Safeguarding Scientific Algorithms: A Framework for Detecting Algorithmic Plagiarism in Metaheuristic Research

Colin L. Heffern[1], Halbert N. Nguyen[1], Walter H. Cook[1], Gunni U. Kamran[1], Giovanna Camacho[1], and Matthew L. Bolton[1*]
*mlb4b@virginia.edu

[1] Department of Systems and Information Engineering, University of Virginia, Charlottesville, VA

***Abstract*—Academic misconduct increasingly occurs through algorithmic duplication, in which existing, lightly modified methods are presented as novel. This problem is especially relevant in metaheuristic optimization, a field that has experienced rapid growth along with concerns about derivative algorithms. This paper presents an AI-driven framework for detecting algorithmic plagiarism in metaheuristic research by comparing the computational logic of algorithms rather than their surface-level similarity. The proposed pipeline extracts pseudocode from academic PDFs, canonicalizes algorithms, and compares them against a reference database using four levels of similarity analysis: surface, structural, primitive, and semantic. Scores from each level are combined into an overall similarity score. The system was tested through controlled verification cases and real non-novel examples identified by experts. Results showed clear separation between disguised derivations of the same algorithm and genuinely distinct algorithms, supporting the framework's ability to measure algorithmic novelty. These findings suggest that automated detection of algorithmic plagiarism is feasible. It could support research integrity efforts by helping journals and reviewers identify submissions in need of close expert review.**

***Keywords*—Algorithm design and analysis, detection algorithms, metaheuristics, plagiarism, semantic analysis**

I. Introduction

In 2023, Hindawi, an open-access scientific publisher, retracted a record 8,000 papers across its journals after discovering evidence of paper mills, fabricated results, and other academic misconduct [1]. This scandal revealed the scale of academic fraud and the "systematic manipulation of the publication and peer review process" [1–3]. In 2024, over 14,000 papers were retracted globally. This represents a 40% increase from 2023 and a 460% increase over five years [4]. With retracted papers representing only about 0.2% of total publications, this is just the tip of the iceberg. Hundreds of thousands of suspected fraudulent papers have not yet been retracted [1]. This trend shows that research misconduct is becoming more common and higher in quality; a problem driven by increasing publishing pressure and better tools that make misconduct easier than ever [5]. This has put the credibility of the global scientific knowledge base at risk, upon which future research and decisions depend [6].

Among all academic disciplines, Electrical Engineering and Computer Science have the highest rate of academic misconduct retractions [7]. In computational research, misconduct is increasingly carried out through algorithmic duplication. This means reusing or slightly modifying existing algorithms and presenting them as entirely new findings. Widespread duplication and low-quality publications undermine the cumulative nature of science by creating confusion and noise in the literature [8]. This problem is especially common in the optimization field of metaheuristics, which "attempts to find the best (feasible) solution out of all possible solutions to an optimization problem" [9]. This project addresses the problem by developing an AI-driven detection pipeline capable of flagging publications whose algorithms exhibit suspicious amounts of similarity with existing algorithms.

The remainder of this paper describes how we developed and evaluated our detection approach. It does this by first providing background on metaheuristic algorithms and academic fraud detection tools. This is followed by a more precise formulation of the project's objectives. We then describe our method and the analyses we performed to calibrate and evaluate it. Finally, we discuss the implications of our results and opportunities for future development.

II. Background

A. Metaheuristic Optimizations Algorithms

Metaheuristic optimizations find problem solutions (e.g., parameters that minimize cost while satisfying design constraints) using stochastic search techniques [10]. By avoiding the use of derivatives or truly exhaustive searches, they can find near-optimal solutions while being very computationally efficient. As such, they are useful when the problems being evaluated are extremely complex and/or NP-hard, where other approaches fail. They generally work by iteratively switching between two modes. In the first exploration, the search attempts to explore the solution space broadly to identify promising regions. In the second exploitation, a narrower, more refined search attempts to find the optimal solution. By switching between these modes, metaheuristic algorithms can find near-optimal solutions while avoiding getting stuck in local extrema. Common algorithms in this space include the Genetic Algorithm, Particle Swarm Optimization, Simulated Annealing, and the Grey Wolf Optimizer. As these names suggest, metaphors are common in this space where natural phenomena or animal behaviors inspire behavior within each mode and the switching between them. There is a high volume of metaheuristic algorithms to the extent that open source

This work was supported by the project "Safeguarding Science: Developing Knowledge and Tools to Prevent Scientific Fraud" from the Jefferson Trust

libraries like MEALPY [11] have been created to enable people to develop and publish new ones with ease.

