AutoDFBench: A Framework for AI Generated Digital Forensic Code and Tool Testing and Evaluation

Akila Wickramasekara
School of Computer Science
University College Dublin
Dublin, Ireland
akila.wickramasekara@ucdconnect.ie

Alanna Densmore
Florida State University
Tallahassee, FL, USA
amd22c@fsu.edu

Frank Breitinger
Institute of Computer Science
University of Augsburg
Augsburg, Germany
frank.breitinger@uni-a.de

Hudan Studiawan
Department of Informatics
Institut Teknologi Sepuluh Nopember
Surabaya, Indonesia
hudan@if.its.ac.id

Mark Scanlon
School of Computer Science
University College Dublin
Dublin, Ireland
mark.scanlon@ucd.ie

Abstract

Generative AI (GenAI) and Large Language Models (LLMs) show great potential in various domains, including digital forensics. A notable use case of these technologies is automatic code generation, which can reasonably be expected to include digital forensic applications in the not-too-distant future. As with any digital forensic tool, these systems must undergo extensive testing and validation. However, manually evaluating outputs, including generated DF code, remains a challenge. AutoDFBench is an automated framework designed to address this by validating AI-generated code and tools against NIST's Computer Forensics Tool Testing Program (CFTT) procedures and subsequently calculating an AutoDFBench benchmarking score. The framework operates in four phases: data preparation, API handling, code execution, and result recording with score calculation. It benchmarks generative AI systems, such as LLMs and automated code generation agents, for DF applications. This benchmark can support iterative development or serve as a comparison metric between GenAI DF systems. As a proof of concept, NIST's forensic string search tests were used, involving more than 24,200 tests with five top-performing code generation LLMs. These tests validated the output of 121 cases, considering two levels of user expertise, two programming languages, and ten iterations per case with varying prompts. The results also highlight the significant limitations of the DF-specific solutions generated by generic LLMs.

CCS Concepts

• **Applied computing** → **Computer forensics**; *Evidence collection, storage and analysis*; • **Software and its engineering** → Software verification and validation.

DFDS 2025, Brno, Czech Republic
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1076-6/25/04
https://doi.org/10.1145/3712716.3712718

Keywords

Digital Forensics, Large Language Models, Investigative Process, Automation, Challenges

ACM Reference Format:
Akila Wickramasekara, Alanna Densmore, Frank Breitinger, Hudan Studiawan, and Mark Scanlon. 2025. AutoDFBench: A Framework for AI Generated Digital Forensic Code and Tool Testing and Evaluation. In *Digital Forensics Doctoral Symposium (DFDS 2025), April 01, 2025, Brno, Czech Republic.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3712716.3712718

1 Introduction

Digital forensics (DF) relies on software tools to gather and analyse digital data, which can range from small single-function scripts to advanced software suites [2, 18]. With recent advances in generative artificial intelligence (GenAI), including Large Language Models (LLMs) [12], and the growing interest in the application of AI to digital forensics [4], it is reasonable to expect GenAI to be used for digital forensic purposes in the not-too-distant future. This has the potential to streamline workflows and allow practitioners to develop bespoke solutions faster for specific DF tasks [16]. However, this evolution raises critical questions about the evaluation of these tools, i.e., scalable and robust testing.

To address these challenges, this article introduces a novel framework, AutoDFBench (Automatic Digital Forensic Code and Tool Benchmarking), along with its corresponding AutoDFBench score. The framework provides a structured approach to evaluate AI-generated DF code, comparable to unit testing in software development, where specific functions are validated against expected results. A forensic task is defined with a dataset containing a 'ground truth'. The task is executed through a pipeline, enabling comparison of the tool's output with the ground truth. This methodology is exemplified in the National Institute of Standards and Technology's (NIST) Computer Forensic Tool Testing Programme (CFTT)[1].

The framework is validated using forensic string search, a common task in digital investigations [7]. This validation uses AutoDFBench to query five LLMs to generate executable string search code,

[1]https://www.nist.gov/itl/ssd/software-quality-group/computer-forensics-tool-testing-program-cftt

execute it, and evaluate the generated code. The CFTT string search dataset serves as the ground truth.

This work makes the following contributions:

(1) Design and implementation of an open-source benchmarking framework, AutoDFBench: A robust framework for evaluating AI-generated scripts in digital forensic use cases, accessible at the following GitHub link: https://github.com/akila-UCD/AutoDFBench.
(2) Demonstration of the framework: A comparison of five state-of-the-art LLMs performing string search, resulting in the most extensive known comparison of generated DF code with 24,200 unique tests.
(3) Insights on LLM performance: The findings reveal that none of the LLMs performed satisfactorily, highlighting the need for practitioners to validate the LLM output and the importance of well-tested tools in forensic investigations.

