AutoMalDesc: Large-Scale Script Analysis for Cyber Threat Research

Alexandru-Mihai Apostu[1,2], **Andrei Preda**[1], **Alexandra Daniela Damir**[1], **Diana Bolocan**[1], **Radu Tudor Ionescu**[2], **Ioana Croitoru**[1], **Mihaela Gaman**[1]

[1]CrowdStrike
[2]University of Bucharest, Romania

Abstract

Generating thorough natural language explanations for threat detections remains an open problem in cybersecurity research, despite significant advances in automated malware detection systems. In this work, we present AutoMalDesc, an automated static analysis summarization framework that, following initial training on a small set of expert-curated examples, operates independently at scale. This approach leverages an iterative self-paced learning pipeline to progressively enhance output quality through synthetic data generation and validation cycles, eliminating the need for extensive manual data annotation. Evaluation across 3,600 diverse samples in five scripting languages demonstrates statistically significant improvements between iterations, showing consistent gains in both summary quality and classification accuracy. Our comprehensive validation approach combines quantitative metrics based on established malware labels with qualitative assessment from both human experts and LLM-based judges, confirming both technical precision and linguistic coherence of generated summaries. To facilitate reproducibility and advance research in this domain, we publish our complete dataset of more than 100K script samples, including annotated seed (0.9K) and test (3.6K) datasets, along with our methodology and evaluation framework.

Code — https://github.com/CrowdStrike/automaldesc

1 Introduction

As large language models continue to excel across diverse domains (Roberts et al. 2024; Wang et al. 2024a; Thirunavukarasu et al. 2023; Wang et al. 2023a; Rivas and Zhao 2023), their increasing data demands has sparked significant interest into self-rewarding and self-improving approaches (Dong et al. 2025; Zelikman et al. 2022; Liang et al. 2024; Huang et al. 2023; Gülçehre et al. 2023). This is particularly relevant in specialized fields where expert-labeled data is scarce and expensive to obtain.

The cybersecurity sector exemplifies these challenges, where rapidly evolving threats and restricted access to malware samples create substantial barriers to dataset creation (Guo et al. 2024b). The dynamic threat landscape, combined with scarce expert annotations and classified data, makes self-improvement techniques essential for advancing security-focused LLM applications. Beyond the data constraints, another challenge lies in generating detailed, human-readable descriptions of malware behavior. While current analysis tools excel at threat detection and classification (Wang et al. 2025; Rudd et al. 2024; Dambra et al. 2023; Eren et al. 2023; Wu et al. 2018), they fall short in automatically producing comprehensive behavioral descriptions (He et al. 2025). This gap between technical detection capabilities and the need for interpretable analysis represents a key bottleneck in current research (He et al. 2025; Jelodar et al. 2025; Fujii and Yamagishi 2024).

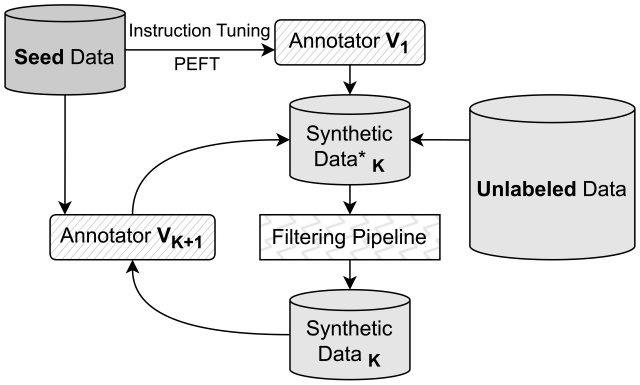

Figure 1: Self-training methodology. A *seed dataset* of 900 high-quality labeled samples initiates semi-supervised learning through pseudo-labeling. An LLM trained on seed data generates filtered pseudo-labels for unlabeled data, enabling iterative improvement with expanding training data.

To bridge these disparities, we present the first comprehensive study on self-improving LLMs for malware script analysis, with three key contributions:

- **Self-improving pipeline:** a novel iterative methodology for enhancing cybersecurity script analysis through LLM-generated annotations, with empirical validation of its effectiveness (see Figures 1 and 2).
- **Public dataset:** a balanced collection of 157K scripts (78K malicious, 79K benign), spanning 5 programming languages relevant for malware analysis (see Table 1).
- **Evaluation framework:** a comprehensive benchmark

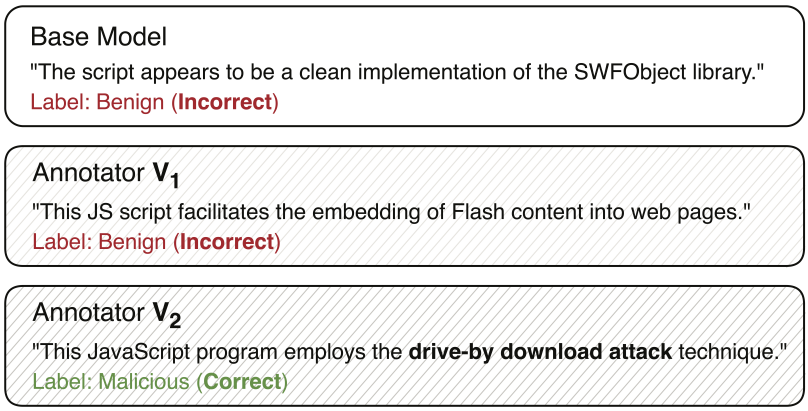

Figure 2: Evolution of model predictions for a malicious JavaScript program. The pretrained (base) model is unable to recognize the malicious intent of the code, focusing on surface-level details. Through self-learning, the model learns to detect malware and identify the attack mechanism.

