



Article

Phase-Adaptive Model Routing in LLM-Driven SSH Honeypots: Balancing Response Fidelity and Latency Across the Attack Lifecycle

Raiymbek Magazov ^{1,2}, Fatima Uralova ^{1,*}, Kuanysh Abeshev ³, Guldana Akhmedi ¹ and Gulnur Aksholak ²

¹ Department of Cybersecurity, Institute of Automation and Information Technology, Satbayev University, Almaty 050013, Kazakhstan; g.akhmedi@satbayev.university (G.A.)

² Department of Cybersecurity and Cryptology, Faculty of Information Technology and Artificial Intelligence, Al-Farabi Kazakh National University, Almaty 050040, Kazakhstan; gaksholak@gmail.com

³ School of Digital Technologies, Almaty Management University, Almaty 050060, Kazakhstan; k.abeshev@almu.edu.kz

* Correspondence: f.uralova@satbayev.university

Abstract

Secure Shell (SSH) intrusions against Linux servers remain a dominant vector of opportunistic and targeted cyber incidents, yet operational honeypots treat attacker commands either as isolated lookup keys for static templates or as input to a single uniform language model. Two research directions partially address this gap: LLM-driven response generation raises interaction realism but incurs a fidelity–latency trade-off, while semantic command analytics classifies attack stages from embeddings yet applies a single fine-tuned model uniformly across sessions. This work introduces the phase-adaptive model routing framework (PAMR), which explicitly couples both views. A lightweight online phase estimator infers the current attack stage—reconnaissance, exploitation, or persistence—from a sliding window of recent commands via compact semantic embeddings; a router then dispatches each command to one of several heterogeneous LLM backends according to estimated phase, command complexity, and a confidence-weighted latency budget. The routing decision is formulated as a constrained optimization problem and realized at runtime as an $O(1)$ lookup in a precomputed dispatch table. PAMR is evaluated on a controlled, reproducible benchmark of 412 stage-annotated command sessions that combines representative Linux command–response pairs with synthesized attacker traces; we explicitly state that this is a laboratory benchmark rather than live attacker traffic, and we scope our claims accordingly. Relative to uniform-model baselines, PAMR reduces mean response latency by approximately 38% against an API-hosted high-capacity backend and by approximately 45% against a locally hosted mid-capacity backend, while keeping token-level response-fidelity metrics close to the high-capacity baseline (cosine similarity ≈ 0.39 vs. 0.40) and maintaining an online stage-classification macro-F1 above 0.87. We further provide a first-order analytical treatment of the timing side-channel that any backend-routing architecture introduces, and we frame the contribution as a latency/token-fidelity trade-off, leaving validation of operational realism against live adversaries to future work.

Keywords: SSH honeypot; large language models; model routing; attack lifecycle; semantic embeddings; cyber deception; command pattern recognition; threat intelligence



Academic Editors: Ajay Bandi and Zhengrui Qin

Received: 11 June 2026

Revised: 2 July 2026

Accepted: 9 July 2026

Published: 14 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article

distributed under the terms and

conditions of the [Creative Commons](https://creativecommons.org/licenses/by/4.0/)

[Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Linux servers exposed to the public Internet through Secure Shell (SSH) services continue to absorb a disproportionate share of opportunistic and semi-targeted intrusion attempts. The defining property of such intrusions is that they are command-driven: an adversary, once authenticated, operates directly within a shell and produces a temporally ordered sequence of commands whose collective semantics reveal intent, capability and progress through the kill chain. Defensive value extracted from a honeypot deployment therefore depends not only on what individual commands are captured, but also on whether the deceptive environment remains coherent long enough for later, more diagnostic phases of the intrusion to unfold.

Two complementary advances have recently reshaped the design space of SSH honeypots. First, replacing the static command–response dictionaries of traditional medium-interaction honeypots with on-the-fly generation from a pre-trained large language model has been shown to substantially extend the range of inputs that the system can plausibly answer [1]. Second, embeddings produced by LLMs have been used to encode attacker command streams into semantic vectors that support stage-level classification with markedly higher accuracy than n-gram or token frequency baselines [2]. These contributions resolve, respectively, the realism problem and the analytics problem—but they do so independently, using a single, fixed model configuration for the entire interaction.

This single-configuration assumption is increasingly difficult to justify in practice. Empirical observations reported in [1] indicate that response latency dominates the perceived realism of the honeypot and that latency scales sharply with model capacity. At the same time, the behavioural analysis presented in [2] makes it clear that not all commands carry equal diagnostic weight: enumeration commands at the reconnaissance stage are linguistically simple, highly repetitive and tolerant of brief, formulaic outputs, whereas exploitation-stage interactions involve elaborate command chains for which an inconsistent or hallucinated response is much more likely to trigger early session abandonment. A model that is appropriate for one phase is therefore systematically inappropriate for another.

The central question motivating this study is whether the latency–fidelity trade-off exposed by LLM-driven honeypots can be relaxed by allowing the backend to vary as a function of the inferred attack phase. To answer this, we formulate phase-adaptive model routing as a constrained optimization problem over a heterogeneous pool of LLM backends, develop a lightweight online phase estimator that operates on rolling command windows, and design a router that selects per-command backends so as to minimize expected latency subject to a per-phase fidelity floor.

The contributions of this paper are threefold. The first contribution is a formalization of phase-adaptive routing for LLM-driven honeypots, including an explicit objective function, the confidence-weighted decision rule, and the assumptions under which it admits a tractable online solution. The second contribution is an architectural design, named PAMR, that decouples the response generation pool from the phase estimator and from the command dispatcher, making each component independently replaceable. The third contribution is a controlled benchmark—its construction and stage-annotation protocol documented in full and scheduled for public release (Data Availability Statement)—that combines the command-level evaluation protocol of [1] with the stage-level annotation scheme of [2] and is used to compare PAMR against uniform-model baselines on identical inputs. We are deliberately precise about the scope of the resulting evidence: all results are obtained on a curated laboratory benchmark, the reported fidelity figures are token-level proxies rather than direct measures of attacker-perceived realism, and validation against live adversaries is identified as the principal item of future work.

The remainder of the paper is organized as follows. Section 2 reviews prior work on dynamic honeypots, semantic command analytics, and adaptive inference for cost-sensitive deployments and positions the present study relative to those lines. Section 3 introduces the system model, the routing problem, the architecture of PAMR and the experimental protocol. Section 4 reports and discusses the experimental results. Section 5 summarizes the findings and outlines directions for further work.

2. Literature Review

The deception literature has accumulated more than two decades of work on honeypot architectures, and the recent emergence of generative language models has reopened several questions that the community had previously considered closed. Three threads are particularly relevant to the present study: the evolution of SSH honeypot realism, the use of LLMs as response generators or feature extractors in deceptive systems, and the broader field of adaptive inference under heterogeneous compute budgets.

2.1. Evolution of SSH Honeypot Realism

Early SSH honeypots emulated a Unix-like shell through a fixed catalogue of command–output pairs and a synthetic file system [3]. This design proved adequate for capturing scripted credential-stuffing attempts and basic post-authentication probes, but it remained brittle in the face of moderately skilled operators who could fingerprint the deception with a small number of probing commands [4]. Subsequent work attempted to widen the supported command space through deeper system call emulation and increased session state retention, at the cost of higher operational risk and considerable maintenance effort. Surveys of the field consistently identify the realism-versus-safety dilemma as the most persistent design tension in the area and observe that medium-interaction honeypots have remained dominant in practice precisely because they sit at an acceptable point on this trade-off curve [5–7]. PAMR belongs to this medium-interaction class: it presents a convincing interactive shell but never executes attacker commands on a real host, inheriting the safety profile of medium-interaction designs while raising their response realism through LLM generation.

2.2. Language Models as Response Generators

A first wave of work uses pre-trained or fine-tuned language models to replace, in whole or in part, the static response catalogue of medium-interaction honeypots [8,9]. Malhotra introduced LLMHoney [1], a real-time SSH honeypot that combines a dictionary-based cache for frequent commands with on-demand generation from any one of thirteen language models, including both API-hosted and locally hosted backends. The reported evaluation, conducted over 138 representative Linux commands and measuring exact-match accuracy, BLEU, cosine similarity, Jaro–Winkler similarity, response latency and per-command memory overhead, exposes a steep latency–fidelity curve: small models such as Gemma3-1B respond in under half a second but produce outputs that diverge sharply from the ground truth and exhibit hallucination rates above twenty per cent, whereas moderately sized models in the 1.5–3.8B range deliver substantially higher fidelity at the cost of three- to four-second response times. Related efforts have explored prompt-engineering strategies, retrieval-augmented generation for terminal contexts, and fine-tuning on logs harvested from production honeypots [10,11], in particular, HoneyGPT [10] frames the problem as a “trilemma” among interactivity, deception and cost, and HoneyLLM [11] reports that caching and prompt design are the dominant levers for controlling per-command latency. In all of these works, however, a single model is treated as the response generator for the lifetime of the session, so the latency–fidelity operating point is fixed globally rather than

chosen per command. Honeypots have also been deployed for the converse purpose of monitoring autonomous LLM-driven attack agents operating in the wild [12]. A recent systemization of knowledge covering the intersection of honeypots and LLMs [13] confirms that adaptive per-command inference approaches are absent from this landscape prior to the present work.