2 Background

2.1 Computer Forensics Tool Testing Program (CFTT)

The typical phases of the DF process commonly rely on both proprietary and open-source tools for accurate data acquisition and analysis, leading to evidence discovery. Ensuring the reliability of these tools is critical, and the National Institute of Standards and Technology (NIST) addresses this through the Computer Forensics Tool Testing Program (CFTT). The program establishes a framework to test forensic tools by defining specifications, criteria, procedures, and test sets to validate their effectiveness and reliability[2]. CFTT provides protocols for various subfields, including Windows registry forensics, deleted file recovery, disk imaging, file carving, mobile devices, cloud data extraction, SQLite forensics, string search, and write blockers.

2.2 Large Language Models

An LLM is a neural network-based model with billions of parameters trained on extensive text datasets. These models understand and generate human language by recognising relationships between words and phrases [5, 15]. LLMs have revolutionised natural language processing (NLP), excelling in various tasks rather than being limited to specific functions.

Fine-tuned LLMs are adapted for specialised tasks in domains such as security, medicine, engineering, and business. This involves retraining the model on domain-specific datasets, enabling them to perform tasks such as threat detection, clinical decision-making, and business process automation [16].

2.3 HumanEval

HumanEval is a benchmark for assessing the code generation capabilities of generative AI systems, including LLMs. It consists of Python programming tasks, where each task includes a problem description and a test suite to verify the generated code [3].

The evaluation metric, pass@k, measures the percentage of correctly solved problems, with pass@1 indicating success on the first attempt. HumanEval is widely used to evaluate and compare LLMs such as GPT-4 and StarCoder for their accuracy and functionality in code generation.

[2]https://www.nist.gov/itl/ssd/software-quality-group/computer-forensics-tool-testing-program-cftt

3 Related Work

The admissibility of electronic evidence requires a rigorous and scientific approach to validate DF tools. Guo et al. [6] proposed a functionality-orientated paradigm for tool testing, focussing on search functions. Their methodology includes mapping functions, specifying requirements, and developing reference test cases to assess tools' performance against standardised criteria, thereby aiding in the design of new validation frameworks.

The use of LLMs in DF is nascent but promising, as highlighted in prior studies. ChatGPT has been explored for tasks such as programming, recovering encryption keys, file carving, and keyword searching, demonstrating the potential for investigation and education [13]. Henseler and van Beek [8] showed ChatGPT's utility in the analysis phase, acting as a copilot to examine artefacts and provide instructions. However, they noted inherent issues with hallucinations in LLMs, suggesting that Augmented Language Models (ALMs) might mitigate these shortcomings [8]. Other studies have explored the potential for automating script generation, question answering, and sentiment analysis while acknowledging risks such as hallucinations, bias, and legal concerns [14].

Wickramasekara et al. [16] introduced a usability matrix categorising DF phases as low, medium, or high potential for LLMs. They highlighted Multimodal Large Language Models (MLLMs) for investigations and analysis. Similarly, Michelet and Breitinger [10] demonstrated LLMs' ability to generate DF reports using Llama2 and ChatGPT 3.5. By inputting data from a Universal Forensic Extraction Device (UFED) Physical Analyser, the researchers demonstrated that while LLMs can automate report generation, the quality depends on the model size, and hallucinations remain a drawback, stressing the need for standardised reports.

The potential of LLMs for generating forensic intelligence graphs (FIGs) was explored by Xu et al. [19]. Using GPT-4-turbo, they reconstructed FIGs from data extracted from mobile devices. Their approach achieved 91.7% coverage for evidence entities and 93.8% for relationship coverage, showcasing the utility of LLMs in building evidence networks.

Wickramasekara and Scanlon [17] proposed a framework that uses Microsoft AutoGen for DF investigations. This framework utilises AI agents with multiple pretrained LLMs to perform DF tasks, bridging knowledge gaps among investigators and enhancing efficiency. However, the researchers highlighted language proficiency as a dependency that affects agent performance.

volGPT, a Volatility 3 plugin, employs LLMs to evaluate ransomware in memory dumps through prompt-based techniques [11]. It uses JSON-formatted process data to identify ransomware. Limitations include challenges in detecting fileless malware or obscured processes [11].

