ANNALES MATHEMATICAE ET INFORMATICAE

VOLUME 63. (2026)

INSTITUTE OF MATHEMATICS AND INFORMATICS
ESZTERHÁZY KÁROLY CATHOLIC UNIVERSITY
HUNGARY, EGER

Selected papers of the 13th International Conference on Applied Informatics

Eszterházy Károly Catholic University,
Eger, Hungary, February 19–21, 2026

HU ISSN 1787-6117 (Online)

A kiadásért felelős az
Eszterházy Károly Katolikus Egyetem rektora
Megjelent a Líceum Kiadó gondozásában
Kiadóvezető: Dinnyés Patrik
Műszaki szerkesztő: Dr. Tómács Tibor

Contents

Annales Mathematicae et Informaticae
63 (2026) pp. 1–17
DOI: 10.33039/ami.2026.06.001
URL: https://ami.uni-eszterhazy.hu

Fair team formation for STEAM education using exact and heuristic constraint-based solvers

Ali Adil Ali[a], **Gergely Kovásznai**[b], **Marcell Herbák**[b], **Mustafa Khalid**[c]

[a]University of Debrecen
ali.adil@inf.unideb.hu

[b]Eszterházy Károly Catholic University
kovasznai.gergely@uni-eszterhazy.hu,herbak.marcell@uni-eszterhazy.hu

[c]Imam Jaafar Al-Sadiq University
mustafa.khalid@ijsu.edu.iq

Abstract. STEAM (Science, Technology, Engineering, Arts, and Mathematics) education promotes interdisciplinary learning through collaboration and project-based activities. Constructing fair and balanced student teams under different limitations, due to their skills, gender, and academic training, is a difficult combinatorial issue. In this paper, we model the team formation task as a constraint satisfaction problem and provide a comparative analysis of the Z3 SMT solver, OR-Tools' CP-SAT constraint programming solver, and the heuristic Genetic Algorithm. We carry out experiments on synthetic datasets of varying sizes (50, 100, and 500 students) under identical constraints and evaluation criteria. Findings reveal an obvious trade-off between fairness and computational efficiency. Specifically, the two exact methods (Z3 and CP-SAT) both produce more balanced teams with lower gaps in fairness consistently, as compared to the Genetic Algorithm, which has great variability in solution quality. Based on performance, Z3 has the fastest solving time; however, its overall execution time increases dramatically for larger datasets. CP-SAT, on the other hand, tends to achieve more stable and scalable results across datasets of different sizes. These results aid in establishing a practical framework that supports the selection and implementation of the most effective strategies for creating educational teams.

Keywords: team formation, STEAM education, constraint satisfaction, Sat-

Submitted: March 31, 2026
Accepted: June 30, 2026
Published online: July 17, 2026

isfiability Modulo Theories (SMT), OR-Tools, CP-SAT, Genetic Algorithm, fairness optimization

2020 *Mathematics Subject Classification:* Primary 60F15, 60G42, 60G50; Secondary 60F10

1. Introduction

STEM and STEAM education refers to combinations of technology, engineering, arts, and arithmetic into a unified framework aimed at growing learners' multidisciplinary competencies [1]. With the rapid acceleration of virtual transformation, the call for competencies in those fields is developing exponentially, with projections indicating that STEM jobs will develop at a substantially faster price than other sectors in the course of the period 2021–2031 [15]. In this context, instructional projects including Coding for Kids Iraq [5] play a pivotal role in growing early programming talents and fostering computational questioning among students. In the domain of collaborative STEAM education, the formation of balanced and equitable learning groups plays an important role in efficiently achieving learning goals [10]. Traditional grouping methods, such as the use of randomness, self-selection, or inference based on the instructor's judgment, often do not optimize learning goals [9]. In addition, the use of guided group formation faces significant challenges in dealing with various overlapping constraints. Due to the combinatorial nature of the problem, it is classified as NP-hard, where the search space of the problem tends to increase exponentially along with the size of the dataset, thus calling for the use of advanced computational methods for efficient processing. To achieve this objective, a comparative analysis of three computational models of automated team formation is presented: the Satisfiability Modulo Theories (SMT) solver Z3, the constraint programming solver CP-SAT from Google's OR-Tools, and an evolutionary optimization approach, the Genetic Algorithm (GA). Through the evaluation of these models, the objective is to elucidate the effectiveness of each approach in the context of education. The following research questions are addressed by the paper:

RQ1. How successful is automated optimization in dealing with multi-dimensional constraints to ensure fairness in STEAM training contexts?

RQ2. What are the points of assessment among Constraint Programming (CP-SAT), Genetic Algorithms (GA), and SMT (Z3) in terms of adherence to constraints and efficiency?

RQ3. To what extent are the overall performance and scalability of the solving approaches (Z3, CP-SAT, and GA) affected by the growth in dataset size?

2. Related work

The problem of team formation has been studied in educational and organizational settings, and its goal has been to form balanced and effective teams based on dif-

ferent criteria such as skills, diversity, and collaboration, among others. Due to the satisfaction of different constraints, team formation problems are complex and can be categorized as NP-hard problems. To cope with computation complexity, different approaches have been proposed, such as metaheuristic, exact optimization, and hybrid models, among others. For instance, metaheuristic approaches such as evolutionary and genetic algorithms have been effectively used for team formation problems due to their ability to efficiently search through large solution spaces. For example, in [3], a team formation problem using an evolutionary metaheuristic approach has been proposed, considering different criteria such as team size, compulsory and forbidden combinations, among others, in a classroom setting. Similarly, in [12], a multi-objective genetic algorithm has been proposed for collaborative learning, considering different attributes such as knowledge, communication, and leadership, among others, for forming effective groups among students. The proposed approaches in these studies are highly scalable and flexible, and they do not guarantee optimal results, which might affect reproducibility depending on parameter tuning.

Recently, there have been some studies that investigate the use of computational algorithms in real classroom settings. For instance, [14] discusses various team formation algorithms and highlights the significance of choosing suitable algorithms depending upon the educational setting. Nevertheless, there has been a lack of thorough comparison between fundamentally diverse paradigms, especially between heuristic and exact optimization algorithms. To overcome the limitations associated with heuristic algorithms, exact optimization algorithms have been proposed. In [2], an SMT-based framework using the Z3 solver has been proposed to form fair teams considering various constraints, including team size, skill thresholds, gender, and diversity of backgrounds. The results show that SMT is capable of efficiently handling complex constraints; however, only small-scale problems have been considered in this study. Mathematical optimization techniques are also being implemented in team formation problems. In [4], the authors presented an integer linear programming model in forming teams of students while satisfying certain constraints like the size of the teams and pairing constraints. This model proves the achievement of optimal results for small and medium-sized problem sets. However, the problem's size affects the computational complexity of the problem.

More recent work has been done on fairness and diversity in team formation. In [8], an algorithm for diversity-aware team formation is proposed, where priority-based optimization and hill-climbing methods are utilized. In this work, intra-heterogeneity and inter-homogeneity metrics are proposed for evaluating the quality of formed teams. Despite its competitive performance, this method is based on heuristics and lacks comparison with other optimization approaches. On the whole, existing research works demonstrate the trade-off between the quality of the solution and the efficiency of the computation, as summarized in Table 1.

Nevertheless, there is a lack of unified experimental approaches to compare various solving paradigms under the same conditions, especially with regard to the aspects of fairness and scalability. This study aims to address the aforementioned

problem and present a comparative analysis of the SMT approach (Z3), constraint programming (CP-SAT), and genetic algorithms.

Table 1. Comparison of related work in team formation.

Approach	**Advantages**	**Limitations**
Evolutionary Metaheuristic [3]	Supports flexible team sizes; handles complex constraints (e.g., compulsory and forbidden conditions); demonstrates good scalability	Does not guarantee optimality; lacks comparison with exact optimization methods; fairness is not explicitly modeled
Genetic Algorithm (Multi-objective) [12]	Considers multiple student characteristics; validated through real-world classroom experiments; improves collaborative learning outcomes	Heuristic approach with no guarantee of optimality; no comparison with exact solvers; fairness is not explicitly quantified
SMT Solving (Z3) [2]	Guarantees strict constraint satisfaction; effective for fair team formation; fast on small datasets	Limited scalability evaluation; tested on small dataset; no comparison with other solvers
Integer Linear Programming [4]	Finds optimal solutions; supports rich constraints (pairing, size, composition); evaluated with multiple solvers	Scalability issues for large datasets; exponential growth in runtime; limited to ILP paradigm
Diversity-Aware Heuristic (Hill Climbing) [8]	Considers diversity and fairness; introduces evaluation metrics; tested on real and synthetic data; supports multiple constraints	Heuristic approach without optimality guarantee; lacks comparison with SMT and CP-based methods
Comparative Algorithm Study (Classroom-based) [14]	Multiple algorithms are evaluated in the classroom, offering practical validation, along with the differences in the quality of the solutions obtained	Theoretical analysis is limited, along with the lack of comparison between the exact methods and the constraint-based methods, and the scalability analysis was not deeply evaluated

2.1. Research gap and motivation

Despite the above-mentioned articles, there are many issues in the field of team building. Metaheuristic approaches show great scalability and flexibility; however, they do not ensure optimal solutions and, frequently, do not provide formal constraint satisfaction.

SMT guarantees strict constraint satisfaction, and so does integer programming, which also guarantees optimality but has scalability issues on large datasets. Moreover, the bulk of prior research on such issues as preference, skill, or diversity addresses certain factors alone, rather than integrating multiple factors in the context of unified criteria. Recent research has drawn attention to fairness and diversity in the form of teams, but there is a lack of standardized rubrics for evaluating and comparing team building, and so it is uncertain whether a variety of methodological methods are equally effective in different settings. Moreover, we find that there seems to be a lack of comparing fundamentally diverse solving paradigms, specifically exact methods and metaheuristic methods against common constraints and criteria. In light of these issues, this paper conducts a joint study of SMT (Z3), Constraint Programming (CP-SAT), and Genetic Algorithms for team formation in STEAM education. The presented approach assesses these techniques through

common constraints which refer to skill thresholds, gender balance, and background diversity, and analyses their performance for scalability, execution time, and solution quality. The model, therefore, allows for a more pragmatic understanding of the trade-offs between optimality and scalability for suitable methods in real-world education settings.

3. Dataset and constraints

To experiment with multiple team formation approaches, a synthetic dataset [6, 7] was used, which was created to represent the characteristics of real-life students in order to simulate scenarios with a focus on STEAM-targeted activities and to represent realistic student demographics. This dataset contains representative content features in relation to gender, education status, and skill level across the main STEAM disciplines. It was rigorously designed to capture diversity; it made sure that any constraints were possible.

3.1. Dataset description

The dataset in Table 2 contains 50, 100, and 500 records of students. Each record is for a student possessing several skill attributes. The dataset can be used to construct teams that have people of different genders, backgrounds, and talents. Using this information, one can utilize exact and heuristic approaches to construct balanced teams that meet certain skill and fairness standards.

Table 2. Dataset attributes.

ID	Unique numeric identifier
Name	Student's first name (anonymized)
Gender	Female (F), Male (M)
Background	Arts, Engineering, Science
Skills	Rated from 1 (lowest) to 5 (highest): • Programming (technical ability) • Design (visual creativity) • Math (quantitative skills) • Creativity (original thinking) • Science (theoretical knowledge)

3.2. Fairness constraints

To make sure teams are fair, we defined a set of constraints that the solver must follow. These constraints help to create balanced and diverse teams.

- **Team Size:** Each team must have exactly 5 students.
- **Minimum Skill Totals:**
  - Programming ≥ 10
  - Design ≥ 10
  - Math ≥ 8

  - Creativity ≥ 6
  - Science ≥ 8

  These totals are the sum of individual student scores in each team. For each threshold, it was decided how many minimum skills should be considered in order to realistically represent the minimum competencies required from multidisciplinary teams working in STEAM. Considering the fact that the skills will be evaluated from 1 to 5 points, and each team consists of five members, the selected minimum levels of skills ensure an adequate base competence of the team as a whole while being feasible solutions.
- **Gender Balance:** Each team must include at least 2 males and at least 2 females.
- **Academic Background Diversity:** Each team must include at least one student from each of the 3 backgrounds (Arts, Engineering, Science).

These constraints help ensure a fair distribution of skills and expertise across teams, thereby supporting effective collaboration and promoting diversity in STEAM contexts.