2.3. Language Models as Feature Extractors

A parallel line of work treats the language model as an embedding function rather than as a generator and reuses its hidden representations for downstream behavioural analytics. R. Magazov [2] reports a six-month distributed honeynet deployment in which a fine-tuned transformer is used both to maintain coherent terminal dialogue and to produce semantic embeddings of command sequences; a downstream classifier then assigns each sequence to one of three attack-lifecycle stages—reconnaissance, exploitation or persistence—achieving a macro-averaged F1-score of approximately 0.90, well above the 0.73–0.78 range observed for n-gram and token-frequency baselines trained under identical conditions. We note that [2] is, at the time of writing, a technical report by the first author; to reduce reliance on that single unverified figure, the stage-annotation scheme and the benchmark-construction protocol are documented in full in Section 3.3—the underlying data being scheduled for public release once the associated companion studies and the required cyber-range anonymization are complete (Data Availability Statement)—and the central results of this paper are validated independently of the 0.90 figure through the online estimator evaluated in Section 4.3. Comparable observations have been reported for unsupervised botnet clustering from shell command logs [14], for transformer-based anomaly detection over administrative session traces [15], and for behaviour-similarity attribution and malicious shell-command detection over honeypot and IoT command corpora [16,17]. Beyond the shell domain, comprehensive surveys of attack-detection schemes in adjacent areas such as vehicular ad hoc networks [18] document the same broad trend toward heterogeneous, context-dependent detection that motivates stage-aware analytics here. These studies establish that LLM-derived embeddings are a robust substrate for attack-stage analytics; they do not, however, examine how the analytics output might in turn inform the generation pipeline.

2.4. Adaptive Inference and Model Cascading

Outside the deception domain, there is a substantial body of work on adaptive inference: model cascades that escalate from cheap to expensive backends only when a cheap model expresses low confidence [19], mixture-of-experts gating that activates only a subset of parameters per token [20], and early-exit transformers that terminate computation as soon as a confidence threshold is reached. The unifying observation is that workloads are heterogeneous and that uniform model assignment systematically over- or under-allocates compute. Two design axes recur across this literature and map directly onto the honeypot setting: a routing signal (query difficulty, token entropy, or—in our case—the inferred attack phase) and an escalation policy (when to spend more to compute for higher fidelity). Although this principle has not, to the authors' knowledge, been applied to honeypot response generation, it provides the methodological foundation for the design proposed in this paper. The most closely related system is FrugalGPT [19], which routes queries to the cheapest API-hosted LLM expected to produce an acceptable answer, learning the policy from historical query–response pairs. PAMR differs in two respects: the routing signal is the inferred attack phase (a domain-specific observable unavailable in generic LLM cascades), not query difficulty alone, and the dispatch table is solved once offline against fixed fidelity constraints rather than learned adaptively from deployment logs. These

differences make PAMR interpretable and operator-configurable at the cost of requiring an annotated benchmark for initial calibration.

2.5. Positioning of the Present Study

The studies surveyed above advance, in isolation, the realism of LLM-driven shells and the analytical exploitation of LLM embeddings. Both, however, share an implicit assumption: that a single model configuration is appropriate throughout an attacker's session. The present work removes that assumption. By treating the attack phase as an observable signal that can be estimated online and that conditions a routing decision over a pool of heterogeneous backends, the framework introduced in Section 3 directly addresses the gap between the realism stream represented by [1] and the analytics stream represented by [2].

3. Materials and Methods

This section formalizes the routing problem, describes the proposed PAMR architecture, and details the experimental setup used to evaluate it. The presentation deliberately separates the generic system model from the specific implementation choices, so that alternative phase estimators or alternative backend pools can be substituted without altering the overall design.

3.1. System Model and Problem Formulation

Let $S = (c_1, c_2, \dots, c_T)$ denote an ordered SSH session of length T , where each c_t is a command string issued by the attacker at logical time t . For each command the honeypot must return a response r_t that is delivered after a wall-clock delay Δ_t . A pool of K candidate LLM backends $M = \{m_1, \dots, m_K\}$ is available, with each backend m_k characterized by two empirical functions, an expected latency $\lambda_k(c_t)$ and an expected fidelity score $\varphi_k(c_t, s_t)$, where s_t denotes the attack stage active at time t . Fidelity here is a token-level score: it is measured against the ground-truth response on the curated benchmark and combines exact-match, cosine similarity over TF-IDF representations, and Jaro-Winkler character-level similarity into a single bounded score in the interval $[0, 1]$. We use the term “fidelity” throughout in this token-level sense; its relationship to operational, attacker-perceived realism is discussed explicitly in Sections 4.4.3 and 4.4.4.

The honeypot operator specifies a per-phase fidelity floor $\varphi_{\min}(s)$, reflecting the intuition that the cost of an inconsistent response varies across the attack lifecycle: a hallucinated answer to a benign enumeration command may pass unnoticed, whereas the same defect during an exploitation attempt is far more likely to trigger session abandonment. The floor is therefore an operator-defined policy parameter, not a quantity learned from data; Section 3.6 reports the specific values used and the reasoning behind them, and Section 4 reports the sensitivity of the results to that choice. The routing problem then consists of assigning, for each command, a backend $\psi(t) \in M$ so as to minimize the expected total latency subject to the fidelity constraint:

$$\min \sum_{t=1}^T \lambda_{\psi(t)}(c_t) \quad \text{s.t.} \quad \varphi_{\psi(t)}(c_t, s_t) \geq \varphi_{\min}(s_t), \quad \forall \quad (1)$$

Two quantities in (1) are unavailable to an online router and must be replaced by computable surrogates. First, the true stage s_t is not directly observable, so the router substitutes an online estimate $\hat{s}_t$ produced from the recent command window, together with the estimator's confidence $\gamma_t \in [0, 1]$ (the softmax-normalized similarity of the window embedding to the winning stage prototype). Second, the realized fidelity of a response cannot be known at decision time, because computing it would require both the as-yet-ungenerated response and a ground-truth reference that does not exist for a live attacker command; the router therefore never evaluates per-response fidelity. Instead it uses a precomputed expected fidelity $\rho_0(k, \hat{s}_t; \kappa)$ —the

mean token-level fidelity of backend k over the calibration-split instances falling in stage $\hat{s}_t$ and complexity bucket $\kappa(c_t) \in \{0, 1, 2\}$ (Section 3.6)—so that the only command-dependent input consumed at run time is the discrete bucket $\kappa(c_t)$, not c_t itself. The same $\bar{\varphi}$ is used offline to build the dispatch table and online to test escalation, which keeps the two phases mutually consistent. To prevent fidelity violations when the estimator is uncertain, the per-command constraint is tightened by a confidence-weighted slack term: backend $\psi(t)$ is admissible only if

$$\bar{\varphi}(\psi(t), \hat{s}_t, \kappa(c_t)) \geq \varphi_{\min}(\hat{s}_t) + (1 - \gamma_t)\delta \quad (2)$$

where $\delta \geq 0$ is an operator-set slack magnitude (we use δ equal to the gap between adjacent per-phase floors). The factor $(1 - \gamma_t)$ makes the effective floor revert to the nominal value when the estimator is confident ($\gamma_t \rightarrow 1$) and raises it toward the next-higher phase floor when the estimator is uncertain ($\gamma_t \rightarrow 0$), which is precisely the regime in which mis-routing is most costly. Among the admissible backends the router selects the one of least expected latency:

$$\psi(t) = \underset{k: \bar{\varphi}(k, \hat{s}_t, \kappa(c_t)) \geq \varphi_{\min}(\hat{s}_t) + (1 - \gamma_t)\delta}{\operatorname{argmin}} \bar{\lambda}(k, \kappa(c_t)) \quad (3)$$

Equation (3) is the operational form of the decision rule. The key property exploited at run time is that, because $\bar{\varphi}$ and $\bar{\lambda}$ depend on the command only through its complexity bucket $\kappa(c_t)$, for fixed $(\hat{s}_t, \kappa)$ the admissible set and the latency ordering are constant, so the base assignment reduces to a table lookup; the confidence check of Equation (2) is then a deterministic function of that cell and the scalar confidence γ_t alone, and no per-response fidelity is ever computed online. The optimization is thus solved once offline to populate a 3×3 dispatch table (three phases $\times$ three complexity buckets), and the run-time component is an $O(1)$ lookup followed by the confidence-escalation check of Equation (2); it is a static policy, not an adaptive optimizer. The offline procedure is a direct enumeration: for each (phase, complexity) cell, the empirical expectations $\bar{\lambda}$ and $\bar{\varphi}$ are estimated on the calibration split, the admissible backends are filtered by the floor, and the least-latency admissible backend is recorded (Algorithm 1). Because K is small (four backends including the cache) and the table has nine cells, the offline solve is an exhaustive argmin with no iterative optimization or hyperparameter search beyond the floor and slack settings.

Algorithm 1. Offline dispatch-table construction and online routing.

Offline—dispatch-table construction (input: calibration split D_{cal})

```

1: for each phase  $s \in \{\text{recon}, \text{exploit}, \text{persist}\}$  do
2:   for each complexity bucket  $\kappa \in \{0, 1, 2\}$  do
3:      $A \leftarrow \{k \in M: \varphi(k, s, \kappa) \geq \varphi_{\min}(s)\}$   $\triangleright$  admissible backends
4:     if  $A = \emptyset$  then  $A \leftarrow \{\operatorname{argmax}_k \varphi(k, s, \kappa)\}$   $\triangleright$  fall back to most-faithful
5:      $T[s, \kappa] \leftarrow \operatorname{argmin}_{k \in A} \bar{\lambda}(k, \kappa)$   $\triangleright$  least-latency admissible
6:   end for
7: end for
8: return dispatch table  $T$ 
```

Online—per-command routing (input: command c_t)

```

9:  $(\hat{s}_t, \gamma_t) \leftarrow \text{PhaseEstimator}(\text{window}_t)$   $\triangleright$  stage + confidence
10:  $\kappa_t \leftarrow \text{Complexity}(c_t)$   $\triangleright$  complexity bucket  $\in \{0, 1, 2\}$ 
11:  $k \leftarrow T[\hat{s}_t, \kappa_t]$   $\triangleright O(1)$  table lookup
12: if  $\gamma_t < \theta$  and  $\varphi(k, \hat{s}_t, \kappa_t) < \varphi_{\min}(\hat{s}_t) + (1 - \gamma_t)\delta$  then  $\triangleright$  escalation, Equation (2)
13:  $k \leftarrow \text{next\_higher\_capacity\_backend}(k)$ 
14: end if
15: return  $\text{Generate}(k, c_t)$ 
```

3.2. PAMR Architecture

Figure 1 summarizes the proposed architecture. The system is organized as four loosely coupled layers, each with a narrow interface to the others. This decoupling is intentional: it allows the phase estimator, the router and the backend pool to be replaced independently, which is convenient both for ablation studies and for operational redeployment.