	Initial Corpus*			Training Data*			Test Set			Seed Dataset		
Lang	Benign	Mal.	Total	Benign	Mal.	Total	Benign	Mal.	Total	Benign	Mal.	Total
sh	16,207	20,616	36,823	9,059	19,926	28,985	295	512	807	66	100	166
bat	5,000	2,838	7,838	2,775	2,575	5,350	594	226	820	64	103	167
js	35,000	30,000	65,000	21,242	14,132	35,374	380	475	855	95	139	234
ps	12,827	17,007	29,834	9,337	6,820	16,157	311	345	656	61	105	166
py	10,236	7,395	17,631	7,584	7,827	15,411	236	262	498	66	101	167
all	79,270	77,856	157,126	49,997	51,280	101,277	1,816	1,820	3,636	352	548	900

Table 1: Distribution of scripts across datasets by language and maliciousness labels. The seed and test sets underwent rigorous validation through multiple sources, including expert verification. *Initial corpus and training set labels are derived from internal labeling automation and shown for reference only. Due to the inherent noise of these labels, the training set was treated as unlabeled data. Languages: Bash (sh), Batch (bat), JavaScript (js), PowerShell (ps) and Python (py).

combining quantitative metrics (maliciousness label and language accuracy) with human and LLM-Judge assessments of technical precision and summary quality.

2 Related Work

Our research builds upon two key areas of prior work: self-improving language models that can enhance their capabilities through iterative learning, and static security analysis techniques for script-based threats.

2.1 Iterative and Self-Improving Approaches

Prior research shows that language models can enhance their capabilities through self-generated training data. Earlier work by Wang et al. (2023b) introduced Self-Instruct, bootstrapping from a small set of human prompts to generate and filter instruction-response pairs for model fine-tuning. Building on this concept, the STaR system (Zelikman et al. 2022) implemented a feedback loop where models generate step-by-step reasoning and learn from successful examples, significantly improving performance on mathematical and commonsense tasks. Several strategies have emerged to tackle self-improvement with limited ground-truth data. Huang et al. (2023) leveraged Chain-of-Thought prompting and majority voting across multiple reasoning paths to identify reliable predictions, while Li et al. (2024b) developed instruction backtranslation, by deriving prompts from web documents. For resource-constrained settings, Dong et al. (2025) proposed Self-Boost, which uses a single model to generate and verify new examples. Recent innovations include Yuan et al. (2024)'s self-rewarding models using LLM-based judgment, Gülçehre et al. (2023)'s reinforcement learning strategy, combining data generation with offline fine-tuning, and Wang et al. (2024b)'s extension of self-improvement to evaluation through model-generated comparisons.

Our work adapts these self-improvement techniques to cybersecurity, specifically addressing the critical challenge of data scarcity in malware analysis.

2.2 Static Security Analysis of Scripts

In cybersecurity, semi-supervised learning has consistently shown promise for reducing manual labeling requirements. While significant work has focused on Portable Executable malware analysis (Joyce et al. 2023; Plohmann et al. 2018; Corlătescu et al. 2023; Anderson and Roth 2018), script-based threats present unique challenges. Alam, Piplai, and Rastogi (2025) used pseudo-labeling to retrain malware classifiers on their own predictions, while Feng et al. (2025b) demonstrated the effectiveness of LLMs in Android

malware detection through feature extraction and prompt engineering. Regarding our work's main focus, natural language explanations of cyber threats, Fujii and Yamagishi (2024) showed that LLMs can accurately explain malware functions, though they highlighted the significant challenge of limited high-quality training data. Lu et al. (2025) tackled this limitation by creating a novel dataset called MalS, by using LLM-generated draft summaries with light human refinement, and achieving strong results through fine-tuning.

Our work extends these efforts by introducing an iterative self-training approach for script analysis across five languages. Unlike previous work, focused on binary classification or requiring human refinement, our method generates detailed security explanations, demonstrating consistent quality improvements through multiple iterations with minimal human input.

3 Dataset

We first describe our high-quality seed dataset used for initial model training. We then explain how we collected a larger unlabeled dataset from Virus Total, which is used in our iterative learning approach. Finally, we present the test set used for evaluation.

3.1 Seed Dataset

Given the scarcity and cost of expert labeling, we followed prior research (Li et al. 2024b; Wang et al. 2023b) and composed a seed dataset of 900 scripts from Virus Total. This dataset supports three tasks: malware detection, scripting language classification and natural language description generation. Table 1 shows the distribution across languages and maliciousness labels, while Figure 3 provides example entries with their corresponding descriptions.

