

LADE: LLM-Assisted Advanced Persistent Threat Detection and Explanation

Joon-Young Gwak¹, Aubrey Strier¹, Zhaohan Xi¹, Guanhua Yan¹, Xiaokui Shu²,
Scott D. Stoller³, and Ping Yang¹

¹ SUNY Binghamton {jgwak1,bstrier1,zxi1,ghyan,pyang}@binghamton.edu

² IBM Research xiaokui.shu@ibm.com

³ Stony Brook University stoller@cs.stonybrook.edu

Abstract. Advanced Persistent Threat (APT) attacks are sophisticated cyberattacks characterized by stealth, persistence, and long-term engagement with targeted systems. Traditional detection approaches using machine learning and deep learning rely on internal model representations or post hoc explainability techniques, which often lack human-readable context and require significant manual interpretation. This paper investigates the use of large language models (LLMs) for APT detection through code analysis. We evaluate LLMs’ ability to identify APT-related behaviors in code-snippet sequences, localize malicious components, and map them to corresponding MITRE ATT&CK tactics, techniques, and procedures (TTPs). To address context-window limitations, we introduce a segmentation-based approach that divides long code sequences into smaller segments and iteratively processes them while preserving context through summary propagation. Because real-world APT data are scarce due to their stealthy nature, privacy constraints, and long attack lifecycles, we construct an APT dataset derived from Caldera adversary profiles and emulation plans. Experimental results show that LLMs, when guided by rubric-based prompts and supplemented with ATT&CK domain knowledge, achieve robust performance across detection, localization, and TTP mapping tasks.

Keywords: Advanced Persistent Threat Detection · Large Language Models · MITRE ATT&CK Framework.

1 Introduction

Advanced Persistent Threat (APT) attacks represent a sophisticated class of cyberattacks characterized by stealth, persistence, and prolonged engagement with victim systems [96]. Their prolonged nature makes timely and effective detection critical to minimizing impact and preventing further damage.

Existing approaches for APT detection leveraging machine learning (e.g., [26, 36]) and deep learning models (e.g., [27, 43, 49]) typically rely on large amounts of labeled or unlabeled training data. Their decision-making processes are explained either through internal mechanisms [13] or post-hoc explainability techniques [58]. These explanations often lack human-readable context, requiring manual interpretation to extract meaningful insights from highlighted features.

Large Language Models (LLMs) have shown potential in security tasks such as vulnerability assessment and patching [54, 69], malware detection [50, 72], and

code de-obfuscation [71]. However, their application to APT detection remains limited. Recent approaches, such as SHIELD [32] and APT-LLM [8], integrate LLMs with traditional methods for system audit log analysis to detect APT attacks. However, abstracting low-level execution details and transforming them into provenance graphs in these log-based methods can lead to information loss. While LLMs have been utilized to generate natural language explanations in log-based detection, these explanations primarily reflect system-event interactions (e.g., a process writing to a file), which may not fully capture the semantic nuances of attackers’ intent, especially in Living-Off-the-Land (LOTL) attacks where adversaries abuse legitimate tools through subtle parameter choices and execution contexts.

To address these limitations, we investigate the use of LLMs for APT detection from a code-level (command-level) perspective by analyzing the shell commands executed by attackers. Directly analyzing shell commands preserves syntactic and semantic details essential for identifying and explaining malicious intent in LOTL attacks. Our work aims to address the following research questions: **RQ1:** Can an LLM detect the presence of APT activity within a chronologically ordered sequence of code snippets containing both benign and malicious operations? Here, a code snippet refers to a single command line or a multi-line script block captured in a discrete log entry by a host-based logging mechanism or a system monitoring tool. **RQ2:** If an APT attack is detected in a sequence, can an LLM accurately localize the specific snippets responsible for malicious behavior? Accurate identification of malicious snippets can help reduce the workload for security analysts by narrowing their focus to critical code. **RQ3:** For each identified malicious snippet, can the LLM accurately map it to the corresponding MITRE ATT&CK technique (referred to as “TTP mapping”)? **RQ4:** How do prompt engineering strategies and domain knowledge (e.g., ATT&CK technique descriptions) affect the LLM performance on these tasks?

In this paper, we propose LADE, an LLM-assisted APT Detection and Explanation framework that analyzes executed code to identify and characterize APT activity. LADE adopts a layered design that decomposes the APT detection and explanation task into three sequential subtasks: attack detection, snippet-level localization, and mapping to MITRE tactics, techniques, and procedures (TTPs). Each subtask is handled by a dedicated LLM guided with structured prompts, rubric-based reasoning, and domain knowledge. Because existing LLMs are constrained by finite context windows, sequences that exceed this window cannot be processed in a single pass. To overcome this limitation, we introduce a segmentation-based approach that partitions sequences into manageable segments, generates concise summaries for each segment, and propagates these summaries across subsequent analyses, thereby preserving contextual continuity throughout the full analysis.

A key obstacle in studying code-based APT detection is the scarcity of high-quality datasets due to the stealthy nature of these attacks, privacy concerns, and their prolonged, multi-stage attack life cycle. Consequently, we identified only one suitable public dataset, AVIATOR [55], which simulates APT TTPs using MITRE emulation plans in a realistic multi-machine environment. To complement this, we constructed an additional dataset using Caldera [15] based on the ATT&CK framework [16], incorporating additional emulation plans (Carbanak, FIN6, FIN7,

and Menu Pass) and multiple adversary profiles to cover a broader range of attack behaviors. Our data repository is available online¹. A detailed discussion of existing APT datasets is provided in Section 5.1.

We conducted experiments to evaluate the capability of LLMs in detecting APT attacks. Our experimental results show that LLMs, when guided by rubric-based prompts and supplemented with ATT&CK domain knowledge, achieve robust performance across all tasks.

Our contributions are summarized as follows:

- We propose a framework that builds a full pipeline for LLM-based APT detection, localization, and TTP mapping over chronologically ordered sequences of benign and malicious code snippets.
- We introduce a segmentation-based approach to analyze code snippet sequences that exceed the context window of LLMs, enabling iterative processing while preserving contextual information.
- We developed an open-source dataset constructed from MITRE Caldera’s adversary profiles and emulation plans (referred to as “Caldera-derived dataset”).
- Our evaluation results using the AVIATOR dataset and our Caldera-derived dataset show that LLMs, when guided by rubric-based prompts and ATT&CK domain knowledge, achieve robust performance on APT detection, snippet-level localization, and TTP mapping.

Organization: The remainder of this paper is organized as follows. Section 2 provides background information on APT attacks, MITRE ATT&CK techniques, and rubric-based prompt engineering. Section 3 provides an overview of our framework, and Section 4 describes its detailed design. Section 5 reports the experimental results. Related work is discussed in Section 6 and Section 7 concludes the paper and outlines future research directions.

2 Background

This section provides an overview of APT attacks, MITRE ATT&CK framework and Rubric-based prompting.

APT Attacks: APT attacks are sophisticated cyberattacks characterized by stealth, persistence, and long-term engagement with targeted systems. These attacks typically unfold in multiple stages: (1) Initial Compromise, where the attacker gains access; (2) Persistence, established through backdoors or remote access tools; (3) Lateral Movement, enabling the attacker to navigate within the network; and (4) Exfiltration, during which sensitive data is extracted [90]. Command-line interfaces such as PowerShell and Bash are commonly used in APT operations, and their execution can be captured by host-based logging mechanisms such as PowerShell Script Block Logging [63] and system monitoring tools such as Linux Audit [74], Windows Sysmon [64], or CrowdStrike Falcon [28].

¹ <https://github.com/jgwak1/LADE>

T1136 (Create Account): Adversaries may create an account to maintain access to victim systems. With a sufficient level of access, creating such accounts may be used to establish secondary credentialed access that do not require persistent remote access tools to be deployed on the system.

T1136.001 (Local Account): Adversaries may create a local account to maintain access to victim systems. Local accounts are those configured by an organization for use by users, remote support, services, or for administration on a single system or service. For example, with a sufficient level of access, the Windows `net user /add` command can be used to create a local account. On macOS systems the `dscl -create` command can be used to create a local account. Local accounts may also be added to network devices, often via common commands such as `username`, or to Kubernetes clusters using the `kubect1` utility.

Fig. 1: ATT&CK Technique Description Examples

MITRE ATT&CK: MITRE ATT&CK [16] is an open-source knowledge base of adversarial behavior based on real-world threats and threat actors. It organizes behavior into a hierarchy of Tactics, Techniques, and Procedures (TTPs): Tactics represent adversary goals, Techniques describe how those goals are achieved, and Procedures are specific implementations. Figure 1 gives an example of MITRE ATT&CK technique description. Sub-technique “T1136.001 (Local Account)” [19] is a variant of technique “T1136 (Create Account)” [18], describing how an adversary creates a local account to gain persistent access. Throughout this paper, we use the term “technique” to refer to both techniques and sub-techniques unless otherwise noted.

Caldera’s attack simulations are derived from the MITRE Center for Threat-Informed Defense, which maintains a comprehensive library of adversary emulation plans. These plans are designed to help organizations assess their defenses against real-world threats. Although the attacks are emulated, they are based on real-world TTPs of known APT groups such as APT29, Blind Eagle, and others.

Rubric-Based Prompt Engineering: Providing rubrics to LLMs is a prompt engineering strategy used in the education field for tasks such as automated grading [85, 94]. A rubric is defined as a set of assessment criteria, accompanied by guidelines for application, with outcomes that can be categorical labels or numerical scores. Providing rubrics to LLMs is well-suited for tasks where a correct answer (ground truth) exists and answers can be assessed according to a hierarchy of correctness or relevance. These assessment guidelines can range from general measures, such as evaluating the semantic relevance between an input and its ground truth, to more domain-specific analysis, such as assessing how well an input aligns with criteria based on domain-specific knowledge.