Despite these advancements, Breitinger et al. [2] emphasised the rapid growth of AI in DF in recent years and the need for new research avenues to effectively leverage AI and LLMs. Although prior work has explored LLM applications in DF, a structured mechanism to evaluate their results remains absent. This work addresses

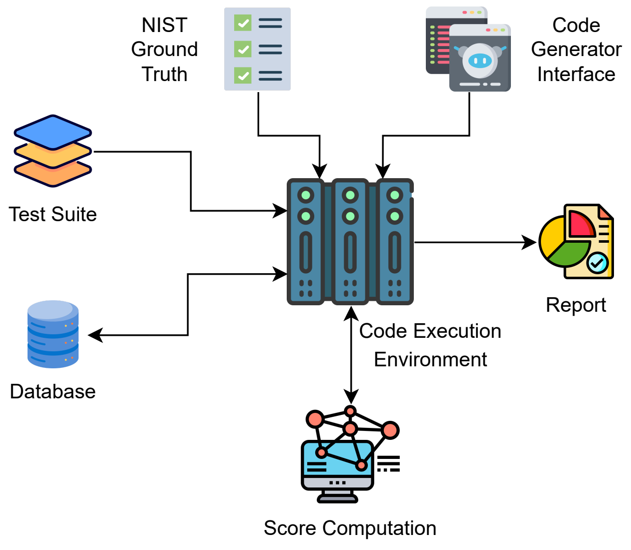

Figure 1: Overview of the proposed framework

this gap by proposing a framework to evaluate AI-assisted contributions, allowing the evaluation of LLM effectiveness in forensic scenarios.

4 Framework Design

This section outlines the design considerations and provides a detailed technical explanation of the framework.

4.1 Design Considerations

The framework is built for extensibility, enabling customisation for diverse experiments. Parameters such as the base prompt, testing disk image types, and the number of cross-validations can be tailored to specific needs. Its flexibility also includes seamless integration with various LLMs, as the framework is LLM agnostic. It supports local models such as Llama 3, StarCoder 2, and WaveCoder, as well as remote models such as GPT-4o and Claude 3.5 Sonnet, accessed via APIs integrated through the LLM Python library[3]. Key variables and configurations are stored in a MySQL database and a configuration file, allowing the smooth incorporation of new models or updates and ensuring adaptability to future technological advancements.

The framework also prioritises reproducibility and architecture agnosticism. By maintaining all variables, prompts, and results in a MySQL database, experiments can be reliably replicated under identical conditions, enabling consistent validation of LLMs in DF. The architecture-agnostic design further enhances its versatility, ensuring compatibility across various computing environments. Whether hosted on GPU-enabled local servers or deployed on cloud platforms, Ollama Docker containers ensure smooth operation regardless of hardware or software infrastructure. These features, illustrated in Figure 1, establish the framework as a flexible and robust solution for a wide range of forensic validation tasks.

[3]https://llm.datasette.io/en/stable/

4.2 Database

The AutoDFBench database is designed to support the management and analysis of experiments. Key components include tables for dynamic configurations, experiment management, and results storage. The configuration table tracks parameters such as API URLs, API keys, and disk paths, while the jobs table manages experiment definitions, including test parameters such as model type, disk image type, and script settings.

Prompt generation and test outcomes are recorded, including metadata such as response times, token counts, and execution statuses. These details enable a comprehensive performance tracking and facilitate analysis in various test scenarios. The results are categorised by hit types (active, deleted, and unallocated), allowing for a detailed performance evaluation and accuracy scoring.

4.3 Software

The framework operates in four key phases: API handling, code preparation, code execution, and summary generation, as illustrated in Figure 3 and detailed in Section 6.

In the API handling phase, the framework retrieves job details and base prompts, combines them with additional input, and generates responses using specified language models. API calls for remote models like GPT-4o and Claude 3.5 Sonnet adhere to rate limits and token usage policies, ensuring reliable operation.

During code preparation, the system processes generated prompts, extracts, and saves code snippets from the LLMs' responses, and organises them into structured output folders for traceability. Once prepared, these scripts are queued for execution.

The code execution phase runs the scripts, verifying disk paths and permissions to ensure valid test environments. A fallback mechanism addresses corrupted or inaccessible disk paths. The results of the execution, including any errors, are recorded for analysis, with a timeout limit ensuring testing efficiency for badly formatted code.

Finally, test results are validated against ground truth data in the summary phase and metrics such as precision, recall, and F1 scores are calculated. These results are stored for detailed performance analysis and benchmarking.

The framework is modular and configurable, allowing each phase to run independently. This flexibility enables users to adapt the system to specific needs and test cases, facilitating comprehensive and scalable evaluations.

4.4 Ground Truth

The framework relies on ground truth data from the NIST CFTT program for validation. These data are formatted and stored in the `ground_truth` table within the database. The framework compares search results with the ground truth during validation to ensure accuracy. More details on the usability of these data are provided in Section 5.1.

4.5 Score Calculation

The framework uses ground truth data to compute precision, recall, and F1 scores for each test run by analysing true positives, false positives, and false negatives.