4. Problem formulation

The team formation problem is formulated as a constrained optimization problem, where the objective is to assign students to teams under a set of predefined constraints. Given a set of $n > 0$ students, the goal is to partition them into T teams such that each team meets all requirements, such all students are assigned to exactly one team, and the teams satisfy fairness and diversity constraints. Moreover, a fairness metric is proposed to measure the balance of the teams in the distribution of total skill set. The above definition makes the assumption that the total number of students can be divided evenly among the specified size of teams, so that there are no students left out. The case where there are leftover students or different sizes of teams is to be considered in the future.

4.1. Decision variables

We define a binary decision variable:

$$x_{i,t} = \begin{cases} 1, & \text{if student } i \text{ is assigned to team } t, \\ 0, & \text{otherwise,} \end{cases}$$

where $i \in \{1, \ldots, n\}$ and $t \in \{1, \ldots, T\}$. This variable determines the assignment of students to teams.

4.2. Team size constraint

Each team must consist of a specific, fixed number of students, denoted by the symbol k, to ensure an even distribution among the teams:

$$\sum_{i=1}^{n} x_{i,t} = k \quad \forall t.$$

4.3. Student assignment constraint

Each student must be assigned to exactly one team:

$$\sum_{t=1}^{T} x_{i,t} = 1 \quad \forall i.$$

This guarantees that no student is assigned to multiple teams or left unassigned.

4.4. Skill constraints

Each team must meet minimum skill requirements for different skill categories such as Programming, Design, Math, Creativity, and Science. Let the binary decision variables skill_i^s represent the skill value of student i for skill s.

$$\sum_{i=1}^{n} x_{i,t} \cdot \text{skill}_i^s \geq \text{MinSkill}_s \quad \forall t, \forall s.$$

This ensures that each team has sufficient capability in all required skill areas.

4.5. Gender constraints

To ensure gender diversity, each team must include a minimum number of students from each gender category, where F and M represent female and male students, respectively.

$$\sum_{i \in F} x_{i,t} \geq F_{\min}, \quad \sum_{i \in M} x_{i,t} \geq M_{\min} \quad \forall t.$$

This constraint promotes balanced gender representation within each team.

4.6. Fairness gap

To evaluate the skill-based balance between teams, we define a fairness metric based on the total skill score of each team.

Let the total skill of team t be:

$$\text{Score}_t = \sum_{i=1}^{n} \sum_{s} x_{i,t} \cdot \text{skill}_i^s$$

The fairness gap is defined as:

$$\text{Gap} = \max_t(\text{Score}_t) - \min_t(\text{Score}_t).$$

A smaller gap means that the teams are more evenly matched, while a bigger gap means that the teams are not that balanced. This measure is used to see how well different solvers perform in assembling fair teams.

5. Experimental setup

The experimental setup, datasets, solver configurations, and evaluation process utilized to gauge the effectiveness of the suggested multi-solver strategy are all covered in this section.

5.1. Datasets

In the experiments to check for scalability and speed, three datasets of different sizes were used:

1. Dataset A of 50 students,
2. Dataset B of 100 students, and
3. Dataset C of 500 students.

Each dataset comprises records of students featuring various attributes such as skill scores (including Programming, Design, Math, Creativity, and Science), gender, and educational background, as explained in Section 3.1.

5.2. System workflow

The team formation problem from Section 4 is solved by three tools: an exact SMT solver called Z3, a constraint programming solver from OR-Tools called CP-SAT, and a heuristic optimization method known as the Genetic Algorithm. Figure 1 illustrates the whole workflow of the system. Z3, CP-SAT, and the Genetic Algorithm access the dataset after loading. Team assignments are generated individually by each solver and combined for further analysis. The team-specific information includes team members' composition, skill levels in general, gender distribution, and background diversity. In addition, the results are exported in Excel format for further analysis.

To make it possible for a fair comparison to be made, all the solvers were set up under consistent parameters and timeouts of 100 seconds. The time allocated to all the solvers was used uniformly so that an analysis could be conducted.

Z3. The SMT solver Z3 checks to see if a set of logical constraints can be met [13]. In this paper, Z3 is used to show the team formation problem as a problem of satisfying constraints. Using Boolean decision variables, all of the constraints

are stored as QF_LIA logical formulas. Z3 ensures fairness if there is a satisfying assignment; otherwise, it returns UNSAT.

CP-SAT. OR-Tools' CP-SAT solver is a constraint programming solver intended for extensive combinatorial optimization problems [11]. This methodology makes use of binary decision variables to represent the assignment of students to a specific team. The constraints are formulated using linear integer equations, and the solver makes use of advanced techniques like presolve, constraint propagation, and parallel search. It is configured with multi-threading (parallel workers) and a time limit.

Genetic algorithm. The Genetic Algorithm (GA), as a heuristic optimization method, is inspired by the evolution principles of nature [16]. Solutions are encoded as chromosomes, representing team assignments, and they evolve over a series of generations by selection, crossover, and mutation operations. It is configured with population size, number of generations, and mutation/crossover probabilities.

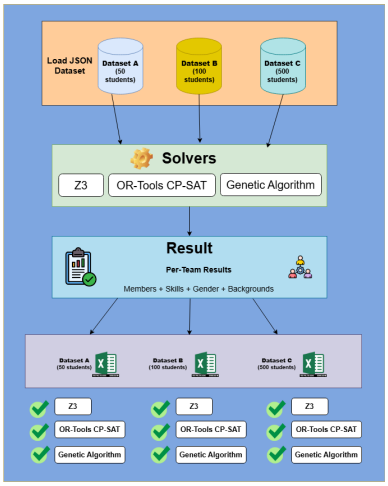


Figure 1. Integrated framework for multi-solver evaluation and comparison.

5.3. Evaluation metrics

The performance of the solvers was evaluated using the following metrics:

- **Solve Time:** Time required by the solver to find a solution.
- **Total Time:** Overall execution time including processing and extraction.
- **Fairness (MIN, MAX, GAP):** The minimum and the maximum team scores, and the difference between those two.

- **Average Skill Score:** Mean skill values across all teams.

This ensures that there is a comprehensive assessment of the efficiency of the computation and the quality of the solutions obtained using various solvers and datasets of different sizes. In Section 6, these measures are presented and analyzed on the results of the experiments. They include the fairness tests and comparisons.

6. Results

The evaluation of the experiments highlights both the operational efficiency and the fairness of the assembled teams. Extensive numerical data is available in Tables 3 and 4.

6.1. Runtime analysis

The runtime performance of the three solvers on the datasets of 50, 100, and 500 students is shown in Table 3, respectively. All solvers showed quick execution times on the smallest dataset of 50 students. Z3 took the lead, followed by CP-SAT and, finally, the Genetic Algorithm. Z3 and CP-SAT maintained their efficiency with a low solve time for the dataset of 100 students. Due to iterative optimization strategy, the Genetic Algorithm's runtime significantly increased by two orders of magnitude. Scalability issues were more noticeable with the largest dataset of 500 students. While Z3 demonstrated strong scalability, CP-SAT and the Genetic Algorithm had long solve time. Overall, Z3 consistently produced the fastest solve times for datasets of all sizes; however, as datasets grew in size, its overall execution time considerably deteriorated. In contrast, the Genetic Algorithm resulted in greater computational costs, due to a methodology in which multiple potential solutions are evaluated across several generations, providing the freedom to explore the solution space but requiring the longest time to compute. CP-SAT demonstrated more consistent and scalable performance in terms of overall runtime.

Table 3. Detailed solver runtime results for synthetic datasets of different sizes.

Size	Solver	Status	Solve (s)	Total (s)
50	Z3	Solved	0.04	1.54
50	CP-SAT	Solved	0.09	0.12
50	Genetic Algorithm	Solved	0.12	0.12
100	Z3	Solved	0.11	5.06
100	CP-SAT	Solved	0.28	0.42
100	Genetic Algorithm	Solved	12.62	12.62
500	Z3	Solved	2.99	123.23
500	CP-SAT	Solved	24.86	27.92
500	Genetic Algorithm	Solved	59.50	59.50

6.2. Fairness analysis

The evaluation of fairness using MIN, MAX, and GAP metrics across different datasets and solvers is shown in Table 4. All solvers produced comparatively equal teams on the dataset of 50 students. The Genetic Algorithm showed a somewhat higher GAP of 9, indicating a modest decline in balance, while Z3 and CP-SAT both recorded a GAP of 7. GAP increased for all solvers on the dataset of 100 students. While Z3 (18) and the Genetic Algorithm (19) produced similar results in terms of the GAP, the team balance was declining in the case of CP-SAT (22). With a GAP of 11, Z3 produced the best fairness results on the largest dataset of 500 students. CP-SAT came in second with a GAP of 12 and the Genetic Algorithm had a GAP of 14. In conclusion, Z3 consistently produced more balanced teams for datasets of all sizes. The reason for this is that Z3 strictly adheres to the constraints by ensuring all conditions are met without trade-offs. On the other hand, the Genetic Algorithm provided competitive fairness but showed less overall stability, because it is a heuristic algorithm and does not ensure an optimal solution. CP-SAT showed somewhat higher variability in the fairness outcomes.

Table 4. Minimum and maximum total STEAM skill scores and fairness gap across datasets.

Dataset Size	Solver	MIN	MAX	GAP
50	Z3	76	83	7
50	CP-SAT	75	82	7
50	Genetic Algorithm	74	83	9
100	Z3	72	90	18
100	CP-SAT	69	91	22
100	Genetic Algorithm	71	90	19
500	Z3	74	85	11
500	CP-SAT	75	87	12
500	Genetic Algorithm	70	84	14

6.3. Scalability considerations

This difference in scalability became more apparent as the size of the datasets increased. Even with a dataset size of 500 students, Z3 exhibited significant scalability. This further suggests that the optimizations made in the solver and the way the constraints are being implemented are well-suited for the application. It is vital to note that solvers such as Z3 may experience scalability issues with increased problem size due to the exponential size of the search space. On the other hand, the Genetic Algorithm presented itself as a viable solution for large-scale problems where an approximate solution is acceptable, despite the lower operational speed associated with increased dataset sizes. CP-SAT also presented itself as a viable solution through the significant scalability and solution quality. Memory consumption was not explicitly measured in this study. A detailed comparison of memory requirements among SMT, CP-SAT, and Genetic Algorithm approaches remains an interesting direction for future research.

6.4. Practical implications

The selection of the solver is dependent on the particular needs of the application. Z3 is the best choice in situations with is a need for stringent fairness and adherence to set limits. In situations where data is large in volume and resources limited in processing capacity, the Genetic Algorithm offers an alternative. At the same time, CP-SAT strikes an optimal balance between efficiency and scalability. The data shows that no particular solver can be considered better than another. Each method has unique advantages depending on the size of the problem and solution accuracy desired.

7. Conclusion and future work

In summary, the research presented in the paper is based on the multi-solution approach, exact (Z3, CP-SAT) and heuristic (Genetic Algorithm), for the team formation problem. The problem has been formulated as a constraint satisfaction problem with the set of criteria, such as team size, skills, gender, and diversity in terms of academic background. The experimental results for the exact approaches show promising results with the balance between the quality of the solutions and the efficiency of the computation compared to the heuristic approach. Z3 demonstrated promising results with the ability to achieve the best solution with the shortest solve time, with low GAP values. On the contrary, CP-SAT demonstrated the ability to achieve the balance between fairness and the efficiency of computation. The Genetic Algorithm approach also demonstrated the ability to achieve the balance, but with higher computation time. The research also demonstrated the need for choosing the most suitable approach based on the scale of the problem to solve.

There exist several possible directions for further research. The most promising future direction is the development of a hybrid framework that combines exact optimization methods with heuristic search techniques, aiming to achieve both high-quality solutions and strong scalability for large-scale team formation problems. The application of the real-life characteristics, such as the preferences of students, the personality, or the changing team, will also enhance the practical application of the model.

Data availability

The datasets generated and used during this study are available from the corresponding author, Ali Adil Ali (ali.adil@inf.unideb.hu), upon reasonable request.

The synthetic datasets used in this study are publicly available at `https://github.com/AliProgrammer85/Three-Solver-Team-Formation`. The repository contains the datasets with 50, 100, and 500 students that were used in the experimental evaluation of the proposed team formation approaches.

References

[1] N. O. Abdulwahid, S. Fakhfakh, I. Amous: *Simulating and Predicting Students' Academic Performance Using a New Approach based on STEAM Education*, JUCS - Journal of Universal Computer Science 28.12 (2022), pp. 1252–1281, ISSN: 0948-695X, DOI: 10.3897/jucs.86340.