3 Overview of LADE

In APT attacks, once a target host is compromised, attackers issue malicious commands and scripts (APT code) from a Command and Control server. This code is often transmitted using evasion techniques, such as encryption or fragmentation, to avoid detection by network monitoring systems. Upon reaching the target, it is decrypted, reassembled, and executed through command-line interfaces, where it can be captured by host-based logging mechanisms and system monitoring tools. In this work, we assume that these logging mechanisms are neither tampered with nor disabled by attackers.

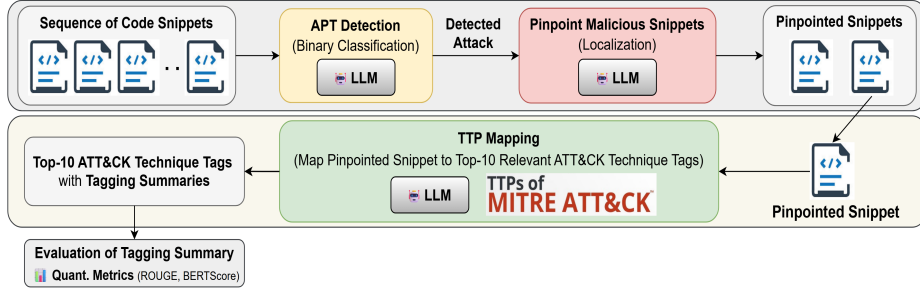


Fig. 2: Architecture of LADE.

LADE employs the LLM to detect and explain APT activities in sequences of code snippets captured by logging mechanisms. Figure 2 presents the overall evaluation pipeline. We use a layered analysis approach that decomposes the problem into three sequential subtasks: attack detection, snippet-level localization, and mapping to MITRE tactics, techniques, and procedures (TTPs). Given a sequence of code snippets, the LLM is first prompted to determine whether the sequence exhibits behavior that indicates an APT attack. If an attack is detected, the LLM is then instructed to identify the specific snippets responsible. Each identified snippet is subsequently mapped to the top ten relevant MITRE ATT&CK techniques (a process we refer to as “TTP mapping” or “TTP tagging”), accompanied by an explanation describing the rationale for each mapping (called “tagging summary”). These tagging summaries are evaluated using quantitative metrics, including ROUGE and BERTScore. To reduce variability in LLM inference, we constrain outputs using rubric-based prompts and aggregate results across multiple independent runs when applicable. To analyze a sequence of code snippets that exceed the LLM’s context window, we partition them into smaller, manageable segments. Each segment is analyzed based on its own content and the summaries of all preceding segments. By propagating summaries to subsequent segments, we maintain contextual continuity across the entire sequence.

4 Design of LADE

This section presents the design details of LADE.

4.1 APT Detection

APT attacks typically unfold as a multi-step progression toward a malicious objective, exhibiting both explicit data dependencies and implicit logical relationships between actions. While some actions are clearly malicious, many leverage dual-use Living-off-the-Land (LOTL) tools that appear benign in isolation but collectively indicate malicious intent. Detecting such attacks requires reasoning over relationships among multiple steps. To address this, we prompt the LLM to analyze sequences of code snippets, identifying coordinated adversarial operations by considering both functional dependencies and higher-level patterns of malicious behavior.

Prompt Design We adopt a zero-shot prompting strategy, providing the model with instructions and domain-specific assessment criteria without in-context

You are an expert cybersecurity analyst. Your task is to analyze a code sample which consists of a chronological sequence of code snippets.

Assessment Criteria

- **Attack:** An “Attack” assessment indicates the sequence of code snippets contains behaviors that provide concrete evidence of coordinated steps toward an adversarial objective, supported by concrete, functional linkages between those steps (e.g., data or process dependency). Do not treat mere co-occurrence as a chain.
- **Benign:** A “Benign” assessment should be assigned to the sequence that exhibit only routine or harmless behaviors. This includes potential Living-off-the-Land (LOTL) behaviors: since LOTL commands are dual-use, they should be considered Benign unless they are part of a functionally-linked sequence that clearly indicates malicious activity.

First Step: Before beginning the analysis, you must identify and exclude any behaviors that are known artifacts of the log-collection process, which should not be considered part of the code’s operational behavior.

Known Artifacts: {known log-collection artifacts}

Provide your assessment using the following output format:

1. Ignored Log-Collection Artifacts: Explicitly list the specific log-collection artifacts identified and excluded from the sequence of code snippets. If none are present, state “None found”.
2. Reasoning: Based only on the remaining behaviors, provide your analysis. Reference specific commands or actions and explain why they do or do not constitute concrete evidence of a coordinated attack.
3. Final Assessment: Based only on the analysis in your ‘Reasoning’ field, assign one of the following labels: “Attack” or “Benign”.

Code Sample: {code-sample}

Fig. 3: Prompt for APT detection.

examples. We choose zero-shot over few-shot prompting, because prior studies indicate that few-shot prompting is highly sensitive to the choice of examples [53]. In tasks with limited labeled data, such as APT detection, the available examples often lack sufficient diversity, which can lead to poorer performance compared to zero-shot inference [33]. Figure 3 presents the APT detection prompt, which guides the model’s analysis via three components:

- **Artifact Filtering:** Our experiments showed that LLMs may interpret benign log-collection artifacts (e.g., auditing configuration and log exports) as malicious. To mitigate this, we explicitly instruct the model to identify and exclude known log-collection artifacts before performing any analysis. This reflects realistic operational settings in which organizations are aware of their logging configurations and can distinguish log-collection artifacts from malicious activity.
- **Chain-of-Thought (Reasoning-First):** We use Chain-of-Thought prompting to guide the model to generate its analysis before producing the final classification label. Generating the label first can lead to post-hoc rationaliza-

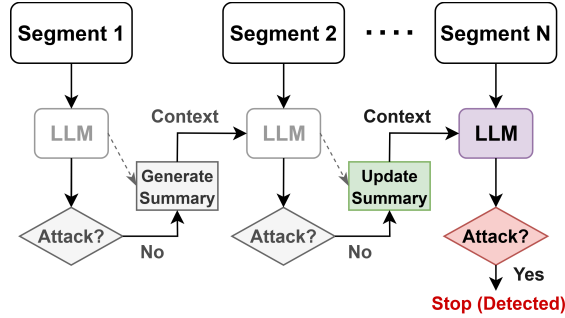


Fig. 4: Segmentation-based detection.

tion [86], where explanations become conditioned on the predicted label rather than grounded in observed evidence. By prompting reasoning first, decisions are based on the analyzed behavior, resulting in more evidence-grounded explanations.

- **Verifiable Evidence:** To reduce false positives arising from coincidental sequences of benign or dual-use commands, we instruct the model to label an input as an “attack” only when it identifies a *functional link* between snippets. A functional link is an explicit and verifiable data or process dependency (e.g., a file created by one snippet is later executed in another snippet). This requirement ensures that detections are grounded in demonstrable malicious behavior rather than inferred intent. We exclude “contextual links” (e.g., clustered discovery commands) at this stage, as they provide circumstantial rather than verifiable evidence. Their interpretation is deferred to the localization step (Section 4.2), which performs a more comprehensive, fine-grained analysis of the full snippet sequence.

Majority Voting Across Multiple Runs LLMs can produce different outputs on different runs of the same input due to inherent randomness in non-deterministic decoding. To reduce this variability, we run each experiment N times (where N is an odd number), and the final label is determined via majority voting. This approach captures the model’s overall consensus rather than relying on a single stochastic prediction, and is well-suited for binary classification tasks where outputs can be easily aggregated.

Beyond LLM Context Windows Current LLMs are constrained by a finite context window size, which limits the amount of input they can process at once. Some sequences in our dataset exceed this limit, making direct analysis infeasible. To overcome this, we propose a segmentation-based approach (shown in Figure 4), which enables the LLM to process long sequences iteratively while maintaining contextual continuity. Sequences that exceed the LLM’s context window are divided into multiple segments. Analysis begins with the first segment, where the LLM determines whether it contains concrete evidence of an attack. If such evidence is found, the sequence is labeled as malicious and the process terminates. Otherwise, the LLM is instructed to generate a factual, chronological summary of the segment, capturing key behavioral aspects. Subsequent segments are analyzed in conjunction with the cumulative summary from previous segments. By maintaining a compact running summary, this approach preserves contextual

Category	Description
Inherent Malice	Assign this label to Snippets that are malicious on their own, without any surrounding context. Examples include: (a) Use of known malicious tools (e.g., Mimikatz); (b) Presence of known Indicators of Compromise (IOCs), such as hardcoded C2 domains, malicious IP addresses, or malware file hashes; (c) Explicit obfuscation techniques (e.g., IEX) used to conceal malicious commands.
Functional Link	Assign this label to snippets that are part of the attack due to a direct functional dependency with another malicious snippet, such as: (a) Writing data to a file or variable that another snippet later reads; (b) Reading data produced by another snippet; (c) Executing a command that directly enables another (e.g., Set-ExecutionPolicy -Bypass immediately followed by execution of an unsigned script).
Contextual Link	Assign this label to dual-use (Living-off-the-Land) actions that are benign in isolation, but have a plausible contextual connection to the unfolding attack narrative. This is a temporal or logical link, not a data dependency. For example: whoami or ipconfig immediately preceding privilege escalation or lateral movement; net user issued alongside other discovery or reconnaissance commands.
Benign/Isolated	Assign this to all other snippets. This is the default assessment for all dual-use (LOTL) commands (e.g., whoami, ipconfig, tasklist) that lack a clear Functional or Contextual link.

Table 1: APT domain-expertise informed rubric for pinpointing malicious snippets.

continuity while staying within the model’s context window limit. The process continues until either an attack is detected or all segments are processed.

To reserve space for the prompt and the iteratively updated context summary, segments are limited to 70% of the LLM’s maximum context window. This threshold is computed using the model’s tokenizer to ensure precise segment boundaries. Furthermore, segmentation boundaries are adjusted to avoid splitting individual command-line entries or scripts.