B. Criticism and Analysis of Metaheuristic Optimization

While there are legitimate Metaheuristic approaches, there has been an explosion of these papers in the literature [12–22], with many of the algorithms having strange names such as Harris Hawks optimization, Ebola optimization search, "optimization method based on mimicking cooking training," Coati Optimization, and Slime mold algorithm [23–28]. In fact, a few other researchers have noticed this trend and have done work to manually show that most are highly derivative and thus lack substantive contributions to the field [12, 29–33]. This includes some of the most well-cited algorithms in the area, such as the gray wolf optimizer [34] (22,443 citations at the time of writing) and cuckoo search [35] (9,712 citations).

C. Academic Fraud and Algorithmic Novelty Detection

With publishers and editors scrambling to develop better methods to detect fraudulent submissions, many legitimate researchers face suspicion and stigmatization. While scientific integrity is of the utmost importance, false accusations can damage careers and reputations. This tension illustrates the challenge of upholding integrity while ensuring fairness.

Stakeholders have increasingly turned to automated academic misconduct detection tools to address integrity challenges. While these tools have promising potential, they are often biased, opaque, or faulty, raising concerns about fairness and accountability [36]. For instance, in 2023, Turnitin released an AI writing detector. Following this, universities raised concerns about false positives, limited transparency, and the risk of falsely accusing writers based on a score they could neither verify nor challenge. Turnitin later admitted to a "higher false positive rate" than originally claimed, resulting in the tool being banned from multiple institutions [37]. This example shows how automated misconduct detection tools do not simply solve issues of integrity, but create new disputes over fairness, transparency, and power dynamics within academia.

These are all concerns when developing automated tools for assessing algorithmic novelty. Traditional plagiarism checkers cannot detect algorithmic or pseudocode-level plagiarism as they rely on textual or lexical similarity rather than code-level logic originality [38]. These tools may flag identical wording, but overlook functionally identical algorithms written in different terms or slightly altered pseudocode. This can cause duplicated algorithms to pass through peer review or automated tools undetected.

III. Objectives

To address the limitations of existing detection methods, our proposed approach introduces an automated process that performs a structural comparison to evaluate the underlying algorithmic logic rather than surface-level phrasing. By quantifying similarity through pseudocode analysis rather than textual overlap, we hypothesized that our pipeline would identify patterns of algorithmic duplication more effectively. To address potential issues associated with false alarms (discussed above), our method focuses on generating a descriptive similarity score over a binary decision. More specifically, this project sought to accomplish the following objectives.

A. Develop a Multi-stage Pipeline

The pipeline is intended to extract pseudocode from academic papers, standardize the representations of the algorithm, and structurally compare this against a reference database of existing optimization algorithms. To be effective, the system must accurately distinguish between genuinely novel algorithms and those that are computationally equivalent, but with modified (disguised) syntax.

B. Implement a Quantitative Similarity Scoring Framework

A quantitative similarity scoring framework should integrate multiple levels of analysis, including surface, structural, primitive, and semantic similarity produced by the pipeline. It is intended to weight each of these using a similar metric to flag potentially fraudulent algorithms for further review.

C. Validate System Performance

We sought to use controlled and representative test cases, as well as real-world algorithm implementations, to assess, calibrate, and validate the performance of our approach using confusion-matrix metrics.

IV. Method

The detection framework developed in this project is an AI-driven, multistage pipeline designed to identify algorithmic plagiarism and superficial novelty in metaheuristic optimization research using the flow diagram shown in Fig. 1.

Language model inference is performed locally using Ollama, a lightweight runtime that allows large language models (LLMs) to be downloaded, managed, and queried through a local API. Ollama was chosen over cloud-based alternatives for several reasons. First, running academic papers through external APIs raises data privacy and intellectual property concerns. Second, running them locally eliminates rate limits and per-query costs, enabling batch processing across hundreds of algorithm pairs. Third, a self-contained local environment produces fully reproducible results, which is an essential property for any tool that makes claims about research integrity, as scores must be verifiable and consistent across runs.

Algorithmic content in papers can be embedded as text or images. Therefore, our system uses two different model types through Ollama. For text-based analysis, including algorithm canonicalization, primitive extraction, structural comparison, and semantic similarity scoring, the pipeline uses Llama 3.1. This was selected for its strong instruction-following capability and support for structured output, which is critical for downstream parsing. For visual extraction of pseudocode embedded as images, the pipeline uses LLaVA.

