

SoK: Honeypots & LLMs, More Than the Sum of Their Parts?

| | | | |
|-----------------------------------|------------------------------------|--------------------------------|--------------------------------|
| 1 st Robert A. Bridges | 2 nd Thomas R. Mitchell | 3 rd Mauricio Muñoz | 4 th Ted Henriksson |
| <i>AI Labs</i> | <i>Security R&D</i> | <i>AI Labs</i> | <i>AI Labs</i> |
| <i>AI Sweden</i> | <i>Volvo Group</i> | <i>AI Sweden</i> | <i>AI Sweden</i> |
| <i>Gothenburg, Sweden</i> | <i>Gothenburg, Sweden</i> | <i>Gothenburg, Sweden</i> | <i>Gothenburg, Sweden</i> |
| <i>robert.bridges@ai.se</i> | <i>thomas.mitchell@volvo.com</i> | <i>mauricio.munoz@ai.se</i> | <i>ted.henriksson@ai.se</i> |

Abstract—The advent of Large Language Models (LLMs) promised to resolve the long-standing paradox in honeypot design, achieving high-fidelity deception with low operational risk. Through a flurry of research since late 2022, steady progress from ideation to prototype implementation is exhibited. Since late 2022, a flurry of research has demonstrated steady progress from ideation to prototype implementation. While promising, evaluations show only incremental progress in real-world deployments, and the field still lacks a cohesive understanding of the emerging architectural patterns, core challenges, and evaluation paradigms. To fill this gap, this Systematization of Knowledge (SoK) paper provides the first comprehensive overview and analysis of this new domain. We survey and systematize the field by focusing on three critical, intersecting research areas: first, we provide a taxonomy of honeypot detection vectors, structuring the core problems that LLM-based realism must solve; second, we synthesize the emerging literature on LLM-powered honeypots, identifying a canonical architecture and key evaluation trends; and third, we chart the evolutionary path of honeypot log analysis, from simple data reduction to automated intelligence generation. We synthesize these findings into a forward-looking research roadmap, arguing that the true potential of this technology lies in creating autonomous, self-improving deception systems to counter the emerging threat of intelligent, automated attackers.

Index Terms—honeypot, large language model, LLM, cyber deception, systematization of knowledge, threat intelligence

1. Introduction

Cybersecurity honeypots—decoy systems designed to lure and analyze attackers—have long grappled with a fundamental design paradox. The spectrum of honeypots ranges from “low-interaction” systems, which are safe and easy to deploy but simplistic and easily detected, to “high-interaction” systems, which offer high-fidelity engagement at the cost of significant risk and management overhead. For decades, the central goal of

honeypot research has been to resolve this tension: to achieve the high fidelity of a real system with the low risk and minimal effort of a simulated one.

The recent advent of Large Language Models (LLMs) appeared to offer a silver bullet. With their uncanny ability to generate convincing text, code, and shell interactions, LLMs promised a paradigm shift for honeypots. The vision was clear: low-interaction decoys could finally be imbued with dynamic, adaptive, and believable personalities, luring attackers into revealing their tools and techniques.¹

This vision spurred a flurry of research over the last two years seeking to integrate LLMs into existing honeypot frameworks. As this work was often performed concurrently, many of these formative papers do not cite or build directly upon each other, creating a fragmented landscape. A comprehensive understanding of how this new field is developing is needed to identify its fundamental limitations and most promising directions. To address this, we provide the first systematic review of LLM-powered honeypots, uniquely situating it between surveys of two critical, flanking topics: honeypot fingerprinting, to understand the core problems of realism, and honeypot data analysis, to understand the ultimate goal of intelligence generation. The goal of this Systematization of Knowledge (SoK) paper is to provide the necessary foundational understanding of this burgeoning research area and to accelerate the development of effective LLM-powered defenses.

This paper is organized as follows. Section 2 provides background on cybersecurity and honeypots for readers seeking more context. Section 3 contains our three core systematizations. Section 4 presents our synthesized takeaways and 5 is a forward-looking research roadmap.

1.1. Related Works & Our Contributions

To situate our work, we conducted a review of recent surveys and Systematization

1. While our focus is on enterprise network server emulation, researchers have explored other creative applications, including honeypots for IoT devices [1], systems to misdirect web scrapers [2], and chatbots designed to waste the resources of email and phone scammers [3, 4].

of Knowledge (SoK) papers. Our search on Google Scholar for the terms ("survey" OR "systematization of knowledge" OR "SoK") AND "honeypot" AND ("Large Language Model" OR "LLM") from 2024 to the present yielded no prior SoKs on this specific topic.

The closest works are broader surveys on the impact of LLMs on cybersecurity [5, 6, 7]. While these papers acknowledge honeypots as a potential application, they are discussed only briefly within a much larger context. Our work is therefore complementary, providing a deep and focused systematization solely on this intersection.

The most relevant prior survey on modern deception technology is by Javadpour et al. [8]. This comprehensive work offers a detailed review of cyber deception literature from 2008-2023, establishing a clear baseline for the pre-LLM era.

Contributions. Our work distinguishes itself from these prior surveys not only through its focused scope but by providing several novel systematizations that structure this emerging field. While Javadpour et al. provide a definitive look at the classic era of deception technology, our work provides the first systematic treatment of the LLM-driven era by contributing the following:

- We provide a foundational framework for how honeypots are detected, which systematizes the core problems that LLM-based realism must solve, a perspective not offered in prior work.
- We synthesize the literature on LLM-powered honeypots into an emerging canonical model, identifying the common architectural patterns (e.g., filter/router logic, state management) that have rapidly evolved.
- We chart the progression of honeypot log analysis through four distinct phases of maturity, from simple data reduction to the modern goal of operationalizing intelligence with agents.
- Finally, we synthesize these findings into a forward-looking research roadmap, identifying high-impact directions such as the need for dynamic evaluation ecosystems and autonomous, self-improving feedback loops.

By providing these frameworks, our SoK offers a structured, multi-faceted understanding of the challenges, architectures, and future trajectory of LLM-powered honeypots, a contribution distinct from both the broad cybersecurity surveys and the pre-LLM deception literature.

1.2. Summarized Findings

Our systematization reveals several key findings regarding the current state of the art. Historical honeypot fingerprinting principles often still apply. Even

with LLM simulation, fundamental limitations will give these systems away to a reasonably intelligent adversary.

Architecturally, we identify an emerging canonical model for LLM-honeypots that incorporates components for filtering, state management, and generation, but we note a significant gap in open-source tooling. We find the field has developed a valuable trichotomy of evaluation criteria and methodologies: deceptiveness (human evaluation), accuracy (statistical), overall (real-world deployments). Yet there persists an evaluation paradox: while multiple methodologies exist, their utility is hampered by a real-world “data desert” of low-sophistication traffic, making it difficult to measure progress against meaningful threats. This leads to our finding that the appropriate target for these systems is not skilled humans, whom they are unlikely to deceive, but the emerging threat of sophisticated automated agents. Finally, we find that while automated threat intelligence generation is now possible, it remains in a nascent, proof-of-concept stage, primarily blocked by the lack of large-scale, labeled training data.

Based on these findings, we propose a forward-looking research roadmap centered on the development of autonomous and self-improving deception systems. Architecturally, we highlight the need for open-source, modular frameworks and research into efficient, privacy-preserving models using techniques like LoRA. We advocate for the creation of a dynamic adversarial research ecosystem to solve the “data desert” problem and enable meaningful evaluation against the true target: sophisticated, automated attackers. Finally, we outline a path toward true autonomy through the implementation of feedback loops, enabling systems to automatically generate intelligence for security operations centers (SOCs) and reconfigure themselves to adapt to the evolving threat landscape.

2. Prerequisites: Security, Cybersecurity, Honeypots, & LLMs

The goal of security—viewed broadly, be it for a bike, a home, or a network of computing assets—is to control access, ensure availability, and prevent harm. The practice of security is to create asymmetry, essentially making a breach prohibitively difficult. Implicit in this definition is the realization that no system is 100% secure; for example, in a building, adding doors, locks, thicker walls, and alarm systems cannot guarantee the security of its contents, but may make a successful attack prohibitively difficult or costly.

2.1. Network Cybersecurity Background

Cybersecurity is commonly defined as ensuring the confidentiality, integrity, and availability of data

and networked computing assets. The practice of cybersecurity begins with strategically designing a network with a good security posture (e.g., protecting important assets via layers of defense with increasing restrictions, employing the principle of least privilege). This is complemented by a “defense-in-depth” strategy, which uses many heterogeneous tools for prevention (e.g., firewalls, anti-malware), detection (e.g., signature-based or anomaly-based detectors), and management (e.g., vulnerability scanning) [9]. The overall effect is that well-defended networks enjoy protection against most previously seen attack patterns. This provides asymmetry, as attackers who use known methods have a decreased probability of success and are forced to develop or acquire new attack patterns, which is presumably more costly.

Note that in network audit logs, it is often very difficult and resource-intensive to distinguish malicious from benign activity, except when known attack patterns are flagged by automated detectors. This is partly because widespread alert and logging capabilities produce an enormous volume and variety of data that obscures the activity of a sophisticated attacker. Individual log entries often lack sufficient information, so building a picture of network activity from heterogeneous logs requires a combination of domain expertise and network-specific knowledge—often a slow, labor-intensive process [9, 10, 11, 12, 13, 14, 15].

Automation has aided both attackers and defenders. The ease of access to attacking capabilities and the use of automated attacks have increased the scale of threats [16, 17], which in turn decreases the cost for the attacker. On the other hand, automated prevention techniques remain effective against these known, high-volume attacks. This landscape presents two clear questions for defenders: “How can we continually learn new attack patterns?” and “How can we deter attackers from targeting our network?”

2.1.1. Cyber Attack Modeling Frameworks.

Frameworks for understanding cyber attacks provide a common language to facilitate attack identification, detection, and threat intelligence sharing. The term TTPs (Tactics, Techniques, and Procedures) is common in attack analysis. In this context, *tactics* are high-level adversarial objectives (e.g., Reconnaissance). *Techniques* are the methods used to achieve a tactic (e.g., Port Scanning). Finally, *procedures* are the specific commands or code used to implement a technique (e.g., a specific `nmap` command).

The Lockheed Martin Cyber Kill Chain [18] was influential in framing attacks as logical sequences of tactics. More commonly used today is the MITRE ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) framework [19]. Released in 2015, MITRE ATT&CK is a globally accessible knowledge base detailing adversary behavior organized into 14 tactics, each with numerous techniques and documented procedures. Unlike the Kill Chain’s

linear model, MITRE ATT&CK provides a broad matrix of TTPs based on real-world observations. This organized language allows for the labeling and mapping of attack patterns, and the infrastructure to document and share them assists with attribution and cyber defense.