To ensure quality of the malware tags, we combined multiple validation sources: (1) YARA rules for detecting known malicious patterns (Lockett 2021), (2) SANDBOX detonations for dynamic analysis (Güven 2024) and (3) expert manual verification. For generating high-quality descriptions of the scripts' behavior, we leveraged SANDBOX detonation reports for technical context and employ Llama-3.3-70B-Instruct (with the temperature $\tau = 0.3$). We note that the manual verification was conducted by subject matter experts (SMEs) with specialized cybersecurity expertise. They reviewed both the script content and corresponding SANDBOX detonation reports to ensure accurate classification of malicious and benign samples. This approach follows prior research methodologies (Li et al. 2024b; Corlătescu et al. 2023; Wang et al. 2023b; Anderson and Roth 2018), while addressing the challenges of obtaining expert-labeled data.

3.2 Training Data Collection

We collected 157,126 scripts from Virus Total, across five languages commonly associated with cybersecurity threats (Feng et al. 2025a; Li et al. 2024a; Srinivasan et al. 2023; Sohan and Basalamah 2020): Bash (sh), Batch (bat), JavaScript (js), PowerShell (ps) and Python (py). The initial corpus distribution is shown in Table 1 (first section). While the data appears balanced between benign and malicious samples, these labels – derived from an internal YARA rules powered noisy labeling automation – are shown for reference only and were not used for training due to their inherent noise.

Through multiple filtering and validation steps, including consistency checks and quality thresholds, we refined this initial corpus to 101,277 examples to get our final training dataset, as illustrated in the second section of Table 1. The complete filtering methodology and validation process are described in detail in Section 4.

3.3 Test Set

Following benchmarking research recommendations (Guo et al. 2024a), we sample 700 additional samples per language, creating a test set of 3,636 samples. The quality of the test set assurance follows the methodology described in Subsection 3.1, generating behavior-informed script summaries and validated scripting language and maliciousness labels. Analysis reveals that obfuscation techniques are present in at least 6% of test samples, reflecting real-world malware.

Finally, we release the three subsets described through this section: a filtered training set of 101,277 samples, a high-quality seed set of 900 samples, and a test dataset of 3,636 samples. Each entry contains the script's SHA256 identifier, maliciousness and scripting language labels, as well as the generated script summaries. The seed and test sets feature SANDBOX-informed summaries, while the training set summaries follow our iterative methodology, as detailed in Section 4.1. To ensure responsible disclosure, we do not include script contents in our public release, though all samples remain accessible through VirusTotal's platform to authenticated users who have completed requisite background verification and compliance checks.

4 Methodology

Our self-training methodology (Figure 1) leverages a small, high-quality seed dataset to fine-tune an LLM, which is then used to generate pseudo-labels for a larger corpus of unlabeled scripts. Through iterative refinement, we expand the training data in a self-supervised manner. This section details two key components: iterative dataset extension and self-training strategy. We evaluate our approach on three tasks: malware classification, programming language identification (collectively referred to as metadata in upcoming sections), and natural language behavior explanation.

4.1 Iterative Dataset Extension

This subsection touches on our synthetic data generation process and subsequent dataset refinement (Figure 4). We begin by fine-tuning a large language model on our seed dataset to create LLM Annotator V1, which we use to generate pseudo-labels for our initial 157,126 unlabeled scripts. Given the limited training data, we implement rigorous filtering to ensure quality, as described below.

Initial generation: Using Annotator V1 (Llama-3.3-70B-Instruct fine-tuned on seed data), we generate explanations and metadata (maliciousness labels and language identification) in JSON format for 157,126 scripts across multiple temperature settings.

Ex 1: Malicious Python (Discord Token Stealer)

SHA256: 9a8592356a...
Language: python
Label: malicious
Summary: This script executes a targeted attack, leveraging various techniques to exfiltrate sensitive user data and propagate itself. At its core, the script is designed to steal Discord tokens from multiple platforms, including Discord, Discord Canary, and Discord PTB, as well as Google Chrome, Opera, Brave, and Yandex browsers. The primary objective is to harvest these tokens, extract relevant user information, and transmit the data to a designated webhook.
[...remainder of summary omitted for brevity...]

Ex 2: Benign Bash (Ceph Cluster Monitor)

SHA256: 3277575ad...
Language: bash
Label: benign
Summary: This script is designed to provide an overview of a Ceph storage cluster's configuration and performance. It utilizes various commands, including `ceph osd pool get` and `ceph osd tree`, to gather information about the cluster's pools and OSD tree. The script also leverages `ssh` to connect to remote hosts and execute commands, such as `df -h` and `hdparm`, to retrieve disk usage and hardware information.
[...remainder of summary omitted for brevity...]

Figure 3: Representative examples from our dataset showing malicious and benign scripts. Seed and test set summaries are generated using LLMs informed by SANDBOX detonation reports, while training set summaries use our iterative methodology without detonation data. Code references are formatted in `monospace` for clarity.