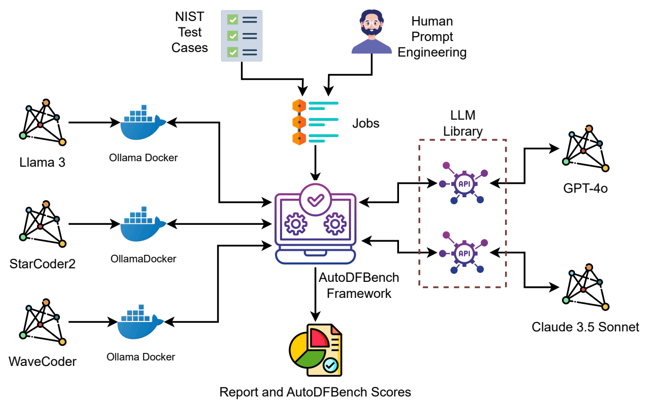

Figure 2: Overview of the proof of concept and evaluation of the framework

True Positives are matches between test results and ground truth data for a specific test case. False Positives are hits in the test results that do not align with the ground truth, indicating incorrect detections. False Negatives are expected outcomes from the ground truth not identified in the test results, representing missed detections.

Based on these metrics, precision, recall and F1 scores are calculated for each subtest. The database stores these scores, and the average F1 score is computed across subtests. This average, referred to as the AutoDFBench score, provides a benchmark to evaluate the performance of different language models and to suggest strategies in the forensic string search.

5 Proof of Concept

This section demonstrates the capabilities of the framework and its practical usability. The evaluation used the Forensic String Search test cases provided by CFTT, employing three open-source LLMs and two closed-source LLMs. Figure 2 provides an overview of the evaluation process for code generated by the LLMs, including Llama 3, StarCoder2, WaveCoder, GPT-4o, and Claude 3.5 Sonnet. The framework integrates human-engineered prompts and NIST test cases as input for testing and validation. It interacts with the LLMs to generate task-specific code, executes the resulting scripts, evaluates outputs against ground truth data, and computes the AutoDFBench Scores to benchmark the models' performance.

5.1 Forensic String Search

String-based evidence is critical in investigations, encompassing data such as natural language text, financial transactions, logs, emails, and other text-based information [1]. String searching is one of the most commonly performed tasks by practitioners [7], making it a practical choice for proof-of-concept testing.

The CFTT defines two core requirements for string search tools: returning exact matches for a given keyword and supporting searches using specific character representations. In addition, 15 non-functional requirements are outlined, including search area specification, stemming, and synonym searches. Based on these criteria, NIST has validated numerous forensic tools [18].

The ground truth data, sourced from NIST's 'Expected Results'[4], lists four-digit numbers each tool must locate, with their file locations specified as Allocated, Unallocated, or Deleted. For Linux, only Allocated and Deleted spaces are covered. Approximately 1,900 test cases for Windows and Linux were added to the `ground_truth` table, focussing on these two locations for Linux experiments.

5.2 LLM Selection

According to Jiang et al. [9], the pass@k = 1 scores for code generation are 84.1% for GPT-4, 82.9% for Claude 3 Opus, 72.6% for StarCoder2-Instruct, 74.4% for CodeFuse, and 81.7% for Llama 3, based on the largest versions of each model. These high HumanEval scores make them suitable for this proof-of-concept.

Anthropic recently reported that Claude 3.5 Sonnet achieves a pass@k = 1 score of 92%, surpassing Claude 3 Opus, while GPT-4o achieves 90.2%, outperforming GPT-4. These evaluations highlight the advances in the code generation capabilities of Claude 3.5 Sonnet and GPT-4o.

For this demonstration, five LLMs were selected: GPT-4o, Claude 3.5 Sonnet, WaveCoder, StarCoder2-Instruct, and Llama 3. These models were chosen for their high HumanEval benchmark performance and compatibility with the Ollama framework, ensuring robust code generation for the framework's tasks [9].

5.3 Implementation

The framework and testing environment were deployed in a Docker-enabled setup. Separate Docker containers were used for the software and MySQL database. The testing server featured a 3.6 GHz Intel Core i7 CPU with 8 cores and 192 GB of RAM, along with NVIDIA GeForce RTX 4090 and RTX 3090 GPUs, each equipped with 24 GB of VRAM, for LLM response generation.

API keys for GPT-4o and Claude 3.5 Sonnet were securely stored in the `config` table. For local LLMs, three individual Ollama Docker containers were configured using `docker-compose.yml` files. These files defined settings such as Docker IP addresses, container names, ports, and GPU preferences. An internal Docker network facilitated communication within the server, and the IP addresses of the configured containers were recorded in the `config` table.

The test disk images provided by NIST, approximately 2 GB each in raw format for Windows and Linux[5], were used for evaluation. These disk images were placed adequately with appropriate permissions to ensure seamless access and execution by the framework.