4.2 Pinpoint Malicious Snippets

When an LLM detects a potential APT attack, we task it with identifying the specific code snippets that lead to this classification. To ensure that the LLM’s reasoning is consistent and aligned with expert analytic practice, we provide it with a domain-informed rubric that specifies the criteria for classifying each snippet (shown in Table 1). A snippet is classified as malicious under three categories: (1) Inherent Malice (a snippet is malicious on its own, such as known hack tools, similar to signature-based detection [78]); (2) Functional Link (a snippet has a direct functional dependency on another malicious snippet, aligning with provenance-based detection methods [46]); and (3) Contextual Link (a snippet is temporally or logically connected to the surrounding attack sequence, consistent with event correlation and sequential analysis [68]). Snippets that are dual-use but lack any of these links are labeled Benign/Isolated.

Before applying the rubric, we instruct the LLM to filter out known log-collection artifacts so they are not mistakenly classified as malicious. The LLM is then asked to examine the entire snippet sequence, labeling each snippet according to one of the categories defined in Table 1. We also instruct the LLM to justify why a dual-use command is classified as benign. After completing this rubric-guided assessment, the LLM produces a structured output listing only the snippets it classifies to be malicious, along with their assigned rubric categories.

4.3 TTP Mapping

Following the localization phase, the final step is to map the identified snippets to MITRE ATT&CK techniques (i.e., “TTP mapping”) to provide standardized intelligence about adversary behavior. Precise technique attribution is often ambiguous because tactically distinct behaviors can exhibit highly similar code-level characteristics. For example, *T1105: Ingress Tool Transfer* [84] and *T1048: Exfiltration Over Alternative Protocol* [83] commonly rely on comparable network utilities or APIs. Limiting the output to a single prediction risks missing the correct technique when the model’s confidence is divided among similar candidates. Moreover, technique attribution often requires semantic matching between snippet behavior and the official ATT&CK technique descriptions, which is difficult to capture with keyword/rule matching alone. To address this issue, we prompt the LLM to produce a ranked list of the top 3 and top 10 candidate mappings, each accompanied by a natural-language rationale grounded in the provided technique descriptions.

The accuracy of this ranking depends on the model’s ability to correctly interpret the semantics of each ATT&CK technique. While LLMs possess internal knowledge of the ATT&CK framework, they can be prone to hallucinations and may rely on outdated information [20, 92]. To mitigate this, we supply the LLM with up-to-date MITRE ATT&CK technique descriptions [17] as external domain knowledge during inference (Prompt 1 in Figure 6). We adopt inference-time knowledge augmentation rather than fine-tuning because the number of ATT&CK techniques is relatively small (approximately 656), which is insufficient to fine-tune the LLM effectively without risking overfitting.

Behavior Description Generation Directly mapping raw code snippets to MITRE ATT&CK techniques may create a granularity mismatch, as MITRE ATT&CK technique descriptions provide high-level concepts and representative examples rather than exhaustive implementation details. To address this, we introduce an intermediate analytical step that instructs LLMs to produce a behavioral description for each snippet that matches the structure and level of abstraction of the ATT&CK descriptions. As illustrated in Figure 5, each LLM is instructed to identify core functions, I/O operations, system interactions, and potential administrative utilities in the code snippet (Steps 1–3). This step produces an objective description of the code snippet’s underlying operational behavior. The LLM then abstracts these details into a high-level behavioral description (Step 4), generating a narrative aligned with MITRE’s linguistic and conceptual style. By focusing on behavioral intent rather than the low-level implementation specifics, this step reduces the semantic gap between code-level actions and ATT&CK’s technique taxonomy. Our experimental results show that this intermediate step yields modest improvements for most models.

Behavior description generation (4-step process)

Initial Prompt: Generate a behavioral description of the code snippet that mirrors the style and tone of an official MITRE technique description. (The initial prompt only presents the code and asks for acknowledgment.)

Code snippet: {code-snippet}

Steps:

1. **Identify Core Function(s)** Identify the snippet’s core function(s) in plain technical language, including the platform(s), required privileges, and any relevant artifacts (e.g., APIs, commands, flags, or paths) when present.
2. **Describe I/O and Outcomes** Describe the snippet’s inputs, outputs, and resulting actions (e.g., scheduling tasks, querying data, reading or writing files).
3. **Summarize Actions** Summarize the snippet’s actions and the system components that enable them. Include prerequisites or limitations only when indicated, and briefly note any plausible benign or administrative use if clearly evident. Maintain an objective and analytical tone throughout.
4. **Produce Final Narrative** Produce the refined behavioral description as a narrative prose. The description must abstract the ‘what’ (mechanisms/actions) into a higher-level behavioral objective, based on prior findings from Steps 1–3. Incorporate relevant identifiers (e.g., commands, paths, registry keys), and system-level interactions to mimic an official MITRE ATT&CK description without any direct mention of TTPs (tags, IDs, or technique names).

Fig. 5: Prompts for behavior description generation.

MITRE ATT&CK Technique Identification We use the prompt structure shown in Figure 6 to guide the LLM in mapping snippets identified as APT to MITRE ATT&CK techniques. The LLM follows a structured reasoning process. First, it analyzes the input and identifies all potentially relevant techniques based on the official technique descriptions. Second, it assesses the semantic alignment between the input and each candidate technique using a five-level rubric, ranging from Score 5 (Excellent) for strong evidence to Score 1 (Minimal) for weak or ambiguous associations. The LLM then ranks the techniques in descending order of alignment scores, breaking ties by prioritizing techniques supported by more specific evidence. The final output consists of the top 3 or top 10 techniques, each accompanied by a structured tagging summary. We further instruct the LLM to provide a rationale for each mapping, including the assigned alignment score, a natural-language explanation of its relevance, and a direct citation (relevant excerpt) from the supplied domain knowledge.

4.4 Evaluation of Tagging Summary

We evaluate the tagging summary using the following three sequentially dependent evaluation criteria.

- **C.1:** “Relevant Excerpt” accurately reflects the content in the “Cited Source”.
- **C.2:** “Relevant Excerpt” is indeed relevant to the “Behavior Description”.

Prompt 1 (optional): Domain Knowledge
 {official MITRE ATT&CK technique descriptions}

Prompt 2: Main Task
 You are an expert cybersecurity analyst specializing in threat detection and the MITRE ATT&CK framework.

Task Instructions: You will be provided with either a behavioral description or a code snippet. Your task is to analyze it and map it to the ten most relevant MITRE ATT&CK techniques. Follow this specific reasoning process:

1. **Analyze and Identify:** Using the provided behavioral description or code snippet, along with official MITRE ATT&CK technique descriptions, identify all potentially relevant techniques. Do not limit the initial selection to ten.
2. **Score Alignment:** For each identified technique, assign an alignment score from 1 (Minimal) to 5 (Excellent) according to the rubric below.
3. **Sort by Alignment:** Sort all identified techniques in descending order based on their alignment score.
4. **Break Ties:** If multiple techniques share the same score, perform a relative assessment to determine their order.
5. **Select Top 10:** Select the top 10 techniques from the sorted list.
6. **Format Output:** Present the top 10 techniques in the required output format in Rank 1–10 order.

Alignment Score Rubric

- **5:** Excellent alignment, with compelling evidence supporting the match.
- **4:** Strong alignment, with significant supporting evidence and only minor uncertainties.
- **3:** Adequate alignment, with sufficient evidence, though some discrepancies remain.
- **2:** Limited alignment, where some shared elements exist, but ambiguities weaken the match.
- **1:** Minimal alignment, with only general traits and weak supporting evidence.

Required Output Structure (for each of the 10 techniques):

- **Rank:** <N>
- **Tag:** Technique-ID: Technique-Name
- **Explanation:** Provide a clear justification.
- **Alignment Score:** Score 1–5.
- **Cited Source:** Provide a precise reference.
- **Relevant Excerpt:** Include an excerpt supporting the match.

Fig. 6: Prompts for TTP mapping.

- **C.3:** “Tagging Explanation” reflects both “Behavior Description” and “Relevant Excerpt” in justifying the tag.

Criterion C.1 assesses the factual alignment between the excerpt and the corresponding MITRE ATT&CK domain knowledge from the cited source. Criteria C.2 and C.3 evaluate the semantic alignment of the excerpt to the behavior

Table 2: APT detection performance

Dataset	Category	gpt-oss 120B		Llama-3.3 70B		Mistral Small-3.2		Gemini 2.5 Pro	
		F1	Acc.	F1	Acc.	F1	Acc.	F1	Acc.
AVIATOR	Unsegmented	0.842	0.842	0.836	0.842	0.815	0.818	0.824	0.803
	Segmented	0.800	0.889	0.800	0.889	0.667	0.778	0.572	0.500
	Total	0.839	0.849	0.833	0.849	0.800	0.813	0.800	0.776
2:1 Caldera-derived	Total	0.970	0.970	0.900	0.910	0.954	0.955	0.952	0.955

description and the coherence of the tagging explanation. Each criterion is evaluated with standard quantitative metrics such as ROUGE and BERTScore, as detailed in Section 5.5.

5 Experimental Results

This section presents our dataset and evaluation results. We evaluated and compared the performance of four widely used LLMs (gpt-oss-120B, Llama-3.3-70B, Mistral-Small-3.2-24B, and Gemini-2.5-Pro) on the proposed tasks. For each model, we developed automated scripts that submit prompts via the model’s API, collect outputs, and record response times. Each experiment was repeated five times as independent runs, except for the malicious snippet localization task, which was run twice due to the time required for result verification. For APT detection, we aggregate the outputs of five runs using majority voting (Section 4.1) to mitigate LLM stochasticity. For malicious snippet localization and TTP mapping, we compute metrics for each run and report the mean performance across runs. We also report the standard deviation to quantify run-to-run variability.

