EVALUATING LLM JUDGES IN CYBERSECURITY SCRIPT ANALYSIS

Alexandra Daniela Damir, Apostu Alexandru-Mihai, Diana Bolocan, Andrei Preda, Ioana Croitoru, Mihaela Gaman, Laura Vasilie, Bilal Issa & Monica-Nicoleta Pascu
CrowdStrike

ABSTRACT

Building on Large Language Models' (LLMs) increasing usage as judges for evaluating natural language outputs, this paper examines which models generate responses ranking higher in expert evaluation of cybersecurity script analyses. Our newly constructed dataset of $1{,}000$+ clean and malicious scripts, with expert-curated natural language summaries, serves as a reference for the evaluation of behavioral script summarization task performed by a candidate LLM. Several judge LLMs are asked to evaluate responses generated by the candidate LLM using human responses as reference. Through manual assessment of judge evaluations, we identify those models with outputs rated higher by experts in cybersecurity contexts and analyze the factors influencing judge quality, including self-preference bias and prompting strategy effects. Our publicly released dataset supports continued research in this domain where accurate evaluation is increasingly vital.

1 INTRODUCTION

The proliferation of LLM-generated content needs scalable evaluation methods that preserve human-quality assessments. LLM-as-a-Judge approaches Li et al. (2025), use LLMs to analyze outputs against defined criteria Badshah & Sajjad (2024), surpassing traditional heuristic metrics by employing reasoning and contextual understanding rather than simple pattern matching Whitehouse et al. (2025). Recent research in this space aims to address scalability challenges while maintaining assessment quality, through improved evaluation methodologies Zhu et al. (2025), bias mitigation Thakur et al. (2024); Ye et al. (2025), and domain-specific applications Zhang et al. (2025); Motlagh et al. (2024).

While LLM judges show promise in general domains Zheng et al. (2023); Chiang & yi Lee (2023), their performance in specialized domains with high-stakes implications, like cybersecurity, remains underexplored. Though LLMs in security and code analysis are gaining research attention Zhang et al. (2025); Sun et al. (2025); Ahmed & Devanbu (2022), studies on LLM judges Shi et al. (2024); Chen et al. (2024); Wang et al. (2023) have primarily addressed general knowledge domains, leaving the nuanced requirements largely unexplored. Existing benchmarks for code summarization Lu et al. (2021); Li et al. (2025); Husain et al. (2019), lack malicious code examples, which reveals a critical gap since malicious code requires specialized expertise to characterize accurately.

Automated evaluation for cybersecurity use-cases presents distinct challenges, requiring technical precision and domain expertise to identify subtle malicious behaviors often concealed Qamar (2023) within innocuous-appearing code. For Threat Analysts who routinely assess scripts of varying risk levels He & Vechev (2023), automated analysis of natural language explanations could significantly enhance threat detection and incident response workflows. Despite growing LLM adoption in security applications Mitra et al. (2024); Deng et al. (2024); Karlsen et al. (2024), a fundamental question persists: Which LLMs align more with human expert judgment when evaluating cybersecurity script behavior explanations?

This paper presents the first comprehensive study of LLM-as-a-judge performance in cybersecurity script analysis, bridging code understanding and security assessment for automated workflows. Our contributions include:

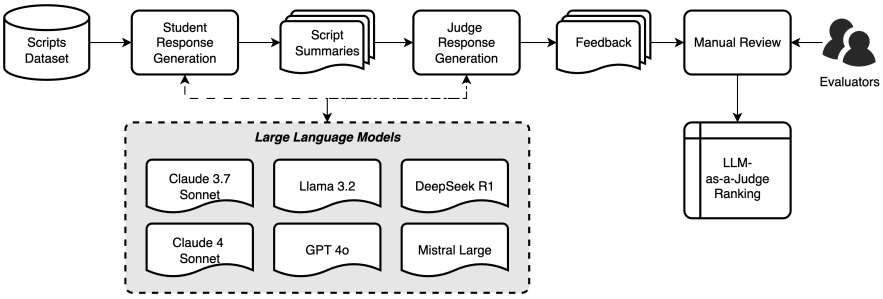

Figure 1: Manual evaluation pipeline with human annotations

1. A benchmark dataset, named ScriptSecEval, of $1{,}085$ manually curated scripts across five programming languages (Bash, Batch, Python, JavaScript, and PowerShell), each with expert-written behavioral summaries capturing security implications.
2. An extensive evaluation of state-of-the-art LLMs in both candidate and judge roles, examining inter-model dynamics, self-preference bias, and assessment reliability in security contexts.
3. Empirical findings identifying which models best align with expert judgment and revealing key factors affecting judge reliability—including hallucination patterns, technical accuracy, and assessment consistency—critical for production security deployments.

2 SCRIPTSECEVAL BENCHMARK

2.1 DATASET

We introduce ScriptSecEval, a benchmark dataset of $1{,}085$ scripts with manually curated natural language behavioral summaries spanning five programming languages relevant in script analysis for threat detection Feng et al. (2025); Li et al. (2024); Srinivasan et al. (2023); Sohan & Basalamah (2020): Bash (*sh*), Batch (*bat*), Python (*py*), JavaScript (*js*), and PowerShell (*ps*), distributed as shown in Table 1. To the best of our knowledge, this represents the largest manually annotated script analysis dataset in the cybersecurity domain and the first to include Batch and PowerShell.