Quality filtering: Our filtering pipeline consists of multiple stages. Initial cleanup removed empty responses (9,095), truncated summaries (17) and incomplete JSONs (7,825), reducing the dataset by 10.76%. Consistency checks across three temperatures (0.4, 0.6, 0.8) eliminated samples with inconsistent malicious labels, reducing data by another 14.28%. Using Phi-3.5-Mini, selected for its 99.5% accuracy on our test set, we removed 219 samples (0.22%) with discrepancies between summaries and maliciousness labels. Finally, we applied a 90% confidence threshold on logit probabilities, with most values clustering near 1 (see Figure 8 from the supplementary material). The resulting filtered dataset contains 101,277 examples (Table 1). This filtering strategy balances correctness with dataset size, though threshold values may need adjustment for different applications.

4.2 Self-Training Strategy

Our full self-training methodology is illustrated in Figure 1. To train the LLM annotators, we used an instruction-tuning approach to specialize these models for our tasks – i.e. malware detection, language classification and summary generation. Given the relatively small size of the datasets and large architectures used throughout our experiments, we decided to use Low-Rank Adaptation (LoRA) (Hu et al. 2022), known to efficiently adapt large models with minimal trainable parameters. Each iteration follows the same training process but uses different datasets: Annotator V1 is trained on the seed data, while Annotator V2 incorporates both seed and V1-generated samples. To prevent overfitting, we monitor the validation loss on a held-out subset, training until convergence (11 epochs for V1, trained on the smaller seed dataset; 13 epochs for subsequent iterations).

Language Detection Accuracy (%)

Model	sh	bat	js	ps	py	Avg.
Base	89.2	92.3	98.4	89.9	100	93.7
V1	98.0	94.2	99.5	92.7	99.8	96.8
V2	99.3	94.9	99.8	91.8	100	97.1

Malware Detection Accuracy (%)

Model	bash	batch	js	ps	py	Avg.
Base	92.4	52.7	90.8	94.2	89.8	83.1
V1	93.2	77.9	90.6	95.3	91.8	89.3
V2	96.3	82.4	92.2	95.3	92.6	91.5

Table 2: Accuracy percentages for language detection and malware detection across three model versions (Base: pretrained Llama-3.3-70B-Instruct, V1: first iteration, V2: second iteration) for five scripting languages. Note the significant improvement in batch script malware detection (52.7% increase from Base to V2) and consistent gains across languages through our iterative training.

5 Experiments

In this section, we succinctly present our experimental setup and configuration parameters, followed by a detailed analysis of our self-improving method's performance and results.

5.1 Experimental Setting

Through our experiments, we fine-tuned Llama-3.3-70B-Instruct using Low-Rank Adaptation (LoRA) on NVIDIA H100 Mega GPUs. Base hyperparameters included: 16k context size, batch size of 1, learning rate of 0.0001, weight decay of 0.001, and warmup ratio of 0.05, with gradient checkpointing and packing enabled for memory optimiza-

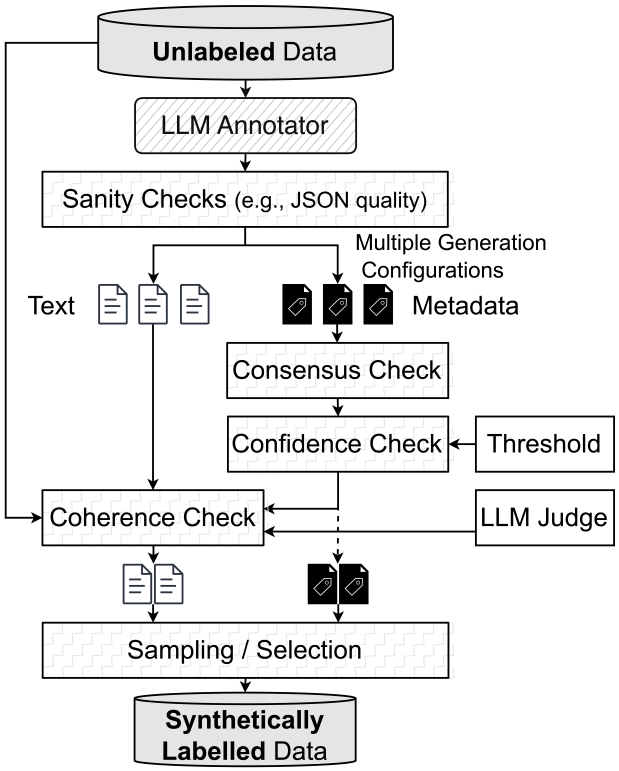

Figure 4: Pseudo-label quality filtering pipeline. The LLM annotator generates metadata and natural language explanations using multiple configurations (e.g. temperature settings) to ensure reliability. Samples pass filtering if metadata remains consistent across configurations and meets confidence thresholds. An LLM judge then verifies coherence between generated explanations and metadata labels.

tion. Annotator V1 used LoRA parameters (rank of 8, $\alpha = 16$, dropout rate of 0.05) and target modules set to "all-linear", training for 15 epochs with epoch 11 providing optimal performance. Annotator V2 maintained the base configuration but doubled the LoRA rank and α (to 16 and 32, respectively), and increased dropout to 0.1. Of its 20 training epochs, epoch 13 proved optimal.

Best-performing checkpoints were selected based on validation loss convergence to prevent overfitting, then evaluated on our held-out test dataset. Additional details are provided in supplementary materials.