2.2. Honeypot Background

Honeypots are decoy computing systems designed to enhance security by gathering threat intelligence and occupying attacker resources. Consequently, honeypots have been a defensive tactic in cybersecurity since at least the 1980s [20, 21]. By convincing unwitting attackers to interact with them, honeypot systems allow for the direct observation of attackers’ actions, which in turn informs defensive mechanisms. This obviates the tedious process of finding malicious activity in a sea of benign system logs. Furthermore, honeypots are generally isolated environments designed so they can be attacked with no risk to production network assets.

The existence of honeypots also serves to slow cyber attackers. Novel attack patterns are presumably more expensive to develop than reusing existing tools. The potential presence of honeypots incentivizes an attacker to ensure they are not interacting with a decoy before using a valuable or secret technique. Notably, scripts to identify common honeypots exist [22], so even automated attacks may avoid uncloaked decoys. It is believed that attackers will not only seek to detect honeypots but will also attempt cheaper, known attacks before risking a more expensive, novel one [23]. All of this slows the attacker, benefiting the defender.

2.2.1. Honeypot Goals & Levels of Simulation.

Honeypots are designed to (1) occupy attacker resources and (2) accelerate threat intelligence (e.g., learn TTPs or reduce the time to detect a breach).

If an attacker is interacting with a honeypot, they are, for that time, occupied and not attacking a real host. This defensive tactic can be the sole goal of a honeypot. For example, CloudFlare uses generative AI to create decoy web networks to occupy malicious web-scraping bots [2]. Similarly, researchers have used AI to create fake Active Directory users [24] or to automatically engage with email and phone scammers to waste their time [3, 4].

In addition to occupying attackers, honeypots are also used to lower the time-to-detection for a cyber breach. Often called “canaries,” these honeypots are designed to entice attacker interaction to discover an active breach faster, thereby preventing further damage [16, 25].

Honeypots are also used to gain threat intelligence by directly observing attacker TTPs. Examples include honeypots that capture malware [26], distributed networks that gain a panoramic view of attack trends [27], and recent designs intended

to identify and counter LLM-powered offensive agents [28, 29].

With these goals in mind, honeypots mimic servers with varying degrees of realism and risk. They are conceptualized on a spectrum from low-interaction to high-interaction. Low-interaction honeypots are almost fully simulated with limited capabilities. Because they have little real connectivity, they pose minimal risk to the defender’s network, but they lack the realism to entice sophisticated attack behavior. Medium-interaction honeypots have some real functionality but are restricted to limit risk. High-interaction honeypots are generally real systems, which require more resources to deploy and entail greater risk. Many templates for these honeypot types are open-sourced [30]. In general, lower interaction means faster deployment and less risk, but at the cost of being less convincing. Experts have stated that low-interaction honeypots will likely never convince a skilled human attacker [31], and our survey (Section 3.1) shows there are many ways to detect a simulated system. The vision driving the integration of LLMs into honeypot systems is to achieve the fidelity of a high-interaction honeypot with the speed, ease, and low risk of a lower-interaction system.

2.2.2. Honeypot Necessities. Overall, we posit two observations. Firstly, for a honeypot to succeed, it must be sufficiently convincing to elicit attacks in the wild. Secondly, if one seeks to gain threat intelligence, the data produced must be efficiently converted into actionable information, meaning the cost to the defender must be less than the value of the intelligence gained.

2.3. LLM Background

Large Language Models (LLMs) are generative machine learning models that produce text and possess an extremely broad knowledge base. They use the transformer architecture [32], which utilizes attention mechanisms to effectively capture long-range dependencies between tokens. Tokens can be understood as generalizations of words or sub-words.

LLMs typically undergo at least two primary training phases. The initial phase, pre-training, involves training the model on next-token prediction using vast amounts of text data. The second phase, fine-tuning, adapts the model using curated input-output text pairs, where inputs might be questions or instructions, and outputs are the desired responses. This two-stage process enables LLMs to perform a wide array of tasks, such as text generation, summarization, and code writing.

Prominent examples include OpenAI’s GPT series [33], Meta’s Llama family [34], and Google’s Gemini models [35]. Furthermore, specialized LLMs are being developed for security applications, such as the offensive security-focused WhiteRabbitNeo

[36] and research into cybersecurity-tuned LLMs [37]. The rapid expansion of LLM research is documented in various surveys [38, 39, 40]. LLMs are often augmented with tools, such as a web search capability or a Python interpreter, that allow for greater functionality. Of particular use are retrieval-augmented generation (RAG) systems, in which an LLM queries a vector store database to gain extended context.

The ability of LLMs to rapidly generate content that deceives humans has ignited exploration of their use in cybersecurity, both for offensive purposes [16, 17, 28, 29, 41] and, as examples discussed in Section 3.2, for defensive ones. A clear dichotomy exists in use between large proprietary models accessible via API for usually per-token fees and open-weight LLMs that can be run locally, but are generally less capable.

3. Related Fields

In this section we survey three areas, namely, honeypot fingerprinting methods, LLM shell honeypots, and honeypot data analysis methods. In each, we organize the works into either categories or developing trends to admit a clearer understanding of the area.

3.1. Honeypot Detection Vectors

Because effective camouflage is necessary for honeypot success, we survey honeypot detection (or fingerprinting) methods not as an exhaustive list or research works in the area, but to identify the fundamental vectors an adversary can exploit to expose a decoy. These vectors form the basis of our systematization and provide a roadmap for where LLMs can offer the most significant improvements. We found four categories of detection vectors for honeypots, each described in a subsection below.

3.1.1. Contents & Network Posture. Inconsistencies in the honeypot’s claimed identity versus its observable properties are a common giveaway. This includes unrealistic ports, services, and banner messages. Similarly, the presence of unrealistic or default data within the system is another key indicator. This ranges from default file system layouts and user accounts in open-source honeypots to a general lack of plausible, lived-in content, such as applications and file contents.

Early detection methods focused on probing for environmental and network inconsistencies. The early work of Provos [42] created a honeypot network named Honeyd. Provos took care to camouflage the system’s fingerprint from scanning tools like NMAP, as well as adjusting outgoing packets (e.g., ISN, IP, and TCP characteristics) to maintain consistency. Subsequent research quickly found new inconsistencies with honeypots. Holz & Raynal

showed that simple Bash commands or hardware details (e.g., MAC addresses) could reveal a simulated environment [43]. Defibaugh-Chavez et al. [44] illustrate many methods of honeypot detection, namely, exercising many different services (identifying ones that should exist but either do not or are unconvincing) and examining packet contents. These red flags have been reported for almost 20 years.

Recent works demonstrate that honeypots often expose an unrealistic network posture, such as running too many disparate services (e.g., multiple databases purportedly running on one IoT device), having mismatched service banners, or possessing insufficient resources (memory, disk space) for the machine they claim to be [45, 46].

A particularly effective and simple vector is the use of default configurations in open-source honeypots, which creates thousands of easily identifiable decoys in the wild [31, 47, 48]. For example, Morishita et al. [47] crafted 20 signatures to identify default responses from open-source honeypots and discovered over 19,000 honeypots in the wild. Cabral et al. [49, 50] have shown empirically that mitigating these static configuration issues by changing banners, file systems, and network services significantly improves evasion. Their study investigates Cowrie [51], a popular open-source, medium-interaction SSH and Telnet honeypot, focusing on how configurations can lead to detection. The follow-on work provides tips on how to configure Cowrie to evade detection, such as changing banners, file system contents, network interactions, and the reported software and OS descriptions. Guan et al. [52] identify four types of problems with LLM honeypot responses: the LLM does not know the input command; the LLM responds with an incorrect format; the LLM provides an unbelievable or incorrect simulated response; and the LLM responds in a manner inconsistent with history. Vetterl & Clayton [53] show that open-source honeypots’ implementations of SSH, TCP, and HTTP protocols can be detected through protocol-specific probing and analysis. Their detection method shows that camouflaging honeypot protocols is necessary to avoid known fingerprinting methods.

3.1.2. Outputs & Behavior. The second major category of fingerprinting stems from outputs that are themselves incorrect or unrealistic, or meta-data associated with the output that is unbelievable. This includes unbelievable TCP/IP characteristics, detectable time delays in communications, or other anomalies, e.g., the `top` command producing a static, nonuniformly spaced table instead of a continuously updated one.

Attackers can identify honeypots not by what they are, but by how they behave. Defibaugh-Chavez et al. [44] also noted that simulated systems often respond much slower to simple ICMP (ping) requests. This timing-based fingerprinting was later formalized to identify Honeyd deployments with high ac-

curacy, as their packet link latencies could differ by an order of magnitude from real systems [54]. This concept was further extended by applying machine learning classifiers to timing statistics from ICMP and TCP/IP to automatically flag decoys [55].

Moving to LLM-based honeypots, timing remains a critical and unresolved challenge [56, 57, 58]. Further, Wang et al. [59] find that “Common impediments such as unsupported tools, insufficient privileges, or the absence of targeted information typically result in the termination of the attack engagements. Therefore, a higher command execution success rate is critical.” This is echoed by Ragsdale & Boppana [57], who cite interactive commands like `Vim` as a pitfall for honeypots. Furthermore, Ragsdale & Boppana, Wang et al., and Guan et al. [52] identify potential vulnerabilities for LLM-based honeypots, particularly denial-of-service via input flooding and prompt injection attacks.

3.1.3. Functional Limitations. A third, more fundamental vector is the exploitation of a honeypot’s inherent functional limitations. This includes simple user interaction problems, such as failed shell tab-completion, and extends to more substantial limitations, like restricted outbound network traffic.

The first documented detection technique of this kind was observed in the wild by Krawetz [60] in 2004 while studying email spammers. The spammers required detection of proxy servers that they used to hide their IP. Their technology, called Honeypot Hunter, used a suspected proxy server to connect back to the attacker’s own host to see if the response was realistic; if the proxy server claimed a successful connection, but none was observed, then the proxy was identified as a honeypot. Krawetz notes that the Honeypot Hunter detection methodology is itself detectable and predicts a cat-and-mouse style progression of detection methodologies.