Furthermore, MealPy (see Section II-A) was chosen as the reference database against which all algorithms were compared. MealPy was chosen because it is the most comprehensive publicly available reference for this domain, consisting of over 100 algorithms that appear frequently in metaheuristic research. The following sections step through each stage of our pipeline (Fig. 1) and describe how they were realized.

A. Pseudocode Extraction

The pipeline's first stage extracts algorithmic pseudocode from PDF publications. Pseudocode appears in papers in several formats, including code, images, and natural language mixed into the body of the paper. Thus, the extraction tool

needed multiple strategies to ensure it captures the algorithm regardless of the format in which it is presented.

Text layer extraction is performed using PyMuPDF (Artifex Software, n.d.). This parses the PDF structure and retrieves text content page by page. For image-embedded pseudocode, the pipeline iterates through every image, writing each one to a temporary file and submitting it to the vision model with the instruction to extract any visible pseudocode precisely as written. Results from both extraction pathways are combined and saved to JSON files for use in subsequent pipeline stages. This dual pathway design ensures that algorithmic content is captured regardless of the paper's format.

B. Algorithm Canonicalization

Once the pseudocode is extracted, it is passed on to the canonicalization stage implemented in the AlgorithmCanonicalizer class. Canonicalization converts an algorithm into a standardized form, stripping away cosmetic differences while preserving the underlying computational logic. This step is essential because the most common forms of algorithmic plagiarism, such as renamed variables, swapped metaphors (e.g., substituting wolf-pack for eagle-flock terminology), and reformatted control structures, involve surface-level modifications which canonicalization removes.

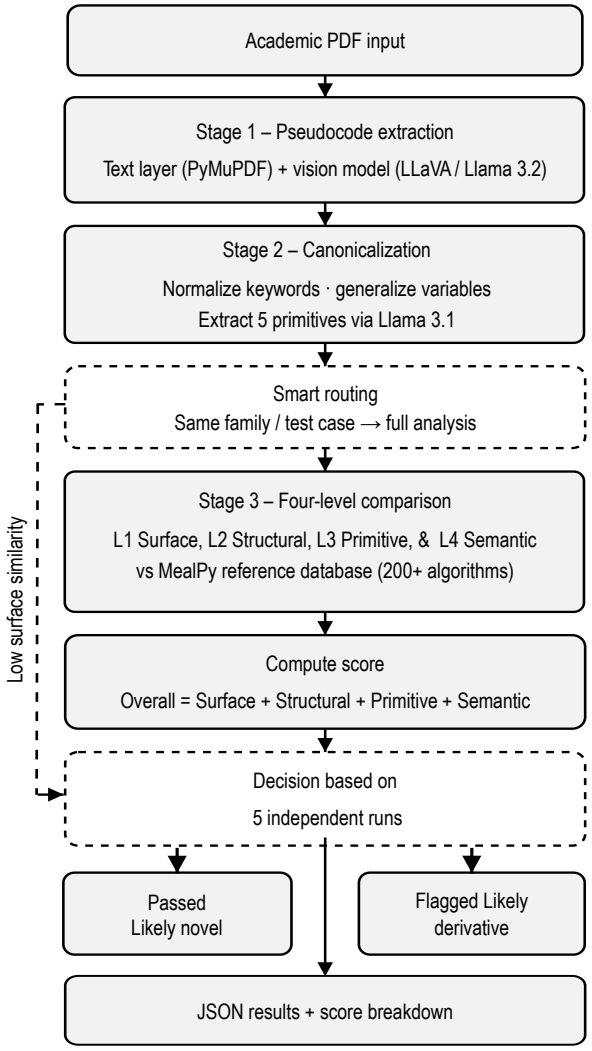

Fig. 1. Detection Pipeline Overview

The canonicalization process operates in four sub-steps. First, basic text normalization is applied, where common control flow keywords are standardized to a fixed vocabulary. Phrases like "for each" and "iterate over" are both mapped to "for," just as "calculate" and "compute" are unified. Second, variable names are replaced with generic placeholders, such as var1 and var2, with the original names preserved for interpretability. Third, the high-level control flow structure is extracted, capturing the presence and nesting of loops, conditionals, and function calls. Fourth, and most critically, the LLM is used to extract five computational primitives from the pseudocode: the initialization pattern, the loop pattern, the update mechanism, the selection mechanism, and the termination condition. These five primitives represent the core computational identity of an optimization algorithm and form the basis for the deepest level of comparison.

C. Four-level Similarity Comparison

Canonicalized algorithms are compared against the MealPy reference database using a four-level weighted scoring system implemented in the DeepComparator class. Each level of comparison captures a different dimension of algorithmic similarity, and the combined score is their sum.