5.2 Results

We evaluate our annotator models, V1 and V2, through both qualitative and quantitative analysis. For detection capabilities (i.e. language identification and malware detection), we use our test set of approximately 3,600 samples (Subsection 3.3), comparing performance against the base Llama-3.3-70B-Instruct model. For script comprehension, we assess 215 samples under 3KB across the five languages in scope (Bash, Batch, Javascript, PowerShell, and Python). Both human and LLM judges performed pairwise evaluations of generated summaries, with the option to rate V1 and V2 outputs as equivalent when appropriate.

Model Evaluation Results (V1 vs. V2)

Evaluator	V1	V2
Human Annotators	46.26%	44.85%
Llama-3.3-70B-Instruct	48.85%	51.15%
Claude 3.7	47.57%	52.43%
Phi-3.5-Mini-Instruct	56.22%	43.78%
Mixtral-7x22B-Large	52.07%	47.93%
GPT-4o	51.15%	48.85%
GPT-5 (low)	47.47%	52.53%

Table 3: Comparison of V1 vs. V2 performance across different evaluators. Both human and LLM judges evaluated approximately 215 samples each. Percentages show the win-rate of the model version where the samples were preferred. The table excludes 19 cases rated equivalent by human evaluators, as LLM judges did not have this option.

Detection capabilities: The base model, Llama-3.3-70B-Instruct, demonstrates strong language detection capabilities even prior to fine-tuning, achieving an average accuracy of 93.70% across all scripting languages. It performs particularly well for Javascript and Python, but it is less robust for Bash (89.22%). Its malware detection varies significantly, from over 90% accuracy for Bash, Javascript and PowerShell, to 52.68% for Batch scripts, averaging at 83.09%.

Our fine-tuning process yielded substantial improvements in both detection tasks. V1 increased Bash language detection by 8.8%, while V2 achieved over 99% language accuracy for Bash and Javascript, and perfect 100% for Python. Similarly, malware detection improved across all languages, with enhancements ranging from 1.1% to 3.8% for most languages, and a significant 29.9% improvement for Batch.

McNemar's test (McNemar 1947) confirmed that the improvements of V2 over V1 are statistically significant ($p < 10^{-5}$), with V2 correcting 110 false positives and 49 false negatives from V1's predictions.

These results highlight the effectiveness of our fine-tuning approach in enhancing the model's ability to correctly identify both scripting languages and malicious samples, particularly addressing specific weaknesses in the base model's detection capabilities for previously challenging script types.

Human evaluation of script comprehension: We evaluated the generated summaries through both human and LLM judges, with human evaluations conducted blindly to prevent bias. Each human judge assessed approximately 20 randomized pairs of summaries, providing their preference and reasoning behind the selection.

Human evaluators found that both models produced similar and factually accurate summaries in 50.42% of the cases, demonstrating comparable script comprehension capabilities. Preferences between models primarily reflected stylistic differences (e.g. verbosity, level of detail, information flow and structure), rather than factual accuracy. Despite these personal preferences influencing decisions, the manual evaluation showed a marginal preference for V1 at 50.46%, with V1 being selected 54 times compared to V2's 53 se-

LLM-as-a-Judge Prompt for Evaluating Script Comprehension

You are a senior malware analyst specializing in technical threat assessment.
Review these two technical analyses of the same script using the additive 5-point scoring rubric described by the criterion below:

- Award 1 point if the analysis identifies the basic nature of the script (language and/or general purpose).
- Award 1 additional point if the analysis describes key technical elements (commands, methods, flags, etc.).
- Award 1 point if the analysis correctly identifies security implications (or lack thereof) and/or system impact.
- Award 1 point if the analysis is comprehensive, well-structured, provides specific technical details, and potential obfuscation techniques.
- Award 1 point for an analysis that demonstrates expert-level understanding, precise cybersecurity terminology, and rigorous explanation of the script's functionality in natural, human-like language.

When comparing analyses that receive the same technical score, consider:

- Which analysis balances technical accuracy and readability?
- Which analysis would be more useful in a real-world threat assessment context?

SCRIPT: <code> {code_content} <\code>

ANALYSIS A:
{gt_language}
{summary_A}

ANALYSIS B:
{gt_language}
{summary_B}

After examining the original script and both responses:

1. Briefly justify your total score for each analysis (up to 100 words), considering technical accuracy, comprehensiveness, clarity, and natural language quality.
2. You MUST declare a single winner - there can be no ties. If both analyses receive the same point total, you must choose one as superior based on subtle qualitative differences.
3. Your final line MUST be EXACTLY "Winner: A" or "Winner: B" - no other text, no asterisks, no variations.

Figure 5: Full evaluation prompt used in evaluating Script Comprehension capabilities for model V1 and V2. This same prompt was used in all four judge models: Claude 3.7 Sonnet, Llama 3.3 70B Instruct, Phi 3.5 Mini Instruct, and Mixtral 7x22B Large.

lections, and 9 instances where both models were rated equally effective. On a less positive note, we observed 27 total instances of hallucinations across both models (V1: 13, V2: 14), with varying distribution across programming languages. V1 had higher susceptibility to hallucination when processing PowerShell (5 instances) and Batch/Python scripts (3 each), with perfect reliability for Javascript samples. In contrast, V2 struggled primarily with Batch scripts (8 instances), but improved on Python (only 1 instance).