Such avenues for detection persist for LLM-powered honeypots. Ragsdale & Boppana [57] identify external connections as a detection vulnerability, as LLMs can simulate successful external communications but have no means of actually performing the connection. “Such scenarios include starting a session with a C&C [command and control] server, downloading malware, or exfiltrating data. Since no traffic is generated, the attacker can determine that their commands are not being executed and that they are in a honeypot.” This principle of “egress control” detection has been leveraged in various ways. Botnets can identify honeypots by checking if a compromised host can be used to launch further attacks [61], and firewalls designed to neuter a honeypot can be detected when they block outbound traffic that a real system would permit [45].

3.1.4. Synthesizing Multiple Features. Finally, modern approaches synthesize all these vectors using machine learning. Huang et al. [62] gathered features of a suspect honeypot and used SVM, k -NN, and

Naive Bayes classifiers to identify honeypots. The features used were based on protocols (e.g., whether services exist), network flow (e.g., average TTL), and system characteristics (e.g., ICMP response times). To obtain labeled data, the authors used online internet scanning and honeypot identification sites and built features from real servers. The approach exhibited high detection accuracy. Srinivasa et al. [48] searched for honeypot identifiers from previous works and signatures from Morishita et al. [47] to identify open-source honeypots in the wild, finding over 21,000. The authors emphasize that these techniques target low- to medium-interaction honeypots only.

Ilg et al. [63] present *Beekeeper*, an LLM-driven system to analyze honeypot realism. Notably, they found that attempted downloads and use of files often gives away honeypots, as the decoys simulate but do not have actual functionality. This work shows not only that LLMs can be used to identify honeypot weaknesses, but that LLMs (and by extension, LLM-powered attackers) can fingerprint honeypots automatically.

3.2. Honeypots Using LLMs

This section surveys papers proposing methods to advance cybersecurity honeypot technologies using LLMs. To the best of our knowledge, this is a comprehensive survey of research that integrates LLMs into network security honeypots. We identified papers in this field using a Google Scholar search for "LLM" AND "Honeypot" through October 2025. We structure our review thematically to systematize the field’s evolution, showing how foundational concepts matured into fidelity evaluations, architectural refinements, and in-the-wild deployments. Table 1 itemizes and summarizes every work in the area.

3.2.1. Foundational Concepts & Challenges. The concept of using LLMs to create dynamic honeypots emerged in late 2022. McKee & Noever [64] were likely the first to propose using chatbots (LLMs) in honeypot environments. They itemized ten honeypot-related tasks and demonstrated (in December 2022) ChatGPT’s ability to perform them with straightforward prompting, thereby illustrating the diverse potential capabilities LLMs offer for honeypots.

Following this proposal, the first implementation was presented by Sladić et al. [56], who integrated an LLM into a Linux shell honeypot. Their system, *ShellLM*, uses OpenAI’s GPT-3.5-turbo-16k API to simulate shell responses, passing each command along with the complete command history to the LLM. In their evaluation, a user study showed that human experts often could not identify the simulated responses. However, in building the first system, they also discovered the foundational challenges

that would define subsequent research: LLM response latency, potentially insufficient context window lengths, and the stochastic nature of LLM outputs, which could lead to inaccuracies. Notably, the authors also publicly released the Prague Dataset, a small dataset of real shell sessions.

3.2.2. LLM Shell Simulation Unit Testing. A lineage of works focused primarily on “unit-tests” (a term coined by Sladić et al. [73]) of LLMs’ abilities to accurately simulate real computer interaction. Many different metrics and experimental methods were developed.

To improve the quality of simulated responses, Otal & Canbaz [66] investigated fine-tuning Llama3-8B on shell command data from Cowrie and empirically verified that the fine-tuning resulted in outputs more similar to Cowrie’s. Weber et al. [65] conducted an experiment where five Computer Science graduate students graded GPT-3.5’s responses for believability (binary) on over 1,400 unique request-response pairs, encompassing 230 different base commands. They concluded that only 52% of the generated responses were convincing. They find that long and compound commands are problematic for LLMs and discuss command features with a high likelihood of producing unbelievable outputs. Their results indicate that the cosine similarity of SBERT text embeddings between simulated and real responses strongly predicts believability, suggesting a path beyond simple string metrics. Malhotra [71] introduces *LLMHoney*, another shell honeypot system, and instantiates it for comparative testing with 13 open-weight LLMs. Using 138 “representative shell commands,” Malhotra evaluates each instantiation using numerous metrics to gauge fidelity, latency, hallucination rate, and memory footprint. He found that latency and memory increase with model size, but fidelity suffers with small models. Christli et al. [68] also used Llama 3 to simulate shell responses, evaluating response fidelity against a real system with similarity metrics.

Fan et al. [58] propose seven metrics for evaluating the fidelity of simulated shell responses. Leveraging these, they tested many powerful LLMs requiring API calls, evaluating them for the fidelity, latency, throughput, and cost of their shell response emulation. Based on their findings, they concluded GPT-4o is best overall and used a Raspberry Pi to implement a simple architecture for filtering problematic inputs, passing only acceptable inputs to the LLM API for server emulation. They observed a higher frequency of longer sessions (in terms of clock time) for GPT-4o over the non-LLM honeypots Cowrie and Amun [75].

3.2.3. Architectures Advancement & Real-World Deployments. In parallel, meaningful architectural advancements for honeypot security, practicality, and cost were also pioneered. The foundational challenges of latency and cost led to a critical archi-

Table 1. LLM-POWERED SHELL HONEYPOT WORKS

| Paper | Type | Core Contribution | LLMs Tested | Evaluation Technique |
|--------------------------------|---|--|---|---|
| McKee & Noever [64] (2023) | General (OS, App, Network) | The first conceptual proposal for using LLMs for a variety of honeypot-related tasks, demonstrated via prompting. | ChatGPT (Dec 2022) | Conceptual Prompting (In Lab) |
| Ragsdale & Boppana [57] (2023) | Shell (SSH, Telnet) | Proposed novel methods for efficiency, including intelligent context pruning and caching deterministic responses for the shell. Presented the first LLM-integrated shell honeypot (<i>ShellLM</i>). Identified foundational challenges like latency and context limits. Released the Prague Dataset. | GPT-3.5 | Statistical (Levenshtein, In Lab); Script Replay (Campaign Length, In Lab) |
| Sladić et al. [56] (2024) | Shell (SSH) | Presented the first LLM-integrated shell honeypot (<i>ShellLM</i>). Identified foundational challenges like latency and context limits. Released the Prague Dataset. | GPT-3.5 | Human Study (12 Experts, In Lab) |
| Wang et al. [59] (2024) | Shell (SSH, Telnet) | Developed a sophisticated Prompt Manager to track system state in a shell honeypot. Introduced a hybrid architecture and a suite of new metrics. | GPT-3.5, 4 | Script Replay (New Metrics, In Lab) ; Comparative Engagement (In the Wild) |
| Guan et al. [52] (2024) | Shell (SSH, Telnet) | Systematized the filter/router architecture for shell honeypots to handle scanners and manage costs. Provided strong evidence for chain-of-thought prompting. | GPT-3.5, 4o, Claude-2, 3 Haiku, 3 Opus, Llama 2 (70B) | Script Replay (Fidelity, In Lab); Statistical (Session Length, In the Wild) |
| Weber et al. [65] (2024) | Shell (SSH) | Conducted a critical analysis of shell fidelity limitations, identifying specific failure modes. Found only 52% of responses were convincing. | GPT-3.5 | Human Study (5 Experts, In Lab); Statistical (SBERT Similarity, In Lab) |
| Otal & Canbaz [66] (2024) | Shell (SSH) | Investigated fine-tuning an open-weights LLM on shell data to improve the quality of simulated responses. | Llama3-8B | Statistical (Similarity, In Lab) |
| Fan et al. [58] (2024) | Shell (SSH) | Proposed seven fidelity metrics for shell simulation. Evaluated multiple commercial LLMs, concluding GPT-4o was superior for the task. | GPT-4, 4o, Gemini Pro 1.5, Claude 3 Opus, Mistral 7B | Statistical (Session Time, In the Wild) |
| Johnson et al. [67] (2024) | Shell (SSH) | Presents a modular architecture (<i>LIMBOSH</i>) with prompt injection mitigation and a separate LLM context for output fidelity. | GPT-4o | Human Study (4 Experts, vs. real server, In Lab) |
| Christli et al. [68] (2024) | General | Evaluates Llama 3’s shell simulation accuracy against a real system | Llama 3 | Statistical (In Lab) |
| Gizzarelli [69] (2024) | Shell (SSH, MySQL) | Introduced SYNAPSE with automated log-to-MITRE ATT&CK mapping. Performed a comparative human study where SYNAPSE was perceived as more realistic than its static equivalent. | GPT-3.5, 4, 4o, Gemini | Human Study (vs. static, In Lab); Comparative (In the Wild) |
| Badran & Niazi [70] (2025) | Shell (HTTP) | Used a prompted LLM to both label HTTP requests by attack type and generate believable server responses. | GPT-4o | Human Study (10 Annotators, In Lab) |
| Malhotra et al. [71] (2025) | Shell (SSH) | Hybrid architecture with state-aware LLM to balance latency and fidelity used to evaluate many open-weight LLMs for shell simulation. | 13 open-weight LLMs, see text | Statistical (Accuracy, latency, hallucinations, In Lab) |
| Jimenez et al. [72] (2025) | LDAP | Design and implement LDAP honeypot, curate LDAP dataset for fine-tuning, and build evaluation methodology. | Llama 3 (8B) | Statistical, (syntax, structural and content accuracy, completeness metrics; In Lab) |
| Sladić et al. [73] (2025) | Shell (SSH, MySQL, POP3, HTTP) | Architecture identifies protocol, and uses different protocol-specific prompts to LLM | GPT-3.5, 4 | Statistical (unit-tests), human study (89 participants), real-world deployment (In Lab & In the Wild) |
| Safargalieva et al. [74] | Shell (SSH, Telnet, HTTP, FTP, SMTP, IPP, SNMP) | Design and implement LLM-powered honeypot for seven protocols, evaluated for accuracy, deceptiveness, and latency. | 10 models, see text | Statistical and qualitative tests for latency, accuracy (In Lab & In the Wild) |

tectural evolution: the hybrid or filtered honeypot, which leverages a component to manage the volume and content of requests sent to the LLM. This design seeks to use the expensive LLM only when necessary, protect against Denial of Service/Wallet (DoS/DoW) attacks, reduce accumulating context throughout an attack session, and enhance fidelity. Many of these works also contributed to testing shell simulation fidelity and architectural advancement began evaluating the whole system through in-the-wild

deployments.