Data gathering. The diverse corpus of raw scripts used for the benchmark was collected from multiple sources: malicious samples from malware repositories (e.g., Hybrid Analysis, ReversingLabs) with metadata including malware classifications via AVClass Sebastián & Caballero (2020) or generated via LLM analysis (used solely for statistical analysis and not incorporated into the evaluation pipeline), and benign samples from three instruction-following datasets Luo et al. (2024); Lin et al. (2018) and Python Code Instruct 18K Alpaca[1]. To address language coverage gaps in the benign sources, which only included Bash, Python, and JavaScript, we utilized LLM-based translation to generate equivalent Batch and PowerShell scripts.

Preprocessing. To obtain a manageable yet diverse corpus base starting from our raw corpus, we implemented Affinity Propagation Frey & Dueck (2007) clustering with Levenshtein distance Navarro (2001), enabling the selection of the most representative scripts across the feature space. We applied additional filtering to remove oversized files that would impede annotation efficiency, setting a maximum file size limit of 5KB to ensure manageable annotation workload, resulting in our final balanced dataset comprising 585 benign and 500 malicious scripts.

Manual annotation. Experts manually annotated all scripts with behavioral summaries describing functionality and security implications. For malicious samples, experts authored summaries from scratch, while for benign samples we utilized instruction-following descriptions as initial drafts that experts subsequently reviewed and refined. Our malicious corpus encompasses diverse threat categories, with downloaders, backdoors, and trojans representing the most prevalent classes.

[1] `https://huggingface.co/datasets/iamtarun/python_code_instructions_18k_alpaca`

Table 1: Data distribution across scripting language and malware label

Label	sh	bat	js	ps	py	all
Benign	122	117	120	107	119	**585**
Malicious	100	96	100	104	100	**500**
all	**222**	**213**	**220**	**211**	**219**	**1,085**

2.2 METHOD

Our experimentation setup, illustrated in Figure 1, employs a two-stage pipeline. First, candidate models generate behavioral summaries for each script in our dataset. We utilize five state-of-the-art models: Claude 3.7 Sonnet Anthropic (2025a), Mistral Large MistralAI (2024) via AWS Bedrock, Llama 3.2-90B Meta (2024), GPT-4o Hurst et al. (2024), and DeepSeek R1 Guo et al. (2025), each with two distinct prompting strategies. In the second stage, these candidate-generated summaries are evaluated by LLM judges against ground truth expert annotations. For the judge role, we employed the same five models plus Claude 4 Sonnet Anthropic (2025b), implementing two different judge prompting approaches.

We conducted manual reviews of judge evaluations, in which human annotators reviewed pairs of judge responses, and then selected those that aligned the most with human judgment. In this manner, we are able to assess judge models' evaluation logic when comparing candidate LLM-generated responses against human baseline. Our pipeline examined all viable candidate-judge combinations, yielding 25 unique pairings per script and facilitating detailed analysis of inter-model dynamics. We deliberately excluded self-evaluation scenarios (where models judge their own outputs) to mitigate documented positive bias effects Ye et al. (2025). To ensure objectivity, we implemented rigorous blind evaluation protocols, anonymizing model identities and randomizing response presentation order, thereby eliminating potential reviewer bias during quality assessment.

3 JUDGE PERFORMANCE ANALYSIS

3.1 JUDGE PROMPT SELECTION

We evaluated two judge prompting strategies through manual assessment of 250 instances: (1) *Detailed Checklist Prompt (J1)*, which requires dimensional scoring with explanations for accuracy, completeness and conciseness and (2) *Universal Evaluation Prompt (J2)*, similar to J1, with the difference that it requests only the overall explanation and final score, without explicit checklist structure.

Manual evaluation revealed a clear preference for J2 (see Table 4 from Appendix A), which outperformed J1 in 156 instances (62.4%) compared to 59 instances (23.6%), with 35 ties (14%). This pattern was consistent across all models, with GPT-4o showing the strongest preference for J2 (77.5%). Human evaluators noted that J2 produced more coherent assessments with specific feedback about discrepancies between candidate responses and reference summaries, whereas J1's responses were often criticized for being "too general" or "lacking specificity". Consequently, we adopted J2 for all subsequent evaluations.

3.2 JUDGE MODEL RANKING

To establish robust judge rankings, we employed four evaluation methodologies for the LLM judges: (1) *Custom Scoring* through a position-based point system (5 for first, 4 for second, etc.) summed across all human evaluations; (2) *Plackett-Luce Model* Xia et al. (2019), which provides statistical estimation of selection likelihood based on ranked preferences; and (3) *ELO Rating* Elo (1978), a pairwise comparison system (K-factor=32, base=1500) over 1000 iterations. Further details can be found in Appendix A (A.3.2).

Table 2: Judge model ranking across four evaluations methods based on human annotations

Model	Custom Score	Plackett-Luce	ELO Rating
Claude-4-Sonnet	1,002	0.4318	1,560.24
Claude-3.7-Sonnet	773	0.1503	1,555.83
GPT-4o	663	0.1136	1,493.90
Mistral Large	611	0.1058	1,468.63
Llama-3.2-90B	595	0.1004	1,470.45
DeepSeek R1	506	0.0980	1,450.92

As shown in Table 2, all four evaluation methodologies produce consistent rankings, with only minor variation between positions 4 and 5 (Mistral Large and Llama-3.2-90B) in the ELO system. These results reveal a performance hierarchy among judge models across three distinct groups: top, mid and low tiers.