LLMs judging script comprehension: To complement human evaluations, we employed multiple LLM judges. Using a standardized prompt (Figure 5), each LLM evaluated summaries for script identification, technical coverage, security impact, composition quality and cybersecurity terminology accuracy. As indicated in Table 3, three of the six LLM judges aligned with human evaluators in preferring V1, Claude 3.7 Sonnet and GPT-5 notably diverged, showing a stronger preference for V2. The other LLM judges consistently favored V1. The split in LLM preferences suggests that different evaluation frameworks may emphasize different qualities in script summaries, when factual accuracy is consistently high. Notably, Phi-3.5-Mini-Instruct, with its smaller architecture (3.8B parameters), was unable to complete the full evaluation set, likely due to its limited capacity compared to larger models.

6 Discussion

This section extends beyond the evaluation presented in Subsection 5.2, focusing on two key aspects. First, we discuss how SANDBOX-informed seed summaries improve generation quality through pattern preservation across iterations. Second, we present detailed human observations on stylistic evolution between Annotator V1 and V2, providing deeper insights into our models' generation characteristics.

Knowledge transfer from detonation reports: To assess the effectiveness of SANDBOX-informed seed annotations, we analyzed behavioral pattern preservation between seed-summaries and descriptions generated by Annotator V1. Table 4 presents a few examples of shorter common patterns found in both datasets, with additional longer examples including *"Queries sensitive IE security settings"*, *"Calls an API typically used to create an HTTP or FTP session"* and *"The input file contains long base64 encoded strings"*. Concretely, over 50 seed summaries, and over $1,300$ V1 sum-

Phrase	Seed	V1
Writes data to a remote process	10	141
Writes registry keys	9	32
Creates new processes	8	98
Modifies proxy settings	8	48
Deletes registry keys	8	26
Executes a shell command	7	214
Executes a JavaScript file	5	269
Queries process information	5	24
Queries sensitive IE security settings	5	30
Drops cabinet archive files	5	9
Invokes the C# compiler	3	10
Searches for file content	2	256

Table 4: Examples of behavioral patterns present in both SANDBOX-informed seed summaries and subsequent V1-generated descriptions.

maries contained exact matches. The presence of these patterns in over 50 seed summaries and 1,300 V1-generated descriptions demonstrates successful transfer of SANDBOX-derived knowledge through fine-tuning.

Especially, for annotating large datasets, our approach offers significant advantages over traditional detonation analysis. While SANDBOX detonations are expensive, time-consuming and dependent on specific environmental conditions, LLM-based analysis provides faster, and, depending on the use-case, even more cost-effective insights. Furthermore, static analysis can identify potential behaviors even when dynamic execution fails, particularly for scripts dependent on inaccessible resources, sandbox-aware malware, or specific environmental configurations.

Summary generation style progression: Our human evaluation revealed distinct stylistic patterns between V1 and V2 summaries despite their similar technical accuracy. V1 maintained a more technical and conservative approach, focusing on specific implementation details like command flags, memory allocation parameters, and API calls. V2 produced more readable summaries with improved narrative flow, while showing several noteworthy characteristics. First, V2 demonstrated a tendency toward dramatic characterization, occasionally describing scripts as "formidable threats" and making broader assumptions about script intent. This was particularly evident in its treatment of benign scripts, where it sometimes overestimated security risks, while V1 maintained a more neutral stance. Annotators consistently noted this difference, often preferring V1's more measured approach for clean scripts. Second, while V2's summaries were generally more readable and fluid, they occasionally exhibited redundancy, particularly when describing obfuscation techniques or encoding schemes. V1's concise, technical descriptions were often preferred despite being less narrative in style. These observations suggest a trade-off between technical precision and natural language fluency across iterations. While V2 improved readability and narrative flow, it also introduced a bias toward assuming malicious intent and occasionally sacrificed precision for expressiveness. This pattern was consistent across multiple annotators, who frequently noted that their preferences were driven by summary style rather than technical accuracy, as both models maintained similar levels of factual correctness. Importantly, this V1/V2 trade-off serves distinct operational needs: V1 offers technical precision for engineers and malware analysts requiring detailed implementation specifics, while V2 prioritizes readability for broader audiences, without sacrificing classification performance.

7 Limitations

Our approach faces three limitations. First, model context length constraints prevent analysis of longer scripts, particularly affecting PowerShell and JavaScript samples with extensive code blocks. Second, language detection accuracy (97.14% in V2) shows minimal improvement across iterations, suggesting a performance ceiling. Third, we observe a clear trade-off between readability and technical precision, with V2 favoring natural language flow at the occasional expense of technical accuracy, a difference that human evaluators found subtle and largely stylistic.

8 Conclusions

In this work, we demonstrated how self-improving language models can enhance malware script analysis, achieving statistically significant improvements in classification and summary generation with minimal human supervision. Our approach notably improved batch script malware detection from near-random (52.7%) to a robust (82.4%) accuracy across two iterations. To support reproducibility and future research, we release a dataset of more than 100K examples, spanning five scripting languages with balanced representation of benign and malicious samples. Furthermore, we make available an evaluation framework, combining quantitative metrics with human and LLM-based qualitative assessment, establishing a robust benchmark for future work in this domain. Results confirm that LLMs can effectively learn to identify malicious behavior patterns from limited seed data, substantially reducing reliance on costly SANDBOX detonations and expert annotations.