Ragsdale & Boppana [57], also a pioneering work in the area, investigated GPT models for simulating shell responses, observing limitations similar to those reported by Sladić et al. [56]. They provide novel methods for handling the volume and quality of what is sent to the LLM by intelligently trimming previous commands that do not alter context. This approach demonstrates significant gains in token efficiency. In the same vein, they propose caching

deterministic responses to reduce latency and load on the LLM. To measure accuracy and deceptiveness, they measured per-command fidelity using Levenshtein distance and found the LLM-powered honeypot to be slightly better than Cowrie. Next, they ran sequences of attack commands comprising an attack campaign against the honeypots to quantify how much of the campaign could be completed before a command fails. The LLM-powered honeypots exhibited longer sessions, with Cowrie often failing in the first few commands. This corresponds to Wang et al.’s [59] observation that the inability to provide a valid command response is a major obstacle for low-interaction honeypots.

Wang et al. [59] introduced HoneyGPT, a system designed to address the honeypot “trilemma” of balancing flexibility, interaction, and deception. Its primary innovation is a sophisticated Prompt Manager that orchestrates LLM interaction. To manage costs, the manager prunes the interaction history to prevent exceeding context limits and routes simple commands to traditional emulators or a cache. This is an important advancement in line with developments by Ragsdale & Boppana [57] and Guan et al. [52], as LLM-powered honeypots introduce a denial-of-service (DoS) vulnerability from overloading the LLM. For honeypot evaluation, the authors created metrics to statistically evaluate honeypots, including methods for quantifying command execution success and logic, session length, and attacker response likelihood. Finally, a four-week in-the-wild deployment comparison shows that HoneyGPT achieves a deeper interaction with the attacker than Cowrie and elicits six MITRE ATT&CK techniques that Cowrie did not.

Building on a similar hybrid concept, Guan et al. [52] also considered LLMs simulating shell responses in a honeypot. Like Wang et al., Guan et al. instantiated a honeypot system with a filtering/routing step to prevent scanning activity from engaging the LLM. The filter leverages signatures to identify scanning scripts. Notably, Guan et al. provided statistics showing that a honeypot’s typical query rate exceeds API limits, proving that this DoS vulnerability is naturally exploited by the high rate of attacks. Their filtering logic keeps rates well within these limits. Guan et al. focus in part on prompting and provide guidance and tests for it. Finally, they provided statistics from “in-vitro” (laboratory) tests and an in-the-wild deployment using many different LLMs alongside a Cowrie-only honeypot. For laboratory tests, they constructed 50 attack sessions and tested each LLM-honeypot with a “vanilla” versus a chain-of-thought (CoT) prompt, identifying the session length until an unbelievable output occurs. Across the board, chain-of-thought prompting provides a large boost. From a three-week internet deployment, they found that for effective attacks (those not deemed to be scanning), the average session length was 2.96 commands with

Cowrie, which was statistically significantly smaller than an average of 5.83 commands from the best LLM-powered version.

Johnson et al. [67] present LIMBOSH, a modular software architecture for an LLM shell. It incorporates prompt injection mitigation with input transformations and a random forest prompt injection classifier. Additionally, an output component employs a separate LLM context to ensure fidelity. A single-blind, within-subjects user study involved four cybersecurity professionals red-teaming a real Ubuntu server and LIMBOSH with the GPT-4o model, without prior knowledge that either was a honeypot. Likert-scale questions about each were compared, showing the LLM honeypot exhibited perceived realism.

3.2.4. Expanding the Attack Surface. With the core concepts established in shell-based honeypots, researchers began applying the same principles to other protocols and services.

Gizzarelli [69] introduced SYNAPSE, a dynamic honeypot that simulates both a Linux terminal and a MySQL server using GPT-4o. The system was evaluated through a unique three-part study: a ten-day in-the-wild deployment, an AI vs. AI attack scenario, and a comparative human study. In the human study, 18 participants interacted with both SYNAPSE and its traditional static equivalent, with nearly 80% perceiving the AI-powered version as more realistic. Hu et al. [76] used an LLM in an SQL honeypot to generate responses to attacker queries. Their system, MySQL-Pot, exhibited longer interaction session lengths compared to an analogous non-LLM powered system. Sezgig & Boyacı [77] created a web API honeypot employing a Retrieval-Augmented Generation (RAG) system. This system queries a database of API requests and responses to generate realistic, automated replies. Badran & Niazi [70] tested GPT-4o in responding to HTTP requests, with a system prompt that directs the LLM to first label the request as an attack type and then respond as a web server. In a human evaluation, they found 87% accuracy on attack labels and that 80% of responses were considered convincing.

Jimenez et al. [72] built an LDAP honeypot based on Llama 3 (8B). This paper describes the full process of implementing a honeypot on a lesser-known protocol, including dataset curation, fine-tuning an open-weight LLM, architectural design, and customized evaluations. They find that the base model is poor at LDAP communications, but fine-tuning with a well-curated dataset provides dramatic gains.

Sladić et al. [73] extended their SSH honeypot *shellLM* to the *VellMes* system, now including support for MySQL, POP3, and HTTP protocols. Each protocol is implemented by prompting an LLM using chain-of-thought techniques. Architecturally, the system identifies the protocol from the input and sends the correct prompt and history to the

LLM. Evaluations were performed to test simulation accuracy via statistical unit tests, deceptiveness via an 89-person user study, and efficacy in a real-world deployment. GPT-3.5 and GPT-4 were evaluated.

Safargaliev et al. [74] present OHRA, a web-facing honeypot that fully supports SSH, Telnet, and HTTP, and partially supports several other protocols. Ten LLMs were tested, and the authors found GPT-4o-mini to be best overall. The authors discuss simulation strengths and weaknesses per protocol and interestingly found that latency could be unrealistically slow at times and unusually fast at others.

3.2.5. Open-Source LLM Honeypots. The concepts from this research are being implemented in publicly available tools. Beelzebub is a Cowrie honeypot with LLM simulations that supports TCP and HTTP, allowing a wide range of services to be emulated; in particular, the developers claim “full support for SSH” [78]. Galah is a honeypot project enabling API connections to various LLMs to “dynamically craft relevant responses—including HTTP headers and body content—to any HTTP request” [79]. The T-Pot platform facilitates the deployment of numerous open-source honeypots and includes a dedicated section for LLM-based honeypots, supporting Beelzebub and Galah deployments with Ollama LLMs [80]. The platform’s maintainers note: “We think LLM-Based Honeypots mark the beginning of a game change for the deception / honeypot field.”

3.3. Honeypot Log Analysis

While many studies report on attack statistics from honeypot deployments [81, 82], this survey concentrates on research aimed at automating the transformation of raw honeypot data into actionable threat intelligence. Our systematization reveals a clear evolution in this research, progressing through four distinct phases of analytical maturity. Importantly, the recent automated methods are promising as they enable previously unprecedented attack labeling via fine-tuning or few-shot learning with LLMs.

3.3.1. Foundational Techniques: Data Reduction & Anomaly Detection. The earliest challenge in honeypot analysis was managing the sheer volume of log data. The goal of this initial phase of research was not to fully understand attacks, but simply to reduce the noise and find interesting sessions worthy of a human analyst’s time. Thonnard & Dacier [83] used clustering based on timing characteristics in honeypot data to reduce the large number of individual attacks to a much smaller number of attack clusters suitable for manual analysis. Ghourabi et al. [84] proposed analysis methods for a web service honeypot, using unsupervised techniques to find a manageable subset of activity. They trained a support

vector regressor to predict message sizes and a classifier to predict message classes, flagging messages with significant deviations for manual inspection. More recently, Aslan et al. [85] used Latent Dirichlet Allocation (LDA) to analyze honeypot logs, learning topics of co-occurring commands to reveal patterns, such as the frequent use of `wget` when downloading malware.

3.3.2. Characterizing Attacks: Session Classification & Visualization. This phase moved beyond just flagging anomalies to trying to categorize and understand them. The focus shifted to building tools that could either label entire sessions or help a human explore the data more effectively. Spyros et al. [86], for example, used Dionaea honeypot logs to train multiple classifiers (e.g., AdaBoost, Random Forest) to categorize attack activity as high-impact or low-impact based on features from each session. Others focused on generating outputs for other security tools. Owezarski [87] identified anomalies in network flows within honeypot attack data to infer filtering rules that could later be used by network intrusion detection systems. Another thread of research focused on human-in-the-loop exploration through visualization. Fraunholz et al. [88] proposed a dashboard consisting of multiple visualizations of informative statistics from honeypot data, a theme also explored by others [89, 90]. Mehta et al. [91] also used the ELK stack for visualization but additionally employed machine learning to predict file and folder traversal, thereby forecasting an attacker’s next steps.

3.3.3. From Patterns to Intent: Automated Mapping to MITRE ATT&CK. A major turning point in log analysis was the shift from statistical classification to semantic understanding: translating raw commands into the standardized language of attacker behavior via MITRE ATT&CK TTPs. This marks an evolution from processing raw data to generating threat intelligence.

An early example is the XT-Pot framework from Ryandy et al. [92], which mapped observed attacker activity to ATT&CK techniques, presumably using manually defined rules to create “soft signatures.” Gizzarelli [69] introduced an LLM-powered honeypot, SYNAPSE, which includes a machine learning classifier to automatically map attacker activity to the MITRE ATT&CK framework. When evaluated against real-world attack data, this mapping extension achieved 75% precision and 68% recall.

The work of Boffa et al. shows a clear progression in this area that mirrors the evolution of the natural language modeling community. They first used Word2Vec embeddings of bash commands to create clusters that were then manually annotated with attacker goals [93]. In a later work, they presented LogPrécis, a fine-tuned BERT model that automatically applies ATT&CK tactic labels to each

command in a honeypot session, achieving over 90% accuracy [94].

With the advent of modern LLMs, researchers began using prompting for this labeling task. Ozkok et al. [95] developed system prompts for GPT-4 to explain attack consequences and label logs with ATT&CK techniques, achieving 72.46% accuracy on the latter task. Similarly, Badran & Niazi [70] prompted GPT-4o to apply one of five labels to HTTP requests, finding 87% accuracy. Lanka et al. [96] prompted an LLM to generate plain-text descriptions of TTPs from shell commands, which were then used to build a vector store for attack detection. However, this approach has limits. Daniel et al. [97], in a related problem of labeling SNORT rules, found that while LLMs provide explainable and efficient mappings, traditional ML models “consistently outperform them in accuracy.” This highlights the significant difficulty of the semantic mapping task, a challenge summarized by Jiang et al. [98]: “Mapping real-world behaviors to ATT&CK techniques is a resource-intensive and subjective process, often prone to bias... Effective mapping relies heavily on dataset quality and expert involvement ...”