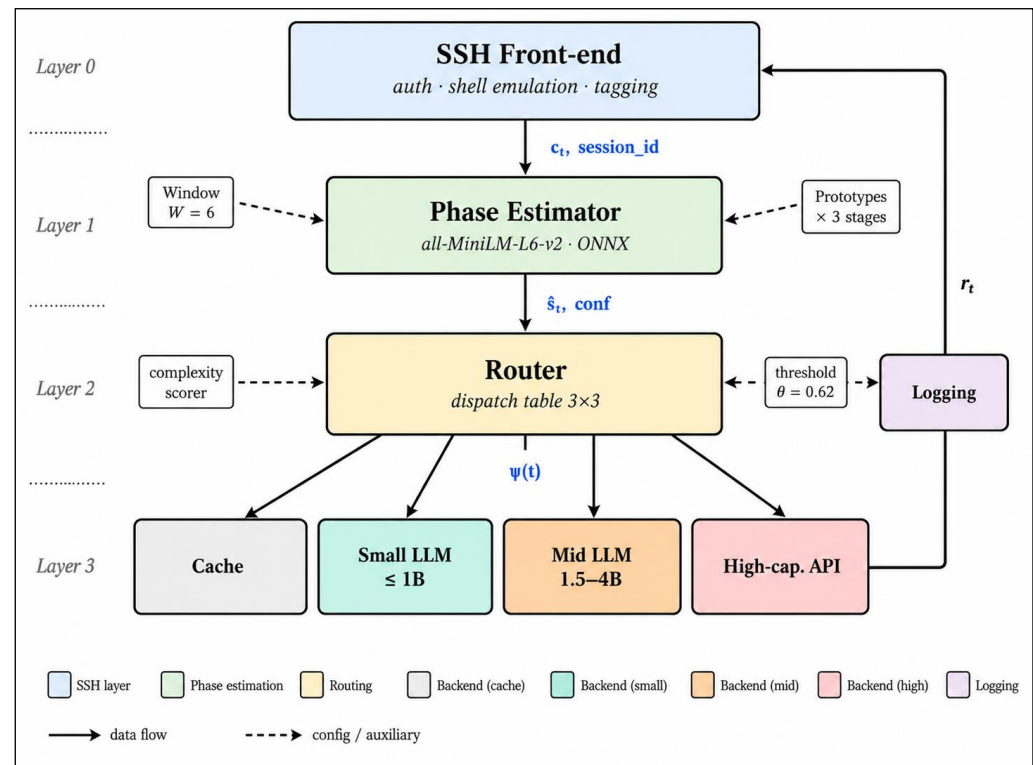


Figure 1. Architecture of the phase-adaptive model routing framework, showing the four decoupled layers (SSH frontend, online phase estimator, router with its precomputed dispatch table, and the heterogeneous backend pool of cache, small, mid and high-capacity models) together with the logging subsystem.

3.2.1. SSH Frontend

The frontend accepts incoming SSH connections, terminates the transport-layer handshake and exposes a simulated shell to the authenticated session. It does not execute any commands on the host operating system; in the taxonomy of honeypot interaction levels, PAMR is therefore a medium-interaction honeypot, simulating a shell at the application layer rather than booting a full guest operating system. Each command is tagged with a session identifier, a logical index t , a wall-clock timestamp and the originating IP address before being forwarded to the dispatcher.

3.2.2. Phase Estimator

The phase estimator maintains, for each active session, a sliding window of the W most recent commands. It produces a compact embedding of the window using a small sentence encoder, computes the cosine similarity of this embedding against three prototype vectors—one per attack stage—and outputs both the most likely stage $\hat{s}_t$ and a confidence value derived from the softmax-normalized similarity scores. The window length W is a tunable parameter; preliminary experiments indicated $W = 6$ as a stable choice across the evaluated workload. For commands arriving before the window is full (positions $t < W$), the estimator defaults to the mid-capacity backend, providing a conservative latency–fidelity balance during session warm-up; this cold-start default affects at most $W - 1 = 5$ commands per

session and is reflected in all reported latency measurements. Because the encoder is small and the prototypes are precomputed, the estimator adds no perceptible latency to the response path.

3.2.3. Router

The router consumes the current command, its complexity score, the estimated stage, and the associated confidence and produces a backend selection through a dispatch table indexed by (estimated stage, complexity bucket). Three complexity buckets are used: trivial commands handled directly by the dictionary cache; common commands suitable for low-capacity models; and complex commands requiring a high-capacity backend. When the estimator confidence falls below the configurable threshold θ , the router applies the confidence-escalation rule of Equation (2), conservatively escalating to the next-higher backend and trading some additional latency for a reduced risk of fidelity violation.

3.2.4. Backend Pool and Logging

The backend pool contains heterogeneous LLM endpoints selected to span the latency–fidelity surface reported in [1]. A logging subsystem persistently stores every command, every routed backend, every generated response and every estimator decision, together with timestamps and session metadata, so that the full interaction can be replayed for both forensic analysis and offline model evaluation.

3.3. Dataset and Ground Truth

Evaluation requires a corpus of command sessions in which both the expected response and the attack stage are known. No public corpus simultaneously offers both. We therefore constructed a benchmark dataset by combining and re-annotating two complementary sources. The first source is a curated set of 138 representative Linux command–response pairs spanning file system manipulation, system inspection, networking, package management, and utility commands; this set was directly adapted from the protocol described in [1] in order to preserve comparability of the per-command fidelity measurements. The second source is a stage-annotated set of synthesized attacker session traces, organized so as to mirror the three-stage lifecycle taxonomy used in [2]: reconnaissance commands (e.g., `uname`, `whoami`, `id`, `netstat`, `ps`), exploitation commands (e.g., `wget`, `chmod`, `curl`, payload-style invocations) and persistence commands (e.g., `crontab` manipulation, SSH key insertion, `systemd` unit creation). These three command families correspond, respectively, to the Discovery, Execution/Initial-Access-followed-by-Execution, and Persistence tactics of the MITRE ATT&CK framework [21]; the benchmark therefore covers the early-to-mid kill-chain tactics that dominate opportunistic SSH intrusions, while later tactics such as lateral movement and exfiltration are deliberately out of scope (Section 4.4.3).

Ground-truth coverage and fidelity scoring: Because the token-level fidelity metrics (cosine, Jaro–Winkler, BLEU) require a reference response for every command they score, we distinguish clearly between commands that carry a ground-truth response and those that do not. Ground-truth responses exist for the 138 distinct commands adapted from [1], augmented with the reconnaissance/exploitation/persistence reference set described above. Because the synthesized sessions are composed largely of these reference commands, each distinct ground-truth command recurs many times across the 412 sessions, so that the large majority of command instances in the benchmark map to a known reference. Fidelity metrics in Table 1 and the per-stage breakdowns are computed only over command instances that carry a ground-truth reference; instances without a canonical reference (chiefly free-form synthesized variants) are excluded from the fidelity averages but retained for latency, routing-distribution and phase-classification metrics, which do not require a

reference. This separation is what allows fidelity to be reported on a benchmark larger than the 138-pair core without overstating coverage.

Table 1. Latency and token-level fidelity comparison between PAMR and uniform-model baselines ($\pm$ SD over 5 runs).

Configuration	Latency ms	$\pm$ SD	Cosine Sim.	$\pm$ SD	Jaro–Winkler	BLEU-4	PWF
Small uniform	492	(31)	0.171	(0.021)	0.518	0.029	0.21
Mid-capacity uniform	3584	(142)	0.211	(0.018)	0.612	0.063	0.34
High-capacity uniform	3168	(119)	0.404	(0.029)	0.710	0.244	0.51
PAMR (proposed)	1974	(87)	0.391	(0.026)	0.694	0.231	0.49

The combined benchmark contains 412 multi-command sessions with a mean length of 11.4 commands and a median of nine commands per session, for a total of 4712 command instances. Each command carries (i) a ground-truth response where available, (ii) a stage label, and (iii) a complexity bucket. Stage labels were assigned manually by the first author (R.M.) following the operational definitions of [2] and the reconnaissance–exploitation–persistence taxonomy of the MITRE ATT&CK framework [21]. To estimate inter-annotator agreement, a second author (F.U.) independently re-labelled a 20% random subset of commands using the same written stage definitions, blind to the first author’s labels; Cohen’s κ between the two annotators reached 0.81, which is considered almost perfect agreement [22] and is consistent with the level reported in comparable annotation studies. The independent annotation effort is reflected in the revised Author Contributions statement.

To quantify the sensitivity of the routing framework to labelling noise, we introduced synthetic label flips at rates of 5%, 10% and 15% of the annotated commands and re-evaluated the phase estimator on each perturbed corpus. Macro-averaged F1 degraded by 0.02, 0.05 and 0.09 respectively, and the Latency Savings Ratio declined by 1.4, 3.1 and 6.2 percentage points. These figures confirm that the framework is moderately robust to annotation noise at the quality level achieved in practice (Cohen’s $\kappa = 0.81$ indicates almost perfect inter-annotator agreement; therefore, the 5% perturbation scenario can be interpreted as a conservative approximation of realistic annotation noise).