1) Surface Similarity: Surface similarity is computed as the word overlap between canonicalized algorithms. Because this comparison operates on post-canonicalization text, it is more robust than raw text matching. However, it is still susceptible to vocabulary changes.

2) Structural Similarity: Structural similarity measures the similarity of control flow structures: loop nesting depth, the presence of conditionals, and the overall organization of the algorithmic skeleton. This level is designed to catch cases where identical computational logic was reformatted with different variable names or statement orders.

3) Primitive Similarity: Primitive similarity compares the extracted computational primitives between two algorithms using LLM-based analysis. For each primitive dimension (initialization, loop pattern, update mechanism, selection mechanism, and termination), the model assesses whether the two descriptions represent the same underlying operation, scoring each from 0 to 1, with the focus on computational equivalence.

4) Semantic Similarity: The LLM is asked to perform a holistic comparison of both algorithms, considering their overall computational behavior, the type of optimization strategy employed, and whether the two could reasonably be considered instances of the same underlying method. An important design choice in this layer is the introduction of a scoring rubric that treats shared metaheuristic traits (such as population-based iteration, fitness evaluation, and stochastic position updates) as a baseline rather than as evidence of similarity. Without this rubric, virtually all metaheuristic comparisons (using the cases we will describe subsequently) scored in the 0.7 to 0.8 range, regardless of whether the algorithms were actually related; all metaheuristics share these general patterns. It was hypothesized that this semantic layer would correctly distinguish between genuinely related algorithms and those that merely belong to the same broad class.

5) The Combined Score: The combined score is computed as the sum of the similarity scores from all four levels:

$$\begin{aligned} Overall = Surface + Structural \\ + Primitive + Semantic. \end{aligned} \tag{1}$$

Weighting of layers is discussed later in Section VI-A.

D. Smart Routing and Efficiency Optimizations

Full, four-level, LLM-based comparison is computationally expensive and impractical at scale. To address this, the system implements a smart routing strategy that determines the appropriate level of analysis for each pair. Pairs within the same algorithm family (an organizational feature of MealPy), same paper comparisons, and test cases will always receive the full four-level analysis, as these are the highest priority comparisons; a surface-level check would be at risk of missing a disguised derivative. For pairs from completely different families with very low surface-level similarity, the system skips the full LLM analysis and records the surface score directly. This routing logic significantly reduces the number of LLM calls required for large-scale processing, while preserving depth for the comparisons that matter most.

V. Testing and Validation

The testing and validation stage focused on two efforts. The first verified the performance of our approach using the test cases we created. The second sought to validate the method's performance by assessing the model's ability to detect non-unique algorithms that were identified as such by experts in the scientific literature.

A. Verification Testing

1) Test Cases and Analyses: To verify that our method was performing as intended (and to identify an appropriate threshold), we constructed a set of test cases. This approach was used because it allowed us to vary the level that an existing algorithm was disguised when comparing our method's similarity score (Eq. (1)). These cases spanned three distinct algorithm families: Genetic Algorithm (GA), Artificial Bee Colony (ABC), and Brown Bear Optimization Algorithm (BBOA). These families were chosen to represent the three major categories of metaheuristic optimization: evolutionary, swarm-based, and bio-inspired approaches.

Each family contributes three algorithm variants to the calibration set based on the degree to which the algorithm was disguised. The **light** disguise applied only variable renaming, replacing identifiers such as "solution" with "location" and "pop_new" with "batch_new," while leaving all equations and control flow untouched. The **medium** disguise added structural changes on top of the light version, including reordering of non-dependent initialization lines, splitting single-line update equations into named intermediate variables, and using tuple assignment instead of sequential assignment. The **heavy** disguise further rewrote equations into algebraically equivalent forms, extracted logic into helper methods, and simplified compound expressions. For example, in the BBOA heavy disguise, the phase one update equation $x + (-pp \cdot r \cdot x)$ was rewritten as the equivalent multiplicative form $x \cdot (1 - pp \cdot r)$, and the phase three expression was algebraically simplified from $(ww \cdot best - |x|) - (ww \cdot worst - |x|)$ to $ww \cdot (best - worst)$.