Top tier. In the top tier, Claude-4-Sonnet and Claude-3.7-Sonnet consistently outperformed all competitors, with Claude-4-Sonnet maintaining the lead across all metrics. The Claude models excelled in identifying technical details and security implications relevant for the cybersecurity domain, while maintaining alignment with human preferences. These models demonstrated superior ability to identify technical omissions, provide detailed comparative analyses between candidate and reference summaries, and maintain appropriate critical assessment without being overly lenient.

Mid tier. The mid tier per our results, comprises GPT-4o, Llama-3.2-90B and Mistral Large, with relatively close scores. GPT-4o was the strongest judge of this group, consistently ranking third across all evaluation methodologies, though with a notable gap to the Claude models. Despite competent performance, GPT-4o occasionally produced factual errors and inappropriate suggestions omitting relevant security information. These middle-performing models showed adequate technical understanding although sometimes they missed subtle security implications or provided less comprehensive evaluations of script functionality.

Low tier. Finally, the lower tier consists solely of DeepSeek R1, which ranked consistently last across all evaluations, showing a significant performance gap compared to other models. DeepSeek R1 frequently provided superficial analyses that missed core technical details and malicious behaviors, demonstrating the highest rate of factual errors and misleading insights, sometimes failing to identify core malicious behaviors or returning misleading insights about candidate summaries.

A recurring challenge across all models was the tendency to either over-penalize or under-penalize candidate responses regarding maliciousness classifications, with some of the judges incorrectly labeling security-relevant observations as "irrelevant" or "unnecessary" even when contextually appropriate. These findings provide strong empirical evidence for model selection in cybersecurity evaluation tasks, with Claude models demonstrating clear advantages as judges in this specialized domain.

4 Conclusions

This paper presents the first comprehensive study of LLM-as-a-judge performance in cybersecurity script analysis, introducing ScriptSecEval, a publicly available benchmark of $1{,}085$ manually annotated scripts across five programming languages with expert-curated behavioral summaries. This benchmark addresses an important gap in cybersecurity research by providing the first publicly available resource for malicious script analysis with expert natural language annotations across multiple programming languages.

We show that model selection significantly impacts evaluation quality in cybersecurity contexts. Claude-4-Sonnet emerged as the most reliable judge among our tested models, substantially outperforming alternatives and showing superior alignment with human expert judgment. While our findings are specific to tested models and task, they should be considered within the limitations of our study, detailed in Appendix A. Further research across broader model selections and diverse security contexts would enhance understanding LLM capabilities in the cybersecurity domain.

REFERENCES

Toufique Ahmed and Premkumar Devanbu. Few-shot training llms for project-specific code-summarization. In *Proceedings of the 37th IEEE/ACM international conference on automated software engineering*, pp. 1–5, 2022.

Anthropic. Claude 3.7 sonnet system card. https://api.semanticscholar.org/CorpusID:276612236, 2025a. Accessed: 2025-10-06.

Anthropic. Claude 4 sonnet. https://www.anthropic.com/news/claude-4, 2025b. Accessed: 2025-10-06.

Sher Badshah and Hassan Sajjad. Reference-guided verdict: Llms-as-judges in automatic evaluation of free-form text. *arXiv preprint arXiv:2408.09235*, 2024.

Guiming Hardy Chen, Shunian Chen, Ziche Liu, Feng Jiang, and Benyou Wang. Humans or llms as the judge? a study on judgement bias. In *Conference on Empirical Methods in Natural Language Processing*, 2024. URL https://api.semanticscholar.org/CorpusID:267740522.

Cheng-Han Chiang and Hung yi Lee. Can large language models be an alternative to human evaluations? In *Annual Meeting of the Association for Computational Linguistics*, 2023. URL https://api.semanticscholar.org/CorpusID:258461287.

Gelei Deng, Yi Liu, Víctor Mayoral-Vilches, Peng Liu, Yuekang Li, Yuan Xu, Tianwei Zhang, Yang Liu, Martin Pinzger, and Stefan Rass. {PentestGPT}: Evaluating and harnessing large language models for automated penetration testing. In *33rd USENIX Security Symposium (USENIX Security 24)*, pp. 847–864, 2024.

Arpad E. Elo. The rating of chessplayers, past and present, 1978. URL https://api.semanticscholar.org/CorpusID:142610973.

Ruirui Feng, Hui Chen, Shuo Wang, Md Monjurul Karim, and Qingshan Jiang. Llm-maldetect: A large language model-based method for android malware detection. *IEEE Access*, 2025.

Brendan Frey and Delbert Dueck. Clustering by passing messages between data points. *Science (New York, N.Y.)*, 315:972–6, 03 2007. doi: 10.1126/science.1136800.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Jingxuan He and Martin Vechev. Large language models for code: Security hardening and adversarial testing. In *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, pp. 1865–1879, 2023.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019.

Egil Karlsen, Xiao Luo, Nur Zincir-Heywood, and Malcolm Heywood. Benchmarking large language models for log analysis, security, and interpretation. *Journal of Network and Systems Management*, 32(3):59, 2024.

Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, et al. From generation to judgment: Opportunities and challenges of llm-as-a-judge, 2025. *URL https://arxiv. org/abs/2411.16594*, 2025.

Ruijie Li, Chenyang Zhang, Huajun Chai, Lingyun Ying, Haixin Duan, and Jun Tao. Powerpeeler: A precise and general dynamic deobfuscation method for powershell scripts. *Proceedings of ACM SIGSAC*, 2024. URL https://api.semanticscholar.org/CorpusID:270286055.

Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D. Ernst. NL2Bash: A corpus and semantic parser for natural language interface to the linux operating system. In Nicoletta Calzolari, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Koiti Hasida, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asuncion Moreno, Jan Odijk, Stelios Piperidis, and Takenobu Tokunaga (eds.), *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA). URL https://aclanthology.org/L18-1491/.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, MING GONG, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie LIU. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. URL https://openreview.net/forum?id=6lE4dQXaUcb.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=UnUwSIgK5W.

Meta. Llama 3.2: revolutionizing edge ai and vision with open, customizable models. https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_2/, 2024. Accessed: 2025-10-06.

MistralAI. Mistral Large bedrock. https://huggingface.co/mistralai/Mistral-Large-Instruct-2407, 2024. Accessed: 2025-10-06.

Shaswata Mitra, Subash Neupane, Trisha Chakraborty, Sudip Mittal, Aritran Piplai, Manas Gaur, and Shahram Rahimi. Localintel: Generating organizational threat intelligence from global and local cyber knowledge. In *International Symposium on Foundations and Practice of Security*, pp. 63–78. Springer, 2024.

Farzad Nourmohammadzadeh Motlagh, Mehrdad Hajizadeh, Mehryar Majd, Pejman Najafi, Feng Cheng, and Christoph Meinel. Large language models in cybersecurity: State-of-the-art. In *International Conference on Information Systems Security and Privacy*, 2024. URL https://api.semanticscholar.org/CorpusID:267406465.

Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, March 2001. ISSN 0360-0300. doi: 10.1145/375360.375365. URL https://doi.org/10.1145/375360.375365.

Suleman Qamar. Smart omvi: Obfuscated malware variant identification using a novel dataset. *arXiv preprint arXiv:2310.10670*, 2023.

Silvia Sebastián and Juan Caballero. Avclass2: Massive malware tag extraction from av labels. In *Proceedings of the 36th Annual Computer Security Applications Conference*, ACSAC '20, pp. 42–53, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450388580. doi: 10.1145/3427228.3427261. URL https://doi.org/10.1145/3427228.3427261.

Lin Shi, Chiyu Ma, Wenhua Liang, Xingjian Diao, Weicheng Ma, and Soroush Vosoughi. Judging the judges: A systematic study of position bias in llm-as-a-judge. *arXiv preprint arXiv:2406.07791*, 2024.

Md. Fahimuzzman Sohan and Anas Basalamah. A systematic literature review and quality analysis of javascript malware detection. *IEEE Access*, 8:190539–190552, 2020. URL https://api.semanticscholar.org/CorpusID:226230804.

Dhamodharan Srinivasan, A Muthuvel, M Prakash Kumar, R Dinesh, and V G Prasannakumar. Advanced malware analysis and prevention. *STCR*, 1:1–6, 2023. URL https://api.semanticscholar.org/CorpusID:267193164.

Weisong Sun, Yun Miao, Yuekang Li, Hongyu Zhang, Chunrong Fang, Yi Liu, Gelei Deng, Yang Liu, and Zhenyu Chen. Source Code Summarization in the Era of Large Language Models . In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 1882–1894, Los Alamitos, CA, USA, May 2025. IEEE Computer Society. doi: 10.1109/ICSE55347.2025.00034. URL https://doi.ieeecomputersociety.org/10.1109/ICSE55347.2025.00034.

Aman Singh Thakur, Kartik Choudhary, Venkat Srinik Ramayapally, Sankaran Vaidyanathan, and Dieuwke Hupkes. Judging the judges: Evaluating alignment and vulnerabilities in llms-as-judges. *arXiv preprint arXiv:2406.12624*, 2024.

Yidong Wang, Zhuohao Yu, Zhengran Zeng, Linyi Yang, Cunxiang Wang, Hao Chen, Chaoya Jiang, Rui Xie, Jindong Wang, Xing Xie, et al. Pandalm: An automatic evaluation benchmark for llm instruction tuning optimization. *arXiv preprint arXiv:2306.05087*, 2023.

Chenxi Whitehouse, Tianlu Wang, Ping Yu, Xian Li, Jason Weston, Ilia Kulikov, and Swarnadeep Saha. J1: Incentivizing thinking in llm-as-a-judge via reinforcement learning, 2025. URL https://arxiv.org/abs/2505.10320.

Tian Xia, Shaodan Zhai, and Shaojun Wang. Plackett-luce model for learning-to-rank task, 2019. URL https://arxiv.org/abs/1909.06722.

Jiayi Ye, Yanbo Wang, Yue Huang, Dongping Chen, Qihui Zhang, Nuno Moniz, Tian Gao, Werner Geyer, Chao Huang, Pin-Yu Chen, Nitesh V Chawla, and Xiangliang Zhang. Justice or prejudice? quantifying biases in LLM-as-a-judge. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=3GTtZFiajM.

Jie Zhang, Haoyu Bu, Hui Wen, Yongji Liu, Haiqiang Fei, Rongrong Xi, Lun Li, Yun Yang, Hongsong Zhu, and Dan Meng. When llms meet cybersecurity: A systematic literature review. *Cybersecurity*, 8(1):55, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Lianghui Zhu, Xinggang Wang, and Xinlong Wang. JudgeLM: Fine-tuned large language models are scalable judges. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=xsELpEPn4A.

A Appendix

A.1 Data Composition Details

To ensure a diverse set of scripts, both malicious and benign, covering multiple languages, we employed a complex data composition pipeline (see Figure 2) that leveraged both human annotations and synthetic data generation.