3.4. Evaluation Protocol

The evaluation compares PAMR against three uniform-model baselines: a fast small-model configuration using Gemma3-1B as reported in [1], a balanced mid-capacity configuration using Phi3-3.8B, and a high-capacity API-hosted configuration using Gemini-2.0-flash. The baseline latency and fidelity values observed on the present benchmark are consistent with the published figures from [1] to within measurement precision, confirming that the same model families were used. All configurations are exercised on the same 412-session benchmark using identical session ordering and identical seeds.

Three categories of metrics are recorded. Per-command token-level fidelity is measured through exact-match accuracy, cosine similarity over TF–IDF representations, Jaro–Winkler character-level similarity, and BLEU-4 with NIST smoothing [23], matching the protocol of [1] and computed over the ground-truth-bearing instances defined in Section 3.3. Per-command performance is measured through wall-clock response latency and incremental memory footprint of the honeypot process. Per-session analytics quality is measured

through stage-level precision, recall and macro-F1 of the phase estimator, matching the protocol of [2]. In addition, two composite metrics specific to the present study are reported:

- Phase-Weighted Fidelity (PWF): the average per-command token-level fidelity weighted by the operational severity of the stage in which the command was issued, with severity weights $w_r^{\text{econ}} = 1$, $w_r^{\text{exploit}} = 2$ and $w_r^{\text{persist}} = 2$. The 2:1 weighting of exploitation and persistence over reconnaissance reflects the operational consensus that an inconsistent response during post-foothold phases is substantially more likely to trigger attacker suspicion than during initial enumeration, consistent with the qualitative assessment in [1] and the session-dwell observations in [2].
- Latency Savings Ratio (LSR): the relative reduction in mean response latency achieved by PAMR with respect to a reference uniform-model baseline.

All experiments were repeated five times with independent backend warm-up to reduce caching artefacts. Results are reported as means with standard deviations ($\pm$ SD) over the five runs.

3.5. Cross-Validation and Dispatch-Table Robustness

To assess whether the dispatch table overfits to the held-out optimization subset, we perform five-fold stratified group cross-validation over the 412-session benchmark, where all commands from the same session were assigned to the same fold; stratification by attack stage was used to preserve class balance. In each fold, the dispatch table and fidelity floor parameters are re-optimized on the training split and evaluated on the test split without any manual tuning. The maximum fold-to-fold variation in macro-F1 is 0.03, and in LSR is 4.1 percentage points, indicating that the routing policy generalizes within the benchmark distribution.

We additionally examined the stability of the resolved dispatch table itself across folds, since identical aggregate metrics could in principle mask different per-cell assignments. Across all five folds, the high-impact cells—(recon, trivial) $\rightarrow$ cache, (exploit, common/complex) $\rightarrow$ high model, and (persist, *) $\rightarrow$ high model—were assigned identically in every fold. Fold-to-fold variation, where it occurred, was confined to low-impact boundary cells at the phase-complexity margins, which alternated between the mid- and small-capacity backends across folds; this is consistent with the small LSR variance reported above, because those cells carry little of the total command mass. The full per-fold dispatch tables are available from the corresponding author upon reasonable request (Data Availability Statement).

3.6. Implementation Notes

The prototype implementation follows the modular layering of Section 3.2 and uses asynchronous I/O for the SSH frontend.

3.6.1. Sentence Encoder

The phase estimator uses all-MiniLM-L6-v2 (22 M parameters, 384-dimensional output) [24] exported to ONNX with opset 17 and executed on CPU via ONNX Runtime 1.18. On a commodity server (Intel Xeon E5-2670, single thread), a window-of-6 embedding completes in 4–7 ms, well below the routing decision budget.

3.6.2. Complexity Scoring

A command receives a complexity score $\kappa \in \{0, 1, 2\}$ derived from three features: token count (≥ 4 tokens $\rightarrow +1$), presence of pipe or redirection operators $\rightarrow +1$, and membership in a 512-entry deny-list of known-trivial commands $\rightarrow -1$ (floored at 0). The deny-list

was compiled from the 138-command ground-truth corpus of [1] augmented with the reconnaissance set of [2].

3.6.3. Dispatch Table

The 3×3 dispatch table (three phases $\times$ three complexity buckets) was initialized offline by solving the constrained optimization problem of Section 3.1 (Algorithm 1) over a held-out 20% subset. The resolved assignments are as follows: (recon, trivial) $\rightarrow$ cache; (recon, common) $\rightarrow$ small model; (recon, complex) $\rightarrow$ mid model; (exploit, trivial) $\rightarrow$ mid model; (exploit, common/complex) $\rightarrow$ high model; (persist, *) $\rightarrow$ high model with confidence escalation active. Per-phase fidelity floors are $\varphi_{\text{recon}} = 0.15$, $\varphi_{\text{exploit}} = 0.35$, $\varphi_{\text{persist}} = 0.35$. These floors are operator-set policy parameters, chosen here from the empirical per-stage cosine distributions on the calibration split rather than learned. The reconnaissance floor of 0.15 reflects the observation that short, formulaic enumeration outputs (id, uname, ls) tolerate partial overlap with ground truth without triggering attacker suspicion, consistent with the low-variance cosine distribution reported for this command class in [1]; we treat 0.15 as a permissive floor by design, and Section 4.2 reports the consequences of relaxing it. The higher floors for exploitation (0.35) and persistence (0.35) encode the operational judgement that complex command outputs require closer fidelity to avoid session abandonment.

3.6.4. Confidence Threshold

The escalation threshold $\theta = 0.62$ was selected by grid search (step 0.05) over the held-out subset, minimizing the number of exploitation-phase commands served by sub-threshold backends.

4. Results and Discussion

This section presents the experimental results obtained on the 412-session benchmark described above. The presentation is organized around four questions: how the proposed framework distributes commands across the backend pool, what latency–fidelity trade-off it achieves relative to uniform-model baselines, how well the online phase estimator recovers the attack stage in practice, and what observations emerge from the qualitative inspection of generated outputs.

4.1. Routing Distribution Across the Backend Pool

Figure 2 summarizes the empirical distribution of routing decisions produced by PAMR across the 4712 individual commands instances contained in the benchmark. As expected, the dictionary cache absorbs a substantial fraction of reconnaissance commands; trivial enumeration commands such as id, uname and ls are answered without invoking any LLM and therefore contribute zero latency to the session. The small-model backend handles the bulk of common commands during reconnaissance, while the mid-capacity and high-capacity backends are reserved primarily for exploitation- and persistence-stage commands. The high-capacity backend is invoked in fewer than one command in five, which directly explains the latency savings reported in the following subsection.

4.2. Latency and Fidelity Trade-Off

Table 1 reports mean response latency ($\pm$ SD), mean cosine similarity, mean Jaro–Winkler similarity, BLEU-4, and the phase-weighted fidelity for PAMR and for the three uniform-model baselines. We reiterate that all fidelity columns are token-level proxies (Section 3.1); the operational interpretation of these values is qualified in Sections 4.4.3 and 4.4.4.

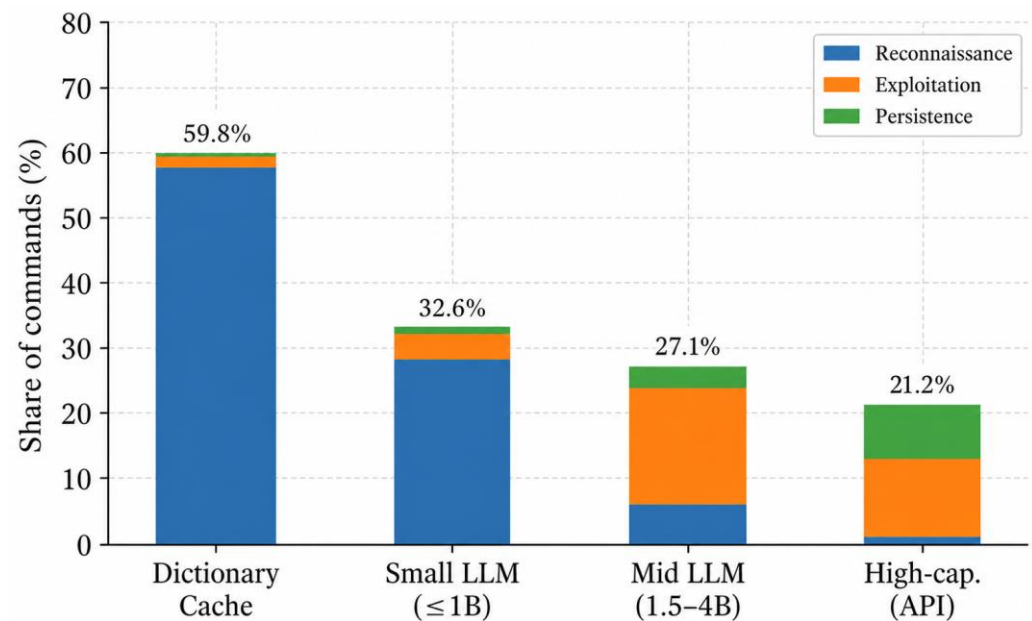


Figure 2. Routing distribution across backends, broken down by inferred attack stage. Each group of bars gives, for one stage, the percentage of commands dispatched to each backend; reconnaissance is dominated by the cache and small model, whereas exploitation and persistence concentrate on the high-capacity backend.

