few dozen validation examples, as here.

Table 12 summarises the results. The strongest policy, **#5, benign floor → relative lift**, raises macro-F1 from 0.295 to 0.489. Improving recall on rare classes would ordinarily be expected to cost precision on the majority class, yet accuracy and routing accuracy also improve. This indicates that the naive argmax was misclassifying a substantial number of rare attacks *as* benign, consistent with the *positive* Δmacro-F1 of every policy.

Per class (Table 13), key attack vectors improve substantially: `distillation` F1 rises from 0.00 to 0.65 and `jailbreak` from 0.006 to 0.83. The per-class results also reveal trade-offs:

(1) The best overall policy, **#5**, lowers `dos` F1 from 0.616 to 0.513, and `credential_abuse` and `bot_farm` performance also declines.
(2) **#3, benign floor → argmax**, shows the opposite trade-off. It improves the middle group (`bot_farm` 0.475 → 0.695, `credential_abuse` 0.451 → 0.608) but barely changes the well-ranked yet rarely selected classes such as `distillation`.
(3) No policy substantially improves `agentic_misuse` or `data_extraction`, whose best F1 scores are 0.042 and 0.244 respectively.
(4) `harmful_use` is an exception: despite a middling ranking (PR-AUC 0.25), it reaches an F1 of 0.47 under the best policy.
(5) **#6, the meta-stacker**, performs well, with a macro-F1 of 0.44 and the most consistent performance across the nine classes. It may be preferable where labels are more abundant and the added complexity of a meta-model is acceptable.

No single policy is best for every class; each represents a different trade-off between the well-separated classes and the difficult ones.

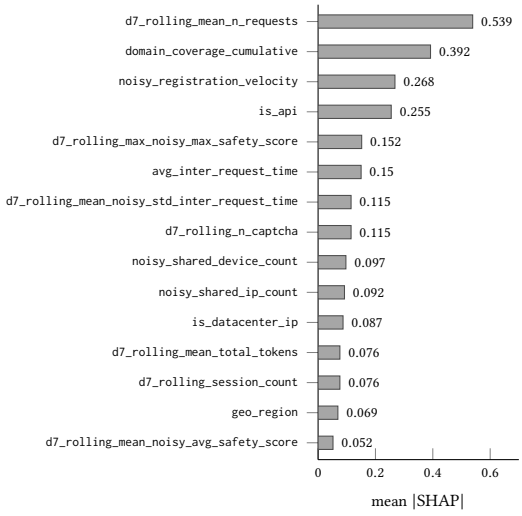


Figure 8: Global feature importance for the attack-type GBDT: mean |SHAP| over a 200k-row test sample.

5.4 Feature Importance

To check which features the model relies on, and whether its behaviour is plausible, we compute SHAP values [19] (Figure 8). The most important feature is request *volume* (the seven-day mean request count), followed by cumulative *domain coverage*, a natural signal for distillation rings that systematically harvest the model's knowledge, consistent with the mechanism encoded in the SCM. Next is a cluster of *infrastructure-graph* signals (registration velocity, shared devices and IPs, and datacenter and API flags), characteristic indicators of coordinated fake-account fleets. Timing cadence and peak safety scores complete the top of the list. The model thus recovers heuristics and priors that were built into the SCM; the extent to which this reflects circularity between generator and detector is a question we leave open.

6 Conclusion

We studied inference-layer defence for a hypothetical LLM provider, *Five Elements Inc.*, starting at the second rung of the Ladder of Abstraction: a per-session point predictor, trained on a synthetic dataset from a structural causal model that simulates coordinated campaigns, platform feedback, and three tiers of label observability. Two findings stand out. First, the labels used for training and evaluation matter as much as the model: the same detector is near-perfect against oracle labels and mediocre against operational ones, and the gap directly measures the value of a labelling pipeline. Second, a model that ranks rare attacks well can still fail to *select* them under an overwhelming benign prior; a simple thresholded decision engine recovers much of this lost signal.