A.1.1 Script Summary Generation

Each script in our benchmark is accompanied by a behavioral summary that provides analysis of the script's functionality, and critically important security implications. To ensure the highest quality and accuracy for our annotation framework, malicious scripts receive exclusively manually-authored summaries created by experts. Benign scripts utilize LLM-generated summaries that follow the same set of annotations rules as the human experts did. Note that this approach is imperfect and there might still be some PIIs left that we potentially missed in our manual curation.

Annotation Rules The annotation framework for generating summaries over scripts follows a structured approach designed to capture critical security-relevant behaviors while maintaining appropriate abstraction levels. Annotators identify a maximum of 4–5 malicious or suspicious actions within each script, prioritizing malicious behaviors over suspicious ones when selection is required. Each identified action is documented with comprehensive contextual details including the characteristics that classify the behavior as threatening, the technical methodology employed to execute the action, and the specific targets or systems affected. The descriptions maintain a medium-level technical depth, avoiding both overly granular implementation details and overly generic characterizations that lack analytical value. To ensure privacy protection and operational security, all personally identifiable information (PII) is excluded from annotations. For example, specific network indicators such as IP addresses, URLs, email addresses, and credentials are replaced with generalized descriptors such as "script establishes connection to remote command-and-control server".

A.1.2 Malware Class Assignment

The malware class assignment task leverages Claude 3.7 Sonnet to categorize scripts into one of the 13 malware classes listed in Table 3. The prompt (see Fig. 3) was designed to contain malware class definitions extracted from CrowdStrike's "The 12 Most Common Types of Malware"[2] and Fortinet's "Types of Malware: How to Identify and Defend Malware"[3].

A.1.3 Language Conversion

As previously mentioned in section 2.1, the utilized data sources provided coverage for only three of our target languages—Bash, Python, and JavaScript—creating a significant gap in our desired comprehensive language representation. To address this limitation and ensure balanced coverage across all five scripting languages, we implemented an LLM-based code generation approach, systematically translating existing benign scripts from the well-represented languages into the missing languages (Batch and PowerShell).

We employed Claude 4 Sonnet to convert scripts between different programming languages using a task-specific prompt that instructed the model to translate functionality while preserving the original script's behavior and intent (see Figure 4). Batch scripts were generated from unused Bash scripts that were sampled from the clustering results (see section 2.1), ensuring representative coverage across different script types and complexities. Similarly, PowerShell scripts were generated from unused PowerShell and Python scripts selected from the same clustering methodology to maintain consistency in our sampling approach. Throughout this conversion process across generations, we identified only 17 failed conversions where the LLM was unable to produce functionally equivalent code, which were removed from the dataset prior to publication to ensure data quality and reliability.

[2] https://www.crowdstrike.com/en-us/cybersecurity-101/malware/types-of-malware/

[3] https://www.fortinet.com/resources/cyberglossary/types-of-malware

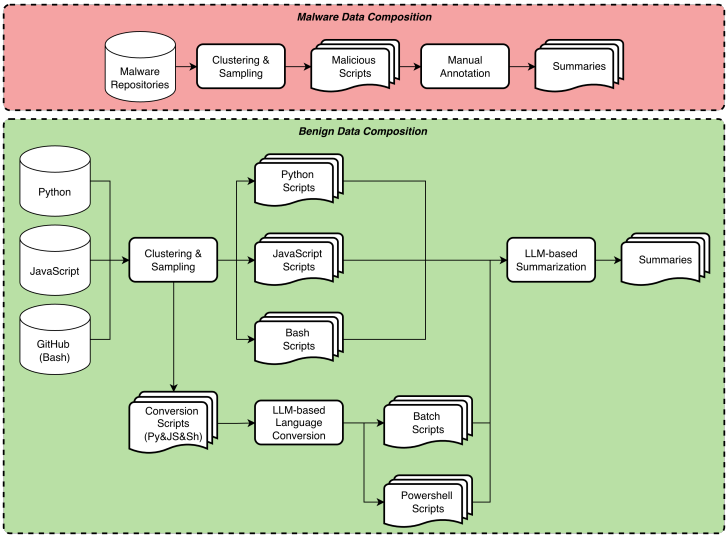

Figure 2: Data composition pipeline for both malicious and benign scripts. Malicious scripts were manually annotated by humans, while the benign ones were synthetically generated with Claude 4 Sonnet. The scripting language gap was addressed by converting scripts from one language to another with the help of Claude 4 Sonnet.

Table 3: Distribution of malware classes

Malware Class	**Count**
Downloader	188
Backdoor	87
Trojan	63
Spyware	27
Grayware	26
Wiper	21
Miner	19
Virus	12
Expkit	12
Adware	12
Grayware:tool	11
Worm	9
Hoax	5
Clicker	3
Ransomware	2
Keylogger	1
Cryptojacking	1
Bot	1
Total	**500**

A.1.4 DATASET ETHICAL CONSIDERATIONS

Due to ethical concerns we cannot openly distribute malicious scripts, however we provided SHA256 hashes of the original scripts that allow researchers to independently source them from established malware repositories (e.g., Hybrid Analysis) while preventing direct distribution of harmful code. In contrast, we provided the full script content for clean scripts as they pose no security risk. Moreover, all behavioral summaries, security analyses and metadata were included for both malicious and clean scripts to strike the appropriate balance between research openness and cybersecurity responsibility.