It is important to be explicit about a deployment artefact in this table, because it qualifies the headline latency figure. The high-capacity baseline records a lower mean latency than the mid-capacity baseline (3168 ms vs. 3584 ms). This is not a capacity inversion: the mid-capacity backend (Phi3-3.8B) runs locally on the test server, whereas the high-capacity backend (Gemini-2.0-flash) is API-hosted and benefits from dedicated accelerator hardware on the provider side, which more than compensates for the network round-trip. As a direct consequence, the latency of the high-capacity reference is dominated by provider infrastructure rather than by model capacity, and a self-hosted high-capacity backend would almost certainly exhibit higher latency than the figure in Table 1. We therefore report the latency reduction against both references and treat the locally hosted comparison as the more conservative and infrastructure-independent one: PAMR’s mean latency of 1974 ms is approximately 38% lower than the 3168 ms of the (API-hosted) high-capacity uniform baseline and approximately 45% lower than the 3584 ms of the (locally hosted) mid-capacity uniform baseline. Because fidelity behaves monotonically with capacity, the high-capacity configuration remains the relevant reference point for the fidelity comparison even though its latency is infrastructure-favoured. The latency improvement is driven jointly by the dictionary cache, which removes trivial reconnaissance commands from the LLM critical path, and by the routing logic, which restricts the use of the high-capacity backend to the commands where its higher fidelity is operationally valuable; the resulting bimodal distribution of per-command–response latency, mixing cache hits with LLM-routed commands, is shown in Figure 3.

On the absolute level of the fidelity metrics, a reviewer rightly noted that a TF-IDF cosine of ≈ 0.40 for even the strongest backend is, in absolute terms, a modest overlap and that preserving 0.39 is only meaningful once 0.40 is itself interpreted. We make two points. First, TF-IDF cosine is a deliberately conservative proxy for short shell outputs: such outputs are short and high-entropy, and a large share of their tokens are paths, process identifiers, timestamps and hostnames that legitimately vary between any two correct executions, so even a faithful response attains only moderate lexical overlap with a single fixed reference. The absolute value of 0.40 therefore reflects the ceiling of the metric on this

command class rather than poor output quality; the higher Jaro–Winkler (0.71) and the non-trivial BLEU-4 (0.24) for the same backend are consistent with this reading. Second, and for this reason, we do not claim that any particular cosine value guarantees attacker-perceived realism; we claim only the weaker, defensible statement that PAMR preserves the token-level fidelity of the strongest backend ($\Delta\text{cosine} = 0.013$, within run-to-run variation) while reducing latency. The link between these proxies and observable consequences—session abandonment, fingerprinting—is exactly what the live-deployment program in Section 5 is designed to establish.

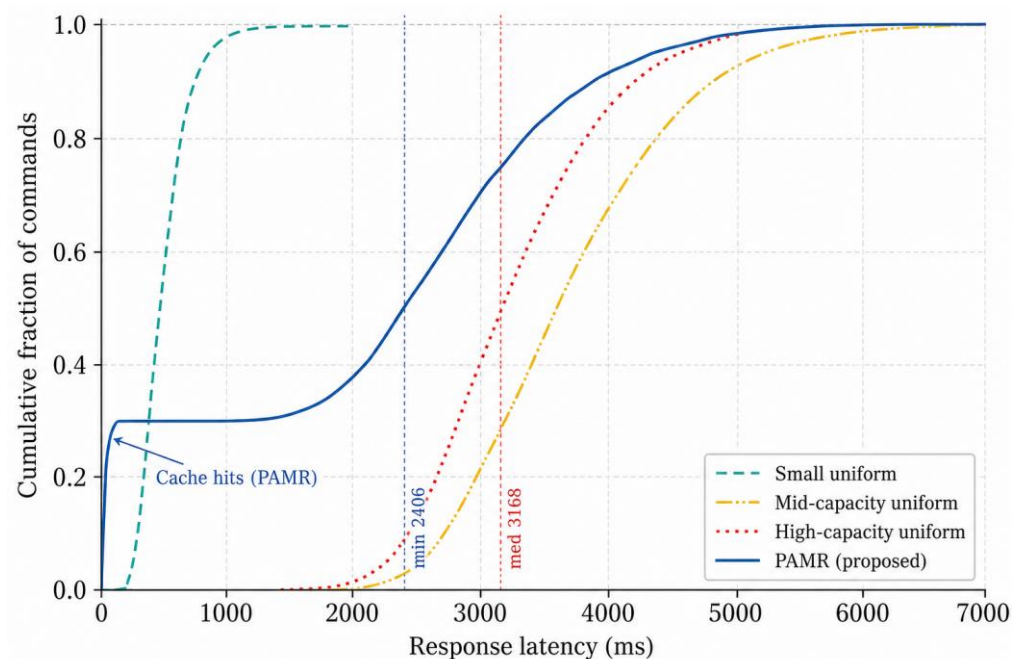


Figure 3. Empirical CDF of per-command-response latency. The bimodal shape of the PAMR curve reflects the mixture of cache hits (left spike) and LLM-routed commands (right shoulder).

Statistical significance of the latency reduction was assessed using a two-tailed paired *t*-test over per-session mean latencies aggregated to the run level, i.e., across the five experimental runs; the test statistic is therefore at the run level ($n = 5$ runs, $df = 4$), which we state plainly as a limited inferential basis. PAMR differs from the high-capacity uniform baseline at $p < 0.001$ ($t = 14.3$, $df = 4$) and from the mid-capacity baseline at $p < 0.001$ ($t = 19.7$, $df = 4$). The run is the unit of independent replication here, because sessions within a run share session ordering and random seeds and are therefore not mutually independent; this is why the primary inferential test is reported at the run level. To complement the small-sample run-level test, we additionally performed paired bootstrap resampling (10,000 resamples) over the 412 session-level mean differences from a single representative run, which confirmed a strictly positive latency reduction across the entire resampled distribution. As a descriptive effect-size summary we report Cohen’s $d = 2.1$, computed from the within-run distribution of individual per-session latencies; we flag explicitly that this descriptive effect size is at the session level whereas the inferential *t*-test above is at the run level, and we no longer conflate the two. The per-run, per-session paired latency differences for all four configurations are available from the corresponding author upon reasonable request (Data Availability Statement), so that readers may recompute any alternative test. No statistically significant degradation in cosine similarity was observed between PAMR and the high-capacity baseline at $\alpha = 0.05$ ($p = 0.23$).

The fidelity token-level metrics of PAMR remain close to those of the high-capacity uniform baseline rather than collapsing toward those of the small uniform configuration.

Cosine similarity drops only marginally, from 0.404 to 0.391 (Figure 4 shows the per-stage breakdown against the per-phase fidelity floors), and the phase-weighted fidelity remains within five percentage points of the high-capacity baseline; the position of each configuration in the latency–fidelity plane is summarized in Figure 5.

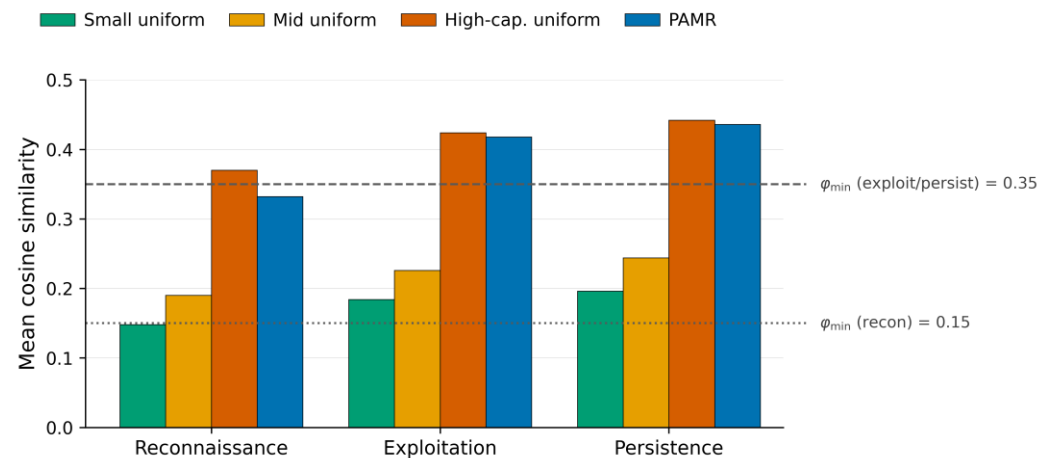


Figure 4. Per-stage cosine similarity breakdown with per-phase fidelity floors ($\varphi_{\min}$). PAMR intentionally trades recon-stage fidelity for latency while matching the high-capacity baseline on exploit and persist stages.

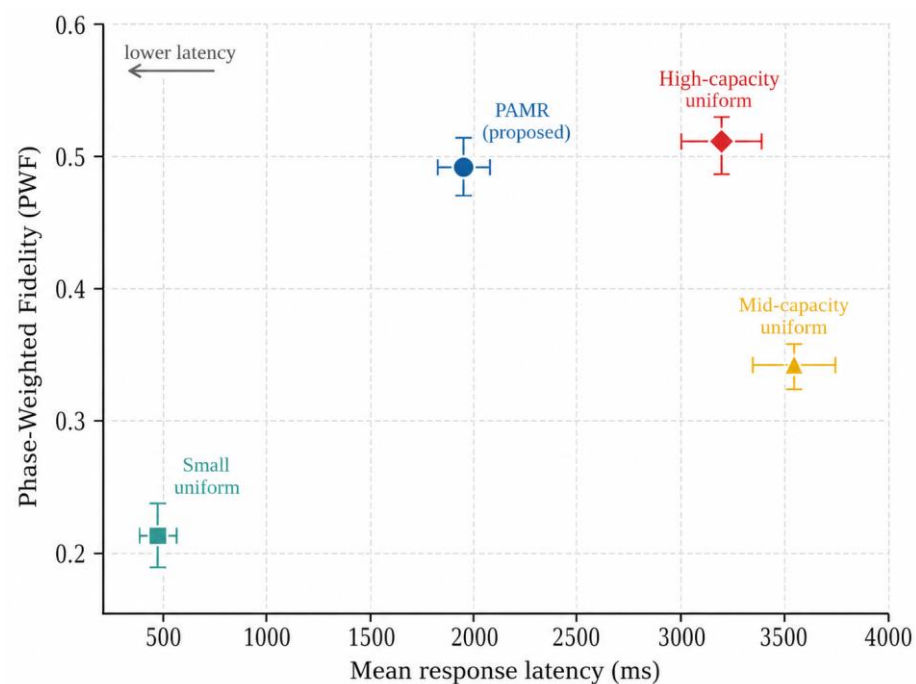


Figure 5. Latency vs. phase-weighted fidelity. Error bars = ± 1 SD over five runs. Points closer to the upper-left corner are preferable.