[2] A. A. Adil, G. Kovásznai, M. Khalid: *Automated fair team formation in STEAM activities using Satisfiability Modulo Theories (SMT)*, Annales Mathematicae et Informaticae 61 (2025), pp. 1–14, DOI: 10.33039/ami.2025.10.001, URL: https://ami.uni-eszterhazy.hu.

[3] G. Candel, V. Sánchez-Anguix, J. M. Alberola, V. Julián, V. Botti: *An evolutionary metaheuristic for forming teams in the classroom with constraints*, Neurocomputing 638 (2025), p. 130138, ISSN: 0925-2312, DOI: 10.1016/j.neucom.2025.130138, URL: https://www.sciencedirect.com/science/article/pii/S0925231225008100.

[4] G. Candel, V. Sánchez-Anguix, J. M. Alberola, V. Julián, V. Botti: *An Integer Linear Programming Model for Team Formation in the Classroom with Constraints*, in: Hybrid Artificial Intelligent Systems (HAIS), vol. 14001, Lecture Notes in Artificial Intelligence (LNAI), Springer, 2023, pp. 397–408, DOI: 10.1007/978-3-031-40725-3_34.

[5] Coding for Kids Iraq: *Coding for Kids Iraq Initiative*, Accessed: 2026-03-20, 2024, URL: https://www.coding4kidsiraq.com/.

[6] M. Dorodchi, E. Al-Hossami, A. Benedict, E. Demeter: *Using Synthetic Data Generators to Promote Open Science in Higher Education Learning Analytics*, in: 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 4672–4675, DOI: 10.1109/BigData47090.2019.9006475.

[7] B. Flanagan, R. Majumdar, H. Ogata: *Fine Grain Synthetic Educational Data: Challenges and Limitations of Collaborative Learning Analytics*, IEEE Access 10 (2022), pp. 26230–26241, DOI: 10.1109/ACCESS.2022.3156073.

[8] B. Hui, O. Adeyemi, K. Phan, J. Schoenit, S. Akins, K. Khademi: *Diversity Considerations in Team Formation Design, Algorithm, and Measurement*, in: Proceedings of the 15th International Learning Analytics and Knowledge Conference (LAK), ACM, 2025, pp. 1–11, DOI: 10.1145/3706468.3706473.

[9] A. Jenkins, E. Rainbow, I. Jaimoukha, F. S. Ng, L. Stankovic, M. Dakovič, D. Mandic, D. Mandic: *Fair and Skill-Diverse Student Group Formation: A graph-theoretic approach*, IEEE Signal Processing Magazine 43.1 (2026), Cited by: 1, pp. 83–97, DOI: 10.1109/MSP.2025.3627096, URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-105030855588&doi=10.1109%2fMSP.2025.3627096&partnerID=40&md5=fc13d3c31d02887705b7673b6dc40319.

[10] K. Krasnić, P. Žigman, T. Babić: *Team Building Elements and Their Importance According to STEAM Students*, in: 2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO), 2022, pp. 739–744, DOI: 10.23919/MIPRO55190.2022.9803519.

[11] B. Lennartson: *Optimization of Timed Petri Nets using CP-SAT*, IFAC-PapersOnLine 58.1 (2024), 17th IFAC Workshop on discrete Event Systems WODES 2024, pp. 90–95, ISSN: 2405-8963, DOI: 10.1016/j.ifacol.2024.07.016, URL: https://www.sciencedirect.com/science/article/pii/S2405896324000946.

[12] J. Moreno, D. A. Ovalle, R. M. Vicari: *A genetic algorithm approach for group formation in collaborative learning considering multiple student characteristics*, Computers & Education 58.1 (2012), pp. 560–569, ISSN: 0360-1315, DOI: 10.1016/j.compedu.2011.09.011, URL: https://www.sciencedirect.com/science/article/pii/S0360131511002284.

[13] L. de Moura, N. Bjørner: *Z3: An Efficient SMT Solver*, in: Tools and Algorithms for the Construction and Analysis of Systems, ed. by C. R. Ramakrishnan, J. Rehof, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340.

[14] S. Peuker, A. Hill: *A Novel Approach to Purposeful Team Formation*, in: 2024 ASEE Annual Conference and Exposition, 2024, doi: 10.18260/1-2--46466.

[15] U.S. Department of Labor: *STEM Day: Explore Growing Careers*, Accessed: 2026-03-20, 2022, url: https://blog.dol.gov/2022/11/04/stem-day-explore-growing-careers.

[16] C. Zhamri, Z. Che Ani, A. Yasin, M. Z. Husin, Z. Hamid: *A Method for Group Formation Using Genetic Algorithm*, International Journal on Computer Science and Engineering 02 (2010), pp. 3060–3064.

A. Algorithmic details

Z3 solver

When using the Z3 solver, the team formation problem is modelled as a set of logical constraints and searches for a solution that satisfies all requirements exactly. It guarantees correctness if a solution exists.

Algorithm 1 Z3-based team formation

Require: Students S, Requirements R
Ensure: Team assignments or UNSAT

```
Initialize solver
Define binary variables x_{i,t}
Add constraint: each student assigned to one team
Add constraint: each team has fixed size
for each team t do
    Apply skill constraints
    Apply gender constraints
    Apply background constraints
end for
Run solver
if solution exists then
    Extract team assignments
else
    Return UNSAT or TIMEOUT
end if
```

OR-Tools' CP-SAT solver

The CP-SAT solver formulates the problem using integer constraints and applies efficient search and propagation techniques to find feasible solutions with good scalability.

Algorithm 2 OR-Tools CP-SAT team formation

Require: Students S, Requirements R
Ensure: Team assignments or UNSAT
Initialize CP-SAT model
Define binary variables $x_{i,t}$
Add constraint: each student assigned to one team
Add constraint: each team has fixed size
for each team t **do**
    Apply skill constraints
    Apply gender constraints
    Apply background constraints
end for
Solve model with time limit
if solution found **then**
    Extract team assignments
else
    Return UNSAT or TIMEOUT
end if

Genetic algorithm

The Genetic Algorithm is a heuristic approach that iteratively improves solutions using evolutionary operators such as selection, crossover, and mutation. It is suitable for large-scale problems where approximate solutions are acceptable.

Algorithm 3 Genetic algorithm for team formation

Require: Students S, Requirements R
Ensure: Best team assignment
Initialize population P
Evaluate fitness of each solution
while not termination condition **do**
    Select parents
    Apply crossover
    Apply mutation
    Evaluate new solutions
    Update population
end while
Return best solution found

B. Example team formation

This section presents typical team configurations generated by the three solvers (Z3, CP-SAT, and GA) using a dataset of 50 students in order to illustrate the

utility of the proposed framework.

Each constructed team meets the established criteria and limitations. Each team has five students representing the different areas of STEAM. All the teams are composed of students of diverse backgrounds and genders. In the results, it is indicated that all the strategies can yield feasible solutions. However, there is a difference in the distribution of the overall skill scores of the teams. Table 5, Table 6, and Table 7 show the exemplary team assignments generated by each solver.

Table 5. A fragment of the team formation generated by Z3 (for 50 students).

Team	Members	F	M	Total Skills
Team1	Kara, Mia, Zane, Max, Nora	3	2	76
Team2	Jack, Quinn, Ryan, Uma, Yusuf	2	3	79
Team3	Grace, Victor, Wendy, Jake, Umar	2	3	81
Team4	Ivy, Noah, Xavier, Elena, Xena	3	2	81
Team5	Yara, Gina, Iris, Quincy, Sam	3	2	78

Table 6. A fragment of the team formation generated by CP-SAT (for 50 students).

Team	Members	F	M	Total Skills
Team1	Eve, Ivy, Owen, Tara, Yusuf	3	2	80
Team2	Jack, Noah, Gina, Iris, Quincy	2	3	79
Team3	Bob, Paul, Ryan, Elena, Luna	2	3	81
Team4	Alice, Charlie, Grace, Finn, Vera	3	2	79
Team5	Diana, Olivia, Yara, Zane, Max	3	2	79

Table 7. A fragment of the team formation generated by GA (for 50 students).

Team	Members	F	M	Total Skills
Team1	Tara, Riley, Ryan, Ben, Sam	2	3	82
Team2	Vera, Olivia, Max, Hank, Tom	2	3	77
Team3	Clara, Will, Henry, Nora, Jack	2	3	79
Team4	Xavier, Quincy, Grace, Noah, Uma	2	3	83
Team5	Xena, Elena, Piper, Frank, Jake	3	2	79

The results show that the exact (Z3 and CP-SAT) and heuristics solvers (GA) are able to satisfy the necessary constraints, such as the balance of genders and

diversity in terms of academics; however, the distribution of the skills among the teams is not uniform, with more consistency observed in the results of Z3 and OR-Tools than in the genetic algorithm.

Annales Mathematicae et Informaticae
63 (2026) pp. 18–31
DOI: 10.33039/ami.2026.06.003
URL: https://ami.uni-eszterhazy.hu

Adaptive sentiment evaluation in social media analysis

Anikó Apró[a b], **Levente Sasi**[c]

[a]University of Debrecen, Doctoral School of Informatics,
apro.aniko@inf.unideb.hu

[b]Eszterházy Károly Catholic University
apro.aniko@uni-eszterhazy.hu

[c]Eszterházy Károly Catholic University
sasi.levente@gmail.com

Abstract. Social media sentiment analysis faces a persistent aggregation problem: lexicon-based and transformer-based models often produce inconsistent outputs for the same short, informal, and stylistically heterogeneous texts. This paper introduces ADRTW (Adaptive Dynamic Reliability-Triggered Weighting), an interpretable sentiment fusion framework that combines heterogeneous sentiment estimators using rule-guided reliability weights derived from textual cues, inter-model disagreement, and consistency patterns [5, 8].

The framework is evaluated on a Reddit dataset containing 1,577 posts, 354,050 comments, and 187,666 authors collected between 2017 and 2025, together with a controlled synthetic benchmark for aggregation comparison. The results show that ADRTW remains competitive with static averaging in controlled settings while preserving context-sensitive local variation in large-scale discourse analysis.

Beyond sentiment fusion, the ADRTW-derived signal supports complementary analyses of online discussions, including temporal trend inspection, toxicity-aware interpretation, and participation-based clustering. Overall, the proposed framework provides a transparent and reusable basis for examining emotional dynamics in social media discourse.

Keywords: sentiment analysis, social media, Reddit, adaptive weighting, toxicity detection, clustering

Submitted: March 31, 2026
Accepted: June 30, 2026
Published online: July 17, 2026

1. Introduction

The rapid growth of social media has resulted in an unprecedented volume of user-generated text. Platforms such as Reddit provide rich sources of opinions, discussions, and emotional expressions across diverse topics. Sentiment analysis has become a fundamental tool for extracting meaningful insights from such data [11, 18].

However, existing sentiment analysis approaches often rely on individual models, such as lexicon-based or deep learning-based methods. These models exhibit varying performance depending on linguistic context, domain, and textual characteristics. As a result, their outputs may differ significantly when applied to the same input. Benchmark studies show that model choice strongly affects sentiment estimation quality, especially in short, informal texts [11, 13, 18].

This inconsistency challenges reliable sentiment interpretation, especially in dynamic environments such as Reddit discussions.

To address this limitation, this paper proposes an adaptive sentiment evaluation framework called ADRTW (Adaptive Dynamic Reliability-Triggered Weighting). The method dynamically combines multiple sentiment models by assigning context-dependent reliability weights.

The main contribution of this paper is not a new sentiment classifier, but an interpretable adaptive fusion mechanism that reweights heterogeneous sentiment signals according to contextual reliability cues. In contrast to static averaging, ADRTW adjusts model influence using disagreement patterns, surface-level text characteristics, and inter-model consistency cues. This makes the final sentiment signal more suitable for discourse-level analysis in heterogeneous social media environments.

The contributions of this paper are as follows:

- We propose ADRTW, an interpretable adaptive late-fusion mechanism for combining lexicon-based and transformer-based sentiment estimators in short social media texts.
- We provide an explicit rule-guided weighting formulation based on contextual text features, inter-model agreement, and consistency penalties.
- We evaluate ADRTW against individual base models and static aggregation on a controlled benchmark, and we examine its large-scale behavior on Reddit through discourse-level and alignment-based analyses.
- We show that ADRTW-derived sentiment can also serve as a useful discourse-level signal in complementary analyses such as temporal trend inspection, toxicity-aware interpretation, and participation clustering.