Limitations. Our results are obtained on synthetic data whose generator fixes account roles, models label delay as a per-type prior rather than an account-level mechanism, and encodes the same heuristics that the detector later recovers; real traffic is likely to be harder. The weighting and decision rules are heuristic and tuned on a single dataset. Finally, the remaining failures, `agentic_misuse`

Table 13: Per-class performance on the test split (M3): one-vs-rest PR-AUC (ranking) and F1 under each decision engine. Grouped by whether ranking and decision agree. Best F1 per class in bold.

		F1 by decision engine						
Class	**PR-AUC**	argmax	abs. lift	rel. lift	floor→argmax	floor→abs.	floor→rel.	meta
Good ranking, compatible with argmax								
benign	1.000	0.992	0.992	0.995	0.993	0.992	**1.000**	0.996
dos	0.661	**0.616**	0.592	0.561	0.571	0.585	0.513	0.381
bot_farm	0.801	0.475	0.641	0.444	**0.695**	0.667	0.379	0.634
credential_abuse	0.615	0.451	0.370	0.289	**0.608**	0.426	0.337	0.509
Good ranking, incompatible with argmax								
jailbreak	0.883	0.006	0.119	0.826	0.140	0.071	**0.830**	0.700
distillation	0.794	0.000	0.005	0.018	0.022	0.009	**0.653**	0.271
Weak ranking								
harmful_use	0.249	0.040	0.100	0.400	0.095	0.071	**0.466**	0.332
data_extraction	0.135	0.073	0.085	**0.244**	0.093	0.063	0.184	0.080
agentic_misuse	0.006	0.000	0.010	0.022	0.008	0.002	**0.042**	0.009

and data_extraction, call for more data or for the higher rungs of the ladder, which we leave to future work.

Data Availability

The full dataset, including the raw user-session log with all three label channels and the engineered feature matrix, is publicly available on Hugging Face [18] at https://huggingface.co/datasets/DaoistDurian/lang-adversarial-inference-01.

References

[1] Acronis. [n. d.]. *What is a Zero-Day Attack and How Can You Defend Against One?* https://www.acronis.com/en/blog/posts/what-is-a-zero-day-exploit/

[2] AI Learned Today. [n. d.]. *Detecting and Preventing Distillation Attacks.* https://ailearnedtoday.com/ref/anthropic-distillation-attacks/

[3] Anthropic. [n. d.]. *Newsroom.* https://www.anthropic.com/news

[4] Anthropic. 2025. *Detecting and Countering Misuse of AI: August 2025.* https://www.anthropic.com/news/detecting-countering-misuse-aug-2025

[5] Anthropic. 2025. *Disrupting the First Reported AI-Orchestrated Cyber Espionage Campaign.* https://www.anthropic.com/news/disrupting-AI-espionage

[6] Anthropic. 2025. *A Small Number of Samples Can Poison LLMs of Any Size.* https://www.anthropic.com/research/small-samples-poison

[7] Anthropic. 2026. *Detecting and Preventing Distillation Attacks.* https://www.anthropic.com/news/detecting-and-preventing-distillation-attacks

[8] Aman Arora. 2020. *What is Focal Loss and When Should You Use It?* https://amaarora.github.io/posts/2020-06-29-FocalLoss.html

[9] Joel Becker, Nate Rush, Elizabeth Barnes, and David Rein. 2025. Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity. arXiv:2507.09089 https://arxiv.org/abs/2507.09089

[10] BleepingComputer. [n. d.]. *Google Says 90 Zero-Days Were Exploited in Attacks Last Year.* https://www.bleepingcomputer.com/news/security/google-says-90-zero-days-were-exploited-in-attacks-last-year/

[11] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 785–794.