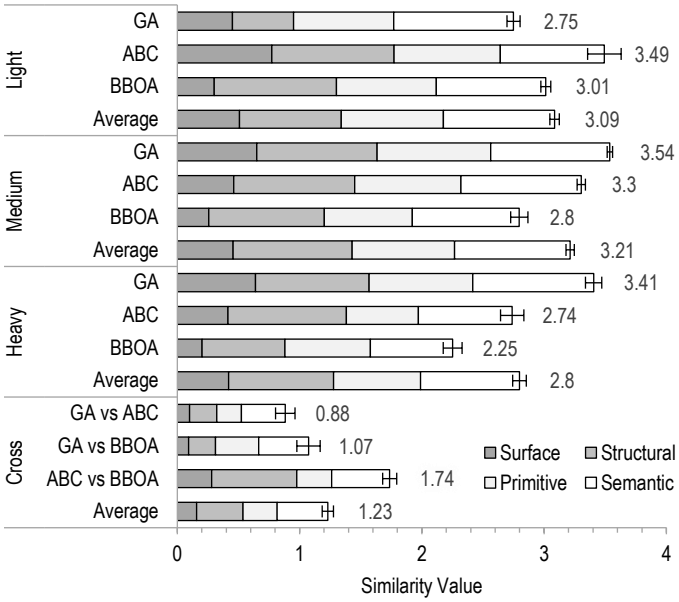

Fig. 2. Bar chart showing the average values (and their 95% confidence intervals) returned from our algorithmic similarity checker across each of our verification test cases. The topmost nine results are grouped based on their level of disguise (Light, Medium, or High) and then their algorithmic family (GA, ABC, BBOA). The bottom three results are those for comparisons across (Cross) families. Averages within each broad category (Light, Medium, High, Cross) are also reported. Scores are decomposed based on the values the checker returned for Surface, Structural, Primitive, and Semantic similarity, all stacked to show their contribution to overall similarity (Eq. (1)).

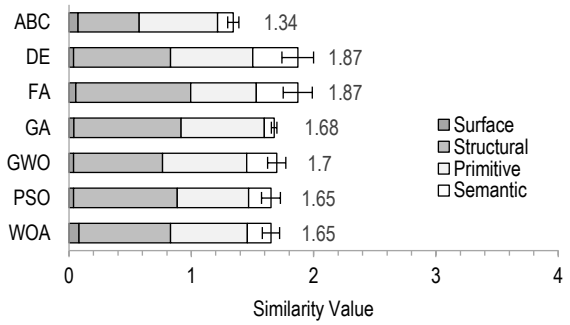

Fig. 3. Bar chart showing the average values (and their 95% confidence intervals) returned from our algorithmic similarity checker across each of our validation test cases. See Fig. 2 for instructions for interpreting the plot.

This design produced 9 test cases. This allowed us to perform multiple tests, where each disguised case was compared against its original method. High similarity was expected because the algorithms were functionally identical. The original (unaltered) algorithms were also compared across families. In these, low similarity was expected because the algorithms were fundamentally different. Each of these evaluations was performed 5 times to collect data on performance variance because of the variance introduced by the LLM in the primitive and semantic layers.

2) Results: Each of these analyses was conducted using the team's implementation of the method. An overview of these results is reported in Fig. 2. There are several important observations that can be made by examining this data. First, there is high similarity across all disguises within the same family: all nine within-family comparisons produced overall scores above 2, with the majority exceeding 2.6, including the heavy disguise category. Second, there was clearly lower sim-

ilarity in the cross-family similarity scores: all scoring below 2. Third, the results suggest that our method produces stable results. That is, even with LLM variance, runs maintained a clean gap between the lowest heavy disguise score and the highest cross-family score, with no overlap observed. Together, these results suggested that our scoring system can clearly distinguish between algorithms that were genuinely similar and those that were not.

B. Validation Testing

1) Test Cases and Analyses: Duplicated algorithms, identified by experts within scientific literature, were used to validate the system's ability to detect real cases of non-novel algorithms (two of which were in the verification test as well). Those were the Artificial Bee Colony (ABC), Differential Evolution (DE), Firefly Algorithm (FA), Genetic Algorithm (GA), Grey Wolf Optimizer (GWO), Particle Swarm Optimization (PSO), and Whale Optimization Algorithm (WOA) [29, 31, 33]. These were selected because they have either been explicitly identified as being non-novel (e.g., FA, GWO, and WOA) in the literature or used as the original from which derivatives within MEALPY have been created (e.g., ABC, DE, GA, PSO). The papers for each of these were analyzed using our method and compared against the entries found in MEALPY (excluding the algorithm being evaluated).

2) Results: The results (Fig. 3) show consistently moderate similarity across all analyzed algorithms, with overall similarity ranging from 1.34 to 1.87. Overall, these cases showed low surface and semantic similarity (similar to those from the cross-comparison verification tests), but structural and primitive similarities comparable to the within-class verifications that directly compared disguised algorithms (Fig. 2).