5.1 Dataset

A key obstacle in studying code-based APT detection is the scarcity of high-quality datasets due to the stealthy nature of these attacks, privacy concerns, and their prolonged attack life cycle. Many existing approaches for APT detection are based on system audit logs, which typically do not include the command lines invoked on host machines. For example, DARPA TC E3 and E5 [21, 22] provide custom formatted audit logs without command line fields. Some APT detection datasets include system audit logs containing command-line fields, such as the Linux APT Dataset [44], DARPA OpTC [7], ATLAS-v2 [76], and PublicArena [4]. However, these datasets rely on system events such as Linux audit logs or Windows ETW events to capture command lines. While these events record commands that invoke executable files, they do not capture native shell built-in commands, such as PowerShell cmdlets or Bash built-ins. Because these built-ins are frequently used in Living-off-the-Land (LOTL) techniques during APT attacks, these datasets are insufficient for our approach. Other datasets, such as DAPT2020 [67], consist primarily of network logs where command-line information is not available. In addition, many approaches for detecting

malicious or anomalous activity directly from logged commands or scripts rely on proprietary enterprise datasets [56, 88]. Publicly available datasets [35, 48, 52, 79] mostly consist of Unix shell commands collected for tasks such as masquerade detection or privilege escalation analysis. These datasets are largely outdated and often contain truncated commands that omit flags and subshells due to privacy restrictions, limiting their usefulness for modern APT detection.

The only publicly available dataset we found that is well aligned with our tasks is AVIATOR [55]. This dataset contains eight attack scenarios, covering multiple variants and extended executions of four MITRE emulation plans: APT29, OilRig, Sandworm, and Wizard Spider. AVIATOR contains logs for both attack and benign activity, from which we derive 35 attack sequences and 32 benign sequences. These sequences are constructed by extracting command-line entries from system event logs (e.g., Sysmon process creation events) and script blocks from PowerShell audit logs (e.g., Event ID 4104, Script Block Logging). We define each extracted command-line entry or script block as a code snippet, the atomic unit of our analysis. Code snippets range from single-line commands to multi-line script blocks. We then sort and aggregate snippets by timestamp to form chronological sequences. On average, each sequence derived from AVIATOR contains approximately 990 snippets, with each snippet comprising an average of 4.8 non-empty lines of code. Across AVIATOR attack samples, the median and mean ratios of benign to APT snippets are 42 : 1 and 113 : 1, respectively, indicating that a relatively small number of APT snippets are embedded within a much larger volume of benign activity.

To cover additional MITRE ATT&CK emulation plans absent in AVIATOR (i.e., Carbanak, FIN6, FIN7, and Menu Pass), we constructed a dataset using Caldera adversary profiles and APT emulation plans (referred to as “Caldera-derived dataset”). In addition, we use multiple Caldera adversary profiles to cover a broader range of attack behaviors. We extracted APT code snippets directly from the `command` field of each attack step. To mimic the visibility of PowerShell Script Block Logging, we augmented each extracted snippet with its resolved code, which includes the source code of any invoked external functions where applicable. We have also excluded commands intended for execution on the attacker’s machines, and ensured that all snippets within a sequence are intended for execution on the victim machine. These APT snippets were interleaved with benign snippets randomly sampled from a pool of 744 code snippets aggregated from three public PowerShell repositories [31, 41, 60].

In addition, we generated two sets of APT-labeled sequences derived from Caldera, each containing 33 sequences: a 2 : 1 benign-to-APT ratio setting for controlled evaluation, and a 10 : 1 ratio setting to assess the impact of increased benign noise. In both sets, APT snippets are inserted at random positions among benign snippets while preserving the sequential dependencies of the attack steps. Each set comprises 20 sequences whose APT code snippets are derived from adversary profiles, with each sequence containing approximately 4 APT snippets. Each set also includes 13 sequences derived from MITRE emulation plans, consisting of 5 lateral-movement sequences (each with about 6 APT snippets) and 8 multi-stage attack sequences (each with roughly 25 APT snippets). The number of benign snippets per sequence is determined by the specified benign-to-APT ratio.

For balanced evaluation, we also constructed 33 benign-labeled sequences that match the sequence-length distribution (i.e., the number of snippets per sequence) of the APT-labeled set, using the same benign code pool described above. On average, the snippets contain 6.1 non-empty lines of code. Preprocessing steps involved removing comments and print commands to avoid inadvertently revealing malicious intent. In addition, we replaced metadata placeholders (e.g., `IP-address`) with arbitrary but realistic values, and our dataset contains no personally identifiable information.

The above two datasets present complementary challenges. AVIATOR, which serves as the more realistic setting, combines executable-file commands and native shell built-ins collected from a realistic operational environment, making it challenging to identify sparse APT code snippets hidden among large volumes of benign administrative activity and real-world noise. In contrast, our Caldera-derived dataset, which provides a more controlled setting and relies mainly on native built-in commands for both benign operations and multi-step APT sequences, requires the model to distinguish malicious activity that closely resembles benign command sequences.

5.2 APT Detection

To address **RQ1**, we evaluated the ability of the four LLMs to distinguish APT-labeled sequences from benign ones. As discussed in Section 4.1, we applied majority voting across five independent runs to mitigate the inherent stochasticity of LLM outputs. Results for both datasets are reported in Table 2. Several sequences from AVIATOR exceed the LLMs’ context windows and were processed using the segment-based method described in Section 4. In contrast, all sequences in the Caldera-derived dataset fit within the context window due to their smaller average size (Section 5.1). Because each model uses a different tokenizer, the number of AVIATOR sequences requiring segmentation varies across LLMs. Mistral-Small-3.2-24B required segmentation for 12 sequences, compared to 10 for both gpt-oss-120B and Llama-3.3-70B (all with 128k windows and an effective segment limit of $\approx 89,600$ tokens). Gemini-2.5-Pro (1M window) required segmentation for only 6 sequences.

Table 2 reports the F1 score and detection accuracy for the 2:1 Caldera-derived dataset and AVIATOR dataset. All models achieve strong performance on the 2:1 Caldera-derived dataset (F1/Accuracy > 90%). Under the 10:1 setting, performance decreases: gpt-oss-120B achieves an F1 score of 0.812 and an accuracy of 0.803, Gemini-2.5-Pro has an F1 score of 0.866 and an accuracy of 0.864, and Mistral-Small-3.2-24B exhibits a larger drop (F1 0.700, Accuracy 0.712). These results suggest that heavier benign noise makes detection more challenging.

On the AVIATOR dataset, all models achieved moderate results, consistent with AVIATOR presenting a more difficult detection setting in which APT activity is embedded within a substantially larger volume of benign activities. Within the AVIATOR dataset, performance on unsegmented samples was consistently strong, with all models exceeding 0.8 in both F1 and Accuracy. In contrast, segmented samples introduced notable variability: gpt-oss-120B and Llama-3.3-70B maintained performance above 0.8, whereas Mistral-Small-3.2-24B and Gemini-2.5-Pro performed considerably worse. This suggests that summary propagation may lose or condense contextual details over time. Across segmented

Table 3: Pinpoint malicious snippets performance

Dataset	Metric	gpt-oss 120B	Llama-3.3 70B	Mistral Small-3.2	Gemini 2.5-Pro
AVIATOR	Precision	0.771	0.720	0.610	0.752
	Recall	0.835	0.574	0.746	0.935
2:1 Caldera-derived	Precision	0.767	0.741	0.669	0.861
	Recall	0.751	0.617	0.712	0.792

samples, most misclassifications were false positives, meaning that at least one segment of a benign sequence was incorrectly labeled as malicious. However, even in these cases, the vast majority of benign segments were correctly classified as benign.

We have also analyzed the consistency of predictions across the five independent runs. On average, 76.3% of predictions were unanimous (5-0), 13.8% showed strong consensus (4-1), and 10% of cases resulted in split decisions (3-2).

5.3 Pinpointing Malicious Snippets

To address **RQ2**, we evaluated the accuracy of LLMs in pinpointing specific malicious snippets within sequences correctly detected as attacks. Table 3 summarizes the performance of the four LLMs in localizing malicious snippets. We report average precision (the proportion of predicted malicious snippets that are truly malicious) and average recall (the proportion of all ground-truth malicious snippets correctly identified). On the Caldera-derived dataset, Gemini-2.5-Pro achieved the highest overall performance, followed by gpt-oss-120B. Both models outperformed Llama-3.3-70B and Mistral-Small-3.2. On the AVIATOR dataset, which features longer sequences and sparser APT code snippets, gpt-oss-120B achieves the highest precision, whereas Gemini-2.5-Pro achieved the highest recall. This indicates that Gemini-2.5-Pro demonstrates robust pinpointing performance even in the presence of substantial benign noise. Qualitative analysis suggests that false positives are primarily driven by spurious contextual links, in which benign dual-use commands are mistakenly interpreted as part of the attack chain. For example, a benign command such as `tasklist.exe` was occasionally flagged as malicious solely because it occurred immediately prior to a true attack step (e.g., `m64.exe`), indicating that simple co-occurrence was incorrectly treated as evidence of a contextual link. In addition, we observe higher consistency in results across multiple runs for Gemini-2.5-Pro and gpt-oss-120B (standard deviation $\sigma \approx 0.01$) compared to Llama-3.3-70B and Mistral-Small-3.2-24B ($\sigma \approx 0.03$) for both metrics, where lower σ indicates more stable outputs across runs.

5.4 TTP Mapping

To address **RQ3** and **RQ4**, we evaluated the TTP mapping performance of four LLMs using all distinct APT snippets extracted from the AVIATOR and Caldera-derived datasets. Each snippet was mapped to the top 10 and top 3 relevant ATT&CK techniques. Performance was measured using multiple ranking metrics – Mean Reciprocal Rank (MRR), Hit Rate (HR), and Normalized Discounted

Cumulative Gain (NDCG) – to provide a balanced evaluation, where higher scores indicate more accurate mappings.