[12] Cloudflare. [n. d.]. *What is AI Security?* https://www.cloudflare.com/learning/ai/what-is-ai-security/

[13] HawkEye. 2026. *How Hackers Used Anthropic's Claude to Breach the Mexican Government.* https://hawk-eye.io/2026/02/how-hackers-used-anthropics-claude-to-breach-the-mexican-government/

[14] Help Net Security. 2026. *Google Researchers Uncover Criminal Zero-Day Exploit Likely Built with AI.* https://www.helpnetsecurity.com/2026/05/11/google-ai-vulnerability-exploitation/

[15] Help Net Security. 2026. *Week in Review: High-Severity LPE Vulnerability in the Linux Kernel, cPanel 0-Day Exploited for Months.* https://www.helpnetsecurity.com/2026/05/03/week-in-review-high-severity-lpe-vulnerability-in-the-linux-kernel-cpanel-0-day-exploited-for-months/

[16] Ferenc Huszár. [n. d.]. *Causal Inference 3: Counterfactuals.* https://www.inference.vc/causal-inference-3-counterfactuals/

[17] InfoWorld. [n. d.]. *Are We Ready to Give AI Agents the Keys to the Cloud? Cloudflare Thinks So.* https://www.infoworld.com/article/4165857/are-we-ready-to-give-ai-agents-the-keys-to-the-cloud-cloudflare-thinks-so.html

[18] Keifer Lee. 2026. *lang-adversarial-inference-01: A Synthetic Dataset of Adversarial LLM Usage.* https://huggingface.co/datasets/DaoistDurian/lang-adversarial-inference-01 Hugging Face dataset.

[19] Scott M. Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*. https://arxiv.org/abs/1705.07874

[20] MachineLearningMastery.com. [n. d.]. *ROC AUC vs Precision-Recall for Imbalanced Data.* https://machinelearningmastery.com/roc-auc-vs-precision-recall-for-imbalanced-data/

[21] MEXC News. [n. d.]. *Brian Armstrong Wants Half of Coinbase Code To Be AI-Written By October.* https://www.mexc.com/news/85424

[22] OpenAI. 2025. *Findings from a Pilot Anthropic–OpenAI Alignment Evaluation Exercise: OpenAI Safety Tests.* https://openai.com/index/openai-anthropic-safety-evaluation/

[23] Judea Pearl. 2009. Causal Inference in Statistics: An Overview. *Statistics Surveys* 3 (2009), 96–146. https://ftp.cs.ucla.edu/pub/stat_ser/r350.pdf

[24] Judea Pearl. 2009. *Causality: Models, Reasoning, and Inference* (2nd ed.). Cambridge University Press. https://archive.illc.uva.nl/cil/uploaded_files/inlineitem/Pearl_2009_Causality.pdf

[25] Pillar Security. [n. d.]. *Operation Bizarre Bazaar.* https://www.pillar.security/resources/operation-bizarre-bazaar

[26] PYMNTS.com. 2026. *Vibe Coding Breaks Into Banking Before Regulators Can React.* https://www.pymnts.com/artificial-intelligence-2/2026/vibe-coding-breaks-into-banking-before-regulators-can-react/

[27] Security Affairs. [n. d.]. *U.S. CISA Adds a Flaw in Google Dawn to its Known Exploited Vulnerabilities Catalog.* https://securityaffairs.com/190282/security/u-s-cisa-adds-a-flaw-in-google-dawn-to-its-known-exploited-vulnerabilities-catalog.html

[28] SecurityWeek. 2023. *Bad Bots Account for 73% of Internet Traffic: Analysis.* https://www.securityweek.com/bad-bots-account-for-73-of-internet-traffic-analysis/

[29] SecurityWeek. 2026. *Hackers Weaponize Claude Code in Mexican Government Cyberattack.* https://www.securityweek.com/hackers-weaponize-claude-code-in-mexican-government-cyberattack/

[30] SecurityWeek. 2026. *LLMs Hijacked, Monetized in 'Operation Bizarre Bazaar'.* https://www.securityweek.com/llms-hijacked-monetized-in-operation-bizarre-bazaar/

[31] Statista. 2024. *Human and Bot Web Traffic Share 2024.* https://www.statista.com/statistics/1264226/human-and-bot-web-traffic-share/