Thus, ADRTW is positioned as a transparent late-fusion strategy rather than as a replacement for existing sentiment classifiers, adaptively reweighting heterogeneous sentiment signals using textual and inter-model consistency cues.

2. Related work

Sentiment analysis methods can be broadly categorized into lexicon-based and machine learning-based approaches [11, 18]. Lexicon-based methods, such as VADER and TextBlob, rely on predefined sentiment dictionaries and rule-based heuristics [2, 6]. These approaches are computationally efficient but often struggle with context-dependent language, sarcasm, and domain-specific expressions.

In contrast, transformer-based models such as RoBERTa and related BERT-style architectures provide context-aware representations and often achieve stronger performance in complex linguistic scenarios. Related Hungarian NLP results also suggest that monolingual transformer models may outperform multilingual alternatives, although such models remain computationally expensive and may still exhibit inconsistencies across domains [3, 9, 16].

Several studies have explored ensemble approaches that combine multiple sentiment models [8, 12]. While these methods improve robustness, they typically rely on static weighting schemes, which do not adapt to the characteristics of individual texts.

The present work addresses this limitation by introducing a dynamic, context-sensitive weighting mechanism for fusing heterogeneous sentiment signals.

Recent work has also emphasized that social media analysis should not be reduced to polarity estimation alone. Toxicity detection, discourse quality assessment, and behavior-aware clustering provide complementary perspectives for understanding online communities, especially in environments where emotional intensity, disagreement, and participation patterns interact [10, 15, 17].

Taken together, these studies suggest that the main unresolved challenge is not merely sentiment classification accuracy, but the context-sensitive fusion of heterogeneous sentiment signals in socially complex environments. This is the specific problem addressed by ADRTW in the present work.

Although prior studies have explored sentiment ensembles, most of them optimize predictive robustness rather than interpretable context-sensitive fusion at the individual-text level. The present work addresses this gap by proposing a rule-guided adaptive weighting mechanism designed for heterogeneous social media discourse.

3. Dataset and data collection

Data were collected through the official Reddit API (PRAW) from multiple topic-oriented communities, including *love*, *politics*, *gaming*, *war*, *technology*, *nature*, *science*, *health*, *education*, *climate*, *family*, and *religion*. The resulting dataset consists of 1,577 posts, 354,050 comments, and 187,666 authors collected between 2017 and 2025. Each record includes textual content, metadata, and relational links between posts and comments.

Preprocessing included text normalization, noise removal, and filtering of non-English content. To reduce topical bias, the selected subreddits were intention-

ally balanced to include both emotionally positive (e.g., *love*, *family*) and conflict-oriented contexts (e.g., *war*, *politics*).

Data collection was performed using multiple listing strategies (*hot*, *rising*, *new*, and *top*) to capture diverse engagement patterns. Duplicate handling was implemented based on post identifiers and comment-level matching (post ID, text content, timestamp). Deleted and empty content were excluded, and non-English texts were filtered using rule-based language detection.

The data were stored in a structured SQLite database preserving post–comment relationships and author-level aggregation statistics. This sampling strategy may introduce topical and community-specific biases, which should be considered when interpreting the results.

Only publicly available Reddit content was processed, and the analysis was conducted at aggregate level without attempting to identify individual users.

4. Methodology

The sentiment analysis pipeline includes data preprocessing, model evaluation, and aggregation.

Three sentiment analysis models were applied to each text: VADER, TextBlob, and a RoBERTa-based sentiment model fine-tuned for social media text.

VADER and TextBlob represent lexicon-based sentiment estimation, while the transformer-based component relies on a RoBERTa-style contextual language model [2, 3, 6, 9].

In addition to simple averaging and fixed-weight aggregation, we examine the proposed ADRTW framework, which dynamically adjusts model weights based on reliability indicators.

4.1. Operational definition of ADRTW

Let x denote an input text and let $s_i(x) \in [-1, 1]$ be the sentiment score produced by model $i \in \{1, \dots, N\}$. In this work, $N = 3$ and the models are VADER, TextBlob, and a RoBERTa-based transformer model. The final ADRTW score is defined as

$$S_{\mathrm{ADRTW}}(x) = \sum_{i=1}^{N} w_i(x) s_i(x), \qquad \sum_{i=1}^{N} w_i(x) = 1, \quad w_i(x) \geq 0.$$

The weight computation consists of three steps: context-sensitive base weighting, consistency correction, and normalization [12].

Step 1: Context-sensitive base weighting. For each text, we extract a small set of surface features:

$$L(x) = |x|, \quad d_{\mathrm{emoji}}(x) = \frac{n_{\mathrm{emoji}}(x)}{\max(|x|, 1)}, \quad n_{\mathrm{sent}}(x) = \text{sentence-fragment count}.$$

Each model receives a prior π_i and a compatibility score $g_i(x)$:

$$w_i^{(\text{base})}(x) = \pi_i \cdot g_i(x).$$

The compatibility score is defined heuristically:

$$g_i(x) = \begin{cases} 1 + \lambda_{\text{short}} & \text{for lexicon models if } L(x) < \tau_L, \\ 1 + \lambda_{\text{emoji}} & \text{for VADER if } d_{\text{emoji}}(x) > \tau_E, \\ 1 + \lambda_{\text{long}} & \text{for RoBERTa if } L(x) \geq \tau_L \text{ and } n_{\text{sent}}(x) \geq \tau_S, \\ 1 & \text{otherwise.} \end{cases}$$

In practice, short and emoji-rich texts favor lexicon-based models, while longer and structurally richer texts increase the influence of the transformer-based model.

Step 2: Consistency correction. Let

$$m(x) = \text{median}\{s_1(x), \ldots, s_N(x)\}$$

and

$$\delta_i(x) = |s_i(x) - m(x)|.$$

We define the global disagreement level as

$$D(x) = \frac{1}{N} \sum_{i=1}^{N} \delta_i(x).$$

The adaptive sensitivity parameter is

$$\alpha_{\text{eff}}(x) = \alpha_0 + \alpha_1 D(x),$$

where $\alpha_0 > 0$ and $\alpha_1 \geq 0$ control the base and disagreement-dependent penalty strength.

The consistency term is then

$$w_i^{(\text{cons})}(x) = \exp\big(-\alpha_{\text{eff}}(x)\delta_i(x)\big).$$

If $\delta_i(x) > \tau_{\text{out}}$, an additional outlier penalty $\rho_{\text{out}} \in (0, 1)$ is applied. If the polarity sign of $s_i(x)$ contradicts a sufficiently strong consensus ($|m(x)| > \tau_{\text{sign}}$), a contradiction penalty $\rho_{\text{sign}} \in (0, 1)$ is introduced. These corrections reduce the influence of unstable or conflicting model predictions.

Step 3: Final weight and normalization. The unnormalized weight is

$$\widetilde{w}_i(x) = w_i^{(\text{base})}(x) \cdot \big(w_i^{(\text{cons})}(x)\big)^{\gamma},$$

where $\gamma > 0$ controls the strength of the consistency correction. The final normalized weight is

$$w_i(x) = \frac{\widetilde{w}_i(x)}{\sum_{j=1}^{N} \widetilde{w}_j(x)}.$$

The ADRTW score is thus obtained by combining transparent priors, text-dependent compatibility, and disagreement-aware consistency correction. Since all components are explicit, the method remains interpretable and can be reproduced by following the described weighting procedure and parameter settings.

Parameterization and practical behavior. The ADRTW weighting mechanism is rule-guided and interpretable rather than end-to-end learned. Its parameters control the relative influence of surface-level textual cues, model priors, and inter-model consistency penalties. Short and emoji-rich texts increase the relative influence of lexicon-based estimators, whereas longer and structurally richer texts increase the relative influence of the transformer-based component. Two key hyperparameters were selected on the synthetic benchmark: the consistency correction strength and the prior weight assigned to the RoBERTa-based estimator. After this tuning step, the selected parameters were kept fixed for all Reddit-based analyses. Consequently, the Reddit corpus was not used for supervised parameter optimization, but only for exploratory discourse-level analysis.

5. Experiments and results

The primary focus of the evaluation is the ADRTW aggregation mechanism. The toxicity and clustering analyses are included as complementary interpretive layers demonstrating how ADRTW-derived sentiment signals can support higher-level discourse interpretation.

5.1. Experimental setup

The empirical evaluation was conducted in two settings: a controlled synthetic benchmark for aggregation comparison and the full Reddit dataset for discourse-level analysis. Temporal sentiment trajectories were smoothed using a 60-day moving average. Toxicity detection was performed using a TF–IDF + logistic regression baseline with a rule-augmented extension, and clustering was applied to interaction and sentiment features with silhouette-based evaluation and PCA visualization.

5.2. Synthetic benchmark construction

To provide a controlled reference setting for evaluating aggregation behavior, we used a synthetic benchmark designed to simulate sentiment estimation inconsistencies commonly observed in short social media texts. The benchmark consists of 500 sentiment instances with reference polarity scores distributed across $[-1, 1]$,

covering negative, neutral, positive, borderline, and ambiguous sentiment conditions.

The reference sentiment score of each synthetic instance was treated as the controlled ground-truth value. The constructed examples were designed to include short, stylistically marked, and potentially ambiguous cases in which lexicon-based and transformer-based models may react differently. To emulate heterogeneous model behavior, controlled perturbations were introduced into the sentiment estimates, including polarity shifts, disagreement amplification, and context-sensitive noise.

The actual sentiment estimators were then applied to the synthetic text instances, allowing the benchmark to compare VADER, TextBlob, RoBERTa, static averaging, and ADRTW against the same controlled reference labels. The purpose of this benchmark is not to replace manually annotated social media data, but to isolate the aggregation problem under controlled and reproducible disagreement conditions.

5.3. Controlled benchmark evaluation

Hyperparameter tuning. Two ADRTW hyperparameters were tuned on the synthetic benchmark using Gaussian process-based Bayesian optimization. The optimized parameters were the consistency correction strength and the prior weight assigned to the RoBERTa-based sentiment estimator. The objective function minimized the mean absolute error (MAE) between the ADRTW score and the synthetic reference sentiment labels. The search space was defined as $[0.2, 1.0]$ for the consistency strength and $[1.0, 1.3]$ for the RoBERTa prior. The optimization was run for 30 function evaluations with 8 initial random points and a fixed random seed for reproducibility. The selected parameters were then kept fixed for all subsequent Reddit-based analyses.

Table 1 reports the performance of the individual sentiment models, the static averaging baseline, and ADRTW on the synthetic benchmark after Bayesian optimization of the ADRTW hyperparameters.

On the synthetic benchmark, ADRTW achieves the lowest MAE among all compared methods. Relative to static averaging, the MAE decreases from 0.2122 to 0.2003, corresponding to an improvement of approximately 5.6%. ADRTW also yields the highest correlation with the benchmark labels (0.9201), slightly exceeding both the strongest individual base model (RoBERTa: 0.9105) and the static averaging baseline (0.9176).

The MSE of ADRTW is marginally higher than that of static averaging (0.0633 vs. 0.0618). This suggests that the proposed weighting scheme is slightly more responsive to disagreement between models and therefore less conservative than simple averaging in some cases. In the context of this paper, this behavior is acceptable because the goal of ADRTW is not only error minimization, but also the preservation of context-sensitive variation in aggregated sentiment.

Taken together, the synthetic benchmark indicates that ADRTW remains competitive with static averaging in overall predictive quality while offering a more

adaptive aggregation behavior.

Table 1. Benchmark results on the synthetic sentiment dataset ($n = 500$). MAE = mean absolute error; MSE = mean squared error; Corr = Pearson correlation with ground-truth labels. ADRTW hyperparameters selected by Bayesian optimization within the predefined search space: consistency strength = 0.200, RoBERTa prior = 1.300.

Model / Aggregation	**MAE↓**	**MSE↓**	**Corr↑**
VADER	0.3708	0.2344	0.7311
TextBlob	0.3394	0.1636	0.7698
RoBERTa	0.3086	0.1436	0.9105
Static avg (avg3)	0.2122	**0.0618**	0.9176
ADRTW	**0.2003**	0.0633	**0.9201**

Alignment analysis on the Reddit corpus. Since no manually annotated sentiment ground truth was available for the Reddit corpus, we do not interpret the large-scale Reddit evaluation as supervised performance validation. Instead, we report an auxiliary alignment analysis relative to the transformer-based model, which showed the strongest individual benchmark performance among the base estimators.