VI. Discussion

The verification and validation results demonstrate that our method can distinguish between computationally equivalent algorithms and genuinely distinct methods. The verification set showed clean separation between within-family and cross-family comparisons. Given standard practices within the metaheuristic algorithm space, this should allow our tool to be useful for detecting algorithmic non-novelty. Thus, the framework developed here provides a quantitative basis for what has previously been a qualitative, expert-driven assessment. However, the validation results show that the method may need modification to detect some real-world non-novel algorithms. We assess our results around several important topics below.

A. The Contribution of Each Algorithmic Layer

Layers were evenly weighted in our analysis (1) because it was unclear how important each type of similarity would be for identifying non-novel algorithms. While the within-family disguised cases from our verification tests appeared to differentiate themselves from the cross comparisons across all four dimensions, the validation cases only appeared to differentiate for the structural and primitive layer similarities.

Based on these results, we sought to combine the similarities across layers into a weighted average that maximally differentiated between non-novel (within-family comparisons and validations) and novel (cross-verification) cases. This was accomplished by using a Generalized Reduced Gradient algorithm to find values of each x in

$$\begin{aligned} Overall = x_{Sur} \cdot Surface + x_{Str} \cdot Structural \\ + x_{prim} \cdot Primitive + x_{Sem} \cdot Semantic \end{aligned} \tag{2}$$

that would maximize the distance between the minimum *Overall* score of the non-novel cases and the maximum of the novel cases. This was constrained such that each $x \in [0, 1]$ and $x_{Sur} + x_{Str} + x_{prim} + x_{Sem} = 1$.

This resulted in an updated overall similarity score of

$$\begin{aligned} Overall = 0 \cdot Surface + 0.125 \cdot Structural \\ + 0.875 \cdot Primitive + 0 \cdot Semantic. \end{aligned} \tag{3}$$

This result suggests that (contrary to our expectations for this project) semantic similarity did not differentiate between real-world cases, while primitive similarity (and to a lesser extent structural similarity) did. Future work should explore ways to improve the contribution of semantic similarity so that it can help differentiate cases of algorithms disguised beyond the methods used in our verification tests. Additional verification and validation cases are also warranted to assess whether our results generalize.

B. Threshold Determination

A similarity threshold could help determine whether an algorithm is sufficiently novel. Based on our results from the previous section and the overall similarity produced by Eq. (3), 0.46 is an obvious threshold as it is the midpoint between the maximum similarity seen for the legitimately novel (cross) comparisons and the minimum of the others. When applied, this threshold cleanly separates the genuinely novel comparisons from the others, achieving perfect detection accuracy for all cases. Future work should systematically evaluate whether this threshold generalizes to other cases in the heuristic algorithm space.

C. Consequential Design Decisions

Several design decisions significantly impacted the system's effectiveness. The most consequential was the switch from Jaccard word overlap to LLM-based comparison for the primitive similarity layer. The LLM-based approach evaluated computational equivalence, which proved to be the correct criterion for this application. Finally, the choice to use local LLM inference through Ollama rather than cloud-based APIs introduced both benefits and limitations. This ensured data privacy, eliminated per-query costs, and made the system reproducible, but at the potential cost of accuracy at the semantic layers. A future version of this system could offer the option to use more powerful models when appropriate.

D. Additional Limitations and Future Work

This work has several additional limitations that could be improved with future work. First, the set of evaluated test cases was intentionally small. While sufficient for our proof-of-concept analysis, a larger and more diverse set would strengthen the results. Future work should expand the test cases to encompass more algorithmic families and disguise strategies. Second, system performance depends on the quality of pseudocode extraction from PDFs, which can produce incomplete or inaccurate representations. Document parsing Improvements may mitigate this risk. Third, LLM-induced

variance necessitates multi-run averaging, increasing computational cost. Future work should explore alternative, more-deterministic LLM models. Fourth, the system is currently limited to metaheuristic models (and those in MealPy). Future work should explore methods to expand the scope of the considered algorithms beyond metaheuristic optimization. Finally, integration with journal submission or review workflows represents a promising path toward practical implementation.

VII. Conclusions and Broader Impacts

Despite its limitations, our framework demonstrates that automated detection of algorithmic plagiarism is feasible and can produce consistent and reliable results. Such a tool could be integrated into the academic publishing pipeline as a screening step, similar to how text-based plagiarism detection tools are used for manuscript submissions. Rather than replacing expert review, the system would serve as a triage mechanism, flagging papers that warrant closer inspection by domain experts.