[32] The Alan Turing Institute. 2025. *LLMs May Be More Vulnerable to Data Poisoning Than We Thought.* https://www.turing.ac.uk/blog/llms-may-be-more-vulnerable-data-poisoning-we-thought

[33] The Deep Dive. [n. d.]. *Coinbase Trading Platform Goes Down Three Days After CEO Bragged About AI Writing Production Code.* https://thedeepdive.ca/coinbase-trading-platform-goes-down-three-days-after-ceo-bragged-about-ai-writing-production-code/

[34] Yuxin Wang, Yuhan Chen, Zeyu Li, Xueze Kang, Yuchu Fang, Yeju Zhou, Yang Zheng, Zhenheng Tang, Xin He, Rui Guo, Xin Wang, Qiang Wang, Amelie Chi Zhou, and Xiaowen Chu. 2024. BurstGPT: A Real-World Workload Dataset to Optimize LLM Serving Systems. arXiv:2401.17644 https://arxiv.org/abs/2401.17644

[35] Wikipedia. 2026. *Causal Model.* Retrieved September 2026 from https://en.wikipedia.org/wiki/Causal_model

[36] Wikipedia. 2026. *Judea Pearl.* Retrieved September 2026 from https://en.wikipedia.org/wiki/Judea_Pearl

[37] Wikipedia. 2026. *Stackelberg Competition.* Retrieved September 2026 from https://en.wikipedia.org/wiki/Stackelberg_competition

[38] Wikipedia. 2026. *Zero-Day Vulnerability.* Retrieved September 2026 from https://en.wikipedia.org/wiki/Zero-day_vulnerability

[39] XGBoost Developers. [n. d.]. *XGBoost Parameters.* https://xgboost.readthedocs.io/en/stable/parameter.html

[40] Yuxing Xiang, Xue Li, Kun Qian, Yan Zhang, Wenyuan Yu, Ennan Zhai, Xin Jin, and Jingren Zhou. 2026. ServeGen: Workload Characterization and Generation of Large Language Model Serving in Production. In *Proceedings of the 23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI '26).* https://www.usenix.org/conference/nsdi26/presentation/xiang-servegen

A Dataset Sample

Table 14 shows one raw user-session record per attack type, with a subset of columns. `label` is the M2 channel; "—" marks a session that is unlabelled on that channel (for `benign` accounts, the attack type itself is empty). The full dataset, including the engineered feature matrix with the materialised M1/M2/M3 label triplets and the trailing seven-day rolling features, is available on Hugging Face [18].

Table 14: A stratified sample of raw user-session records (oracle M3 fields included). IRT = inter-request time (s); QSE = query-structure entropy; cov. Δ = domain coverage delta.

Attack type	Phase	Day	Req.	API	Avg IRT	Std IRT	QSE	Safety	Cov. Δ	Soph.	Label (M2)
benign	dormant	9	6	–	22.74	14.94	0.214	0.001	0.000	0.00	—
bot_farm	active	104	17	✓	2.15	0.67	0.069	0.005	0.000	0.12	bot_farm
dos	active	1	382	✓	0.49	0.43	0.014	0.035	0.039	0.60	dos
credential_abuse	exploitation	36	26	✓	8.37	2.92	0.747	0.079	0.000	0.65	credential_abuse
distillation	active	27	28	✓	3.84	15.53	0.069	0.002	0.000	0.40	—
jailbreak	cooldown	59	5	–	30.10	11.30	0.693	0.028	0.000	0.84	jailbreak
harmful_use	active	22	9	–	3.60	10.95	0.067	0.478	0.000	0.34	—
data_extraction	active	19	23	✓	12.69	2.99	0.012	0.106	0.000	0.88	—
agentic_misuse	probing	57	18	✓	3.73	0.62	0.382	0.040	0.000	0.70	—
cover_traffic	probing	93	7	–	50.06	24.29	0.393	0.002	0.000	0.70	—