A.1.5 HUMAN ANNOTATOR DETAILS

Our benchmark construction and LLM judge evaluations involved 11 human annotators with cybersecurity experience ranging from 1 to 5 years. To ensure consistency and quality across annotations, all annotators followed structured guidelines that specified:

- Annotation rules for behavioral summaries (Section 2)
- Evaluation criteria for judge quality assessment
- Standardized comparison operators for ranking ($>$, $\geq$, $=$)
- Requirements for documenting reasoning in evaluation notes

These guidelines were combined with blind evaluation protocols where model identities were anonymized and response orderings randomized to eliminate potential annotator bias during quality assessments.

A.2 EXPERIMENTATION DETAILS

A.2.1 JUDGE PROMPTS

In Figure 5 we present the complete judge prompts used in our evaluation pipeline. Both prompts instruct models to evaluate candidate-generated code summaries based on three key criteria: accuracy (correctness of functional description), completeness (coverage of key functionality), and conciseness (appropriate brevity without omitting essentials). The primary difference between the two approaches lies in the output structure: Judge Prompt 1 (J1) requires explicit dimensional scoring with detailed explanations for each criterion, while Judge Prompt 2 (J2) requests only an overall explanation and final score, eliminating the structured checklist format that proved less effective in our manual evaluation.

A.2.2 CANDIDATE PROMPTS

In Figure 6 we present the two candidate prompting strategies employed in our evaluation pipeline. Candidate Prompt 1 (S1) enforces a structured JSON output format requiring specific fields including summary, maliciousness label, programming language identification, and malware classification. Candidate Prompt 2 (S2) adopts a free-form approach, allowing models to provide natural language analysis without format constraints while maintaining the same core analytical requirements. We employed two distinct prompting approaches to ensure diversity in our evaluation and avoid bias toward any specific response style.

A.3 EVALUATION METHODOLOGY DETAILS

A.3.1 JUDGE PROMPT SELECTION

As presented in Table 4, the manual evaluation of 250 instances revealed a consistent preference for the holistic evaluation prompt (J2) across all judge models, with J2 outperforming the structured checklist approach (J1) by a substantial margin of 156 to 59 wins. This preference pattern was remarkably consistent across different model architectures, with GPT-4o showing the strongest inclination toward J2 (31 wins vs. 5 for J1), while even the most balanced model, Claude-4-Sonnet, humans still favored J2 over J1.

Table 4: Preference patterns for structured checklist (J1) versus holistic evaluation (J2) prompts across judge models

Judge Model	J1 Wins	J2 Wins	Ties
Claude-4-Sonnet	18	24	8
Claude-3.7-Sonnet	9	26	5
Mistral-Large	13	24	3
DeepSeek R1	5	25	10
Llama3-2-90B	9	26	5
GPT-4o	5	31	4
Overall	**59**	**156**	**35**

A.3.2 JUDGE MODEL RANKING

Our evaluation framework includes four distinct ranking systems, providing comprehensive methodological details for the ranking approaches summarized in the main paper:

Custom Scoring System The first approach is a position-based scoring system that assigns points based on the relative ranking position of each model in the human evaluations. For each comparison (e.g. A=B>C=D>E), models receive points inversely proportional to their effective position. Specifically, in our comparison of 5 models, the highest-ranked model(s) receive 5 points, the second-ranked receive 4 points, and so on. When models are tied (indicated by the strict equal "="), they receive equal points corresponding to the leftmost position in their group. For example, in the ranking "A=B>C=D>E" with five models, A and B each receive 5 points, C and D each receive 3 points, and E receives 1 point. The final score for each model is the sum of points across all comparisons, with higher scores indicating better performance.

Plackett-Luce Model To capture the probabilistic nature of preferences, we also employed the Plackett-Luce model, a generalization of the *Bradley-Terry model* for list-wise comparisons. This statistical approach estimates the relative *strength* of each model based on its position in ranked lists, calculating the likelihood of a model being selected by a human annotator. This methodology also treats preference ties as a **strict preference** (">=" is treated as "=").

ELO Rating System We converted our multi-model comparisons into pairwise "duels" to be able to compute an adapted ELO rating system. Inspired from the competitive games field, this technique is widely-used to calculate relative performance ratings for each individual in tournaments or matches. Using the standard parameters *K-factor*=32 and *base ELO*=1500 running over 1000 iterations with randomized duel matchups to minimize bias, this approach provides a powerful metric of relative model strength.

The evaluation results reveal significant performance gaps between models. In the Plackett-Luce evaluation, Claude-4-Sonnet's selection likelihood (0.4318) is nearly three times higher than Claude-3.7-Sonnet's (0.1503), indicating substantial qualitative differences even among top performers. The position-based scoring confirms this gap, with Claude-4-Sonnet $(1,002$ points) outperforming Claude-3.7-Sonnet (773 points) by 29.6%. Mid-tier models show closer competition: GPT-4o (663 points) leads Mistral Large (611 points) by only 8.51%, while the ELO ratings reveal Mistral Large (1468.63) and Llama-3.2-90B (1470.45) perform nearly equivalently. These findings provide strong empirical evidence for model capabilities in malicious script evaluation, with Claude models demonstrating clear advantages in this specialized domain.

A.3.3 CANDIDATE MODEL RANKING

Having established Claude-4-Sonnet as the judge best aligned with human expert judgment, we also evaluated candidate model performance for cybersecurity script summarization. Claude 3.7 Sonnet emerged as the top performer with an average score of 4.07 out of 5, followed by GPT-4o (3.80), Mistral-Large (3.45), Llama 3.2 90B (3.34), and DeepSeek (3.20). These results reveal significant performance variations in security-focused analysis capabilities.