On comments, ADRTW reduces the mean absolute deviation from the RoBERTa scores from 0.315 (static averaging) to 0.261, while increasing the correlation from 0.848 to 0.894. On posts, the corresponding values improve from 0.258 to 0.202 for mean absolute deviation and from 0.848 to 0.915 for correlation.

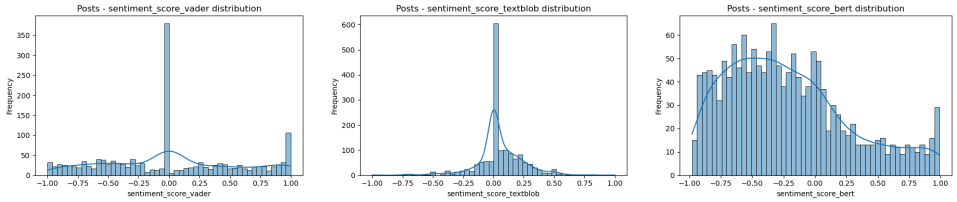
These results do not establish superiority with respect to ground truth on the Reddit corpus; rather, they indicate that ADRTW produces a more coherent aggregation relative to the strongest individual context-aware component while still retaining the contribution of the lexicon-based models.

5.4. Distributional analysis of sentiment estimators

We examine the distributional characteristics of the individual models and the ADRTW aggregation.

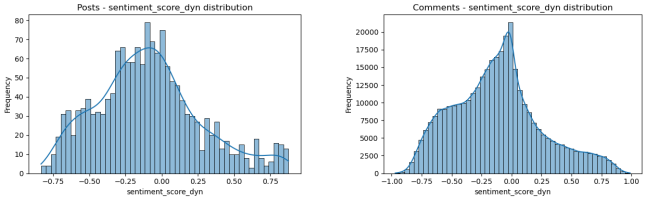
Figure 1 highlights systematic differences between the underlying sentiment estimators. The lexicon-based models (VADER and TextBlob) tend to produce distributions centered closer to neutral or slightly positive values, whereas the transformer-based model exhibits a noticeable shift toward negative sentiment. This suggests that ADRTW reduces the dominance of divergent individual outputs while remaining more aligned with the strongest context-aware component.

The ADRTW aggregation produces a more expressive and context-sensitive sentiment representation at both post and comment level (Figure 2). The resulting



(a) VADER **(b)** TextBlob **(c)** RoBERTa

Figure 1. Distribution of sentiment scores produced by the individual sentiment models on the dataset.



(a) Posts **(b)** Comments

Figure 2. Distribution of ADRTW sentiment scores at post and comment level. The aggregated scores remain expressive while reducing the dominance of any single underlying model.

distributions preserve meaningful variation while reducing the dominance of individual models and mitigating systematic biases. This indicates adaptive integration rather than simple arithmetic smoothing.

Table 2 further summarizes the distributional properties of the individual models and aggregation methods on the full Reddit comment corpus.

Table 2. Distributional statistics of sentiment estimators on the full Reddit comment corpus ($n = 354{,}050$).

Statistic	**VADER**	**TextBlob**	**RoBERTa**	**Avg**	**ADRTW**
Mean	+0.039	+0.059	−0.314	−0.072	−0.119
Std	0.509	0.264	0.517	0.348	0.369
Median	0.000	0.000	−0.417	−0.075	−0.124
Q1	−0.340	−0.007	−0.743	−0.321	−0.384
Q3	+0.440	+0.182	−0.028	+0.139	+0.079

The statistical results confirm the visual observations. While the lexicon-based models exhibit slightly positive central tendencies, the transformer-based model is shifted toward negative values. ADRTW remains closer to the transformer-based signal while preserving contributions from the lexicon-based models, resulting in a

more balanced aggregated distribution.

5.5. Temporal dynamics and collective rebalancing

Figure 3 presents the smoothed sentiment trajectories (60-day moving average) over time. While both methods follow similar global trends, ADRTW exhibits sharper local variations, indicating higher sensitivity to contextual changes.

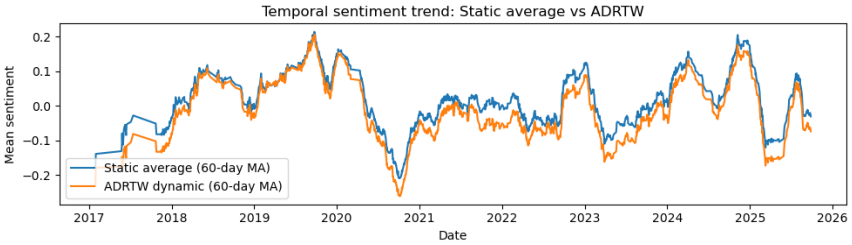


Figure 3. Temporal comparison of static averaging and ADRTW-based sentiment using moving averages. While both methods follow similar global trends, ADRTW reacts more sensitively to local contextual changes.

This suggests that ADRTW preserves context-sensitive variation rather than acting as a simple smoothing mechanism.

Figure 4 shows the temporal evolution of ADRTW sentiment at post and comment level. Posts tend to exhibit stronger emotional amplification, whereas comments display a moderating effect over time. This suggests that collective discussion may partially rebalance initially polarized emotional responses.

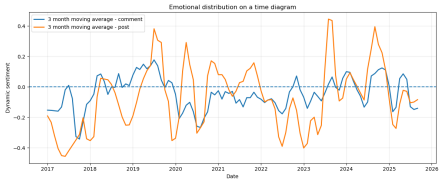


Figure 4. Temporal evolution of post- and comment-level ADRTW sentiment using moving averages. Posts often show stronger emotional amplification, whereas comments tend to moderate the discourse over time.

5.6. Toxicity as a complementary behavioral signal

Negative sentiment and toxic discourse are not equivalent phenomena. A comment may express dissatisfaction, criticism, or emotional tension without targeting another participant in a hostile or abusive manner [4, 10, 15]. To evaluate whether

ADRTW-based sentiment remains meaningful beyond polarity estimation alone, we include toxicity analysis as a complementary discourse-quality layer. Toxicity is not treated as an alternative sentiment model, but as an orthogonal signal that helps distinguish emotional negativity from socially harmful interaction.

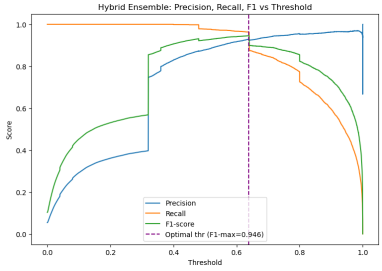


Figure 5. Precision, recall, and F1-score of the hybrid toxicity model across classification thresholds.

The toxicity baseline model represents comments using TF–IDF features and applies logistic regression to estimate the probability of toxic discourse. A hybrid ensemble extends this baseline with a rule-based component tailored to community-platform language patterns, including explicitly insulting, threatening, or personalized hostile expressions. The final toxicity score is adjusted by combining statistical prediction with rule-based signals, allowing the system to remain sensitive to both lexical regularities and explicit toxic markers.

The evaluation results indicate that the hybrid model does not primarily improve the overall F1-score, but rather shifts the precision–recall balance toward higher sensitivity. In toxicity-related classification tasks, such threshold-dependent trade-offs are especially important because aggregate metrics alone may hide practically relevant differences in model behavior [1]. This supports the methodological claim that toxicity should be modeled as a distinct discourse-quality dimension rather than inferred directly from negative sentiment alone [4, 15]. The threshold-dependent trade-off between precision, recall, and F1-score is illustrated in Figure 5.

5.7. Clustering analysis and participation archetypes

We further examined whether ADRTW-based sentiment supports higher-level discourse interpretation through clustering of posts and comments. Here, clustering serves not as a predictive task but as an interpretive layer to identify recurring participation patterns and behavioral archetypes.

For posts, clusters were formed using interaction and sentiment-related features (e.g., score, comment score, aggregated sentiment), while for comments we used like score, text length, and sentiment. Although smaller cluster numbers yielded higher silhouette scores, a four-cluster solution proved more informative for interpretation. Silhouette analysis assessed cluster quality [14], while PCA was used only for visualization [7].

The resulting clusters suggest four recurring roles: critical/debating participants, constructive contributors, short positive acknowledgers, and low-visibility passive participants. Rather than simply summarizing engagement, clustering reveals how emotional polarity, cognitive effort, and interaction patterns jointly structure community discourse.

Overall, the findings support the view that large-scale social media discussions are not emotionally uniform, but organized into recurring activity profiles and topic-dependent emotional dynamics.

6. Discussion and limitations

The joint analysis of sentiment, toxicity, and clustering suggests that social media interaction is better understood as a layered process rather than a simple positive–negative continuum. Short, highly polarized reactions often reflect immediate affective responses, whereas longer comments tend to be less extreme and more argumentative. This indicates that discourse structure is shaped not only by emotional polarity, but also by cognitive effort and interactional intent.

The synthetic benchmark and the Reddit corpus serve different evaluative purposes. The benchmark provides a controlled setting for comparing aggregation behavior against known reference scores, whereas the Reddit corpus demonstrates the applicability of the resulting ADRTW signal in large-scale exploratory discourse analysis. Therefore, conclusions about predictive accuracy are drawn only from the benchmark, while Reddit-based findings are interpreted in terms of coherence, alignment, and interpretability rather than supervised sentiment validity.

The study has several limitations. First, the real-world analysis is restricted to Reddit, so the observed weighting behavior may partly reflect platform-specific discourse patterns. Second, the transformer component was originally fine-tuned on social media text from another platform, which may introduce domain-transfer bias. Third, the ADRTW weighting mechanism is rule-guided and interpretable rather than end-to-end learned; this improves transparency but may limit cross-domain adaptability. Finally, the toxicity and clustering analyses should be interpreted as complementary interpretive layers rather than universal downstream guarantees.

7. Conclusion and future work

This paper introduced ADRTW, an interpretable and context-sensitive sentiment aggregation framework for heterogeneous social media text. Instead of replacing existing sentiment models, ADRTW combines them through adaptive reliability weighting based on textual cues and inter-model consistency. The results show that ADRTW remains competitive with static averaging on a controlled benchmark while providing a flexible signal for large-scale discourse-level analysis.

Future work will focus on author-aware and predictive analysis, including behavioral profiling, recurring user roles, and early ADRTW-based signals for modeling

later engagement trajectories. Further evaluation on manually annotated datasets and platforms beyond Reddit is also needed.

References

[1] D. Borkan, L. Dixon, J. Sorensen, N. Thain, L. Vasserman: *Nuanced Metrics for Measuring Unintended Bias with Real Data for Text Classification*, Proceedings of the 2019 World Wide Web Conference (2019), pp. 491–500, doi: 10.1145/3308560.3317593.

[2] T. De Smedt, W. Daelemans: *Pattern for Python*, Journal of Machine Learning Research 13 (2012), pp. 2063–2067.

[3] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova: *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics, 2019, doi: 10.18653/v1/N19-1423.

[4] P. Fortuna, S. Nunes: *A Survey on Automatic Detection of Hate Speech in Text*, ACM Computing Surveys 51.4 (2018), 85:1–85:30, doi: 10.1145/3232676.

[5] C. Guo, G. Pleiss, Y. Sun, K. Q. Weinberger: *On Calibration of Modern Neural Networks*, in: Proceedings of the 34th International Conference on Machine Learning (ICML), 2017, pp. 1321–1330.

[6] C. J. Hutto, E. Gilbert: *VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text*, in: Proceedings of the International AAAI Conference on Web and Social Media, 2014, doi: 10.1609/icwsm.v8i1.14550.

[7] I. T. Jolliffe, J. Cadima: *Principal Component Analysis: A Review and Recent Developments*, Philosophical Transactions of the Royal Society A 374.2065 (2016), p. 20150202, doi: 10.1098/rsta.2015.0202.

[8] B. Lakshminarayanan, A. Pritzel, C. Blundell: *Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles*, in: Advances in Neural Information Processing Systems (NeurIPS), 2017, pp. 6402–6413.

[9] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov: *RoBERTa: A Robustly Optimized BERT Pretraining Approach*, arXiv preprint arXiv:1907.11692 (2019), arXiv: 1907.11692.

[10] S. MacAvaney, H.-R. Yao, E. Yang, K. Russell, N. Goharian, O. Frieder: *Hate Speech Detection: Challenges and Solutions*, PLoS ONE 14.8 (2019), e0221152, doi: 10.1371/journal.pone.0221152.