- **Mean Reciprocal Rank (MRR)**: Mean $\frac{1}{\text{rank}}$ of ground-truth technique across snippets (0 if not in top 10/3). MRR favors higher ranks.
- **Normalized Discounted Cumulative Gain (NDCG)**: Mean $\frac{1}{\log_2(\text{rank}+1)}$ of ground-truth technique across snippets (0 if not in top 10/3). NDCG imposes a softer penalty for lower ranks than MRR.
- **Hit Rate (HR)**: The fraction of snippets for which the ground-truth technique appears in the top 10/3 predictions. HR measures coverage without penalizing lower-ranked but correct placements.

We evaluate the benefit of incorporating MITRE ATT&CK technique descriptions as an external knowledge base by comparing performance across three settings: (1) no domain knowledge, (2) coarse-grained knowledge containing descriptions of the 203 techniques, and (3) full knowledge covering all 203 techniques and 453 sub-techniques. While coarse-grained descriptions fit within standard context windows, the full knowledge exceeds them. To address this, we prompted gpt-oss-120B to generate summaries of the full knowledge, while trying to preserve essential semantic and operational details.

The MRR, HR, and NDCG scores for mapping to top 3 and top 10 relevant ATT&CK techniques are presented in Table 5 and Table 4. Each reported value represents the average over five independent runs. The standard deviations, ranging from 0.01 to 0.04, indicate that the results are consistent across runs.

The tables show that performance when mapping to the top 3 ATT&CK techniques is consistently lower than when mapping to the top 10 techniques. Incorporating domain knowledge consistently improves performance. Coarse-grained knowledge yields performance gains across all models. Summarized full knowledge produces mixed outcomes: it improves performance for gpt-oss-120B but leads to worse results for the other models compared to coarse-grained knowledge. This discrepancy likely reflects an inductive bias: since the summaries were generated using gpt-oss-120B, they may be structurally aligned with its architecture while introducing information loss that negatively affects other models. In the future, we plan to investigate whether generating domain-knowledge summaries using the same LLM as the one being evaluated can improve performance.

We also compare two tagging approaches: (1) behavior-description tagging, in which the model first generates a natural-language description of the code’s behavior before performing TTP mapping (i.e., *code* \rightarrow *behavior description* \rightarrow *TTP mapping*), and (2) direct code tagging, in which the model maps code snippets to TTPs without an intermediate behavior description generation step (i.e., *code* \rightarrow *TTP mapping*). As shown in Table 4, behavior-description tagging performs similarly or better across all models. This suggests that generating an intermediate behavior description slightly helps bridge the semantic gap between low-level code operations and the higher-level TTP concepts. In addition, Gemini-2.5-Pro and gpt-oss-120B significantly outperform Llama-3.3-70B and Mistral-Small-3.2-24B across all metrics. Among them, Gemini-2.5-Pro performs best with coarse-grained domain knowledge. An MRR of 0.812 for top 10 mapping suggests that the ground-truth technique ranks highly, on average appearing

Table 4: TTP mapping performance at Top-10 across three domain-knowledge (DK) levels: No DK, Coarse-grained DK, and Full (Summarized) DK.

Input	Model	No DK			Coarse DK			Full (Summ.) DK		
		MRR	HR	NDCG	MRR	HR	NDCG	MRR	HR	NDCG
Behavior Desc.	gpt-oss-120B	0.700	0.926	0.755	0.718	0.926	0.764	0.751	0.943	0.798
	Llama-3.3-70B	0.440	0.607	0.481	0.558	0.795	0.615	0.438	0.662	0.492
	Mistral-Small-3.2	0.502	0.630	0.534	0.657	0.777	0.687	0.636	0.729	0.659
	Gemini-2.5-Pro	0.782	0.950	0.823	0.803	0.961	0.841	0.784	0.963	0.827
Direct Code	gpt-oss-120B	0.683	0.926	0.746	0.715	0.926	0.767	0.734	0.926	0.781
	Llama-3.3-70B	0.420	0.607	0.466	0.509	0.738	0.564	0.324	0.559	0.381
	Mistral-Small-3.2	0.398	0.591	0.445	0.592	0.742	0.630	0.502	0.649	0.538
	Gemini-2.5-Pro	0.768	0.936	0.803	0.812	0.952	0.847	0.785	0.954	0.826

within the top two positions. The NDCG metric, which penalizes lower-ranked correct techniques less severely than MRR, also indicates strong performance (0.847). A high Hit Rate of 0.952 further confirms that Gemini-2.5-Pro ranks the correct technique within the top 10 positions in approximately 95% of cases.

During our experiments, we observe that when a key indicator is omitted in the intermediate behavior description, the mapping can become overly generic. For example, consider a Run-key persistence command such as `reg add "... \CurrentVersion\Run" /v "Updater" /d "C:\Temp\calc.exe"`. If the registry path `... \CurrentVersion\Run` (a key signal of autostart persistence) is omitted, the behavior may be incorrectly mapped to T1112 (*Modify Registry*) instead of the correct T1547.001 (*Registry Run Keys*). This example shows that an LLM’s ability to preserve key indicators in behavior descriptions is essential for distinguishing between closely related techniques.

5.5 Evaluation of Tagging Summary

We evaluate tagging summaries generated using behavior description tagging with coarse-grained domain knowledge, chosen for its strong performance across models (Table 4). To assess the quality of these summaries, we apply the three sequentially dependent evaluation criteria (C.1–C.3) introduced in Section 4.4.

We define the *candidate* and *reference* pairs for each criterion as follows:

- **C.1 (Excerpt Validity):** The candidate is the extracted excerpt; the reference is the cited source text. This measures whether the excerpt faithfully appears in the source.
- **C.2 (Excerpt-Behavior Alignment):** The candidate is the extracted excerpt; the reference is the behavior description. This measures whether the excerpt conceptually supports the described behavior.
- **C.3 (Explanation Alignment):** The candidate is the tagging explanation; the reference is the combination of the behavior description and the excerpt. This evaluates if the explanation effectively aligns with both.

We evaluate tagging summaries for samples where LLMs successfully identify the ground-truth technique. We report precision for both ROUGE-1 [51] (unigram

Table 5: TTP mapping performance at Top-3 across three domain-knowledge (DK) levels: No DK, Coarse-grained DK, and Full (Summarized) DK.

Input	Model	No DK			Coarse DK			Full (Summ.) DK		
		MRR	HR	NDCG	MRR	HR	NDCG	MRR	HR	NDCG
Behavior Desc.	gpt-oss-120B	0.678	0.825	0.714	0.698	0.832	0.726	0.732	0.840	0.760
	Llama-3.3-70B	0.425	0.524	0.450	0.528	0.623	0.552	0.413	0.529	0.443
	Mistral-Small-3.2	0.490	0.563	0.510	0.645	0.717	0.663	0.627	0.674	0.639
	Gemini-2.5-Pro	0.764	0.883	0.795	0.800	0.887	0.822	0.766	0.862	0.791
Direct Code	gpt-oss-120B	0.661	0.820	0.701	0.695	0.814	0.725	0.717	0.836	0.748
	Llama-3.3-70B	0.396	0.492	0.421	0.481	0.584	0.507	0.298	0.416	0.328
	Mistral-Small-3.2	0.374	0.471	0.400	0.584	0.701	0.614	0.485	0.560	0.504
	Gemini-2.5-Pro	0.764	0.848	0.786	0.787	0.887	0.827	0.766	0.853	0.788

overlap) and BERTScore [98] (semantic similarity), as our criteria focus on how well the candidate reflects or aligns with the reference content.

Table 6 presents the results. Excerpt Validity (C.1) shows high ROUGE scores, indicating that the snippets extracted by the LLMs faithfully appear in the original text. Gemini-2.5-Pro achieves near-perfect extraction fidelity (0.99), while the other models achieve slightly lower scores (0.92–0.94). The comparatively lower BERTScore values (0.77–0.85) result from the length mismatch between short extracted excerpts and their longer source passages, which shifts embedding representations even when lexical overlap is substantial.

Excerpt-Behavior Alignment (C.2) yields the lowest scores. This is expected, as the candidate (extracted excerpt) and reference (code-derived behavior description) operate at different levels of abstraction. The behavior descriptions captures a specific implementation of the technique, whereas the extracted excerpt defines the general technique class or a representative implementation. As a result, ROUGE-1 scores (0.51–0.65) are constrained by limited lexical overlap between these distinct narrative levels. In contrast, the moderate BERTScore values (0.61–0.66) indicate that the extracted excerpts still capture the underlying semantics of the behaviors, supporting the presence of a valid conceptual link despite vocabulary differences.

Explanation Alignment (C.3) scores consistently fall between those of C.1 and C.2. Most models achieve high ROUGE-1 scores (0.79–0.80), indicating that their explanations (candidates) often closely mirror the language of the behavior descriptions and excerpts (references). Gemini-2.5-Pro is a notable exception, with lower ROUGE (0.67) and BERTScore (0.64) values. This suggests a qualitative difference in its generation style: Gemini-2.5-Pro tends to produce more abstract or paraphrased explanations, resulting in reduced lexical and embedding overlap with the behavior descriptions and excerpts compared to other models.

5.6 LLM Response Time

LLM response times are influenced by multiple factors, including server location, available computational resources, network latency, model size, system load, and prompt length. In our experiments, all models were accessed via official or third-party APIs. The average response time varies considerably across different

Table 6: Evaluation of tagging summaries generated by LLMs.

Model	C.1		C.2		C.3	
	Excerpt Validity		Excerpt-Behavior Alignment		Explanation Alignment	
	ROUGE	BERTSc.	ROUGE	BERTSc.	ROUGE	BERTSc.
gpt-oss-120B	0.92	0.77	0.59	0.63	0.79	0.69
Llama-3.3-70B	0.93	0.78	0.65	0.66	0.79	0.69
Mistral-Small-3.2	0.94	0.84	0.55	0.63	0.80	0.69
Gemini-2.5-Pro	0.99	0.85	0.51	0.61	0.67	0.64

models and tasks. Llama-3.3-70B demonstrated the fastest average latencies, with 17.3 seconds for detection, 32.5 seconds for pinpoint, and 43.7 seconds for TTP Mapping, totaling 93.5 seconds. gpt-oss-120B followed closely, with a total response time of 97.6 seconds. Gemini-2.5-Pro and Mistral-Small-3.2-24B exhibited higher latencies, with total response times of 206.3 seconds and 237.1 seconds, respectively.