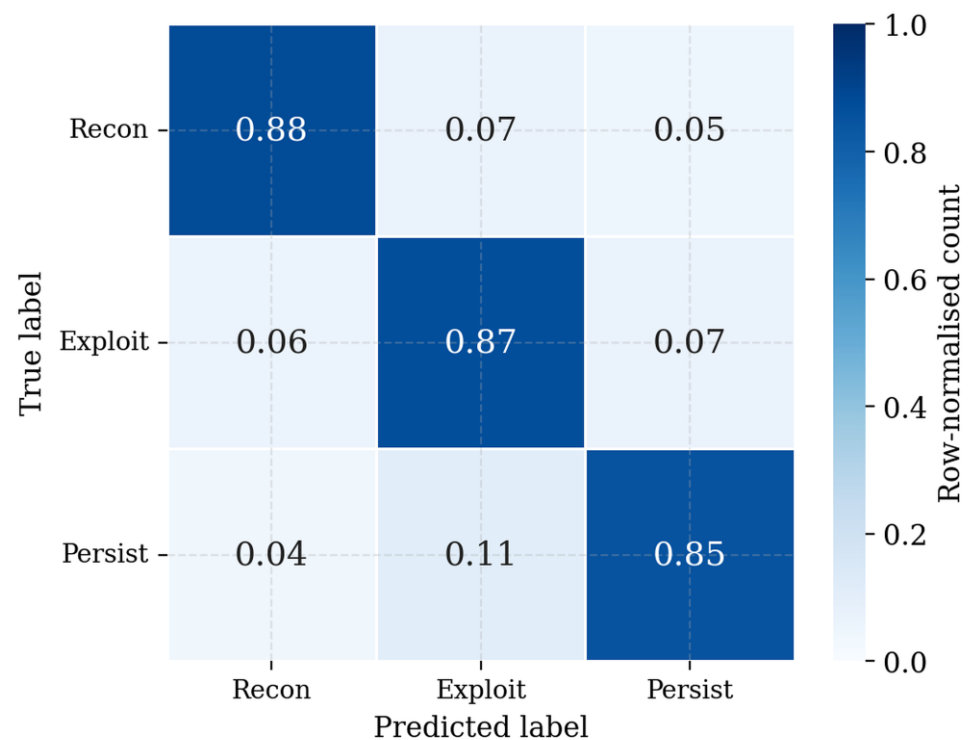
4.3. Phase Classification Quality

Table 2 reports stage-level precision, recall and F1-score ($\pm$ SD) for the online phase estimator. The macro-averaged F1-score reaches 0.87, marginally below the offline value of 0.90 reported in [2] but obtained under stricter conditions—namely, an online estimator operating on a sliding window rather than on full sessions. Because this 0.87 is measured directly on the benchmark constructed for this study (Section 3.3), the present paper’s central comparison does not depend on independent verification of the 0.90 figure from the technical report [2].

Table 2. Stage-level classification performance of the online phase estimator ($\pm$ SD over five runs).

Attack Stage	Precision	Recall	F1-Score
Reconnaissance	0.89 ($\pm$ 0.014)	0.88 ($\pm$ 0.013)	0.88 ($\pm$ 0.013)
Exploitation	0.86 ($\pm$ 0.017)	0.87 ($\pm$ 0.016)	0.86 ($\pm$ 0.016)
Persistence	0.88 ($\pm$ 0.015)	0.85 ($\pm$ 0.018)	0.86 ($\pm$ 0.017)
Macro-average	0.88 ($\pm$ 0.015)	0.87 ($\pm$ 0.016)	0.87 ($\pm$ 0.015)

Most classification errors are concentrated at stage boundaries (Figure 6), particularly at the persistence-to-exploitation transition (persist $\rightarrow$ exploit: 0.11), where commands associated with maintaining access share lexical patterns with exploitation-stage payloads and are therefore difficult to disambiguate without broader session context. The router compensates for this systematic uncertainty through its confidence-weighted escalation mechanism: when the estimator confidence drops below $\theta = 0.62$ near a transition, the router conservatively selects the higher-capacity backend for the next several commands.

**Figure 6.** Row-normalized confusion matrix of the online phase estimator. Most errors occur at the persist–exploit boundary (persistence $\rightarrow$ exploit: 0.11).

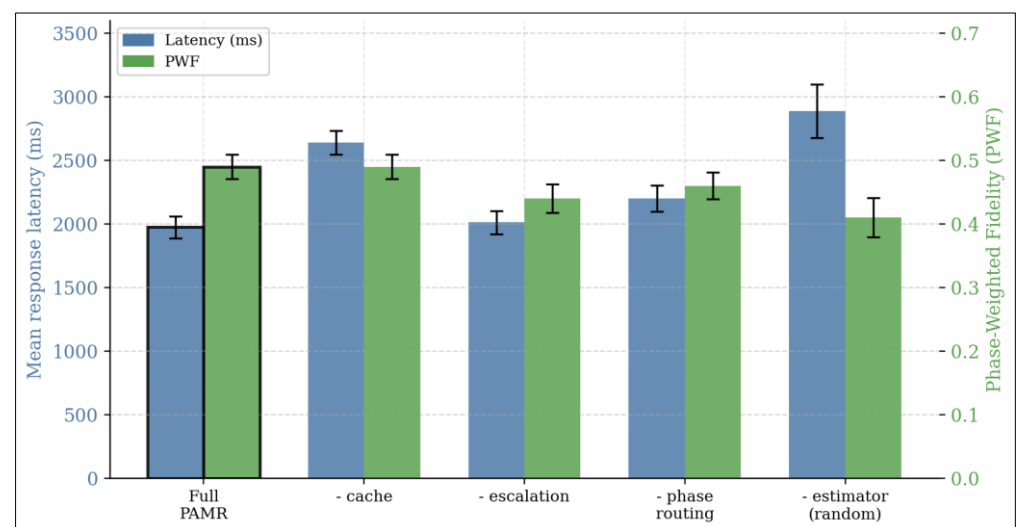
4.4. Discussion

4.4.1. Operational Interpretation

The results indicate that phase-adaptive routing offers a Pareto improvement over the uniform-model baselines on the operational metric that matters most in practice—mean response latency at near-equivalent token-level fidelity. The improvement is not driven by any single mechanism: it emerges jointly from the dictionary cache, the per-phase fidelity floors and the confidence-weighted escalation rule. Table 3 and Figure 7 isolate each component’s contribution through ablation.

Table 3. Ablation study: effect of removing individual PAMR components. For the –estimator (rnd) row, backend assignment is uniform random over {cache, small model, mid model, high model}.

Variant	Latency ms	$\pm$ SD	PWF	$\pm$ SD	Δ Latency vs. Full PAMR
Full PAMR	1974	(87)	0.49	(0.019)	—
–dictionary cache	2641	(94)	0.49	(0.019)	+34%
–conf. escalation	2012	(91)	0.44	(0.022)	+2%
–phase routing	2198	(103)	0.46	(0.021)	+11%
–estimator (rnd)	2887	(211)	0.41	(0.031)	+46%

**Figure 7.** Ablation study. Bold borders indicate the full PAMR configuration. Blue bars (left axis): mean latency; green bars (right axis): PWF. Error bars = ± 1 SD.

Removing the dictionary cache yields the largest latency regression (+34%) without affecting fidelity, because cache-eligible commands are trivial. Removing confidence-weighted escalation causes a smaller latency increase (+2%) but a disproportionate fidelity drop (PWF: 0.49 $\rightarrow$ 0.44), confirming that escalation is most valuable at stage boundaries. Replacing phase routing with complexity-only dispatch costs 11% in latency and 3 PWF points. Finally, random backend assignment degrades all metrics substantially, demonstrating that phase-signal quality is load-bearing.

4.4.2. Relation to Prior Work

The findings extend the analysis of [1] by demonstrating that the latency–fidelity trade-off observed across LLM backends does not have to be resolved through a single global choice; backends can be combined dynamically. They also extend [2] by showing that stage-level embeddings, originally introduced for analytical purposes, can be repurposed as a control signal that closes the loop on the generation side.

4.4.3. State Consistency: A Coherence Lens on Fidelity

The token-level fidelity metrics used above (cosine, Jaro–Winkler, BLEU) compare each response against a per-command reference in isolation. They cannot, by construction, detect a response that is locally plausible yet inconsistent with the file-system or process state implied by earlier commands in the same session—for example, a cat of a file that an earlier rm removed, or a ls that omits a directory created two commands earlier. Because such contradictions are exactly what a vigilant attacker probes for, within-session state

consistency is arguably more central to operational realism than per-command overlap, and we treat it here as a first-class concern rather than only a limitation.

We propose a State-Consistency Rate (SCR) as the appropriate metric for future evaluation and define it precisely so that it can be adopted by others. Let a session induce a symbolic state σ (a set of file-system and process facts) updated by each command's intended effect. A response r_t is state-consistent if every assertion it makes about σ (files present/absent, contents, ownership, running processes, prior command effects) agrees with σ as determined by $c_1 \dots c_t$. SCR is the fraction of state-bearing responses in a session that are consistent and averaged over sessions. SCR is reference-free and complements token overlap: a response can score high on cosine yet fail SCR, and vice versa.