Ethical Statement

To balance research accessibility with security concerns and responsible disclosure practices, we release only SHA256 identifiers of scripts along with the associated artefacts derived from our research, requiring VirusTotal authentication for accessing the scripts content. Thus, aligned with established security research practices and industry standards, our publicly available dataset includes comprehensive metadata, labels and summaries, enabling research without exposing harmful code, while maintaining reproducibility.

Acknowledgments

The authors would like to thank Bilal Issa, Ioana Angela Ileni, Mihai Alestar and Monica Pascu for their input and expertise. The authors would further like to thank Sven Krasser, Cheryl Houser, Ciprian Bejan, Dragos Corlatescu, Alexandru Dinu, John Zuehlke and Marian Radu for their help and support.

References

Alam, M. T.; Piplai, A.; and Rastogi, N. 2025. ADAPT: A Pseudo-labeling Approach to Combat Concept Drift in Malware Detection. *arXiv preprint arXiv:2507.08597*.

Anderson, H.; and Roth, P. 2018. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *arXiv preprint arXiv:1804.04637*.

Corlătescu, D. G.; Dinu, A.; Găman, M.; and Sumedrea, P. 2023. EMBERSim: a large-scale databank for boosting similarity search in malware analysis. In *Proceedings of NeurIPS*.

Dambra, S.; Han, Y.; Aonzo, S.; Kotzias, P.; Vitale, A.; Caballero, J.; Balzarotti, D.; and Bilge, L. 2023. Decoding the Secrets of Machine Learning in Malware Classification: A Deep Dive into Datasets, Feature Extraction, and Model Performance. In *Proceedings of CCS*, 60–74.

Dong, Q.; Dong, L.; Zhang, X.; Sui, Z.; and Wei, F. 2025. Self-Boosting Large Language Models with Synthetic Preference Data. In *Proceedings of ICLR*.

Eren, M. E.; Bhattarai, M.; Joyce, R. J.; Raff, E.; Nicholas, C.; and Alexandrov, B. S. 2023. Semi-Supervised Classification of Malware Families Under Extreme Class Imbalance via Hierarchical Non-Negative Matrix Factorization with Automatic Model Selection. *ACM Transactions on Privacy and Security*, 26(4): 1–27.

Feng, P.; Xi, N.; Jin, J.; Zhang, J.; and Ma, J. 2025a. Flash: Federated Graph Learning-Based Malicious Bash Script Detection for Industrial Cyber–Physical Systems. *IEEE Transactions on Industrial Informatics*, 21(6): 4979–4989.

Feng, R.; Chen, H.; Wang, S.; Karim, M. M.; and Jiang, Q. 2025b. LLM-MalDetect: A Large Language Model-Based Method for Android Malware Detection. *IEEE Access*, 13: 81347–81364.

Fujii, S.; and Yamagishi, R. 2024. Feasibility study for supporting static malware analysis using LLM. In *Proceedings of ESORICS*, 5–28.

Gülçehre, Ç.; Paine, T. L.; Srinivasan, S.; Konyushkova, K.; Weerts, L.; Sharma, A.; et al. 2023. Reinforced Self-Training (ReST) for Language Modeling. *arXiv preprint arXiv:2308.08998*.

Guo, C.; Liu, X.; Xie, C.; Zhou, A.; Zeng, Y.; Lin, Z.; Song, D.; and Li, B. 2024a. RedCode: risky code execution and generation benchmark for code agents. In *Proceedings of NeurIPS*, volume 37, 106190–106236.

Guo, W.; Xu, Z.; Liu, C.; Huang, C.; Fang, Y.; and Liu, Y. 2024b. An Empirical Study of Malicious Code In PyPI Ecosystem. In *Proceedings of ASE*, 166–177.

Güven, M. 2024. Dynamic Malware Analysis Using a Sandbox Environment, Network Traffic Logs, and Artificial Intelligence. *International Journal of Computational and Experimental Science and Engineering*, 10(3): 480–490.

He, Y.; She, H.; Qian, X.; Zheng, X.; Chen, Z.; Qin, Z.; and Cavallaro, L. 2025. On Benchmarking Code LLMs for Android Malware Analysis. In *Proceedings of ISSTA*, 153–160.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *Proceedings of ICLR*.

Huang, J.; Gu, S. S.; Hou, L.; Wu, Y.; Wang, X.; Yu, H.; and Han, J. 2023. Large Language Models Can Self-Improve. In *Proceedings of EMNLP*, 1051–1068.

Jelodar, H.; Bai, S.; Hamedi, P.; Mohammadian, H.; Razavi-Far, R.; and Ghorbani, A. 2025. Large Language Model (LLM) for Software Security: Code Analysis, Malware Analysis, Reverse Engineering. *arXiv preprint arXiv:2504.07137*.