5.4 Base Prompts

The framework requires base prompts to guide LLMs in structuring their outputs and formulating their approaches. For this experiment, two distinct base prompts were used, each designed to simulate different levels of forensic expertise.

The first base prompt was comprehensive and detailed, providing extensive guidance on conducting a string search. It encompassed 418 words and included suggestions for libraries, Linux commands, and examples. This prompt aimed to replicate the instructions that an advanced forensic investigator might typically provide.

[4]https://cfreds.nist.gov/all/NIST/StringSearch,V11
[5]https://www.nist.gov/itl/ssd/software-quality-group/computer-forensics-tool-testing-programme-cftt/federated-testing

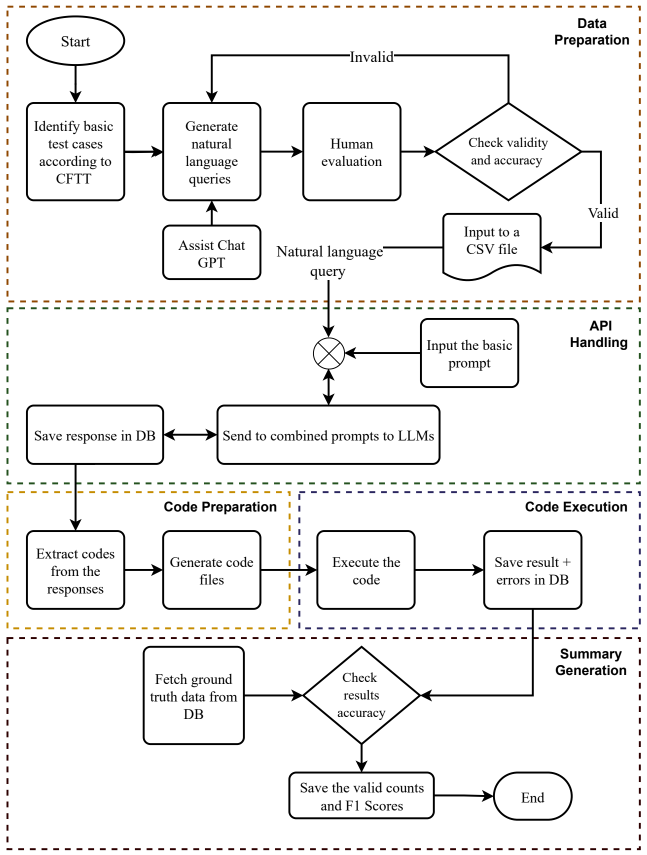

Figure 3: Flow diagram of the experimentation

The second base prompt was concise, containing 105 words. It instructed the LLM to act as an investigator and focus on delivering results without offering detailed procedural hints.

In both cases, the expected output format was standardised to ensure consistency in the results generated by the LLMs.

6 Experiment Flow

As shown in Figure 3, the test data was prepared for seamless integration into the framework. All test cases were compiled into a spreadsheet, and 1,330 human-like prompts were generated using ChatGPT. These prompts were manually reviewed to ensure accuracy before being input into the framework.

During the API Handling phase, these natural language prompts were combined with base prompts and sent to LLMs, generating responses stored in the database. The Code Preparation phase extracted code from these responses and generated the corresponding files. In the Code Execution phase, the framework executed the generated code and saved the outcomes in the database. The executions were performed in a Conda environment pre-configured with libraries for string extraction in Python and tools and dependencies essential for the process. This environment was defined in an `environment.yml` file containing approximately 180 libraries and dependencies. Finally, during the summary generation phase, the outputs were cross-validated against ground truth data, and summary statistics were recorded.

For this experiment, 40 job configurations were defined. These configurations included combinations of five LLMs, two base prompts, two script types, and two disk types, resulting in 40 jobs. Windows test cases included 59 cases, while Unix involved 62, each repeated 10 times for validation, generating 24,200 unique tests. Each job was run within the Conda environment and summary results were logged in the database.

The process of code generation, execution, and evaluation was computationally intensive. Local LLMs averaged approximately 1 minute for code generation and 2 minutes for execution per test, requiring about 484 hours. Cloud-based LLMs (Claude 3.5 Sonnet and GPT-4o) completed code generation in approximately 5 seconds per test but required 3 minutes for execution, totalling 496 hours. Overall, the experimentation took approximately 980 hours for all test cases.

7 Results

This section presents the findings from the proof-of-concept experiments, examining the performance of LLMs in string search tasks. The analysis focusses on the impact of different triggers, the capabilities of LLMs, and the factors influencing their accuracy, efficiency, and reliability. All experiments were conducted as 0-shot tests with the LLMs and results were obtained by averaging hits across subtests compared to the ground truth. Table A.1 details the results of the Windows and Linux test cases, classified by LLM, OS, coding language, and prompt level (beginner or advanced). The average F1 scores were also calculated for each test case. These F1 scores are then averaged across all test cases to produce the AutoDFBench forensic string search (FSS) score for each LLM.