The growing scale of the problem underscores the need for automated approaches. Recent surveys have documented that over 500 metaheuristic algorithms have been proposed to date, with more than 350 appearing in the last decade alone [13], many lacking novelty [12, 29–33]. Manual detection of duplicates has traditionally required significant time from experts. Our framework offers a scalable alternative that processes large volumes of material and identifies the most suspicious cases for deeper human review.

References

[1] R. Van Noorden, "More than 10,000 research papers were retracted in 2023—a new record," *Nature*, Dec. 2023.

[2] F. Downer, S. Ferguson, A. Fisher, R. Tomek, M. Bolton, W. T. Scherer, S. Johnson, E. Koca, and O. C. King, "Academic integrity in crisis: A systematic analysis of questionable research practices," in *2025 Systems and Information Engineering Design Symposium (SIEDS)*. IEEE, 2025, pp. 13–18.

[3] A. Fisher, R. Tomek, D. Downer, S. Ferguson, M. Bolton, W. Scherer, E. Koca, S. Johnson, and O. C. King, "Academic integrity in crisis: A systematic analysis of questionable research practices; taxonomy and dictionary," University of Virginia, Tech. Rep., 2025, doi:10.18130/ekr4-2e60.

[4] K. Travis, "Springer nature retracted 2,923 papers last year," *Retraction Watch*, Feb. 2025. [Online]. Available: https://retractionwatch.com/2025/02/17/springer-nature-journal-retractions-2024/

[5] H. Else, "Biomedical paper retractions have quadrupled in 20 years—why?" *Nature*, vol. 630, no. 8016, pp. 280–281, 2024.

[6] H. Lumbard and D. Routledge, "Open science and transparency are our strongest tools in the fight against fraudulent publishing activities," *PLOS Medicine*, vol. 22, no. 9, p. e1004774, 2025.

[7] M. Li and Z. Shen, "Science map of academic misconduct," *Innovation*, vol. 5, no. 2, p. 100593, 2024.

[8] J. Ioannidis and K. W. Boyack, "Citation metrics for appraising scientists: Misuse, gaming and proper use," *The Medical Journal of Australia*, vol. 212, no. 6, pp. 247–249, 2020.

[9] F. Glover, M. Laguna, and R. Martí, "Fundamentals of scatter search and path relinking," *Control and cybernetics*, vol. 29, no. 3, pp. 653–684, 2000.

[10] E.-G. Talbi, *Metaheuristics: From design to implementation*. John Wiley & Sons, 2009.

[11] N. Van Thieu and S. Mirjalili, "MEALPY: An open-source library for latest meta-heuristic algorithms in python," *Journal of Systems Architecture*, vol. 139, p. 102871, 2023.

[12] L. Velasco, H. Guerrero, and A. Hospitaler, "A literature review and critical analysis of metaheuristics recently developed," *Archives of Computational Methods in Engineering*, pp. 1–22, 2023.

[13] K. Rajwar, K. Deep, and S. Das, "An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges," *Artificial Intelligence Review*, pp. 1–71, 2023.

[14] S. Kaur, Y. Kumar, A. Koul, and S. Kumar Kamboj, "A systematic review on metaheuristic optimization techniques for feature selections in disease diagnosis: open issues and challenges," *Archives of Computational Methods in Engineering*, vol. 30, no. 3, pp. 1863–1895, 2023.

[15] A. M. Nassef, M. A. Abdelkareem, H. M. Maghrabie, and A. Baroutaji, "Review of metaheuristic optimization algorithms for power systems problems," *Sustainability*, vol. 15, no. 12, p. 9434, 2023.

[16] A. H. Halim, I. Ismail, and S. Das, "Performance assessment of the metaheuristic optimization algorithms: An exhaustive review," *Artificial Intelligence Review*, vol. 54, pp. 2323–2409, 2021.

[17] N. Khanduja and B. Bhushan, "Recent advances and application of metaheuristic algorithms: A survey (2014–2020)," *Metaheuristic and evolutionary computation: algorithms and applications*, pp. 207–228, 2021.

[18] V. Kumar and S. Yadav, "A state-of-the-art review of heuristic and metaheuristic optimization techniques for the management of water resources," *Water supply*, vol. 22, no. 4, pp. 3702–3728, 2022.

[19] F. Peres and M. Castelli, "Combinatorial optimization problems and metaheuristics: Review, challenges, design, and development," *Applied Sciences*, vol. 11, no. 14, p. 6449, 2021.