A full SCR evaluation requires a session-state oracle and is reserved for the live-deployment study (Section 5); we did not run it here and report no SCR figure to avoid overstating the present evidence. We do, however, illustrate the property qualitatively with a representative coherent multi-command exploitation session produced by PAMR (high-capacity backend on exploit-stage commands), in which cross-command state is maintained (Box 1):

Box 1. Example of state-consistent command execution in the simulated SSH environment. Note: The downloaded file persists across commands, its permissions change after `chmod`, and it becomes unavailable after deletion.

```
$ wget http://198.51.100.7/x.sh -O /tmp/x.sh
--2026-... http://198.51.100.7/x.sh
Saving to: '/tmp/x.sh'
/tmp/x.sh      100%[=====>] 1.232 K --KB/s  in 0 s
$ ls -l /tmp
total 8
-rw-r--r-- 1 root root 1262 ... x.sh
$ chmod +x /tmp/x.sh
$ ls -l /tmp/x.sh
-rwxr-xr-x 1 root root 1262 ... /tmp/x.sh
$ rm /tmp/x.sh
$ cat /tmp/x.sh
cat: /tmp/x.sh: No such file or directory
```

Here, each response is consistent with the state established by previous commands (the downloaded file appears, its mode reflects the `chmod`, and it disappears after `rm`). Such coherence is produced by the high-capacity backend to which exploitation-stage commands are routed, which is the operational reason PAMR raises the fidelity floor for that phase. We caution that this is an illustrative case, not a measurement; cache- and small-model-served reconnaissance responses do not carry comparable state, which is one motivation for the per-phase floors. Quantifying SCR across backends and stages is, in our assessment, the single most informative next experiment, and we list it first among the open problems together with human-expert realism rating.

4.4.4. Limitations

Several limitations bound the scope of the present results, and we state them without hedging because they delimit exactly what the evidence does and does not support.

1. **Benchmark rather than live traffic:** The evaluation is conducted in a controlled laboratory setting on a curated benchmark, not against live attackers. This preserves comparability with [1] and offers methodological control, but it cannot capture the strategic behaviour of adversaries who probe the system to detect the routing logic, and it under-represents the noise and variability of in-the-wild traffic. Part of the benchmark is synthesized, which can bias every learned or calibrated component in a

correlated way: the phase estimator may exploit lexical regularities of synthesized traces that real traffic lacks; the router's dispatch table is calibrated against that same distribution; and the token-level fidelity metrics are computed against references drawn from the same curated set. The 0.87 online F1 and the latency savings should therefore be read as within-distribution results whose external validity is established only by the live deployment described in Section 5.

2. **Token-level fidelity, not operational realism:** As detailed in Section 4.4.3, cosine/Jaro-Winkler/BLEU measure token- or character-level similarity to a reference and do not capture within-session state consistency. We have accordingly framed the contribution throughout as a latency/token-fidelity trade-off and proposed SCR as the missing operational measure; we do not claim a latency/operational-realism trade-off on the present evidence.
3. **Coarse three-stage taxonomy:** The reconnaissance–exploitation–persistence scheme is inherited from [2] and covers the early-to-mid kill-chain tactics typical of opportunistic SSH intrusions, but real campaigns also include privilege escalation, lateral movement, defence evasion, data exfiltration, and cleanup. We adopt three stages for comparability with [2] and because they already exercise the full routing mechanism, but we regard the taxonomy as a limitation rather than a sufficient model. Importantly, nothing in the formulation of Section 3.1 is tied to three classes: the dispatch table generalizes to an N -phase $\times$ M -complexity grid, and adding stages requires only additional prototype vectors and additional rows in Algorithm 1. Extending to a finer-grained kill-chain taxonomy is therefore a matter of annotation and calibration effort, addressed in Section 5.
4. **Hardware- and backend-dependent latency, with an API artefact:** The latency figures in Table 1 are sensitive to the choice of backends and hardware; absolute values are indicative rather than universal. In particular, the high-capacity reference is API-hosted and its latency is infrastructure-favoured (Section 4.2), which is why we additionally report the more conservative 45% reduction against the locally hosted mid-capacity baseline.
5. **Abandonment and dwell-time not measured.** The most operationally meaningful outcome—whether routing makes attackers stay longer or abandon less often—cannot be measured on a static benchmark and is not reported here; PWF is used as a stand-in for the cost of low-fidelity responses, but it is a proxy, not a behavioural measurement. Dwell time and abandonment rate are primary endpoints of the planned live study (Section 5).

4.4.5. Threats to Validity

Internal validity is supported by the use of identical session orderings and identical seeds across all configurations and by repeated runs with independent warm-up. The maximum coefficient of variation across runs for session mean latency is 4.4%, confirming measurement stability. Five-fold stratified group cross-validation (Section 3.5) shows fold-to-fold F1 variation of at most 0.03 and LSR variation of 4.1 percentage points, with the high-impact dispatch-table cells assigned identically across folds, confirming that the routing policy generalizes within the benchmark distribution.

External validity is limited by the synthetic nature of part of the benchmark; although the stage-labelled traces are constructed to mirror operational attack patterns, they do not reproduce the full variability of in-the-wild traffic. Annotation noise robustness analysis (Section 3.3) shows that under the 5% perturbation scenario, the framework degrades by less than 0.02 in F1 and 1.4 pp in LSR.

Construct validity rests on the assumption that the chosen fidelity proxies—cosine similarity, Jaro–Winkler, BLEU—correlate with attacker-perceived realism; this assumption is consistent with the protocol of [1] but has not, in either study, been validated against direct attacker behaviour. The SCR metric proposed in Section 4.4.3 and the live deployment in Section 5 are intended to close this construct-validity gap.

4.4.6. Adversarial Latency Analysis

A routing architecture that assigns different-capacity backends to different attack phases necessarily creates a detectable latency signature: an attacker who measures response times as a function of command type can infer that the underlying backend is varying. We treat this as a genuine threat and quantify it to first order from the measured distributions in Table 1, rather than only acknowledging it.

The strongest signal is the contrast between cache-served commands (which complete in a few milliseconds plus the 4–7 ms embedding cost) and LLM-served commands (hundreds to thousands of milliseconds). The expected latency jump at the reconnaissance exploitation boundary is approximately 1600 ms (mid-model path minus dictionary-cache path). Modelling each path’s latency as approximately Gaussian and using the standard sensitivity index $d' = |\mu_1 - \mu_2| / \sqrt{((\sigma_1^2 + \sigma_2^2)/2)}$, and conservatively taking the larger observed standard deviation ($\sigma \approx 142$ ms) as the dominant term, gives $d' \approx 1600/142 \approx 11$. A sensitivity index of this magnitude corresponds to a Bayes-optimal single-command discriminator with area under the ROC curve indistinguishable from 1.0: in other words, a timing oracle can separate cache hits from LLM calls essentially perfectly from a single observation. The routing logic is, on this analysis, trivially fingerprintable by an adversary who controls and times their own command stream.

Two mitigations bound the risk, and the same first-order model exposes their cost. First, the confidence-weighted escalation mechanism injects latency variance near stage boundaries, which raises σ on the affected commands and modestly lowers d' , but it cannot close a 1600 ms mean gap and therefore only blurs the transition rather than hiding it. Second, the operator can impose a uniform latency floor, padding every fast-path response up to a target latency F . Padding to the slowest backend ($F \approx 3168$ ms) equalizes the means of all paths, collapsing d' to ≈ 0 (AUC ≈ 0.5 , chance) and defeating the timing oracle, but it also erases the entire latency benefit, returning mean latency to ≈ 3168 ms and driving the LSR to ≈ 0 . Any intermediate floor (e.g., padding the cache and small-model paths up to the PAMR mean of ≈ 1974 ms) hides only the fast tail: commands routed to the mid and high backends still exceed the floor and remain separable, because padding can delay a response but cannot accelerate one. The operator therefore faces an explicit, quantifiable Pareto choice between timing-channel resistance and latency savings, parameterized by F . We deliberately stop at this analytical bound: it is derived from already-measured distributions under a Gaussian assumption and should be read as an upper bound on achievable resistance, not as an empirical detection result. A controlled detection experiment against a trained adversarial timing classifier, and an evaluation of partial and randomized padding schedules, are reserved for the live-deployment study (Section 5).

4.4.7. Deployment Considerations: Heterogeneous (Linux and Windows) Environments

PAMR as evaluated targets Linux SSH servers, which dominate the opportunistic-intrusion threat model that motivates the work. Two aspects determine its transfer to mixed Linux and Windows estates. The protocol-facing layer (the SSH frontend, Section 3.2.1) and the analytics layer (phase estimator, router) are operating-system agnostic: they operate on command strings and embeddings and would apply unchanged to a Windows command-line surface exposed over SSH or to a PowerShell-over-WinRM frontend. The OS-specific

dependencies are concentrated in two places—the ground-truth response corpus and the complexity deny-list (Section 3.6), and the stage prototypes (Section 3.2.2)—both of which would need a Windows-specific counterpart (e.g., PowerShell and cmd.exe command–response pairs, and Windows-oriented persistence patterns such as registry run-keys, scheduled tasks and service creation). The practical implementation path is therefore as follows: (i) keep the four-layer architecture intact; (ii) add a Windows command corpus and prototypes; (iii) re-run the offline calibration of Algorithm 1 per platform, yielding one dispatch table per shell family; and (iv) route by detected shell type at the frontend. The advantage of this approach is reuse of the entire routing and estimation machinery across platforms; the principal limitations are the additional annotation cost for Windows ground truth and the fact that the MITRE-derived stage definitions, while cross-platform in principle, have here been validated only on Linux command streams. A genuinely cross-platform evaluation is out of scope for this paper and is noted as future work.