3.3.4. From Analysis to Action: Operationalizing Intelligence with Agents. The most recent phase of research seeks to move beyond post-facto log analysis and use the intelligence gathered from honeypots to power real-time detection on production systems. Lanka et al. [96] exemplify this frontier with a system that leverages a shell honeypot with a Retrieval-Augmented Generation (RAG) system. After preprocessing, attack commands from the honeypot are stored as vectors in a RAG knowledge base. For real-time detection, commands from a user on a live system are compared against the known malicious commands in the RAG system, with a prompt to GPT-4o crafted for final classification. This system shows a new potential for using agents with shell honeypots for more sophisticated detection. This work connects to a parallel, emerging area of using LLM agents to enhance threat intelligence and detection [99], suggesting a promising future in merging these fields.

4. Discussion & Limitations

Now we synthesize the key takeaways and limitations to LLM honeypots.

4.1. LLM-Honeypot Architecture

Our survey indicates a clear convergence toward a multi-component canonical architecture. Early systems that simply passed commands to a prompted LLM have evolved into more sophisticated designs that add layers for security, efficiency, and fidelity. This architecture typically includes a pre-LLM filter

to block scanners and cache responses, a core LLM engine for generation, and a state manager to track context. Other components include context pruning modules and prompt generators. Figure 1 provides a general overview of the basic architectural components of an LLM honeypot system that have developed in the literature.

Evidence of the need for these more complex architectures is established, yet no comprehensive, easily configurable architecture that implements all these features exists as open source. This is a direct engineering gap hindering reproducible research and rapid community progress.

LLM Tradeoffs. A wide variety of LLMs have been evaluated head-to-head by the current literature, and this provides clear guidelines for users. A fundamental dichotomy exists between using proprietary LLMs, which are generally the largest and most capable models accessible only via API, versus an open-weight model that is usually smaller and weaker but can be run locally. Research suggests the former requires the user to forfeit inference privacy and entails potentially costly API fees and greater latency, but provides greater deceptiveness and fidelity, focusing development on prompt engineering. Using the latter—smaller but open-weight models—allows all computation to happen locally with generally faster response times, but entails less accuracy and believability. When using open-weight models, most research shows worthwhile gains in accuracy and believability after fine-tuning models for specific protocols. Establishing and managing one’s own hardware or cloud infrastructure is the primary cost.

Notably, fine-tuning with Low-Rank Adaptation (LoRA) methods [100] promises exciting possibilities for honeypot deployments requiring open-weight LLMs. As LoRA can be considered a lightweight add-on to the base model, custom LoRA heads can be trained for each protocol to be simulated, essentially providing a fine-tuned model for each protocol while all use the same base model. This allows a single model to be held in memory and quickly used for different types of inference to enhance the attack surface of the honeypot.

4.2. LLM-Honeypot Evaluation

The community has developed an evaluation triad for measuring success: (1) human user studies for deceptiveness, (2) in-lab statistical and script-replay tests for fidelity, and (3) in-the-wild deployments for engagement. This convergence on a robust set of metrics and reusable evaluation methods allows for statistical comparison and rigorous growth in the field. We note that very recent work of Aradi et al. [101] provides a more comprehensive honeypot evaluation framework, including metrics for attack interaction depth and fingerprinting resistance, and

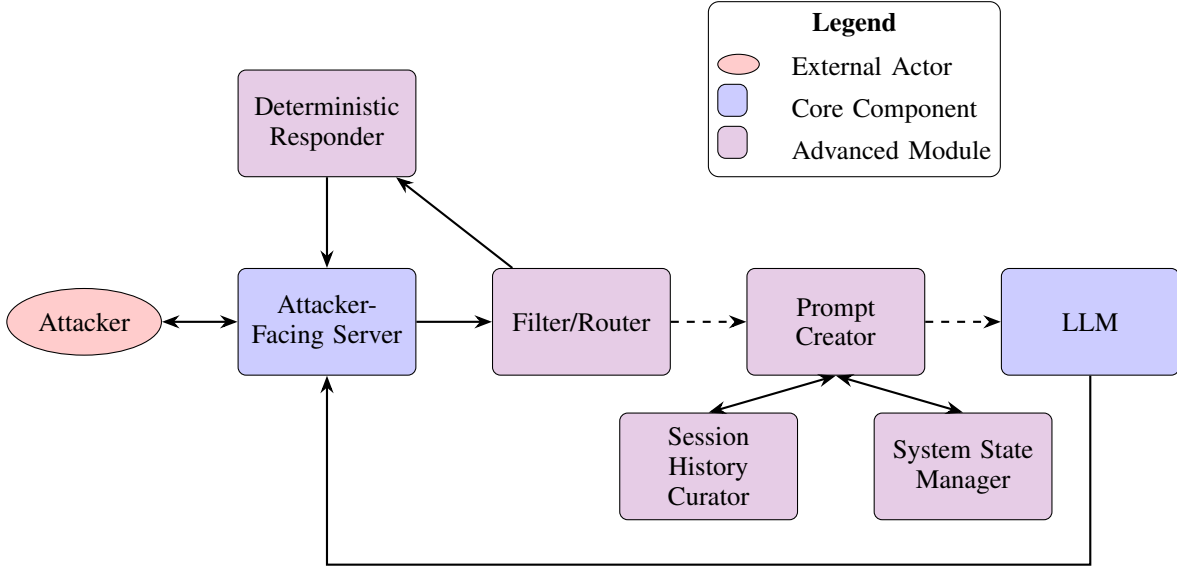


Figure 1. The Canonical Architecture of an LLM-Powered Honeypot. This diagram synthesizes the architectural patterns that have emerged from recent literature. Components are color-coded, with blue nodes denoting core, needed components, and purple nodes advanced/optional components introduced in the literature. An **Attacker** connects to the **Attacker-Facing Server**, which mimics a network service (e.g., SSH). Each command is passed to a **Filter/Router**, a critical component that can prevent prompt injection, Denial-of-Service (DoS), and Denial-of-Wallet (DoW) attacks [52, 59, 67]. The filter can pass the command to a **Deterministic Responder** for simple or cached responses or, for novel interactions, forward it toward an LLM [57, 59]. The **Prompt Creator** component constructs a rich, context-aware prompt for the **LLM**. This can incorporate data from a **System State Manager**, which tracks changes to the virtual environment, and a **Session History Curator**, which intelligently prunes the command history to manage context length [59]. The LLM component itself may represent multiple models or fine-tuned LoRA heads, each specialized for a different protocol. All interactions are logged in the **Honeypot Data Store** (not depicted) for analysis.

are recommended to be considered in future evaluations.

Evaluation of LLM honeypots is hampered by two core problems.

4.2.1. The Data Desert. First, real-world deployments are inundated by a “data desert” of low-sophistication traffic. Guan et al. [52]’s in-the-wild study demonstrated an increase in average session length from just 2.96 commands to only 5.83 with the best-tested LLM, and found that 99.2% of all honeypot activity was scanning scripts. Similarly, Wang et al. [59] found that in two large datasets, only 0.58% and 0.048% of connections resulted in a valid post-login attack session. This highlights a significant gap: current systems show only marginal improvements in a landscape dominated by low-quality attack noise. In short, the attack landscape is dominated by attacks that are too simplistic to reveal the efficacy of LLM honeypots against sophisticated attackers. Nor is the data sufficient for iterative, data-driven development.

4.2.2. No Target Adversary. The pre-LLM honeypot era held the belief that low-interaction honeypots will never trick an intelligent human adversary [23], and despite the hype, our findings extend this belief to current LLM-driven honeypots. This stems from fundamental limitations that cannot be solved by realistic simulation; examples persist throughout honeypot history (see Section 3.1), such as the

inability to convincingly simulate `vim` or `htop` outputs without custom architectures, and the even harder challenge of safely downloading and running an adversary’s file. These findings beg the question, “If humans will not be tricked and bots have limited attack functionality, what is the target adversary of such systems?” Without a clear attacker model, targeted development of next-generation honeypots is not possible.

4.3. Automated Threat Intelligence

The most significant recent breakthrough in honeypot data analysis is the newfound ability to automate the labeling of attacker behavior with MITRE ATT&CK tactics, largely thanks to fine-tuned language models [94]. This marks the “Semantic Leap” from processing raw data to generating structured intelligence. However, this research is still in a nascent stage. The current state-of-the-art, LogPrécis, is a proof-of-concept, limited by the lack of large-scale, well-labeled datasets. A clear need for real-world use is a sufficiently capable labeler, which depends on the creation of a larger and more attack-representative dataset.

We also note the inherent difficulty and subjectivity of ATT&CK mapping [98]; perhaps researchers can find a more appropriate taxonomy of honeypot data for aiding cyber defense. Meanwhile, more advanced concepts, such as using honeypot data to power real-time detection via RAG systems,

are emerging but remain exploratory yet promising [96].

5. Future Research Directions

The takeaways from our systematization point to a clear and actionable research roadmap to address the gaps and capitalize on new synergies.

5.1. Redefining the Target Adversary

Explicitly, the appropriate target for these systems is not skilled humans, but sophisticated automated attackers (LLM-powered bots) [102, 103]. Historically, automated attacks have been pre-scripted and often repeated. However, recent research demonstrates that LLMs can be used to create a new class of powerful, autonomous attacker capable of reasoning, using tools, and adapting its strategy mid-attack [102, 103, 104]. This marks a fundamental shift in the nature of automated threats, moving from brittle scripts to intelligent agents. Further, there is now evidence that LLM-powered attack agents are being used in the wild [29]. This emerging threat creates an urgent imperative for a new generation of high-fidelity honeypots with increased dynamism and believability.

5.1.1. The Adversarial Research Ecosystem.

Moreover, the emergence of intelligent and automated attackers presents an unprecedented opportunity for honeypot research. We therefore propose the development of a dynamic adversarial research ecosystem, a contained platform where LLM-powered attackers and honeypots can be pitted against each other for continuous, co-evolving study. Such an ecosystem promises to solve the “data desert” problem by generating high-quality attack data at scale and provide a testbed for developing meaningful metrics for a honeypot’s two fundamental goals, occupying attacker resources and gathering novel threat intelligence. Notably, labeled attacks could be generated by such attackers, which would permit automated attack labeling techniques to advance from proof-of-concept to viable honeypot components.