[11] W. Medhat, A. Hassan, H. Korashy: *Sentiment analysis algorithms and applications: A survey*, Ain Shams Engineering Journal 5.4 (2014), pp. 1093–1113, doi: 10.1016/j.asej.2014.04.011.

[12] R. Polikar: *Ensemble based systems in decision making*, IEEE Circuits and Systems Magazine 6.3 (2006), pp. 21–45, doi: 10.1109/MCAS.2006.1688199.

[13] S. Rosenthal, N. Farra, P. Nakov: *SemEval-2017 Task 4: Sentiment Analysis in Twitter*, in: Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017), Association for Computational Linguistics, 2017, doi: 10.18653/v1/S17-2088, url: https://aclanthology.org/S17-2088/.

[14] P. J. Rousseeuw: *Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis*, Journal of Computational and Applied Mathematics 20 (1987), pp. 53–65, doi: 10.1016/0377-0427(87)90125-7.

[15] Z. Waseem, T. Davidson, D. Warmsley, I. Weber: *Understanding Abuse: A Typology of Abusive Language Detection Subtasks*, in: Proceedings of the First Workshop on Abusive Language Online, 2017, pp. 78–84, doi: 10.18653/v1/W17-3012.

[16] Z. G. Yang, Á. Agócs, G. Kusper, T. Váradi: *Abstractive text summarization for Hungarian*, Annales Mathematicae et Informaticae 53 (2021), pp. 299–316, doi: 10.33039/ami.2021.04.002.

[17] M. Zampieri, P. Nakov, S. Rosenthal, P. Atanasova, G. Karadzhov, H. Mubarak, L. Derczynski, Z. Pitenis: *SemEval-2020 Task 12: Multilingual Offensive Language Identification in Social Media*, in: Proceedings of the Fourteenth Workshop on Semantic Evaluation, 2020, pp. 1425–1447, doi: 10.18653/v1/2020.semeval-1.188, url: https://aclanthology.org/2020.semeval-1.188/.

[18] L. Zhang, S. Wang, B. Liu: *Deep Learning for Sentiment Analysis: A Survey*, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery (2018), doi: 10.1002/widm.1253.

Annales Mathematicae et Informaticae
63 (2026) pp. 32–41
DOI: 10.33039/ami.2026.07.001
URL: https://ami.uni-eszterhazy.hu

Bayesian adaptive assessment with distribution-aware item selection: An empirical study on uncertainty reduction and test efficiency

Anikó Apró[a], **Tibor Tajti**[bc]

[a]University of Debrecen, Doctoral School of Informatics
apro.aniko@inf.unideb.hu

[b]University of Debrecen, Faculty of Informatics
tajti.tibor@inf.unideb.hu

[c]Eszterházy Károly Catholic University
tajti.tibor@uni-eszterhazy.hu

Abstract. Computerized adaptive testing (CAT) combines ability estimation with an item-selection rule that usually favors the currently most informative item. Although randomization is often used for exposure control, security, or robustness, the probability distribution used for item sampling is rarely treated as an explicit design variable. This paper studies this choice in a Bayesian adaptive-assessment framework in which posterior updating is fixed, while the next item is sampled from a distribution over information-ranked candidates. Five kernels are compared: uniform, binomial, normal, exponential, and Poisson. Using interaction logs from 33 test sessions completed by 18 participants, we analyze observed session-level accuracy, test length, early stopping, and posterior uncertainty reduction. The results indicate that the selection kernel affects operational behavior: concentrated kernels tend to produce more stable accuracy and reduce interaction variability, whereas flatter kernels increase exploration and may prolong sessions. The study contributes a compact system-level formulation of distribution-aware item selection and shows how the exploration–exploitation trade-off appears in deployable Bayesian CAT systems.

Keywords: computerized adaptive testing, Bayesian adaptive testing, item-selection distribution, exploration–exploitation, early stopping, uncertainty reduction

Submitted: March 31, 2026
Accepted: July 1, 2026
Published online: July 17, 2026

1. Introduction and related work

Computerized adaptive testing (CAT) dynamically selects items to estimate an examinee's latent ability with fewer responses than fixed-form testing. Classical CAT typically combines an item response model with a deterministic selection rule, such as maximum Fisher information, and a stopping criterion based on precision or maximum test length [9, 10]. Bayesian CAT replaces point estimates with posterior distributions, making uncertainty available throughout the session and supporting uncertainty-aware termination [2, 5, 8, 11].

In practical adaptive systems, item selection is not always deterministic. Randomization may be introduced to support item exposure control, test security, or operational robustness [4, 6]. Recent studies have also examined broader item-selection strategies, including heuristic search, cognitive-diagnostic variants, and reinforcement-learning-based policies [1, 3, 7]. However, the probability distribution used to randomize item choice is often treated as a secondary implementation detail.

The central claim of this paper is that the distribution governing randomized item selection should be treated as a system-level design parameter. A concentrated distribution behaves similarly to greedy exploitation, whereas a flat distribution increases exploration. Even when the Bayesian update is unchanged, this choice can affect test length, uncertainty convergence, and final observed accuracy. The contribution is therefore narrow but practically relevant: given the same posterior update and information ranking, we isolate the operational effect of the probability law used to sample the next item.

This perspective is aligned with applied informatics. In a deployed adaptive-assessment system, the item-selection rule is not only a psychometric component; it also determines interaction cost, latency, user fatigue, item-pool usage, and the reliability of the final decision. Prior work on exposure control and utilization already shows that operational constraints shape item administration patterns [4, 6]. We extend this view by making the sampling distribution itself explicit and by examining its effect in real interaction logs.

The paper is organized into five main sections to keep the revision compact. Section 2 defines the distribution-aware selection mechanism and the Bayesian update. Section 3 describes the empirical setting and defines the outcome metrics, including the accuracy measure requested by the reviewers. Section 4 presents the empirical results. Section 5 discusses limitations and concludes the study.

2. Distribution-aware Bayesian adaptive assessment

At step t, the system has observed the response history

$$D_t = \{(i_1, y_1), \ldots, (i_t, y_t)\},$$

where i_s is the administered item and $y_s \in \{0, 1\}$ is the binary response outcome. Let θ denote latent proficiency and let I_t be the set of unused candidate items.

Given the current posterior $p(\theta \mid D_t)$, each candidate item $i \in I_t$ receives an information score

$$J_t(i) = \mathbb{E}_{\theta \sim p(\theta \mid D_t)}[I(i, \theta)],$$

where $I(i, \theta)$ is item information under the chosen measurement model. Classical maximum-information CAT selects the item with maximal $J_t(i)$.

In the proposed framework, information scores are first converted into ranks. Let $r_t(i) \in \{1, \ldots, |I_t|\}$ denote the rank of item i, where rank 1 is the currently most informative unused item. A kernel $K_\phi(r)$ parameterized by ϕ defines a probability distribution over ranks and items:

$$P_t(i \mid D_t, \phi) = \frac{K_\phi(r_t(i))}{\sum_{j \in I_t} K_\phi(r_t(j))}. \tag{2.1}$$

The next item is sampled according to

$$i_{t+1} \sim P_t(\cdot \mid D_t, \phi).$$

Thus, the measurement model determines the ranking, while the kernel determines how greedily or diffusely the system exploits that ranking. A degenerate kernel concentrated on rank 1 reproduces deterministic greedy CAT; a nearly flat kernel approaches uniform exploration. The implementation compares uniform, binomial, normal, exponential, and Poisson-shaped kernels.

The operational system uses a Bayesian mastery-style update for binary outcomes. Let p denote the latent mastery probability for a session. After t responses,

$$p \mid D_t \sim \mathrm{Beta}(\alpha_t, \beta_t), \quad \alpha_t = \alpha_0 + \sum_{s=1}^{t} y_s, \quad \beta_t = \beta_0 + \sum_{s=1}^{t} (1 - y_s). \tag{2.2}$$

In Equation (2.2), $y_s = 1$ indicates a correct answer and $y_s = 0$ an incorrect answer. The parameters α_0 and β_0 are the prior Beta parameters before any responses are observed; in the empirical reconstruction we use the non-informative prior $\alpha_0 = \beta_0 = 1$.

The posterior standard deviation is used as an analytic scalar measure of uncertainty:

$$U_t = \mathrm{SD}(p \mid D_t) = \sqrt{\frac{\alpha_t \beta_t}{(\alpha_t + \beta_t)^2 (\alpha_t + \beta_t + 1)}}. \tag{2.3}$$

Early stopping is defined as

$$\text{stop at step } t \iff U_t \leq \tau, \tag{2.4}$$

where τ is a fixed uncertainty threshold. The stopping rule is distribution-agnostic; differences in stopping behavior therefore arise from how the selection kernel changes the administered item sequence and the resulting uncertainty path.

3. Experimental setting and metrics

The empirical study uses interaction logs from an operational programming-assessment deployment. The adaptive tests targeted programming and algorithmic problem-solving knowledge. The item pool contained tasks involving code interpretation, correctness of program behavior, selection of appropriate solutions, and basic algorithmic reasoning. Items were organized by programming language and difficulty level. The available sessions mainly involve C# and Python tasks, with levels including beginner-oriented material; these dimensions are later considered when interpreting possible level and language confounds.

In total, the evaluation covers 33 test sessions from 18 participants. This is a limitation, and the results should be read as a preliminary empirical system study rather than a large-scale confirmatory psychometric validation. Nevertheless, the dataset is useful for studying operational behavior because it contains real session-level interaction traces generated by a deployable adaptive-testing implementation.

For each session, four outcome dimensions are analyzed: test length, final observed accuracy, stopping behavior, and uncertainty reduction. Test length is the number of administered items. Stopping behavior records whether the uncertainty threshold in Equation (2.4) was reached. Uncertainty reduction is measured from the posterior trajectory. Test accuracy is measured at the session level as the proportion of correctly answered items among all administered items:

$$\text{Accuracy}_q = \frac{1}{n_q} \sum_{s=1}^{n_q} y_{qs}, \tag{3.1}$$

where q denotes a test session, n_q is the number of administered items in that session, and $y_{qs} = 1$ if item s was answered correctly and 0 otherwise. Thus, the reported accuracy is observed response accuracy within a session, not an external criterion-validity coefficient. Bayesian posterior uncertainty is analyzed separately as a measure of estimation confidence and convergence.

All compared variants use the same Bayesian update and differ only in the kernel used in Equation (2.1). This design isolates the operational effect of distribution-aware item selection. Standard deterministic CAT baselines remain important future comparison points, but the present paper focuses on the effect of alternative sampling kernels inside one fixed Bayesian framework.

Because the number of sessions per distribution is small, the analysis combines descriptive summaries with nonparametric comparisons. Kruskal–Wallis tests are used for multi-group comparisons, Mann–Whitney tests for two-group contrasts, and Spearman rank correlation for ordinal concentration effects. These tests are interpreted cautiously and primarily as evidence of systematic empirical tendencies.

4. Results

Table 1 summarizes the session-level results by item-selection distribution. The normal kernel yields the highest average accuracy in the available runs and one of

the shortest average test lengths, whereas the uniform kernel produces the lowest average accuracy and the longest average tests. These descriptive patterns indicate that the kernel changes the system's operating profile.

Table 1. Summary statistics by item-selection distribution.

Distribution	N	Test length (mean ± SD)	Accuracy (mean ± SD)	Stop rate
Binomial	6	15.50 ± 2.26	0.702 ± 0.159	0.833
Exponential	7	16.43 ± 4.04	0.795 ± 0.098	0.857
Normal	8	14.88 ± 5.64	0.878 ± 0.095	0.250
Poisson	5	15.00 ± 6.86	0.710 ± 0.157	0.400
Uniform	7	16.57 ± 6.13	0.646 ± 0.198	0.429

The small per-kernel counts require caution. A five-way Kruskal–Wallis test on session accuracy across the individual distributions yields $H = 8.14$, $p = 0.087$, which does not reach conventional significance. Therefore, the strongest defensible interpretation is not that one individual distribution is universally best, but that the distributions generate different empirical operating profiles. The monotonic trend from concentrated to flat kernels motivates the grouped analysis below.

Because the five kernels differ primarily in how sharply they concentrate probability mass on top-ranked items, we introduce a qualitative grouping: concentrated kernels (normal and exponential), moderate kernels (binomial and Poisson), and a flat kernel (uniform). Table 2 shows that concentrated kernels have the highest mean accuracy, while the flat kernel has the lowest mean accuracy and the longest average test length.