Claude 3.7 Sonnet consistently delivered the most accurate and comprehensive behavioral summaries, excelling at detailed technical specifications, including API functions, obfuscation techniques, and execution flows while maintaining appropriate conciseness. However, minor limitations were observed in its tendency to include unnecessary security assessments for benign code and occasional gaps in contextual insights regarding broader security implications.

Lower-ranked models exhibited more pronounced weaknesses affecting their reliability in security-critical applications. While GPT-4o provided accurate functional descriptions, it demonstrated concerning inconsistencies in threat detection, including missed malicious behaviors and mischaracterized threat severity levels. Most notably, DeepSeek had the weakest performance with frequent critical accuracy errors, incomplete technical analysis, and fundamental misunderstandings of command syntax. These performance deficiencies could have significant implications in operational security environments where accurate script analysis is crucial for threat detection and mitigation. These findings underscore the importance of careful model selection for security analysis tasks, as performance gaps can significantly impact the quality and reliability of code assessment outcomes.

A.4 Infrastructure and Computational Resources

For the majority of model inferences, we utilized API-based services, which eliminated the need for dedicated local hardware. The exception was DeepSeek R1, for which we established a dedicated computational environment consisting of 8 NVIDIA H100 GPUs.

A.5 Limitations

Our study presents several limitations that should be considered when interpreting the results and applying our findings. The limited sample size per programming language (200–220 scripts) may not capture the full diversity of scripting behaviors in real-world environments. Our findings are specific to script analysis tasks and might not generalize beyond this domain to other applications.

Additionally, our judge model evaluation relied on human annotations to establish rankings. We do not have an automatic way to evaluate other judge models, and our results are limited to the six models included in this study.

Finally, the study presents a comparison between several LLM-as-a-Judge systems in terms of evaluation capabilities with given human reference. However, the current method does not measure the extent to which judge LLMs align with human expertise. We intend to assess the reliability of such systems in future work.

Malware Class Assignment Prompt

You are a cybersecurity expert with over 10 years of experience with cyberattacks. Your role is to classify malicious scripts based on their content and their description in one malware type. In order to do so, you must:
1. Analyze the received script and its description and understand what it is doing
2. Select based on the script and its description which type of malware from below fits best

Types of malware and their description:
1. **Ransomware**: software that uses encryption to disable a target's access to its data until a ransom is paid. The victim organization is rendered partially or totally unable to operate until it pays, but there is no guarantee that payment will result in the necessary decryption key or that the decryption key provided will function properly.
2. **Spyware**: Spyware collects information about users' activities without their knowledge or consent. This can include passwords, pins, payment information and unstructured messages. The use of spyware is not limited to the desktop browser: it can also operate in a critical app or on a mobile phone. Even if the data stolen is not critical, the effects of spyware often ripple throughout the organization as performance is degraded and productivity eroded.
3. **Adware**: Adware tracks a user's surfing activity to determine which ads to serve them. Although adware is similar to spyware, it does not install any software on a user's computer, nor does it capture keystrokes. The danger in adware is the erosion of a user's privacy — the data captured by adware is collated with data captured, overtly or covertly, about the user's activity elsewhere on the internet and used to create a profile of that person which includes who their friends are, what they've purchased, where they've traveled, and more. That information can be shared or sold to advertisers without the user's consent.
4. **Trojan**: A Trojan disguises itself as desirable code or software. Once downloaded by unsuspecting users, the Trojan can take control of victims' systems for malicious purposes. Trojans may hide in games, apps, or even software patches, or they may be embedded in attachments included in phishing emails.
5. **Worms**: Worms target vulnerabilities in operating systems to install themselves into networks. They may gain access in several ways: through backdoors built into software, through unintentional software vulnerabilities, or through flash drives. Once in place, worms can be used by malicious actors to launch DDoS attacks, steal sensitive data, or conduct ransomware attacks.
6. **Virus**: A virus is a piece of code that inserts itself into an application and executes when the app is run. Once inside a network, a virus may be used to steal sensitive data, launch DDoS attacks or conduct ransomware attacks. A virus cannot execute or reproduce unless the app it has infected is running. This dependence on a host application makes viruses different from trojans, which require users to download them, and worms, which do not use applications to execute. Many instances of malware fit into multiple categories: for instance, Stuxnet is a worm, a virus and a rootkit.
7. **Rootkits**: A rootkit is software that gives malicious actors remote control of a victim's computer with full administrative privileges. Rootkits can be injected into applications, kernels, hypervisors, or firmware. They spread through phishing, malicious attachments, malicious downloads, and compromised shared drives. Rootkits can also be used to conceal other malware, such as keyloggers.
8. **Keylogger**: A keylogger is a type of spyware that monitors user activity. Keyloggers have legitimate uses; businesses can use them to monitor employee activity and families may use them to keep track of children's online behaviors. However, when installed for malicious purposes, keyloggers can be used to steal password data, banking information and other sensitive information. Keyloggers can be inserted into a system through phishing, social engineering or malicious downloads.
9. **Bots/Botnets**: A bot is a software application that performs automated tasks on command. They're used for legitimate purposes, such as indexing search engines, but when used for malicious purposes, they take the form of self-propagating malware that can connect back to a central server. Usually, bots are used in large numbers to create a botnet, which is a network of bots used to launch broad remotely-controlled floods of attacks, such as DDoS attacks. Botnets can become quite expansive. For example, the Mirai IoT botnet ranged from 800,000 to 2.5M computers.
10. **Wiper**: A wiper is a type of malware with a single purpose: to erase user data and ensure it can't be recovered. Wipers are used to take down computer networks in public or private companies across various sectors. Threat actors also use wipers to cover up traces left after an intrusion, weakening their victim's ability to respond.
11. **Backdoor**: A backdoor is a malware that creates a hidden entry point into a system, enabling unauthorized access without triggering security alerts. It can be used for malicious activities like data theft or deploying additional malware.
12. **Scareware**: Scarewares are designed to scare users into taking action by displaying fake warnings that mimic legitimate system alerts, urging downloads or payments for fake fixes.
13. **Cryptojacking**: Uses the victim's device to mine cryptocurrency without consent, hijacking processing power for mining tasks.