[20] M. A. Rahman, R. Sokkalingam, M. Othman, K. Biswas, L. Abdullah, and E. Abdul Kadir, "Nature-inspired metaheuristic techniques for combinatorial optimization problems: overview and recent advances," *Mathematics*, vol. 9, no. 20, p. 2633, 2021.

[21] B. Toaza and D. Esztergár-Kiss, "A review of metaheuristic algorithms for solving tsp-based scheduling optimization problems," *Applied Soft Computing*, p. 110908, 2023.

[22] L. Abualigah, M. A. Elaziz, A. M. Khasawneh, M. Alshinwan, R. A. Ibrahim, M. A. Al-Qaness, S. Mirjalili, P. Sumari, and A. H. Gandomi, "Meta-heuristic optimization algorithms for solving real-world mechanical engineering design problems: A comprehensive survey, applications, comparative analysis, and results," *Neural Computing and Applications*, pp. 1–30, 2022.

[23] A. A. Heidari, S. Mirjalili, H. Faris, I. Aljarah, M. Mafarja, and H. Chen, "Harris hawks optimization: Algorithm and applications," *Future Generation Computer Systems*, vol. 97, pp. 849–872, 2019.

[24] O. N. Oyelade, A. E.-S. Ezugwu, T. I. A. Mohamed, and L. Abualigah, "Ebola optimization search algorithm: A new nature-inspired metaheuristic optimization algorithm," *IEEE Access*, vol. 10, pp. 16 150–16 177, 2022.

[25] A. Faramarzi, M. Heidarinejad, B. Stephens, and S. Mirjalili, "Equilibrium optimizer: A novel optimization algorithm," *Knowledge-Based Systems*, vol. 191, p. 105190, 2020.

[26] E. Trojovská and M. Dehghani, "A new human-based metahurestic optimization method based on mimicking cooking training," *Scientific Reports*, vol. 12, no. 1, p. 14861, 2022.

[27] M. Dehghani, Z. Montazeri, E. Trojovská, and P. Trojovskỳ, "Coati optimization algorithm: A new bio-inspired metaheuristic algorithm for solving optimization problems," *Knowledge-Based Systems*, vol. 259, p. 110011, 2023.

[28] S. Li, H. Chen, M. Wang, A. A. Heidari, and S. Mirjalili, "Slime mould algorithm: A new method for stochastic optimization," *Future generation computer systems*, vol. 111, pp. 300–323, 2020.

[29] A. P. Piotrowski, J. J. Napiorkowski, and P. M. Rowinski, "How novel is the "novel" black hole optimization approach?" *Information Sciences*, vol. 267, pp. 191–200, 2014.

[30] C. Aranha, C. L. Camacho Villalón, F. Campelo, M. Dorigo, R. Ruiz, M. Sevaux, K. Sörensen, and T. Stützle, "Metaphor-based metaheuristics, a call for action: The elephant in the room," *Swarm Intelligence*, vol. 16, no. 1, pp. 1–6, 2022.

[31] C. L. Camacho-Villalón, M. Dorigo, and T. Stützle, "Exposing the grey wolf, moth-flame, whale, firefly, bat, and antlion algorithms: six misleading optimization techniques inspired by bestial metaphors," *International Transactions in Operational Research*, vol. 30, no. 6, pp. 2945–2971, 2023.

[32] C. L. Camacho Villalón, T. Stützle, and M. Dorigo, "Grey wolf, firefly and bat algorithms: Three widespread algorithms that do not contain any novelty," in *International conference on swarm intelligence*. Springer, 2020, pp. 121–133.

[33] K. Sörensen, "Metaheuristics—the metaphor exposed," *International Transactions in Operational Research*, vol. 22, no. 1, pp. 3–18, 2015.

[34] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in Engineering Software*, vol. 69, pp. 46–61, 2014.

[35] X.-S. Yang and S. Deb, "Cuckoo search via lévy flights," in *2009 World congress on nature & biologically inspired computing (NaBIC)*. IEEE, 2009, pp. 210–214.

[36] C. O'Neil, *Weapons of Math Destruction*. Penguin Books, 2017.

[37] S. D'Agostino, "Turnitin's ai detector: Higher-than-expected false positives," *Inside Higher Ed*, Jun. 2023. [Online]. Available: https://www.insidehighered.com/news/quick-takes/2023/06/01/turnitins-ai-detector-higher-expected-false-positives

[38] M. Abdelhamid, F. Azouaou, and S. Batata, "A survey of plagiarism detection systems: Case of use with English, French and Arabic languages," *arXiv*, 2022, arXiv:2201.03423.