Despite a poor performance, Claude 3.5 Sonnet and GPT-4o achieved the highest benchmarks, with values of 0.043 and 0.036, respectively. The results highlight that advanced prompts consistently produce higher F1 scores, demonstrating the critical role of input prompt detail in the generation of effective code. Among open-source LLMs, WaveCoder achieved the highest F1 score with advanced prompts, suggesting its potential for DF fine-tuning.

In particular, Claude 3.5 Sonnet and StarCoder2-Instruct successfully identified all social security numbers in the Linux environment. High hits in the phone number test case suggest that most LLMs, except Llama 3, generated an accurate code to identify numeric string values. For ASCII-related string searches, the accuracy in locating all search strings was also consistently high.

Furthermore, of the 4,840 test runs per LLM, it is found that GPT-4o and WaveCoder achieved an F1 score of 1 in just 11 instances, while Claude 3.5 Sonnet and StarCoder2-Instruct achieved it 9 and 3 times, respectively. However, Llama 3 failed to achieve an F1 score of 1 in any run.

To determine the best performance run among the ten trials for each test case, the highest F1 score was selected, as shown in Table A.1. The AutoDFBench score was calculated by averaging the F1 scores in all subtest cases, with equal weight assigned to each subtest. The results indicate that Claude 3.5 Sonnet achieved the highest AutoDFBench score of 0.421 using the advanced prompt,

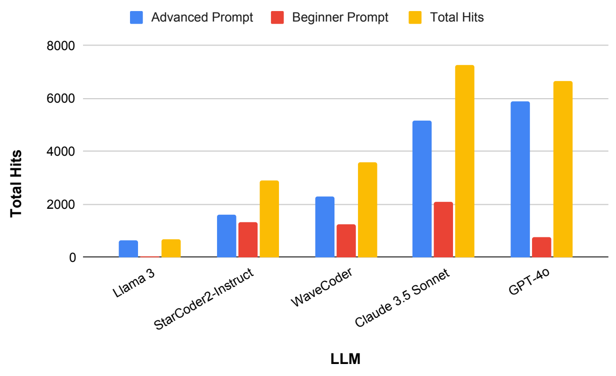

Figure 4: Summary of total keywords found

followed by GPT-4o. Among open-source models, WaveCoder outperformed Llama 3 and StarCoder2-Instruct in average F1 score, reaffirming its potential for AI-driven digital forensic applications.

The significant gap between the 'best run' scores and the average scores from the ten runs highlights the variability inherent in generative AI systems. Although the higher 'best run' scores suggest promising future possibilities for fine-tuned LLMs tailored for digital forensic code generation, the consistently low average scores underscore the current limitations of generic LLMs in delivering reliable and consistent performance for this use case.

The framework also facilitates evaluating the impact of user prompt quality/detail. In the evaluation performed as part of this paper, two levels of simulated expertise were analysed with each of the LLMs. Figure 4 summarises the total number of keyword search hits for each LLM. The results show that advanced prompts, which offer more detailed instructions, lead to higher hits than beginner prompts, which provide minimal guidance.

Although GPT-4o achieved the highest hit count with advanced prompts, Claude 3.5 Sonnet performed best considering total hits across all prompt levels. Among open-source LLMs, WaveCoder ranked highest in finding accurate keywords. Llama 3 had the lowest hit count, suggesting that it may not be a suitable choice for DF code generation fine-tuning.

7.1 Discussion

GPT-4o and Claude 3.5 Sonnet accurately identified only 5.5% and 4.5% of cases, respectively, demonstrating the limited capabilities of advanced commercial LLMs for digital forensic string searches. WaveCoder outperformed StarCoder2-Instruct and Llama 3 among open-source models, with 5.5%, 1.5%, and 0% keyword search accuracy, respectively. Llama 3 showed negligible utility for DF tasks.

Despite being 0-shot responses from generic LLMs, the models tested achieved relatively satisfactory keyword search hit rates and F1 scores for *some* of the individual test cases, indicating reasonable performance without specific training for those specific tasks. However, this performance was inconsistent across all forensic string search test cases used as part of the validation of the benchmarking framework. Significant improvement could be achieved through a combination of better prompt design, AI agents, and fine-tuned models tailored to digital forensics, improving both accuracy and reliability. This experiment used only two base prompts, suggesting that future work could explore a wider variety of prompts. Although character encoding was not specified or validated, the framework's flexibility allows such considerations to be incorporated into future evaluations.