5.2. Architectural Needs

Future architectural work should focus on building open-source, modular honeynet frameworks to build complex environments. Significant research is also needed into the use of Small Language Models (SLMs) for privacy- and cost-constrained deployments. This includes developing efficient fine-tuning and distillation techniques. The use of LoRA to enable a single base model to flexibly simulate multiple protocols or perform internal tasks is a particularly promising direction for enhancing capability with minimal overhead.

5.2.1. Unsolved Technical Challenges in Simulation. While modular architectures and efficient models provide a path forward, significant research is still needed to overcome the core technical hurdles of simulation. Our surveys highlight several recurring, unsolved problems that future work must address to enhance realism:

- **Dynamic & Interactive Commands:** Honeypots consistently fail to convincingly simulate commands with continuously updating output (e.g., `top`) or those that require an interactive, stateful application environment (e.g., `vim`) [65].
- **Complex & Compound Commands:** LLMs often struggle to correctly parse and execute long, chained, or complex shell commands, leading to unrealistic error messages or incorrect behavior [65].
- **Outbound Network Actions:** Simulating actions that require real outbound network connectivity, such as downloading a file with `wget` or connecting to a C&C server, remains a fundamental limitation that can quickly expose the honeypot [57].

5.2.2. Hardening the Honeypot Architecture. A driver for the more advanced honeypot architectures has been securing the honeypot itself against prompt injections, flooding attacks, or context overflow. Continued research to identify and mitigate these inherent vulnerabilities will be needed.

5.2.3. Potential Architectural Directions. To extend the canonical LLM-honeypot model (Figure 1) and address ongoing challenges in realism and adaptability, we identify avenues for research into alternative architectures, grounded in recent LLM advancements [32, 38]. These directions merit investigation to develop efficient autonomous honeypots that can engage sophisticated automated attackers [28, 29], subject to empirical validation in cybersecurity contexts.

- Domain-specific tokenizers and RAG/tool-use integrations to improve handling of cybersecurity data and enable dynamic, interactive responses [37, 39, 94].
- Hybrid selective state space models (e.g., Mamba [105]) for scalable state management of extended sessions, reducing latency in resource-limited deployments.
- Hierarchical reasoning models like Sapient’s HRM [106] to facilitate abstracted deception strategies with low data requirements, alleviating the “data desert.”
- Asynchronous multi-agent frameworks, building on honeypot specific systems [107, 108], for parallel execution of duties such as state tracking and adaptive planning.

Pursuing these avenues could enable autonomous, self-improving deception systems, potentially miti-

gating the evaluation paradox and data scarcity issues synthesized in Section 4.2.1.

5.2.4. REST/JSON API Proposal. While this work focuses on shell-based honeypots, the extension of LLM simulation to REST/JSON APIs represents a promising avenue for future research. As the fundamental backend for mobile and web applications, these APIs constitute a critical attack surface exposed to external threats. The generative capabilities of LLMs are uniquely suited to mimicking complex APIs by producing dynamic, realistic-looking responses that are necessary to sustain attacker engagement. This high-fidelity simulation creates an opportunity to capture sophisticated probing for business logic flaws and other vulnerabilities. We anticipate that such honeypots would observe a broad spectrum of attacks, ranging from credential stuffing and data scraping to command injection attempts aimed at the underlying infrastructure fronted by the HTTP server.

5.3. Toward Real-Time Threat Detection

Previous work, e.g. LogPrécis of Boffa et al. [94] demonstrate the use of LLMs for (offline) attack pattern detection in logs. Future research is likely to expand on this, focusing on techniques for online threat detection. This use case has interesting implications for architectural choices of the underlying models. Boffa et al. [94] use a classifier based on an encoder-only model for this purpose. In parallel, significant progress has been made on scaling decoder-only (e.g. [109]) and encoder-decoder ([110]) model architectures. We propose a re-evaluation of threat detection tasks in light of these recent advances. In particular, causal (decoder-only) architectures more precisely model real-time detection tasks, since no knowledge of future events (honeypot interactions) can be leveraged, is in contrast to threat detection for in offline scenarios. Encoder-decoder models likewise more naturally map the problem as a translation task from, e.g., raw session interactions to, e.g., MITRE ATT&CK labels. Future work should study what further benefits and drawbacks the inductive biases of these architectures have on tasks such as this one.

5.4. The Autonomous Feedback Loop

We now have the potential to create autonomous systems that learn from their interactions. The first step is to operationalize attack labeling by creating the large-scale, open-source datasets needed to train production-quality classifiers. With reliable labeling, two powerful feedback loops become possible: a honeypot-to-SOC loop that automatically feeds intelligence to live defensive tools, and a honeypot-to-honeypot loop that enables self-improvement through automated reconfiguration.

With automated TTP labeling, it becomes possible to create novel metrics that move beyond simple session length. For example, a metric like information gain, based on the novelty of observed attack sequences compared to historical data, could quantify the honeypot’s effectiveness at gathering new intelligence. These new, quantifiable metrics can then serve as a formal objective function for optimization algorithms (e.g., Reinforcement Learning or prompt optimization techniques). This provides a clear path toward data-driven methods for establishing stopping criteria (i.e., when a honeypot’s information gain diminishes) and for guiding the automated reconfiguration and redeployment of the honeypot to maximize its intelligence-gathering goals. Leveraging the already-discovered metrics for honeypot success can lead to clear stopping criteria, signaling a need for honeypot reconfiguration, and data-driven methods for how to reconfigure and automatically redeploy the honeypot. Such a self-adaptive process can ideally speed up the acquisition of threat intelligence.

In this vein, we propose two longer-term research directions. The first is a federated network of autonomous honeypots for shared threat intelligence and dynamic reconfiguration. Such an endeavor will entail new challenges, such as developing privacy-preserving techniques to gain adoption. The second is seeking algorithms for real-time reconfiguration mid-attack. As an example, given the ability to label an attacker’s tactics automatically, real-time prediction of the attacker’s next steps can be trained and integrated to assist the LLM honeypot in changing its simulation state to steer the attacker into revealing new tools.

6. Conclusion

The integration of LLMs into honeypot systems marks a pivotal moment in the field of cyber deception. While LLMs were initially seen as a silver bullet for the classic fidelity-risk paradox, our systematization has shown that their immediate impact has been modest. The true potential of this technology, we argue, is not merely to create slightly more believable decoys, but to confront the next generation of autonomous, intelligent threats. The rise of LLM-powered attackers creates an urgent need for equally sophisticated, AI-driven defenses.

In this paper, we provided a foundational understanding of this new domain. We began by creating a taxonomy of the fundamental ways honeypots are detected, framing the core challenges that must be addressed. We then synthesized the initial literature on LLM-powered honeypots into a canonical architecture and systematized the field’s approaches to evaluation. Finally, we charted the evolution of log analysis, showing a clear progression toward the ultimate goal of automated threat intelligence.

Our key insight is that these distinct research threads—architecture, evaluation, and analy-

sis—converge on a powerful new paradigm: the autonomous, self-improving honeypot that operates in a continuous feedback loop of interaction, analysis, and reconfiguration. By providing these structured frameworks and a clear research roadmap, we hope to guide the community in building the intelligent, adaptive deception systems necessary to secure the networks of tomorrow.

Acknowledgments

This research was funded by Vinnova, the Swedish Innovation Agency.

The authors utilized Google Gemini 2.5 Pro [35] to assist in the preparation of this manuscript. The initial draft of all content was written entirely by the authors. This author-drafted text was then provided to Gemini for suggestions on improving wording, grammar, and organization. Gemini was also used to validate L^AT_EX syntax and to help identify potential errors in the references. All changes suggested by the model were manually reviewed by the authors. The authors retained full editorial control, incorporating or modifying suggestions as they deemed appropriate.

References

- [1] C. Vasilatos, D. J. Mahboobeh, H. Lamri, M. Alam, and M. Maniatakos, “LLMPot: Dynamically configured LLM-based honeypot for industrial protocol and physical process emulation,” in *2025 IEEE 10th European Symposium on Security and Privacy (EuroS&P)*, pp. 963–979, IEEE, 2025.
- [2] R. Tatoris, H. Saxena, and L. Miglietti, “Trapping misbehaving bots in an AI labyrinth,” <https://blog.cloudflare.com/ai-labyrinth/>, 3 2025. Accessed: 2025-03-24.
- [3] E. Cambiaso and L. Caviglione, “Scamming the scammers: Using chatgpt to reply mails for wasting time and resources,” *arXiv preprint arXiv:2303.13521*, 2023.
- [4] A. Desmarais, “A British telecommunications company launched an AI “granny” that will waste scammers’ time by rambling on the phone for as long as possible,” 11 2024. Accessed: April 23, 2025.
- [5] I. Hasanov, S. Virtanen, A. Hakkala, and J. Isoaho, “Application of large language models in cybersecurity: A systematic literature review,” *IEEE Access*, 2024.
- [6] M. Hassanin and N. Moustafa, “A comprehensive overview of large language models (LLMs) for cyber defences: Opportunities and directions,” *arXiv preprint arXiv:2405.14487*, 2024.
- [7] W. Guo, Y. Potter, T. Shi, Z. Wang, A. Zhang, and D. Song, “Frontier ai’s impact on the cybersecurity landscape,” *arXiv preprint arXiv:2504.05408*, 2025.
- [8] A. Javadpour, F. Ja’fari, T. Taleb, M. Shojafar, and C. Benzaïd, “A comprehensive survey on cyber deception techniques to improve honeypot performance,” *Computers & Security*, vol. 140, p. 103792, 2024.
- [9] R. A. Bridges, A. E. Rice, S. Oesch, J. A. Nichols, C. Watson, K. Spakes, S. Norem, M. Huettel, B. Jewell, B. Weber, C. Gannon, O. Bizovi, S. C. Hollifield, and S. Erwin, “Testing SOAR tools in use,” *Computers & Security*, vol. 129, p. 103201, 2023.
- [10] D. Botta, R. Werlinger, A. Gagné, K. Beznosov, L. Iverson, S. Fels, and B. Fisher, “Towards understanding IT security professionals and their tools,” in *Proceedings of the 3rd symposium on Usable privacy and security*, pp. 100–111, 2007.
- [11] R. A. Bridges, M. D. Iannacone, J. R. Goodall, and J. M. Beaver, “How do information security workers use host data? a summary of interviews with security analysts,” *arXiv preprint arXiv:1812.02867*, 2018.
- [12] C. R. De Souza, C. S. Pinhanez, and V. F. Cavalcante, “Information needs of system administrators in information technology service factories,” in *Proceedings of the 5th ACM Symposium on Computer Human Interaction for Management of Information Technology*, pp. 1–10, 2011.
- [13] R. Werlinger, K. Hawkey, and K. Beznosov, “An integrated view of human, organizational, and technological challenges of it security management,” *Information Management & Computer Security*, vol. 17, no. 1, pp. 4–19, 2009.
- [14] R. Werlinger, K. Muldner, K. Hawkey, and K. Beznosov, “Preparation, detection, and analysis: the diagnostic work of it security incident response,” *Information Management & Computer Security*, vol. 18, no. 1, pp. 26–42, 2010.
- [15] J. Goodall, W. Lutters, and A. Komlodi, “The work of intrusion detection: rethinking the role of security analysts,” 2004.
- [16] L. Spitzner, *Honeypots: Tracking Hackers*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.
- [17] V. Valeros, M. Rigaki, and S. Garcia, “Attacker profiling through analysis of attack patterns in geographically distributed honeypots,” *arXiv preprint arXiv:2305.01346*, 2023.
- [18] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, “Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains,” white paper, Lockheed Martin Corporation, 2011.
- [19] MITRE ATT&CK, “Enterprise tactics,” <https://attack.mitre.org/tactics/enterprise/>, 2024. Accessed 2024-09-03.
- [20] C. Stoll, *The cuckoo’s egg: tracking a spy*