Log all these insights in the following JSON template:
{
"malware_type": string, // best fitting malware type from the list above; if the script and its description do not fit any malware types from above, fill with "NA"
"reasonings": string // short explanation of maximum 3-4 sentences explaining with the associated malware type fits the best
}

Here is the script and its description you have to find the best fitting malware type:

<SCRIPT>
{{ script }}
</SCRIPT>

<DESCRIPTION>
{{ description }}
</DESCRIPTION>

Figure 3: Full malware class assignment prompt used in generating synthetic metadata via Claude 3.7 Sonnet. The malware class definitions were taken from CrowdStrike and Fortinet blogs.

Templatized Malware Class Assignment Prompt

You are a senior software engineer with over 10 years of experience in writing code. You are tasked to convert a piece of code to another programming language. You will need to:

1. Read the received piece of code and understand what it is doing
2. Convert the piece of code to {{ conversion_language }} language
3. Make sure that the received piece of code and the converted piece of code do the same thing

Add the converted code to the following JSON structure:
{
"code": string // Converted code from {{ current_language }} to {{ conversion_language}}
}

Additionally:

4. Return only the completed JSON structure
5. Do not add unnecessary code while converting
6. Use the same variable and function names

Here is the code you need to convert from {{ current_language }} to {{ conversion_language }}:
{{ code }}

Figure 4: Full templatized language prompt used in generating synthetic metadata via Claude 4 Sonnet.

Judge Prompt 1: Detailed Checklist Prompt

You are an expert code reviewer evaluating the quality of code summaries.
CODE: {{ code }}
REFERENCE SUMMARY: {{ reference_explanation }}
CANDIDATE SUMMARY: {{ candidate_summary }}

Evaluate the candidate summary against the reference summary and the code itself. Consider:

1. **Accuracy**: Does it correctly describe what the code does?
2. **Completeness**: Does it capture all key functionality?
3. **Conciseness**: Is it appropriately brief without omitting essentials?

Return your evaluation in valid JSON format following this exact structure:

```
{
"checklist": {
"accuracy": {
"score": 0,
"explanation": ""
},
"completeness": {
"score": 0,
"explanation": ""
},
"conciseness": {
"score": 0,
"explanation": ""
}
},
"overall_explanation": "",
"final_score": 0
}
```

For each metric, provide:

1. A score from 1-5 (integer only). Higher scores mean better performance for the specific metric.
2. A brief explanation for the score

The overall_explanation should summarize your evaluation, highlighting key strengths and weaknesses.
The final_score should be an integer from 1-5 representing the overall quality.
Important: Ensure your response is valid JSON that can be parsed programmatically. Do not include any text outside the JSON structure.

Judge Prompt 2: Universal Evaluation Prompt

You are an expert code reviewer evaluating the quality of code summaries.
CODE: {{ code }}
REFERENCE SUMMARY: {{ reference_explanation }}
CANDIDATE SUMMARY: {{ candidate_summary }}

Evaluate the candidate summary against the reference summary and the code itself. Consider:

1. **Accuracy**: Does it correctly describe what the code does?
2. **Completeness**: Does it capture all key functionality?
3. **Conciseness**: Is it appropriately brief without omitting essentials?

Return your evaluation in valid JSON format following this exact structure:

```
{
"overall_explanation": "",
"final_score": 0
}
```

For each metric, provide:

1. A score from 1-5 (integer only). Higher scores mean better performance for the specific metric.
2. A brief explanation for the score

The overall_explanation should summarize your evaluation, highlighting key strengths and weaknesses.
The final_score should be an integer from 1-5 representing the overall quality.
Important: Ensure your response is valid JSON that can be parsed programmatically. Do not include any text outside the JSON structure.

Figure 5: Full Judge prompts designed to instruct the model to evaluate candidate responses based on accuracy, completeness and conciseness. Judge Prompt 2 is a variation of Judge Prompt 1, where the explicit checklist was removed from the generation output.

Candidate Prompt 1: Detailed JSON Format Prompt

Analyze the provided script and respond with a brief explanation of what the code does and whether it is malicious or not.
Provide your analysis in this JSON format:
{
"summary": "Brief description of what the script does and whether it's malicious or not.",
"label": "malicious or clean",
"language": "programming language used",
"malware_class": "specific malware category if malicious (e.g., worm, ransomware, grayware) or clean-class if not malicious"
}
Keep your analysis concise and evidence-based.
Important: Ensure your response is valid JSON that can be parsed programmatically. Do not include any text outside the JSON structure.
Here is the script:
{{ code }}

Candidate Prompt 2: Universal Free-Form Prompt

Analyze the provided script and respond with a brief explanation of what the code does and whether it is malicious or not.
Keep your analysis concise and evidence-based.
Here is the script:
{{ code }}

Figure 6: Candidate prompts used to instruct models to analyze code samples. Candidate Prompt 1 requires a structured JSON response with specific fields, while Candidate Prompt 2 allows for free-form analysis without enforcing a specific output format.