8 Conclusion

This paper introduced AutoDFBench, a novel framework and benchmarking score to test and evaluate AI-generated DF code and tools. The framework encompasses four main components: data preparation, API handling, code execution, and summary and score generation. Built using a MySQL database and Python, AutoDFBench is designed for flexibility, allowing seamless integration with generative AI systems, including multiple LLMs via Ollama Docker containers or remote API calls.

AutoDFBench leverages ground truth data provided by NIST, using forensic string search as a proof-of-concept evaluation method to test the effectiveness of various open-source and commercial LLMs in DF applications. Beyond its current use case, the framework is capable of evaluating prompts and iterating fine-tuned DF-focused generative AI systems. Its modular nature also allows for future integration with an API, enabling centralised retrieval of results and the possibility of assessing non-AI-generated digital forensic tools and code.

To validate its capabilities, the framework was tested using NIST CFTT's forensic string search, encompassing 24,200 tests across five LLMs. These tests demonstrated the robustness of the framework, while revealing that state-of-the-art code-generation LLMs are not yet fully equipped to handle DF-specific tasks. GPT-4o and Claude 3.5 Sonnet achieved the highest performance of the tested models, but with modest F1 scores of 0.043 and 0.036, respectively. Open-source models such as Llama 3, WaveCoder, and StarCoder2-Instruct exhibited even more limited capabilities. However, a marked improvement was observed for all LLMs when provided with detailed "advanced user" prompts, underscoring the critical role of prompt engineering, AI agents and model fine-tuning in enhancing DF task performance.

In conclusion, AutoDFBench represents a significant step forward in the validation and evaluation of AI-generated DF tools. It addresses the growing demand for reliable and scalable solutions in the field, equipping forensic investigators to meet the challenges of the AI era.

The framework will be expanded for future work to evaluate a broader range of AI-assisted DF scenarios. This will include integrating all NIST CFTT test procedures to ensure complete coverage and alignment with established NIST tool testing and validation standards. Such enhancements will bolster the framework's applicability across diverse DF tasks, advancing its utility in identifying the most effective LLMs, prompts, and methodologies for future AI-assisted DF investigations.

A Appendix

Table A.1 details the results of the Windows and Linux test cases, classified by LLM, OS, coding language, and prompt level (beginner or advanced).

Table A.1: AutoDFBench score and the best performing F1-score for LLMs benchmarked over the 10 runs for each subtest

	Llama 3		StarCoder2 Instruct		WaveCoder		Claude 3.5 Sonnet		GPT-4o	
	Beginner	Advanced	Beginner	Advanced	Beginner	Advanced	Beginner	Advanced	Beginner	Advanced
AutoBFBench Score	0	0.001	0.005	0.006	0.011	0.013	0.005	0.036	0.006	0.043
Best Performing Run	0	0.165	0.143	0.197	0.369	0.379	0.354	0.427	0.305	0.411

References

[1] Nicole Lang Beebe and Jan Guynes Clark. 2007. Digital forensic text string searching: Improving information retrieval effectiveness by thematically clustering search results. *Digital Investigation* 4 (2007), 49–54. doi:10.1016/j.diin.2007.06.005

[2] Frank Breitinger, Jan-Niclas Hilgert, Christopher Hargreaves, John Sheppard, Rebekah Overdorf, and Mark Scanlon. 2024. DFRWS EU 10-year review and future directions in Digital Forensic Research. *Forensic Science International: Digital Investigation* 48 (2024), 301685. doi:10.1016/j.fsidi.2023.301685 DFRWS EU 2024 - Selected Papers from the 11th Annual Digital Forensics Research Conference Europe.

[3] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *CoRR* abs/2107.03374 (2021). arXiv:2107.03374 https://arxiv.org/abs/2107.03374

[4] Xiaoyu Du, Chris Hargreaves, John Sheppard, Felix Anda, Asanka Sayakkara, Nhien-An Le-Khac, and Mark Scanlon. 2020. SoK: Exploring the State of the Art and the Future Potential of Artificial Intelligence in Digital Forensic Investigation. In *Proceedings of the 15th International Conference on Availability, Reliability and Security* (Virtual Event, Ireland) *(ARES '20)*. Association for Computing Machinery, New York, NY, USA, Article 46, 10 pages. doi:10.1145/3407023.3407068

[5] Lizhou Fan, Lingyao Li, Zihui Ma, Sanggyu Lee, Huizi Yu, and Libby Hemphill. 2023. A Bibliometric Review of Large Language Models Research from 2017 to 2023. *CoRR* abs/2304.02020 (2023). doi:10.48550/ARXIV.2304.02020 arXiv:2304.02020

[6] Yinghua Guo, Jill Slay, and Jason Beckett. 2009. Validation and Verification of Computer Forensic Software Tools—Searching Function. *Digital Investigation* 6 (2009), S12–S22. doi:10.1016/j.diin.2009.06.015 The Proceedings of the Ninth Annual DFRWS Conference.