Joyce, R. J.; Amlani, D.; Nicholas, C.; and Raff, E. 2023. MOTIF: A malware reference dataset with ground truth family labels. *Computers & Security*, 124: 102921.

Li, R.; Zhang, C.; Chai, H.; Ying, L.; Duan, H.; and Tao, J. 2024a. PowerPeeler: A Precise and General Dynamic Deobfuscation Method for PowerShell Scripts. In *Proceedings of CCS*, 4539–4553.

Li, X.; Yu, P.; Zhou, C.; Schick, T.; Levy, O.; Zettlemoyer, L.; Weston, J. E.; and Lewis, M. 2024b. Self-Alignment with Instruction Backtranslation. In *Proceedings of ICLR*.

Liang, Y.; Zhang, G.; Qu, X.; Zheng, T.; Guo, J.; Du, X.; Yang, Z.; Liu, J.; Lin, C.; Ma, L.; Huang, W.; and Zhang, J. 2024. I-SHEEP: Self-Alignment of LLM from Scratch through an Iterative Self-Enhancement Paradigm. *arXiv preprint arXiv:2408.08072*.

Lockett, A. 2021. Assessing the Effectiveness of YARA Rules for Signature-Based Malware Detection and Classification. *arXiv preprint arXiv:2111.13910*.

Lu, H.; Peng, H.; Nan, G.; Cui, J.; Wang, C.; Jin, W.; Wang, S.; Pan, S.; and Tao, X. 2025. Malsight: Exploring malicious source code and benign pseudocode for iterative binary malware summarization. *IEEE Transactions on Information Forensics and Security*, 20: 6733–6747.

McNemar, Q. 1947. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2): 153–157.

Plohmann, D.; Clauss, M.; Enders, S.; and Padilla, E. 2018. Malpedia: A Collaborative Effort to Inventorize the Malware Landscape. *The Journal on Cybercrime & Digital Investigations*, 3(1): 1–19.

Rivas, P.; and Zhao, L. 2023. Marketing with ChatGPT: Navigating the Ethical Terrain of GPT-Based Chatbot Technology. *AI*, 4(2): 375–384.

Roberts, J.; Luddecke, T.; Sheikh, R.; Han, K.; and Albanie, S. 2024. Charting New Territories: Exploring the Geographic and Geospatial Capabilities of Multimodal LLMs. In *Proceedings of CVPRW*, 554–563.

Rudd, E. M.; Krisiloff, D.; Coull, S.; Olszewski, D.; Raff, E.; and Holt, J. 2024. Efficient Malware Analysis Using Metric Embeddings. *Digital Threats: Research and Practice*, 5(1): 1–20.

Sohan, M. F.; and Basalamah, A. 2020. A Systematic Literature Review and Quality Analysis of Javascript Malware Detection. *IEEE Access*, 8: 190539–190552.

Srinivasan, D.; Muthuvel, A.; Kumar, M. P.; Dinesh, R.; and Prasannakumar, V. G. 2023. Advanced Malware Analysis and Prevention. In *Proceedings of STCR*, volume 1, 1–6.

Thirunavukarasu, A.; Ting, D.; Elangovan, K.; Gutierrez Sinisterra, L.; Tan, T.; and Ting, D. 2023. Large language models in medicine. *Nature Medicine*, 29: 1930–1940.

Wang, H.; et al. 2023a. Scientific discovery in the age of artificial intelligence. *Nature*, 620: 47–60.

Wang, J.; Huang, Y.; Chen, C.; Liu, Z.; Wang, S.; and Wang, Q. 2024a. Software Testing With Large Language Models: Survey, Landscape, and Vision. *IEEE Transactions on Software Engineering*, 50(4): 911–936.

Wang, L.; Xue, J.; Wen, H.; Wang, Y.; Zhang, J.; and Liu, Z. 2025. RL4Mal: Representation learning-based malware classification under long-tailed distribution. In *Proceedings of CNCC*, 41–51.

Wang, T.; Kulikov, I.; Golovneva, O.; Yu, P.; Yuan, W.; Dwivedi-Yu, J.; Pang, R. Y.; Fazel-Zarandi, M.; Weston, J.; and Li, X. 2024b. Self-taught evaluators. *arXiv preprint arXiv:2408.02666*.

Wang, Y.; Kordi, Y.; Mishra, S.; Liu, A.; Smith, N. A.; Khashabi, D.; and Hajishirzi, H. 2023b. Self-Instruct: Aligning language models with self-generated instructions. In *Proceedings of ACL*, volume 1, 13484–13508.

Wu, C.; Shi, J.; Yang, Y.; and Li, W. 2018. Enhancing Machine Learning Based Malware Detection Model by Reinforcement Learning. In *Proceedings of ICCNS*, 74–78.

Yuan, W.; Pang, R. Y.; Cho, K.; Li, X.; Sukhbaatar, S.; Xu, J.; and Weston, J. 2024. Self-rewarding language models. In *Proceedings of ICML*, 57905–57923.

Zelikman, E.; Wu, Y.; Mu, J.; and Goodman, N. D. 2022. STaR: Self-taught reasoner bootstrapping reasoning with reasoning. In *Proceedings of NeurIPS*, 15476–15488.