Table 2. Summary statistics by kernel concentration level.

Concentration	n	Accuracy (mean ± SD)	Test length (mean ± SD)
Concentrated	15	0.840 ± 0.102	15.6 ± 4.9
Moderate	11	0.705 ± 0.150	15.3 ± 4.6
Flat	7	0.646 ± 0.198	16.6 ± 6.1

The grouped comparison gives clearer evidence. The Kruskal–Wallis test across three concentration levels for accuracy gives $H = 7.03$, $p = 0.030$. A Mann–Whitney contrast between concentrated and non-concentrated kernels gives $U = 206.0$, $p = 0.011$, with a large rank-biserial effect. Spearman correlation between concentration level and session accuracy is $\rho = 0.465$, $p = 0.006$. Test length does not differ significantly across concentration levels ($H = 0.20$, $p = 0.90$), indicating that the observed accuracy advantage of concentrated kernels is not simply achieved by administering more items.

Figure 1 shows the distribution of test length by strategy. Uniform sampling exhibits a broader distribution with a longer upper tail, indicating that exploratory

behavior can prolong sessions before the stopping criterion is met. More concentrated kernels reduce the frequency of very long sessions. From a system perspective, test length is a direct proxy for user interaction load and, when per-item response times are comparable, for total session duration.

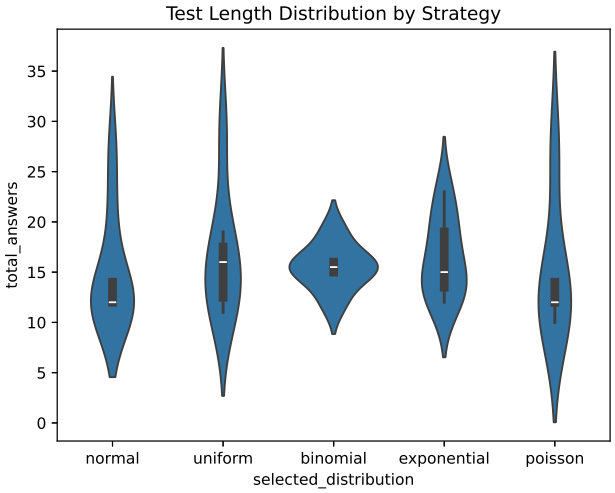


Figure 1. Distribution of test length by item-selection strategy.

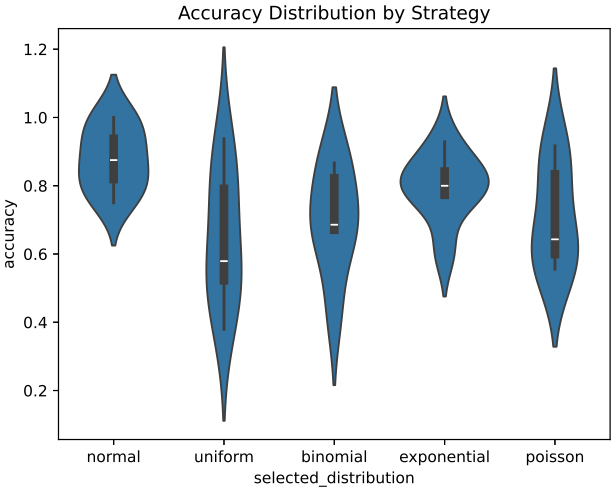


Figure 2. Distribution of final session accuracy by item-selection strategy.

Figure 2 presents the session-level accuracy distributions. Uniform sampling produces a wider and lower-centered distribution, while the normal and exponential

kernels cluster more tightly at higher accuracy levels in the available sample. Again, the defensible claim is not that one kernel is universally optimal, but that the kernels generate measurably different operating profiles.

To quantify how efficiently each kernel reduces uncertainty, we reconstruct the Beta posterior trajectory for every session using $\alpha_0 = \beta_0 = 1$ and compute the per-item uncertainty reduction rate $\Delta U/n = (U_0 - U_n)/n$. Table 3 reports the resulting information-efficiency summary. The normal kernel obtains the largest per-item uncertainty reduction, while the uniform kernel has the lowest value.

Table 3. Information efficiency by item-selection distribution.

Distribution	Final U_n	ΔU total	$\Delta U/n$ per item
Normal	0.0871	0.2016	0.01477
Poisson	0.1082	0.1804	0.01330
Exponential	0.0954	0.1933	0.01228
Binomial	0.1043	0.1843	0.01206
Uniform	0.1046	0.1841	0.01191

The normal kernel extracts approximately 24% more uncertainty reduction per administered item than the uniform kernel in the available sample. This supports the interpretation that the selection distribution changes the information yield of each interaction. Concentrated kernels more often choose items near the current estimated ability boundary, while uniform sampling may spend more interactions on less informative items.

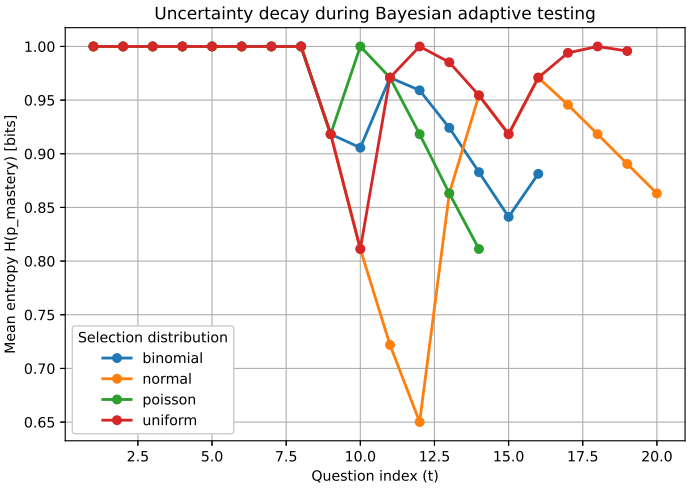


Figure 3. Mean posterior uncertainty as a function of administered items. Steeper decline corresponds to faster information accumulation.

Figure 3 shows the mean decay of posterior uncertainty over the course of the test. Equation (2.3) defines uncertainty analytically through the posterior standard deviation. In the implementation logs, an entropy-based uncertainty indicator was also recorded; the plotted trajectory follows the same convergence interpretation, with steeper decline corresponding to faster information accumulation.

Early stopping is implemented through the fixed threshold in Equation (2.4). The rule itself is distribution-agnostic, but the selection kernel changes the sequence of administered items and therefore the uncertainty path U_t. In the present data, stopped sessions are on average longer and lower-accuracy than non-stopped sessions (Table 4). This apparently counterintuitive pattern is explainable: the stopping threshold measures posterior certainty, not mastery. A student who consistently answers incorrectly also accumulates posterior certainty because the Beta posterior concentrates near low mastery probability.

Table 4. Test length and accuracy: stopped vs. non-stopped sessions.

	Non-stopped ($n = 15$)		Stopped ($n = 18$)	
	Length	Accuracy	Length	Accuracy
Mean	12.1	0.798	18.7	0.717
SD	1.3	0.154	4.9	0.162

This result is important for adaptive-system design. Uncertainty-based stopping should not be interpreted as an implicit pass/fail decision. It is better understood as a convergence detector that can indicate sufficient certainty about either high or low mastery. In practice, the stopping rule should therefore be paired with a separate mastery decision rule if the test is used for pass/fail classification.

Table 5. Robustness of the concentration–accuracy association across subsets and controls.

Subset / control	n	ρ	Result
Full sample	33	0.465	$p = 0.006$
Beginner-only	25	0.453	$p = 0.023$
C# + Python only	29	0.444	$p = 0.016$
Ability-adjusted residual	33	0.369	$p = 0.035$
Excluding normal kernel	25	0.337	$p = 0.099$
Excluding Bayesian method	22	0.306	$p = 0.167$

Finally, we examine whether the observed concentration–accuracy association is robust to co-varying dimensions. With the available sample, kernel, adaptive method, difficulty level, language, and student identity are not fully independent. Restricting the analysis to beginner sessions preserves the association ($\rho = 0.453$, $p = 0.023$). Restricting to the two majority languages, C# and Python, also pre-

serves it ($\rho = 0.444$, $p = 0.016$). After adjusting for student ability by subtracting each student's leave-one-out mean accuracy, the concentration effect remains significant ($\rho = 0.369$, $p = 0.035$). Table 5 summarizes these checks.

These checks support a cautious conclusion. The concentration effect is visible in the full sample and survives controls for level, language, and student ability. However, it weakens when the normal kernel or Bayesian-method sessions are removed. This indicates that the effect is partly amplified by the specific deployment configuration and should be tested in a larger balanced study.

5. Discussion and conclusion

The results support three conclusions. First, the probability distribution used for randomized item selection is not a trivial implementation detail. Once the information ranking is fixed, the kernel still changes which items are administered and therefore changes the session trajectory. This makes the kernel a practical design parameter between purely greedy selection and unrestricted random exploration.

Second, the empirical results are consistent with an exploration–exploitation trade-off. Concentrated kernels tend to stabilize observed accuracy and improve per-item uncertainty reduction, whereas flatter kernels provide broader exploration at the price of higher variability and, in some sessions, longer interaction. This trade-off is relevant for real systems: formative systems may tolerate more exploration, while time-constrained assessment may prefer faster convergence.

Third, uncertainty-based early stopping is best interpreted as a convergence mechanism. It interacts with the information yield of the selected item sequence rather than acting as an independent mastery indicator. The stopped-versus-non-stopped comparison shows that posterior certainty can accumulate in both mastery and non-mastery directions. Therefore, a practical adaptive test should separate the stopping rule from the final mastery or grading decision.

The main limitation is the size of the evaluation: 33 sessions from 18 participants are sufficient for a preliminary system-level analysis but not for broad generalization. The deployment is also partly confounded by method assignment, language, level, and student identity. The robustness checks reduce this concern but do not eliminate it. Future work should extend the sample, balance kernel assignment across languages and difficulty levels, include deterministic maximum-information and randomesque baselines, and report confidence intervals or formal paired comparisons.

In summary, this paper formulated distribution-aware item selection in Bayesian adaptive assessment as a probabilistic layer over information-based ranking. The empirical results indicate that the kernel affects observed accuracy, test length, uncertainty convergence, and stopping behavior. This supports treating the selection distribution and stopping rule as explicit, tunable design parameters in deployable CAT systems.

References

[1] X. Cao, Y. Lin, D. Liu, F. Zheng, H. B.-L. Duh: *Novel item selection strategies for cognitive diagnostic computerized adaptive testing: A heuristic search framework*, Behavior Research Methods 56.4 (2024), pp. 2859–2885, doi: 10.3758/s13428-023-02228-9.

[2] L. Dwahdh, N. Alshraifin: *The impact of computerized adaptive test termination rules on accuracy across different ability estimation methods*, EURASIA Journal of Mathematics, Science and Technology Education 21.1 (2025), em15897, doi: 10.29333/ejmste/15897.

[3] P. Gilavert et al.: *Computerized Adaptive Testing: A Unified Approach Under Reinforcement Learning*, in: Proceedings of EC-TEL, 2022, doi: 10.1007/978-3-031-16290-9_13.

[4] S. Lim, S. W. Choi: *Item exposure and utilization control methods for optimal test assembly*, Behaviormetrika 51 (2024), pp. 125–156, doi: 10.1007/s41237-023-00214-1.

[5] L. Niu, S. W. Choi: *More Efficient Fully Bayesian Adaptive Testing with a Revised Proposal Distribution*, Behaviormetrika 49 (2022), pp. 255–273.

[6] Y. Pan et al.: *Item Selection Algorithm Based on Collaborative Filtering for Item Exposure Control*, Educational Measurement: Issues and Practice 42.4 (2023), pp. 6–18, doi: 10.1111/emip.12578.

[7] Y. Pian et al.: *Improving the Item Selection Process with Reinforcement Learning in Computerized Adaptive Testing*, in: Communications in Computer and Information Science, 2023, doi: 10.1007/978-3-031-36336-8_35.

[8] H. Ren, S. W. Choi, W. J. van der Linden: *Bayesian adaptive testing with polytomous items*, Behaviormetrika 47.3 (2020), pp. 427–449, doi: 10.1007/s41237-020-00114-8.

[9] M. Sahin Kursad: *Effects of Different Item Selection Methods on Test Efficiency in CAT*, Journal of Measurement and Evaluation in Education and Psychology 14.1 (2023), pp. 33–46, doi: 10.21031/epod.1140757.