[7] Christopher Hargreaves, Frank Breitinger, Liz Dowthwaite, Helena Webb, and Mark Scanlon. 2024. DFPulse: The 2024 digital forensic practitioner survey. *Forensic Science International: Digital Investigation* 51 (2024), 301844. doi:10.1016/j.fsidi.2024.301844

[8] Hans Henseler and Harm van Beek. 2023. ChatGPT as a Copilot for Investigating Digital Evidence. In *Proceedings of the Third International Workshop on Artificial Intelligence and Intelligent Assistance for Legal Professionals in the Digital Workplace (LegalAIIA 2023) co-located with the 19th International Conference on Artificial Intelligence and Law (ICAIL 2023), Braga, Portugal, June 19, 2023 (CEUR Workshop Proceedings, Vol. 3423)*, Jack G. Conrad, Daniel W. Linna Jr., Jason R. Baron, Hans Henseler, Paheli Bhattacharya, Aileen Nielsen, Jyothi K. Vinjumur, Jeremy Pickens, and Amanda Jones (Eds.). CEUR-WS.org, 58–69. https://ceur-ws.org/Vol-3423/paper6.pdf

[9] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A Survey on Large Language Models for Code Generation. arXiv:2406.00515 [cs.CL] https://arxiv.org/abs/2406.00515

[10] Gaëtan Michelet and Frank Breitinger. 2024. ChatGPT, Llama, can you write my report? An experiment on assisted digital forensics reports written using (local) large language models. *Forensic Science International: Digital Investigation* 48 (2024), 301683. doi:10.1016/j.fsidi.2023.301683 DFRWS EU 2024 - Selected Papers from the 11th Annual Digital Forensics Research Conference Europe.

[11] Dong Bin Oh, Donghyun Kim, Donghyun Kim, and Huy Kang Kim. 2024. volGPT: Evaluation on triaging ransomware process in memory forensics with Large Language Model. *Forensic Science International: Digital Investigation* 49 (2024), 301756. doi:10.1016/j.fsidi.2024.301756 DFRWS USA 2024 - Selected Papers from the 24th Annual Digital Forensics Research Conference USA.

[12] Partha Pratim Ray. 2023. ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope. *Internet of Things and Cyber-Physical Systems* 3 (2023), 121–154. doi:10.1016/j.iotcps.2023.04.003

[13] Mark Scanlon, Frank Breitinger, Christopher Hargreaves, Jan-Niclas Hilgert, and John Sheppard. 2023. ChatGPT for digital forensic investigation: The good, the bad, and the unknown. *Forensic Science International: Digital Investigation* 46 (2023), 301609. doi:10.1016/j.fsidi.2023.301609

[14] Mark Scanlon, Bruce Nikkel, and Zeno Geradts. 2023. Digital forensic investigation in the age of ChatGPT. *Forensic Science International: Digital Investigation* 44 (03 2023), 301543. doi:10.1016/j.fsidi.2023.301543

[15] Yiqiu Shen, Laura Heacock, Jonathan Elias, Keith D. Hentel, Beatriu Reig, George Shih, and Linda Moy. 2023. ChatGPT and Other Large Language Models Are Double-edged Swords. *Radiology* 307, 2 (2023), e230163. doi:10.1148/radiol.230163 PMID: 36700838.

[16] Akila Wickramasekara, Frank Breitinger, and Mark Scanlon. 2024. Exploring the Potential of Large Language Models for Improving Digital Forensic Investigation Efficiency. arXiv:2402.19366 [cs.CR] https://arxiv.org/abs/2402.19366

[17] Akila Wickramasekara and Mark Scanlon. 2024. A Framework for Integrated Digital Forensic Investigation Employing AutoGen AI Agents. In *2024 12th International Symposium on Digital Forensics and Security (ISDFS)*. 01–06. doi:10.1109/ISDFS60797.2024.10527235

[18] Tina Wu, Frank Breitinger, and Stephen O'Shaughnessy. 2020. Digital forensic tools: Recent advances and enhancing the status quo. *Forensic Science International: Digital Investigation* 34 (2020), 300999. doi:10.1016/j.fsidi.2020.300999

[19] Hanxiang Xu, Shenao Wang, Ningke Li, Kailong Wang, Yanjie Zhao, Kai Chen, Ting Yu, Yang Liu, and Haoyu Wang. 2024. Large Language Models for Cyber Security: A Systematic Literature Review. arXiv:2405.04760 [cs.CR] https://arxiv.org/abs/2405.04760