5. Conclusions

This paper introduced PAMR, a phase-adaptive model routing framework for LLM-driven SSH honeypots that addresses the latency–fidelity trade-off characteristic of uniform-model deployments. The framework combines a lightweight online phase estimator (all-MiniLM-L6-v2, ONNX), which infers the current attack stage from a sliding window of $W = 6$ recent commands, with a router that dispatches each command to one of several heterogeneous LLM backends according to a precomputed 3×3 dispatch table, a complexity score and a per-phase fidelity floor ($\varphi_{\min}$), under the confidence-weighted decision rule of Equation (2).

Experiments on a controlled 412-session benchmark showed that PAMR reduces mean response latency by approximately 38% relative to an API-hosted high-capacity uniform baseline, and by approximately 45% relative to a locally hosted mid-capacity baseline (run-level paired t -test $p < 0.001$; session-level paired bootstrap confirming a strictly positive reduction; descriptive session-level Cohen’s $d = 2.1$), while preserving token-level phase-weighted fidelity within five percentage points and maintaining an online macro-averaged stage-classification F1-score of 0.87. Five-fold cross-validation confirms that these gains generalize within the benchmark distribution, with the high-impact dispatch-table cells stable across folds. Ablation analysis demonstrates that no single mechanism dominates: the gains arise from the interaction of the dictionary cache, the per-phase fidelity floors, and the confidence-weighted escalation rule ($\theta = 0.62$). We explicitly state that these results are token-level and within-distribution: they establish a latency/token-fidelity trade-off on a laboratory benchmark, not yet a latency/operational-realism trade-off against live adversaries.

Several directions are open, and we list them in priority order. First and most critical is a controlled deployment against live attackers, designed to measure dwell time and abandonment rate and to establish the external validity of the present results; the same deployment would supply the session-state oracle needed to measure the State-Consistency Rate (SCR) proposed in Section 4.4.3 and to obtain human-expert realism ratings, closing the construct-validity gap between token-level proxies and attacker-perceived realism. Second is the extension of the stage taxonomy from three classes to a finer-grained kill-chain representation (privilege escalation, lateral movement, defence evasion, exfiltration, cleanup), which the $N \times M$ dispatch formulation already accommodates. Third is an empirical study of the timing side-channel analyzed in Section 4.4.6—training an adversarial timing classifier against PAMR and evaluating randomized and partial latency-padding schedules to characterize the resistance/latency Pareto front quantitatively. Fourth is a

genuinely cross-platform (Linux and Windows) evaluation along the implementation path sketched in Section 4.4.7.

Author Contributions: Conceptualization, R.M. and F.U.; methodology, R.M.; software, R.M.; validation, R.M., F.U. and K.A.; formal analysis, R.M.; investigation, R.M. and G.A. (Gulnur Aksholak); resources, F.U.; data curation, R.M.; independent stage-label annotation and inter-annotator validation, F.U.; writing—original draft preparation, R.M.; writing—review and editing, F.U., G.A. (Gulnur Aksholak) and K.A.; visualization, R.M. and G.A. (Guldana Akhmedi); supervision, F.U.; project administration, F.U. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The benchmark and evaluation data used in this study will be made publicly available in a permanent repository. Public release is currently deferred for two reasons. First, the dataset forms the shared empirical basis of two companion manuscripts by the authors, in preparation on related aspects of this line of work (the benchmark itself and the analysis of LLM-driven autonomous agents); to avoid pre-empting those studies, the data will be released once they have completed peer review. Second, part of the stage-annotated data was generated and validated within a controlled university cyber-range exercise and contains participant- and infrastructure-related information, therefore requiring anonymization and institutional security clearance before public release. As evidence of the authors' commitment to open data, two companion datasets from the same research line are already openly available on Zenodo—an LLM-backend benchmarking dataset and code release <https://doi.org/10.5281/zenodo.20816078> (accessed on 8 July 2026) and a cross-vendor AI-agent shell-behaviour corpus <https://doi.org/10.5281/zenodo.20752273> (accessed on 8 July 2026)—and the present benchmark will be released on the same terms once the conditions above are met. In the interim, the benchmark-construction procedure, stage-annotation scheme, complexity-scoring rules and per-fold dispatch tables are documented in full in Sections 3.3–3.6, the dataset can be provided confidentially to the editor and reviewers for verification, and further material may be made available from the corresponding author upon reasonable request, subject to the clearance described above.

Acknowledgments: During the preparation of this manuscript, the authors used ChatGPT (GPT-5.5 Thinking, OpenAI) to translate parts of the initial draft and improve the English language and readability. After using this tool/service, the authors carefully reviewed and edited the content as needed and take full responsibility for the final content of the publication.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Malhotra, P. LLMHoney: A real-time SSH honeypot with large language model-driven dynamic response generation. *arXiv* **2025**, arXiv:2509.01463.
2. Magazov, R.; Uralova, F.; Aksholak, G.; Abeshev, K. Command pattern recognition in Linux-based cyber incidents using large language models for intelligent honeypot systems. In Proceedings of the 2026 18th International Conference on Electronics, Computer, and Computation (ICECCO), Kaskelen, Kazakhstan, 10–11 April 2026.
3. Oosterhof, M. Cowrie SSH/Telnet Honeypot, Version 2.5.0 Documentation. Available online: <https://docs.cowrie.org/> (accessed on 8 July 2026).
4. Pahal, S.; Priya, P. A comprehensive research study on low-interaction SSH honeypot. *Mapana J. Sci.* **2022**, *21*, 99–118. [CrossRef]
5. Franco, J.J.; Aris, A.; Canberk, B.; Uluagac, A.S. A survey of honeypots and honeynets for IoT, IIoT, and cyber-physical systems. *IEEE Commun. Surv. Tutor.* **2021**, *23*, 2351–2383. [CrossRef]
6. Javadpour, A.; Ja'Fari, F.; Taleb, T.; Shojafar, M.; Benzaid, C. A comprehensive survey on cyber deception techniques to improve honeypot performance. *Comput. Secur.* **2024**, *140*, 103792. [CrossRef]
7. Morić, Z.; Dakić, V.; Regvart, D. Advancing cybersecurity with honeypots and deception strategies. *Informatics* **2025**, *12*, 14. [CrossRef]
8. Otal, H.T.; Canbaz, M.A. LLM honeypot: Leveraging large language models as advanced interactive honeypot systems. *arXiv* **2024**, arXiv:2409.08234.

9. Sladić, M.; Valeros, V.; Catania, C.; García, S. LLM in the shell: Generative honeypots. In Proceedings of the IEEE European Symposium on Security and Privacy Workshops (EuroSPW), Vienna, Austria, 8–12 July 2024.
10. Wang, Z.; You, J.; Wang, H.; Yuan, T.; Lv, S.; Wang, Y.; Sun, L. HoneyGPT: Breaking the trilemma in terminal honeypots with large language models. *arXiv* **2024**, arXiv:2406.01882.
11. Fan, W.; Yang, Z.; Liu, Y.; Qin, L.; Liu, J. HoneyLLM: A large language model-powered medium-interaction honeypot. In Proceedings of the ICICS, Mytilene, Greece, 26–28 August 2024; pp. 253–272.
12. Volkov, D. LLM agent honeypot: Monitoring AI hacking agents in the wild. *arXiv* **2024**, arXiv:2410.13919.
13. Bridges, R.A.; Mitchell, T.R.; Muñoz, M.; Henriksson, T. SoK: Honeypots & LLMs, more than the sum of their parts? *arXiv* **2025**, arXiv:2510.25939.
14. Crespi, V.; Hardaker, W.; Abu-El-Haija, S.; Galstyan, A. Identifying botnet IP address clusters using NLP techniques on honeypot command logs. *arXiv* **2021**, arXiv:2104.10232.
15. Liu, Z.; Buford, J. Anomaly detection of command shell sessions based on DistilBERT. *arXiv* **2023**, arXiv:2310.13247.
16. Hobert, K.; Lim, C.; Budiarto, E. Enhancing cyber attribution through behavior similarity detection on Linux shell honeypots with ATT&CK framework. In Proceedings of the IEEE International Conference on Cryptography, Informatics, and Cybersecurity (ICoCICs), Bogor, Indonesia, 22–24 August 2023. [CrossRef]
17. Alasmary, H.; Anwar, A.; Abusnaina, A.; Alabduljabbar, A.; Abuhamad, M.; Wang, A.; Nyang, D.; Awad, A.; Mohaisen, D. ShellCore: Automating malicious IoT software detection by using shell commands representation. *arXiv* **2021**, arXiv:2103.14221.
18. Dong, S.; Su, H.; Xia, Y.; Zhu, F.; Hu, X.; Wang, B. A comprehensive survey on authentication and attack detection schemes that threaten it in vehicular ad-hoc networks. *IEEE Trans. Intell. Transp. Syst.* **2023**, *24*, 13573–13602. [CrossRef]
19. Chen, L.; Zaharia, M.; Zou, J. FrugalGPT: How to use large language models while reducing cost and improving performance. *arXiv* **2023**, arXiv:2305.05176.
20. Shazeer, N.; Mirhoseini, A.; Maziarz, K.; Davis, A.; Le, Q.; Hinton, G.; Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv* **2017**, arXiv:1701.06538.
21. MITRE Corporation. MITRE ATT&CK: Adversarial Tactics, Techniques, and Common Knowledge. 2025. Available online: <https://attack.mitre.org/> (accessed on 2 June 2026).
22. Landis, J.R.; Koch, G.G. The measurement of observer agreement for categorical data. *Biometrics* **1977**, *33*, 159–174. [CrossRef] [PubMed]
23. Papineni, K.; Roukos, S.; Ward, T.; Zhu, W.J. BLEU: A method for automatic evaluation of machine translation. In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia, PA, USA, 7–12 July 2002; pp. 311–318.
24. Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; et al. Transformers: State-of-the-art natural language processing. In Proceedings of the EMNLP (System Demonstrations), Online, 16–20 October 2020; pp. 38–45.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.