6 Related Work

Conventional Approaches for APT Detection Traditional APT detection techniques primarily rely on signature-based or rule-based mechanisms that match predefined patterns of malicious behavior [34,95] or employ statistical anomaly detection methods [42,62]. These approaches have been applied to a variety of data sources, including network traffic [57,87], system provenance graphs [38,65], and code artifacts [9,82]. However, these approaches rely on manual rules about normal behavior, which makes them prone to high false-positive and false-negative rates, especially when dealing with stealthy or previously unseen attack strategies [47].

Recent advances incorporate machine learning (ML) to capture complex behavioral patterns for improved APT detection. Prior work employs classical ML models such as SVMs [12], Random Forests [24], and clustering-based methods [36], as well as deep learning architectures including LSTMs [27], CNNs [25], GNNs [75], and Bayesian neural networks [3]. Although these approaches generally achieve higher detection performance than traditional rule-driven systems [1], they often require extensive training data and computational resources. Moreover, while many ML-based techniques support post-hoc interpretability using tools such as SHAP and LIME [37,66] or counterfactual explanation [91], the resulting explanations, such as feature attribution scores, often remain low-level and abstract. As a result, analysts still need to invest substantial manual effort to interpret these low-level outputs to infer the intent of attacker.

Prior research on code-level analysis has applied static analysis, dynamic analysis, or machine learning to suspicious scripts or binaries [11,30,77] for APT detection. However, these approaches generally focus on detecting APT activities within individual programs and do not capture the sequential progression of APT attacks, which typically unfold as sophisticated, multi-stage operations

involving the execution of a sequence of commands or scripts on a victim’s system. Other research has focused on detecting malicious activities using rule-based approaches [80,81] or by mapping these activities to MITRE ATT&CK techniques through similarity measures [70]. However, none of them explored the use of LLMs for APT attack detection and explanation.

LLM for Cybersecurity LLMs have been used for various cybersecurity tasks [97], including vulnerability discovery [6, 54, 59], program repair [69], anomaly detection [45], secure code generation [89], threat intelligence [39], malware detection [29,71], penetration testing [14], cyber espionage [5], and security CTF solving [61]. A few recent works have explored using LLMs for APT detection. APT-LLM [8] encodes process-behavior descriptions from system logs using encoder-only LLMs for anomaly detection. APTSniffer [93] applies retrieval-augmented generation by extracting relevant sequences from encrypted traffic to construct few-shot prompts for APT detection. SHIELD [32] analyzes suspicious nodes in provenance graphs derived from system logs and employs chain-of-thought prompting to detect malicious processes and generate attack summaries; however, its explanations primarily describe low-level events (e.g., a process writing to a file), which may not fully capture attacker intent. Works in [2, 10, 73] extract TTPs from unstructured text, such as threat intelligence reports. In contrast, we identify TTP techniques in APT code snippets. RACONTEUR [23] generates behavioral descriptions of malicious shell commands using a fine-tuned local LLM and maps these descriptions to MITRE ATT&CK techniques. Cmd-Caliper [40] leverages LLMs to synthesize a command-line similarity dataset for training an embedding model, which is evaluated on a limited set of TTPs. However, these works do not account for chronologically ordered shell commands in APT attacks nor do they perform APT detection or localization.

7 Conclusion

In this paper, we propose LADE, an LLM-assisted framework for analyzing sequences of code snippets to identify APT activities. We have shown that LLMs have the capability to identify APT-related behaviors, localize malicious code components, and map them to MITRE ATT&CK TTPs. To address the scarcity of real-world APT data, we created an open-source dataset based on Caldera adversary profiles and emulation plans. Experimental results show that, when guided by rubric-based prompts and enriched with ATT&CK knowledge, LLMs achieve strong performance in detection, localization, and TTP mapping, highlighting their potential as practical tools for APT detection and explanation.

In the future, we plan to evaluate an approach in which each LLM summarizes the MITRE ATT&CK techniques and sub-techniques to construct its own domain knowledge, and to explore the use of retrieval-augmented generation (RAG) to access the full MITRE knowledge base without the information loss inherent in LLM-generated summaries.

Acknowledgement: This work is supported in part by a SUNY-IBM AI Research Alliance grant, and NSF grants 2146212 and 2153056. We thank the anonymous reviewers for their constructive comments.

References

1. AL-Aamri, A.S., Abdulghafor, R., Turaev, S., Al-Shaikhli, I., Zeki, A., Talib, S.: Machine learning for apt detection. *Sustainability* **15**(18), 13820 (2023)
2. Ali, A., Peng, M.C.: Ttpmapper: Accurate mapping of ttps from unstructured cti reports. In: 2024 IEEE International Conference on Future Machine Learning and Data Science (FMLDS). pp. 558–563. IEEE (2024)
3. Anjum, M.M., Iqbal, S., Hamelin, B.: Anubis: a provenance graph-based framework for advanced persistent threat detection. In: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing. pp. 1684–1693 (2022)
4. Anonymous: Public arena dataset. <https://github.com/security0528/PublicArena> (2022), accessed: 2025-11-18
5. Anthropic: Disrupting the first reported ai-orchestrated cyber espionage campaign (2025), <https://www.anthropic.com/news/disrupting-AI-espionage>, news post, Accessed: 2026-04-18
6. Anthropic: Assessing claude mythos preview’s cybersecurity capabilities (2026), <https://red.anthropic.com/2026/mythos-preview/>, technical blog post, Accessed: 2026-04-18
7. Arantes, R., Weir, C., Hannon, H., Kulseng, M.: Operationally transparent cyber (optc) (2021). <https://doi.org/10.21227/edq8-nk52>, <https://dx.doi.org/10.21227/edq8-nk52>
8. Benabderrahmane, S., Valtchev, P., Cheney, J., Rahwan, T.: Apt-llm: Embedding-based anomaly detection of cyber advanced persistent threats using large language models. arXiv preprint arXiv:2502.09385 (2025)
9. Bhatt, P., Yano, E.T., Gustavsson, P.: Towards a framework to detect multi-stage advanced persistent threats attacks. In: 2014 IEEE 8th international symposium on service oriented system engineering. pp. 390–395. IEEE (2014)
10. Büchel, M., Paladini, T., Longari, S., Carminati, M., Zanero, S., Binyamini, H., Engelberg, G., Klein, D., Guizzardi, G., Caselli, M., et al.: {SoK}: Automated {TTP} extraction from {CTI} reports—are we there yet? In: 34th USENIX security symposium (USENIX Security 25). pp. 4621–4641 (2025)
11. Chakkaravarthy, S.S., Vaidehi, V., Rajesh, P.: Hybrid analysis technique to detect advanced persistent threats. *International Journal of Intelligent Information Technologies (IJIT)* **14**(2), 59–76 (2018)
12. Chu, W.L., Lin, C.J., Chang, K.N.: Detection and classification of advanced persistent threats and attacks using the support vector machine. *Applied Sciences* **9**(21), 4579 (2019)
13. Čík, I., Rasamoelina, A.D., Mach, M., Sinčák, P.: Explaining deep neural network using layer-wise relevance propagation and integrated gradients. In: 2021 IEEE 19th world symposium on applied machine intelligence and informatics (SAMI). pp. 000381–000386. IEEE (2021)
14. CodeWall: How we hacked mckinsey’s ai platform (2026), <https://codewall.ai/blog/how-we-hacked-mckinseys-ai-platform>, research post, Accessed: 2026-04-18
15. Corporation, T.M.: Caldera, <https://github.com/mitre/caldera>
16. Corporation, T.M.: Mitre att&ck, <https://attack.mitre.org/>
17. Corporation, T.M.: Mitre att&ck enterprise techniques, <https://attack.mitre.org/techniques/enterprise/>
18. Corporation, T.M.: Mitre att&ck: T1136 - create account, <https://attack.mitre.org/techniques/T1136/>
19. Corporation, T.M.: Mitre att&ck: T1136.001 - create account: Local account, <https://attack.mitre.org/techniques/T1136/001/>

20. Corporation, T.M.: Mitre att&ck version history, <https://attack.mitre.org/resources/versions/>
21. DARPA Transparent Computing (TC) Program: Darpa transparent computing engagement 3 data release. <https://github.com/darpa-i2o/Transparent-Computing/blob/master/README-E3.md> (2018), accessed: 2025-11-18
22. DARPA Transparent Computing (TC) Program: Darpa transparent computing engagement 5 data release. <https://github.com/darpa-i2o/Transparent-Computing> (2020), accessed: 2025-11-18
23. Deng, J., Li, X., Chen, Y., Bai, Y., Weng, H., Liu, Y., Wei, T., Xu, W.: Raconteur: A knowledgeable, insightful, and portable llm-powered shell command explainer. arXiv preprint arXiv:2409.02074 (2024)
24. Do Xuan, C.: Detecting apt attacks based on network traffic using machine learning. *Journal of Web Engineering* **20**(1), 171–190 (2021)
25. Do Xuan, C., Duong, D.: Optimization of apt attack detection based on a model combining attention and deep learning. *Journal of Intelligent & Fuzzy Systems* **42**(4), 4135–4151 (2022)
26. Dong, F., Wang, L., Nie, X., Shao, F., Wang, H., Li, D., Luo, X., Xiao, X.: DISTDET: A cost-effective distributed cyber threat detection system. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 6575–6592 (2023)
27. Du, M., Li, F., Zheng, G., Srikumar, V.: Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC conference on computer and communications security. pp. 1285–1298 (2017)
28. Elastic: Crowdstrike exported fields (2025), <https://www.elastic.co/docs/reference/beats/filebeat/exported-fields-crowdstrike>, accessed: 2025-05-01
29. Fang, C., Miao, N., Srivastav, S., Liu, J., Zhang, R., Fang, R., Asmita, Tsang, R., Nazari, N., Wang, H., Homayoun, H.: Large language models for code analysis: Do LLMs really do their job? In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 829–846. USENIX Association, Philadelphia, PA (Aug 2024), <https://www.usenix.org/conference/usenixsecurity24/presentation/fang>
30. Fang, Y., Zhou, X., Huang, C.: Effective method for detecting malicious powershell scripts based on hybrid features. *Neurocomputing* **448**, 30–39 (2021)
31. Fleschutz: Mega collection of powershell scripts. GitHub Repository (2023), <https://github.com/fleschutz/PowerShell>, accessed: March 12, 2025
32. Gandhi, P.A., Wudali, P.N., Amaru, Y., Elovici, Y., Shabtai, A.: Shield: Apt detection and intelligent explanation using llm. arXiv preprint arXiv:2502.02342 (2025)
33. Ghosal, S.S., Pal, S., Mukherjee, K., Manocha, D.: PromptRefine: Enhancing few-shot performance on low-resource Indic languages with example selection from related example banks. In: Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers). pp. 351–365 (2025)
34. Giura, P., Wang, W.: A context-based detection framework for advanced persistent threats. In: 2012 International Conference on Cyber Security. pp. 69–74. IEEE (2012)
35. Greenberg, S.: Using unix: Collected traces of 168 users (1988)
36. Han, X., Pasquier, T., Bates, A., Mickens, J., Seltzer, M.: Unicorn: Runtime provenance-based detector for advanced persistent threats. arXiv preprint arXiv:2001.01525 (2020)
37. Hasan, M.M., Islam, M.U., Uddin, J.: Advanced persistent threat identification with boosting and explainable ai. *SN Computer Science* **4**(3), 271 (2023)