- through the maze of computer espionage. Simon and Schuster, 2024.
- [21] B. Cheswick, “An evening with Berferd in which a cracker is lured, endured, and studied,” in *Proc. Winter USENIX Conference, San Francisco*, pp. 20–24, 1992.
 - [22] @UnaPibaGeek, “honeypots-detection (GitHub repository).” <https://github.com/UnaPibaGeek/honeypots-detection> Accessed: 11 March 2025.
 - [23] D. Sysman, G. Evron, and I. Sher, “YouTube Video from Black Hat Talk.” BlackHat Conference presentation <https://www.youtube.com/watch?v=HiZdkBAFp7Q>, Dec. 2015. Accessed: 25 March 2025.
 - [24] O. Lukas and S. Garcia, “Deep generative models to extend active directory graphs with honeypot users,” *arXiv preprint arXiv:2109.06180*, 2021.
 - [25] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levine, and H. Owen, “Honeystat: Local worm detection using honeypots,” in *Recent Advances in Intrusion Detection: 7th International Symposium, RAID 2004, Sophia Antipolis, France, September 15-17, 2004. Proceedings* 7, pp. 39–58, Springer, 2004.
 - [26] R. Holbel, J. Yerby, and W. Smith, “Utilizing virtualized honeypots for threat hunting, malware analysis, and reporting.,” *Issues in Information Systems*, vol. 25, no. 1, 2024.
 - [27] O. R. Team, “Cyber threat landscape study 2023: Outpost24’s honeypot findings from over 42 million attacks,” 2023.
 - [28] K. M. Heckel and A. Weller, “Countering autonomous cyber threats,” *arXiv preprint arXiv:2410.18312*, 2024.
 - [29] Reworr and D. Volkov, “LLM agent honeypot: Monitoring ai hacking agents in the wild,” *arXiv preprint arXiv:2410.13919*, 2025.
 - [30] The Honeypot Project, “Projects.” <https://www.honeynet.org/projects/>. Accessed: 2025-04-25.
 - [31] D. Sysman, G. Evron, and I. Sher, “Breaking honeypots for fun and profit and itamar sher.” <https://infocondb.org/con/black-hat/black-hat-usa-2015/breaking-honeypots-for-fun-and-profit>, 2015. Presentation at Black Hat USA 2015, accessed April 25, 2025.
 - [32] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, 2017.
 - [33] OpenAI, “Models.” <https://platform.openai.com/docs/models>.
 - [34] Meta, “Llama models website.” <https://ai.meta.com/llama/>. Accessed 2025-05-15.
 - [35] Google, “Gemini - meet the everyday AI assistant from google.” <https://gemini.google/>. Accessed: 2025-04-30.
 - [36] Kindo, “WhiteRabbitNeo: Offensive security Gen-AI model.” <https://www.securityweek.com/whiterabbitneo-high-powered-potential-of-uncensored-ai-pentesting-for-attackers-and-defenders/>, 2024. Accessed: 2025-10-20.
 - [37] N. O. Jaffal, M. Alkhanafseh, and D. Mohaisen, “Large language models in cybersecurity: A survey of applications, vulnerabilities, and defense techniques,” *AI*, vol. 6, no. 9, 2025.
 - [38] L. Fan, L. Li, Z. Ma, S. Lee, H. Yu, and L. Hemphill, “A bibliometric review of large language models research from 2017 to 2023,” *arXiv preprint arXiv:2304.02020*, 2023.
 - [39] S. Qiao, Y. Ou, N. Zhang, X. Chen, Y. Yao, S. Deng, C. Tan, F. Huang, and H. Chen, “Reasoning with language model prompting: A survey,” *arXiv preprint arXiv:2212.09597*, 2022.
 - [40] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, D. Yifan, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, and J.-R. Wen, “A survey of large language models,” *arXiv preprint arXiv:2303.18223*, 2023.
 - [41] M. Gupta, C. Akiri, K. Aryal, E. Parker, and L. Praharaj, “From chatgpt to threatgpt: Impact of generative AI in cybersecurity and privacy,” *IEEE Access*, vol. 11, pp. 80218–80245, 2023.
 - [42] N. Provos, “A virtual honeypot framework,” in *13th USENIX Security Symposium*, USENIX Association, 2004.
 - [43] T. Holz and F. Raynal, “Detecting honeypots and other suspicious environments,” in *Proceedings from the sixth annual IEEE SMC information assurance workshop*, pp. 29–36, IEEE, 2005.
 - [44] P. Defibaugh-Chavez, R. Veeraghattam, M. Kannappa, S. Mukkamala, and A. Sung, “Network based detection of virtual environments and low interaction honeypots,” in *2006 IEEE Information Assurance Workshop*, 2006.
 - [45] D. Wenda and D. Ning, “A honeypot detection method based on characteristic analysis and environment detection,” in *2011 International Conference in Electrics, Communication and Automatic Control Proceedings*, pp. 201–206, Springer, 2011.
 - [46] The Censys Research Team, “Unmasking deception: Navigating red herrings and honeypots,” 2023. Accessed: 2025-04-25.
 - [47] S. Morishita, T. Hoizumi, W. Ueno, R. Tanabe, C. H. Ganai, M. van Eeten, K. Yoshioka, and T. Matsumoto, “Detect me if you... oh

- wait. an internet-wide view of self-revealing honeypots,” pp. 134–143, 2019.
- [48] S. Srinivasa, J. M. Pedersen, and E. Vasilomanolakis, “Gotta catch ’em all: a multistage framework for honeypot fingerprinting,” 2021.
 - [49] W. Cabral, C. Valli, L. Sikos, and S. Wakeling, “Review and analysis of Cowrie artefacts and their potential to be used deceptively,” in *2019 International Conference on computational science and computational intelligence*, pp. 166–171, IEEE, 2019.
 - [50] W. Z. Cabral, C. Valli, L. F. Sikos, and S. G. Wakeling, “Advanced Cowrie configuration to increase honeypot deceptiveness,” in *IFIP International Conference on ICT Systems Security and Privacy Protection*, pp. 317–331, Springer, 2021.
 - [51] M. Oosterhof, “Cowrie SSH/Telnet honeypot.” <https://github.com/cowrie/cowrie>, 2024. Accessed: 2024-09-03.
 - [52] C. Guan, G. Cao, and S. Zhu, “HoneyLLM: Enabling shell honeypots with large language models,” in *2024 IEEE Conference on Communications and Network Security (CNS)*, pp. 1–9, 2024. <https://www.cse.psu.edu/~sxz16/papers/HoneyGPT.pdf>.
 - [53] A. Vetterl and R. Clayton, “Bitter harvest: Systematically fingerprinting low- and medium-interaction honeypots at internet scale,” in *12th USENIX Workshop on Offensive Technologies*, USENIX Association, 2018.
 - [54] X. Fu, W. Yu, D. Cheng, X. Tan, K. Streff, and S. Graham, “On recognizing virtual honeypots and countermeasures,” in *2006 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing*, pp. 211–218, IEEE, 2006.
 - [55] S. Mukkamala, K. Yendrapalli, R. Basnet, M. Shankarapani, and A. Sung, “Detection of virtual environments and low interaction honeypots,” in *2007 IEEE SMC Information Assurance and Security Workshop*, pp. 92–98, IEEE, 2007.
 - [56] M. Sladić, V. Valeros, C. Catania, and S. Garcia, “LLM in the shell: Generative honeypots,” in *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, vol. 220, p. 430–435, IEEE, 2024.
 - [57] J. Ragsdale and R. V. Boppana, “On designing low-risk honeypots using generative pre-trained transformer models with curated inputs,” *IEEE Access*, vol. 11, pp. 117528–117545, 2023.
 - [58] W. Fan, Z. Yang, Y. Liu, L. Qin, and J. Liu, “HoneyLLM: A large language model-powered medium-interaction honeypot,” in *International Conference on Information and Communications Security*, pp. 253–272, Springer, 2024.
 - [59] Z. Wang, J. You, H. Wang, T. Yuan, S. Lv, Y. Wang, and L. Sun, “Honeygpt: Breaking the trilemma in terminal honeypots with large language model,” 2024.
 - [60] N. Krawetz, “Anti-honeypot technology,” *IEEE Security & Privacy*, vol. 2, no. 1, pp. 76–79, 2004.
 - [61] P. Wang, L. Wu, R. Cunningham, and C. C. Zou, “Honeypot detection in advanced botnet attacks,” *International Journal of Information and Computer Security*, vol. 4, no. 1, pp. 30–51, 2010.
 - [62] C. Huang, J. Han, X. Zhang, and J. Liu, “Automatic identification of honeypot server using machine learning techniques,” *Security and Communication Networks*, 2019.
 - [63] N. Ilg, D. Germek, P. Duplys, and M. Menth, “Beekeeper: Accelerating honeypot analysis with LLM-driven feedback,” *IEEE Access*, 2025.
 - [64] F. McKee and D. Noever, “Chatbots in a honeypot world,” *arXiv preprint arXiv:2301.03771*, 2023.
 - [65] S. B. Weber, M. Feger, and M. Pilgermann, “Don’t stop believin’: A unified evaluation approach for LLM honeypots,” *IEEE Access*, 2024.
 - [66] H. T. Otal and M. A. Canbaz, “LLM honeypot: Leveraging large language models as advanced interactive honeypot systems,” in *2024 IEEE Conference on Communications and Network Security (CNS)*, pp. 1–6, IEEE, 2024.
 - [67] S. Johnson, R. Hassing, J. Pijpker, and R. Loves, “A modular generative honeypot shell,” in *2024 IEEE International Conference on Cyber Security and Resilience (CSR)*, pp. 387–394, IEEE, 2024.
 - [68] J. A. Christli, C. Lim, and Y. Andrew, “Ai-enhanced honeypots: Leveraging LLM for adaptive cybersecurity responses,” in *2024 16th International Conference on Information Technology and Electrical Engineering (ICIT-TEE)*, pp. 451–456, 2024.
 - [69] E. Gizzarelli, “Honeypot and generative ai,” Master’s thesis, Politecnico di Torino, 2024.
 - [70] M. Badran and T. Niazi, “Towards adaptive web honeypots, an experimental implementation using LLMs,” Master’s thesis, Malmö University, Malmö, Sweden, 2025.
 - [71] P. Malhotra, “LLMHoney: A real-time SSH honeypot with large language model-driven dynamic response generation,” *arXiv preprint arXiv:2509.01463*, 2025.
 - [72] J. Jiménez-Román, F. Almenares-Mendoza, and A. Sánchez-Macián, “Design and development of an intelligent LLM-based ldap honeypot,” *arXiv preprint arXiv:2509.16682*, 2025.
 - [73] M. Sladić, V. Valeros, C. Catania, and S. Gar-