38. Hossain, M.N., Milajerdi, S.M., Wang, J., Eshete, B., Gjomemo, R., Sekar, R., Stoller, S., Venkatakrishnan, V.: SLEUTH: Real-time attack scenario reconstruction from COTS audit data. In: 26th USENIX Security Symposium (USENIX Security 17). pp. 487–504 (2017)
39. Hu, Y., Zou, F., Han, J., Sun, X., Wang, Y.: Llm-tikg: Threat intelligence knowledge graph construction utilizing large language model. *Computers & Security* **145**, 103999 (2024)
40. Huang, S.Y., Yang, C.L., Lin, C.Y., Huang, C.Y.: CmdCaliper: A semantic-aware command-line embedding model and dataset for security research. In: Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing. pp. 20188–20206 (Nov 2024)
41. Hub, I.D.: Real powershell scripts dataset. Hugging Face Datasets (2024), <https://huggingface.co/datasets/infinite-dataset-hub/RealPowShellScripts>, accessed: March 12, 2025
42. Ioannou, G., Louvieris, P., Clewley, N., Powell, G.: A markov multi-phase transferable belief model: An application for predicting data exfiltration apts. Proceedings of the 16th International Conference on Information Fusion pp. 842–849 (2013), <https://api.semanticscholar.org/CorpusID:12702003>
43. Jia, Z., Xiong, Y., Nan, Y., Zhang, Y., Zhao, J., Wen, M.: MAGIC: Detecting advanced persistent threats via masked graph representation learning. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 5197–5214 (2024)
44. Karim, S.S., Afzal, M., Iqbal, W., Al Abri, D.: Advanced persistent threat (apt) and intrusion detection evaluation dataset for linux systems 2024. *Data in Brief* **54**, 110290 (2024)
45. Karlsen, E., Luo, X., Zincir-Heywood, N., Heywood, M.: Benchmarking large language models for log analysis, security, and interpretation. *J. Netw. Syst. Manag.* **32**(3) (Jul 2024)
46. King, S.T., Chen, P.M.: Backtracking intrusions. In: Proceedings of the nineteenth ACM symposium on Operating systems principles. pp. 223–236 (2003)
47. Krishnapriya, S., Singh, S.: A comprehensive survey on advanced persistent threat (apt) detection techniques. *Computers, Materials & Continua* **80**(2) (2024)
48. Lane, T., Brodley, C.E.: An application of machine learning to anomaly detection. In: Proceedings of the 20th national information systems security conference. vol. 377, pp. 366–380. Baltimore, USA (1997)
49. Li, S., Dong, F., Xiao, X., Wang, H., Shao, F., Chen, J., Guo, Y., Chen, X., Li, D.: Nodlink: An online system for fine-grained apt attack detection and investigation. arXiv preprint arXiv:2311.02331 (2023)
50. Li, X., Zhu, T., Zhang, W.: Efficient ransomware detection via portable executable file image analysis by LLaMA-7b. Research Square preprint (2023). <https://doi.org/10.21203/rs.3.rs-3679775/v1>, <https://www.researchsquare.com/article/rs-3679775/v1>
51. Lin, C.Y.: Rouge: A package for automatic evaluation of summaries. In: Text summarization branches out. pp. 74–81 (2004)
52. Lin, X.V., Wang, C., Zettlemoyer, L., Ernst, M.D.: Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system. arXiv preprint arXiv:1802.08979 (2018)
53. Liu, J., Shen, D., Zhang, Y., Dolan, B., Carin, L., Chen, W.: What makes good in-context examples for GPT-3? In: Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures. pp. 100–114 (2022)
54. Liu, P., Liu, J., Fu, L., Lu, K., Xia, Y., Zhang, X., Chen, W., Weng, H., Ji, S., Wang, W.: Exploring ChatGPT’s capabilities on vulnerability management. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 811–828 (2024)

55. Liu, Q., Bao, K., Hagenmeyer, V.: Aviator: A mitre emulation plan-derived living dataset for advanced persistent threat detection and investigation. In: 2024 IEEE International Conference on Big Data (BigData). pp. 5610–5619. IEEE (2024)
56. Liu, Z., Buford, J.: Anomaly detection of command shell sessions based on distilbert: Unsupervised and supervised approaches. arXiv preprint arXiv:2310.13247 (2023)
57. Lu, J., Chen, K., Zhuo, Z., Zhang, X.: A temporal correlation and traffic analysis approach for apt attacks detection. *Cluster computing* **22**, 7347–7358 (2019)
58. Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. *Advances in neural information processing systems* **30** (2017)
59. Lynch, M.: Claude code found a linux vulnerability hidden for 23 years (2026), <https://mtlynch.io/claude-code-found-linux-vulnerability/>, blog post, Accessed: 2026-04-18
60. May, J.: Common powershell commands. GitHub Gist (2023), <https://gist.github.com/johnathanmay/c83e3fc26d70ed6b7b6b99396740a80e>, accessed: March 12, 2025
61. Mayoral-Vilches, V., Navarrete-Lozano, L.J., Balassone, F., Sanz-Gómez, M., Chavez, C.R., de Torres, M.d.M., Turiel, V.: Cybersecurity ai: The world’s top ai agent for security capture-the-flag (ctf). arXiv preprint arXiv:2512.02654 (2025)
62. Mees, W.: Multi-agent anomaly-based APT detection. In: Information Assurance and Cyber Defence. pp. 1–10. No. STO-MP-IST-111 in NATO Science and Technology Organization (STO) Meeting Proceedings (2012), <https://publications.sto.nato.int/publications/STO%20Meeting%20Proceedings/STO-MP-IST-111/MP-IST-111-03.pdf>, paper MP-IST-111-03
63. Microsoft: About logging in windows - powershell. Microsoft Learn (2024), https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_logging_windows?view=powershell-7.5#enabling-script-block-logging, accessed: March 12, 2025
64. Microsoft: Overview of sysmon capabilities (2025), <https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon#overview-of-sysmon-capabilities>, accessed: 2025-05-01
65. Milajerdi, S.M., Gjomemo, R., Eshete, B., Sekar, R., Venkatakrishnan, V.: Holmes: real-time apt detection through correlation of suspicious information flows. In: 2019 IEEE symposium on security and privacy (SP). pp. 1137–1152. IEEE (2019)
66. Mutalib, N.H.A., Sabri, A.Q.M., Wahab, A.W.A., Abdullah, E.R.M.F., AlDahoul, N.: Explainable deep learning approach for advanced persistent threats (apts) detection in cybersecurity: a review. *Artificial Intelligence Review* **57**(11), 297 (2024)
67. Myneni, S., Chowdhary, A., Sabur, A., Sengupta, S., Agrawal, G., Huang, D., Kang, M.: Dapt 2020-constructing a benchmark dataset for advanced persistent threats. In: International workshop on deployable machine learning for security defense. pp. 138–163. Springer (2020)
68. Ning, P., Cui, Y., Reeves, D.S.: Constructing attack scenarios through correlation of intrusion alerts. In: Proceedings of the 9th ACM Conference on Computer and Communications Security. pp. 245–254 (2002)
69. Nong, Y., Yang, H., Cheng, L., Hu, H., Cai, H.: Automated software vulnerability patching using large language models. arXiv preprint arXiv:2408.13597 (2024)
70. Okuma, M., Watarai, K., Okada, S., Mitsunaga, T.: Automated mapping method for sysmon logs to att&ck techniques by leveraging atomic red team. In: 2023 6th International Conference on Signal Processing and Information Security (ICSPIS). pp. 104–109. IEEE (2023)
71. Patsakis, C., Casino, F., Lykousas, N.: Assessing llms in malicious code deobfuscation of real-world malware campaigns. *Expert Systems with Applications* **256**, 124912 (2024)

72. Qian, X., Zheng, X., He, Y., Yang, S., Cavallaro, L.: Lamd: Context-driven android malware detection and classification with llms. arXiv preprint arXiv:2502.13055 (2025)
73. Rani, N., Saha, B., Maurya, V., Shukla, S.K.: Ttphunter: Automated extraction of actionable intelligence as ttps from narrative threat reports. In: Proceedings of the 2023 Australasian Computer Science Week, pp. 126–134 (2023)
74. Red Hat, I.: Linux Audit Documentation (2024), <https://man7.org/linux/man-pages/man8/auditd.8.html>, accessed: 2025-12-03
75. Ren, W., Song, X., Hong, Y., Lei, Y., Yao, J., Du, Y., Li, W.: Apt attack detection based on graph convolutional neural networks. *International Journal of Computational Intelligence Systems* **16**(1), 184 (2023)
76. Riddle, A., Westfall, K., Bates, A.: Atlasv2: Atlas attack engagements, version 2. arXiv preprint arXiv:2401.01341 (2023)
77. Salim, D.T., Singh, M.M., Keikhosrokiani, P.: A systematic literature review for apt detection and effective cyber situational awareness (ecsa) conceptual model. *Heliyon* **9**(7) (2023)
78. Scarfone, K., Mell, P., et al.: Guide to intrusion detection and prevention systems (idps). NIST special publication **800**(2007), 94 (2007)
79. Schonlau, M., DuMouchel, W., Ju, W.H., Karr, A.F., Theus, M., Vardi, Y.: Computer intrusion: Detecting masquerades. *Statistical science* pp. 58–74 (2001)
80. Splunk: Hunting for malicious powershell using script block logging (2023), https://www.splunk.com/en_us/blog/security/hunting-for-malicious-powershell-using-script-block-logging.html, accessed: 2025-02-19
81. Splunk: Endpoint detection - powershell script block logging. Splunk Research (2024), <https://research.splunk.com/endpoint/bc1dc6b8-c954-11eb-bade-acde48001122/>, accessed: March 12, 2025
82. Su, Y., Li, M., Tang, C., Shen, R.: A framework of apt detection based on dynamic analysis. In: 2015 4th National Conference on Electrical, Electronics and Computer Engineering. pp. 1047–1053. Atlantis Press (2015)
83. The MITRE Corporation: Exfiltration Over Alternative Protocol (T1048). <https://attack.mitre.org/techniques/T1048/> (2024), mITRE ATT&CK Framework, Accessed: November 2025
84. The MITRE Corporation: Ingress Tool Transfer (T1105). <https://attack.mitre.org/techniques/T1105/> (2024), mITRE ATT&CK Framework, Accessed: November 2025
85. Tian, X., Mannekote, A., Solomon, C.E., Song, Y., Wise, C.F., Mcklin, T., Barrett, J., Boyer, K.E., Israel, M.: Examining llm prompting strategies for automatic evaluation of learner-created computational artifacts. In: Proceedings of the 17th International Conference on Educational Data Mining. pp. 698–706 (2024)
86. Turpin, M., Michael, J., Perez, E., Bowman, S.: Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. *Advances in Neural Information Processing Systems* **36**, 74952–74965 (2023)
87. Villeneuve, N., Bennett, J.: Detecting apt activity with network traffic analysis. *Trend Micro Incorporated Research Paper* pp. 1–13 (2012)
88. Vinay, V., Mangal, A.: Scade: Scalable command-line anomaly detection engine. arXiv preprint arXiv:2412.04259 (2024)
89. Wang, J., Cao, L., Luo, X., Zhou, Z., Xie, J., Jatowt, A., Cai, Y.: Enhancing large language models for secure code generation: A dataset-driven study on vulnerability mitigation. arXiv preprint arXiv:2310.16263 (2023)
90. Wang, Y., Liu, H., Li, Z., Su, Z., Li, J.: Combating advanced persistent threats: Challenges and solutions. *IEEE Network* (2024)

91. Welter, F., Wilkens, F., Fischer, M.: Tell me more: Black box explainability for apt detection on system provenance graphs. In: ICC 2023-IEEE International Conference on Communications. pp. 3817–3823. IEEE (2023)
92. Welz, L., Lanquillon, C.: Enhancing large language models through external domain knowledge. In: International Conference on Human-Computer Interaction. pp. 135–146. Springer (2024)
93. Xu, H., Si, C., Wang, C., Sun, P., Liu, Q., et al.: Aptsniffer: Detecting apt attack traffic using retrieval-augmented large language models. In: ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 1–5. IEEE (2025)
94. Yancey, K.P., Laffair, G., Verardi, A., Burstein, J.: Rating short l2 essays on the cefr scale with gpt-4. In: Proceedings of the 18th workshop on innovative use of NLP for building educational applications (BEA 2023). pp. 576–584 (2023)
95. Yu, H., Li, A., Jiang, R.: Needle in a haystack: attack detection from large-scale system audit. In: 2019 IEEE 19th International Conference on Communication Technology (ICCT). pp. 1418–1426. IEEE (2019)
96. Zhang, B., Gao, Y., Kuang, B., Yu, C., Fu, A., Susilo, W.: A survey on advanced persistent threat detection: A unified framework, challenges, and countermeasures. *ACM Comput. Surv.* (2024)
97. Zhang, J., Bu, H., Wen, H., Liu, Y., Fei, H., Xi, R., Li, L., Yang, Y., Zhu, H., Meng, D.: When LLMs meet cybersecurity: a systematic literature review (2024). arXiv preprint arXiv:2405.03644
98. Zhang, T., Kishore, V., Wu, F., Weinberger, K.Q., Artzi, Y.: Bertscore: Evaluating text generation with bert. In: International Conference on Learning Representations (2020), <https://openreview.net/forum?id=SkeHuCVFDr>

Appendix

This appendix provides examples referenced in the main text.

To detect APT-related activities, we employ Chain-of-Thought prompting to guide the model in generating its analysis prior to producing the final classification label. Figure 7 presents an example generated by Gemini-2.5-Pro. The left subfigure contains the prompts, while the right subfigure shows the model’s response, in which it first analyzes the code for APT-related activities and subsequently provides the classification label.

You are an expert cybersecurity analyst. Your task is to analyze a code sample which consists of a chronological sequence of code snippets.

⋮

2) Reasoning: Based only on the remaining behaviors (after excluding artifacts in Step 1), provide your analysis. Reference specific commands or actions and explain why they do or do not constitute concrete evidence of a coordinated attack.

3) Final Assessment: Based only on the analysis in your ‘Reasoning’ field, assign one of the following labels: ‘Attack’ or ‘Benign’.

Code Sample: {code-sample}

*“The code snippets provide **concrete evidence** of a **coordinated data theft operation**..*

*PowerShell command is executed to **recursively search**... file names containing keywords such as ***pass***, ***login***, and ***admin***, clearly targeting credentials..*

*All discovered files are collected and **compressed** into a single archive file named **working.zip** located in the %APPDATA% directory..*

*The actor... uses **Rar.exe** to re-archive the previously created **working.zip**. The command **rar.exe a -hpfGzq5yKw** ... uses the **-hp switch** to **encrypt** the new archive... obfuscating the stolen data...*

*The actor uses **sdelete64.exe**... to wipe the tools and staged data. The commands **.\sdelete64.exe ... "...\Rar.exe"**, **.\sdelete64.exe ... "...\working.zip"**... demonstrate a **deliberate effort to remove traces** of the attack.”*

“Final Assessment: Attack”

Fig. 7: Example of Explanation for APT Detection.

Figure 8 presents an example tagging summary generated by Gemini-2.5-Pro. The left subfigure provides the code snippet to be mapped to a TTP, along with the behavior description generated by the LLM. The right subfigure shows the top TTP technique the LLM identified for this code snippet, along with an explanation, alignment score, and the relevant excerpt.

<p style="text-align: center;">Code Snippet</p> <pre> \$env:APPDATA;\$files=ChildItem -Path \$env:USERPROFILE -Include *.doc;*.xps;*.xls;*.ppt;*.pps;*.wps;*.wpd;*.ods;*.odt;*.lwp;*.jtd; *.pdf;*.zip;*.rar;*.docx;*.url;*.xlsx;*.xltx;*.ppsx;*.pst;*.ost;*.psw; *.pass;*.login;*.admin;*.sifr;*.sifer;*.vpn;*.jpg;*.txt;*.lnk -Recurse -ErrorAction SilentlyContinue Select -ExpandProperty FullName; Compress-Archive -LiteralPath \$files -CompressionLevel Optimal -DestinationPath \$env:APPDATA\Draft.Zip -Force </pre>	<p>1. T1119 - Automated Collection</p> <ul style="list-style-type: none"> • Rank: 1 • Tag: T1119: Automated Collection • Tagging Explanation: The PowerShell script automates the process of identifying and gathering a wide range of files based on a predefined list of extensions and keyword patterns (-Include *.doc...*pass*). This scripted approach to gathering files of interest is the primary purpose of the code and aligns perfectly with automated collection. • Alignment Score: Score 5 (Excellent) • Cited Source: MITRE ATT&CK Technique T1119 • Relevant Excerpt: “Once established within a system or network, an adversary may use automated techniques for collecting internal data. Methods for performing this technique could include use of a Command and Scripting Interpreter to search for and copy information fitting set criteria such as file type, location, or name at specific time intervals.”
<p style="text-align: center;">Behavior Description</p> <p>This PowerShell script systematically searches a user's home directory and its subdirectories for a variety of files, including documents, archives, email data, images, and text files, while also specifically targeting filenames associated with credentials, administrative access, and VPN configurations. Once identified, the files are collected and compressed into a single ZIP archive, Draft.Zip, which is stored in the AppData directory, a hidden system folder often overlooked by users. By suppressing errors and recursively scanning directories, the script ensures it gathers as many files as possible without interruption, even if certain locations are restricted.</p> <p>The script does not initiate exfiltration but effectively stages sensitive data in a single compressed archive, making it easy to transfer through subsequent actions. Storing the archive in the AppData folder minimizes visibility and reduces the likelihood of immediate detection. The capability to systematically locate, collect, and consolidate files into an archive suggests an intent to facilitate data theft, unauthorized access, or operational persistence. While the script does not explicitly exhibit destructive behavior, its ability to aggregate potentially sensitive information—especially files containing credentials or login details—poses a significant risk to confidentiality and security.</p>	

Fig. 8: Example of MITRE ATT&CK Tagging Summary.