- cia, “VeLLMes: A high-interaction ai-based deception framework,” in *2025 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp. 671–679, IEEE, 2025.
- [74] A. Safargalieva, A. Rüffer, and E. Vasilomanolakis, “Ohra: dynamic multi-protocol LLM-based cyber deception,” in *Proceedings of the 30th Nordic Conference on Secure IT Systems (Nordsec 2025)*, Springer, 2025.
- [75] J. G. Göbel, “Amun: A Python honeypot,” *Technical Report, University of Mannheim, Germany* <https://madoc.bib.uni-mannheim.de/2595/1/amunhoneypot2.pdf>, 2009. Accessed: 2025-10-20.
- [76] Y. Hu, S. Cheng, Y. Ma, S. Chen, F. Xiao, and Q. Zheng, “MySQL-Pot: A LLM-based honeypot for MySQL threat protection,” in *2024 9th International Conference on Big Data Analytics (ICBDA)*, pp. 227–232, 2024.
- [77] A. Sezgin and A. Boyacı, “DecoyPot: A large language model-driven web API honeypot for realistic attacker engagement,” *Computers & Security*, vol. 154, p. 104458, 2025.
- [78] B. Labs, “Beelzebub honeypot.” <https://beelzebub-honeypot.com/>. Accessed: April 23, 2025.
- [79] A. Karimi, “Galah: An LLM-powered web honeypot.” <https://github.com/0x4D31/galah>, 2024. GitHub repository, accessed:2025-10-20.
- [80] Telekom Security, “T-Pot Community Edition - LLM-Based Honeypots Section.” GitHub Repository <https://github.com/telekom-security/tpotce?tab=readme-ov-file#llm-based-honeypots>, May 2025. Accessed on May 8, 2025.
- [81] S. Kemppainen and T. Kovanen, “Honeypot utilization for network intrusion detection,” in *Cyber Security: Power and Technology* (M. Lehto and P. Neittaanmäki, eds.), vol. 93 of *Intelligent Systems, Control and Automation: Science and Engineering*, pp. 249–270, Springer International Publishing AG, 2018.
- [82] C. Kelly, N. Pitropakis, A. Mylonas, S. McKeeown, and W. J. Buchanan, “A comparative analysis of honeypots on different cloud platforms,” *Sensors*, vol. 21, no. 7, p. 2433, 2021.
- [83] O. Thonnard and M. Dacier, “A framework for attack patterns’ discovery in honeynet data,” *Digital Investigation*, vol. 5, pp. S128–S139, 2008.
- [84] A. Ghourabi, T. Abbes, and A. Bouhoula, “Characterization of attacks from the deployment of honeypot,” *Security Communication Networks*, 2013.
- [85] c. B. Aslan, E. Türkşanlı, R. E. Erkan, M. Öztürk, and C. Akdeniz, “Unveiling hidden patterns in t-pot honeypot logs: A latent topic analysis,” in *2024 17th International Conference on Information Security and Cryptology (ISCTürkiye)*, pp. 1–6, 2024.
- [86] A. Spyros, A. Papoutsis, I. Koritsas, N. Mengidis, C. Iliou, D. Kavallieros, T. Tsikrika, S. Vrochidis, and I. Kompatsiaris, “Towards continuous enrichment of cyber threat intelligence: a study on a honeypot dataset,” in *International Conference on Cyber Security and Resilience*, pp. 267–272, IEEE, 2022.
- [87] P. Owezarski, “A near real-time algorithm for autonomous identification and characterization of honeypot attacks,” in *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, Apr 2015.
- [88] D. Fraunholz, M. Zimmermann, A. Hafner, and H. D. Schotten, “Data mining in long-term honeypot data,” in *International Conference on Data Mining Workshops (ICDMW)*, pp. 649–656, IEEE, 2017.
- [89] G. Ikuomenisan and Y. Morgan, “Systematic review of graphical visual methods in honeypot attack data analysis,” *Journal of Information Security*, vol. 13, no. 4, pp. 210–243, 2022.
- [90] C. Valli, “Visualization of honeypot data using graphviz and afterglow,” 2009.
- [91] S. Mehta, D. Pawade, Y. Nayyar, I. Siddavatam, A. Tiwart, and A. Dalvi, “Cowrie honeypot data analysis and predicting the directory traverser pattern during the attack,” in *2021 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICES)*, pp. 1–4, 2021.
- [92] Ryandy, C. Lim, and K. E. Silaen, “XT-Pot: eXposing Threat Category of Honeypot-based attacks,” in *The International Conference on Engineering and Information Technology for Sustainable Industry*, (NY, USA), pp. 1–6, ACM, sep 2020.
- [93] M. Boffa, G. Milan, L. Vassio, I. Drago, M. Mellia, and Z. Houidi, “Towards NLP-based processing of honeypot logs,” in *IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, p. 314–321, 2022.
- [94] M. Boffa, I. Drago, M. Mellia, L. Vassio, D. Giordano, R. Valentim, and Z. B. Houidi, “Logprécis: Unleashing language models for automated malicious log analysis,” *Computers & Security*, vol. 141, p. 103805, 2024.
- [95] M. B. Ozkok, B. Birinci, O. Cetin, B. Arief, and J. Hernandez-Castro, “Honeypot’s best friend? investigating chatgpt’s ability to evaluate honeypot logs,” in *Proceedings of the 2024 European Interdisciplinary Cybersecurity Conference*, pp. 128–135, 2024.
- [96] P. Lanka, K. Gupta, and C. Varol, “Intelligent threat detection—AI-driven analysis of honeypot data to counter cyber threats,” *Elec-*

- tronics, vol. 13, no. 13, p. 2465, 2024.
- [97] N. Daniel, F. K. Kaiser, S. Giladi, S. Sharabi, R. Moyal, S. Shpolyansky, A. Murillo, A. Elyashar, and R. Puzis, "Labeling network intrusion detection system (NIDS) rules with MITRE ATT&CK techniques: Machine learning vs. large language models," *Big Data and Cognitive Computing*, vol. 9, no. 2, 2025.
 - [98] Y. Jiang, Q. Meng, F. Shang, N. Oo, L. T. H. Minh, H. W. Lim, and B. Sikdar, "Mitre att&ck applications in cybersecurity and the way forward," *arXiv preprint arXiv:2502.10825*, 2025.
 - [99] B. Bokkena, "Enhancing it security with LLM-powered predictive threat intelligence," in *2024 5th International Conference on Smart Electronics and Communication (ICOSEC)*, pp. 751–756, 2024.
 - [100] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, *et al.*, "Lora: Low-rank adaptation of large language models.," *ICLR*, vol. 1, no. 2, p. 3, 2022.
 - [101] Z. Aradi, S. Bottyán, E. Kail, E. Rigó, and A. Bánáti, "Metrics-driven evaluation and optimization of honeypots: Toward standardized measures of deception effectiveness," *Acta Polytechnica Hungarica*, vol. 22, no. 12, 2025.
 - [102] R. Fang, R. Bindu, A. Gupta, and D. Kang, "LLM agents can autonomously exploit one-day vulnerabilities," *arXiv preprint arXiv:2404.08144*, 2024.
 - [103] S. Glazunov and M. Brand, "Project naptime: Evaluating offensive security capabilities of large language models." <https://googleprojectzero.blogspot.com/2024/06/project-naptime.html>, 2024. Accessed June 2024.
 - [104] M. Xu, J. Fan, X. Huang, C. Zhou, J. Kang, D. Niyato, S. Mao, Z. Han, Xuemin, Shen, and K.-Y. Lam, "Forewarned is forearmed: A survey on large language model-based agents in autonomous cyberattacks," 2025.
 - [105] S. Bae, B. Acun, H. Habeeb, S. Kim, C.-Y. Lin, L. Luo, J. Wang, and C.-J. Wu, "Hybrid architectures for language models: Systematic analysis and design insights." *arXiv preprint arXiv:2510.04800*, October 2025. FAIR at Meta and KAIST AI collaboration; submitted October 6, 2025.
 - [106] G. Wang, J. Li, Y. Sun, X. Chen, C. Liu, Y. Wu, M. Lu, S. Song, and Y. A. Yadkori, "Hierarchical reasoning model," 2025.
 - [107] L. Newsham, R. Hyland, and D. Prince, "Inducing personality in LLM-based honeypot agents: Measuring the effect on human-like agenda generation," 2025.
 - [108] C. R. Landolt, C. Würsch, R. Meier, A. Mermoud, and J. Jang-Jaccard, "Multi-agent reinforcement learning in cybersecurity: From fundamentals to applications." *arXiv preprint arXiv:2505.19837*, 2025. Presented at NATO STO ICMCIS Symposium, Oeiras, Portugal, May 13–14, 2025.
 - [109] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models.," *CoRR*, vol. abs/2302.13971, 2023.
 - [110] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of machine learning research*, vol. 21, no. 140, pp. 1–67, 2020.