[10] Q. Tang et al.: *Item selection methods for cognitive diagnostic computerized adaptive testing*, Advances in Psychological Science 28.12 (2020), pp. 2160–2168, doi: 10.3724/SP.J.1042.2020.02160.

[11] Z. Wang, C. Wang, D. J. Weiss: *Termination criteria for grid multiclassification adaptive testing with multidimensional polytomous items*, Applied Psychological Measurement 46.7 (2022), pp. 551–570, doi: 10.1177/01466216221108383.

Annales Mathematicae et Informaticae
63 (2026) pp. 42–54
DOI: 10.33039/ami.2026.07.002
URL: https://ami.uni-eszterhazy.hu

Student opinion mining: Automated topic extraction from student feedback*

Olivér Czimbalmos[a], Zsolt Szántó[a], Gábor Kőrösi[a], Péter Becsei[a], Beáta Udvari[b], Richárd Farkas[a]

[a]Institue of Informatics, University of Szeged
{czimbalmos,szantozs,korosig,becsei,rfarkas}@inf.u-szeged.hu

[b]Directorate of Academic Affairs, University of Szeged
udvari.beata@szte.hu

Abstract. While Student Evaluation of Teaching Work (SETW) is a cornerstone of quality assurance in higher education, the high volume and linguistic complexity of unstructured qualitative feedback often lead to its underutilization in institutional decision-making. This study addresses this "analysis gap" by developing an automated pipeline to process 34,000 unique Hungarian student responses from a major research university. To empower academic administration and faculty leadership with the ability to uncover latent thematic patterns within these responses, we propose a hybrid NLP framework that utilizes a Large Language Model (LLM) for thematic reclassification and Aspect-Based Sentiment Analysis (ABSA), combined with an unsupervised layer for fine-grained latent topic discovery using transformer-based embeddings and HDBSCAN clustering. Our proposed pipeline successfully identified new granular latent topics – such as lecture pacing, traceability, material accessibility and slides – providing a level of diagnostic detail that remains invisible to standard quantitative metrics. The results prove that modern natural language processing (NLP) techniques can effectively transform raw, unstructured student narratives into objective, actionable diagnostic tools.

Keywords: student feedback, large language models, natural language processing

*This work was supported by project 2024-1.2.3-HU-RIZONT-2024-00017, implemented with the support provided by the Ministry of Culture and Innovation of Hungary from the National Research, Development and Innovation Fund, financed under the 2024-1.2.3-HU-RIZONT funding scheme.

Submitted: March 31, 2026
Accepted: July 3, 2026
Published online: July 17, 2026

1. Introduction

University quality assurance mainly relies on student feedback, containing crucial information about the institution's performance. Student Evaluation of Teaching Work (SETW) provides universities with a systematic mechanism to assess pedagogical effectiveness and identify areas for improvement. While quantitative Likert-scale ratings offer a convenient numerical summary of student satisfaction, their interpretability is inherently limited. The most informative insights are often embedded in textual comments, where students articulate the underlying reasons behind their evaluations. These qualitative responses reveal nuanced perspectives on teaching practices, course organization, and learning environments that remain invisible in purely quantitative assessments. Despite their potential value, extracting actionable insights from large volumes of textual feedback remains a persistent challenge for universities. Large institutions frequently collect tens of thousands of individual responses each semester, making manual analysis both impractical and prone to subjective interpretation. As a result, a substantial portion of qualitative student feedback remains underutilized in institutional decision-making processes. This problem becomes even more pronounced in multilingual educational environments. In the Hungarian context, for example, the morphological complexity of the language significantly increases the difficulty of traditional natural language processing approaches, while institutional differences in evaluation frameworks across faculties further complicate the systematic interpretation of feedback.

Previous research has explored automated methods for analyzing student feedback, including sentiment analysis and topic modeling [5, 10, 13]. While these approaches have demonstrated promising results, they often face limitations when applied to large multilingual datasets.

To address these challenges, this study presents a large-scale case study of automated qualitative feedback analysis conducted at the University of Szeged, involving more than 34,000 unique student responses. Our research identifies previously unrecognized thematic patterns in texts, hence it transforms unstructured student narratives into structured, interpretable insights. Concurrently, this work addresses the following central research question: *How effectively can modern NLP techniques, including Large Language Models and transformer-based embeddings, automatically extract meaningful topics and sentiments from large-scale student feedback in higher education?*

To achieve this goal, we propose a hybrid Natural Language Processing (NLP) pipeline (see Figure 1) that combines large language model–based classification with modern embedding techniques and density-based clustering methods. The framework first harmonizes heterogeneous faculty-level feedback into the new institutional categories while simultaneously identifying sentiment expressed toward specific aspects of teaching. Subsequently, multilingual text embeddings are created to capture semantic similarities between responses, enabling the discovery of latent thematic clusters that reveal recurring student concerns. These clusters then recive an appropiate name based on their text such as **lecture pacing, course**

organization, or material accessibility and slides. To furthure enhance the preocessing the results are summerized in a interactive dashboard.

The primary contribution of this work is the demonstration of a scalable methodology for transforming large-scale qualitative student feedback into actionable institutional knowledge. By integrating automated categorization, sentiment analysis, and unsupervised topic discovery within a single analytical framework, the proposed approach enables universities to move beyond aggregate satisfaction scores and uncover the underlying drivers of student experience. The results illustrate how modern opinion mining techniques can support evidence-based educational governance and help institutions more effectively respond to student needs.

We also contribute to the Hungarian NLP research as we evaluate the effectiveness of open Large Language Models in processing Hungarian-language student feedback, focusing on few-shot learning for both category classification and aspect-based sentiment analysis, and conduct a systematic comparison of state-of-the-art multilingual embedding models to assess their ability to capture semantic structure in Hungarian textual data.

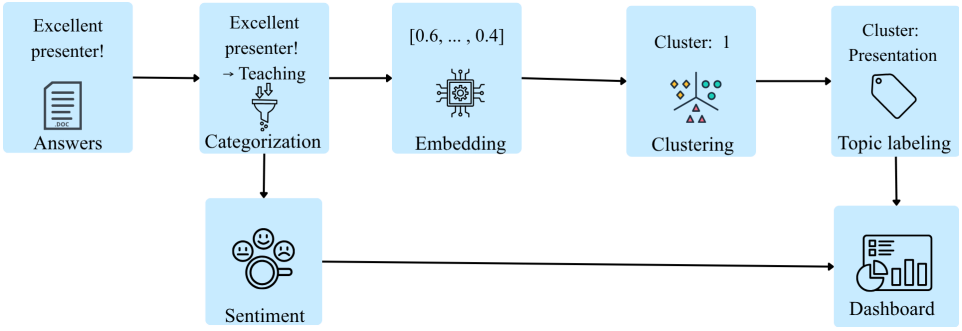


Figure 1. Schematic view of our proposed pipeline.

1.1. Related work

The automated analysis of student feedback through NLP has become an increasingly prominent field of study [1]. In the international literature, various methodologies have been proposed for thematic discovery and sentiment detection within educational datasets. A widely adopted approach is the application of Latent Dirichlet Allocation (LDA) for topic modeling. For instance, Sun, Yan (2023) utilized LDA to extract key themes related to the quality of teaching and learning from student narratives [13]. Beyond internal pedagogical assessment, student reviews from online platforms have also been utilized for institutional SWOT analyses. In this context, Srinivas, Rajendran demonstrated a framework that combines LDA-based topic modeling with rule-based sentiment detection to identify organizational strengths and weaknesses [12]. These foundational attempts at thematic discovery underscore the potential of automated qualitative analysis, which our

research seeks to refine using more contextually aware semantic models.

The rise of Massive Open Online Courses (MOOCs) has further catalyzed the development of more complex student feedback analysis. To determine the polarity of large-scale feedback, Baqach, Battou (2024) implemented a pipeline using BERT-based embeddings coupled with deep neural networks, incorporating both convolutional and dense layers for classification [3]. More recent comparative research has shifted focus toward the efficacy of different embedding strategies. Specifically, it has been observed that high-parameter foundation models and their corresponding embeddings can provide superior semantic representation in clustering tasks when compared to traditional BERT-scale architectures [6]. Our methodology builds upon this shift toward high-parameter embedder models, utilizing their deep semantic representations to enhance the granularity of student feedback clusters.

Paralleling this shift, Mersha, Kalita, et al. (2024) established an end-to-end semantic-driven topic modeling framework that utilizes Sentence-BERT representations combined with UMAP dimensionality reduction and HDBSCAN density clustering to isolate coherent thematic regions from noisy text corpora [8]. In the Hungarian higher education landscape, the quantitative results of Student Evaluation of Teaching Work (SETW) are often made publicly available by institutions such as the University of Veterinary Medicine Budapest and the University of Szeged. However, the systematic application of NLP to the qualitative, textual portions of these surveys remains less documented in the academic literature. Nevertheless, data mining and opinion detection have been applied to other Hungarian-language domains. Osváth, Yang Zijian, Kósa (2022) explored patient experience narratives, utilizing Uniform Manifold Approximation and Projection (UMAP) for dimensionality reduction and subsequent topic modeling [10]. Similar methodologies have appeared in the clinical sector as well; for example, Hinek (2024) utilized LDA to identify key thematic dimensions within Hungarian-language hotel guest reviews [5]. These studies highlight the ongoing transition from traditional statistical methods toward context-aware semantic analysis in the processing of Hungarian textual data. By situating our research within this evolving Hungarian landscape, we aim to extend these localized NLP successes into the specific domain of educational evaluation using more advanced generative architectures.

Methodologically, our research is most closely aligned with the framework presented by Osváth, Yang Zijian, Kósa (2022). Unlike their approach, we utilize Large Language Models (LLMs) for automated topic labeling, leveraging the extensive world knowledge inherent in these foundation models to generate descriptive and contextually accurate titles for the emergent clusters. Furthermore, we implement LLM-based Aspect-Based Sentiment Analysis (ABSA), allowing for a more nuanced evaluation of student feedback compared to traditional sentiment detection methods.

2. Methodology

2.1. Dataset

The dataset utilized in this study was provided by the Directorate of Academic Affairs at the University of Szeged. The corpus comprises student responses from two distinct faculties (Faculty 1 and Faculty 2) spanning three academic semesters: the full 2023/2024 academic year and the first semester of 2024/2025. All data were anonymized at the database level to ensure the privacy of both instructors and students before being released for research purposes.

The dataset is primarily composed of Hungarian textual feedback, though it also contains a small proportion of English and French responses. This multilingual characteristic necessitated the use of a multilingual embedding model in the subsequent stages of the pipeline [14]. The questionnaires used by the two faculties differed significantly in their qualitative framing:

- Faculty 1 employed multiple specific, instructor-focused questions (e.g., **"To what extent was the instructor's delivery conducive to note-taking?"**).
- Faculty 2 used a broader, more general approach, including self-reflective prompts (e.g., **"Self-reflection: What have you done to achieve the best possible performance in this course?"**).

An analysis of participation revealed a participation paradox: Faculty 2, despite offering fewer open-ended questions, generated a higher absolute volume of responses than Faculty 1. However, responses from Faculty 1 were generally longer and more detailed, with an average token count of 17.9 compared to 15.8 for Faculty 2.

The raw dataset underwent an initial preprocessing phase to ensure data quality. This included the removal of duplicate entries and responses consisting solely of non-alphanumeric characters (e.g., punctuation marks or symbols). Following these steps, the final corpus for analysis contained 34,000 unique responses with a vocabulary size of 42,809 tokens. For the statistics summary, see Table 1.

Table 1. Descriptive Statistics of the SETW Corpus.

Metric	Faculty 1	Faculty 2	Token
Number of unique responses	15,688	18,313	34,000
Mean token count	17,9	15,8	16
Median token count	11	13	12
Vocabulary size (Tokens)	27,225	26,763	42,809

From this set of answers 20 samples were selected randomly -a answer could only be picked once- from each question as a result we gathered 200 responses.

From these answers we have annotated the samples into the 6 given categories and also assigned the sentiment values for each category. There were two annotators, who used annotation guidelines provided from the Academic Affairs office[1]. The initial inter annotator agreement – averaged Cohen's kappa for each class – came out as 0.96 and for the aspect based sentiment annotation as 0.94. After each annotator finished they have discuissed the cases of disagreement and assigned the final value.

2.2. Categorization and aspect-based sentiment analysis

To address the structural heterogeneity of the faculty-level questionnaires, we implemented a standardized classification layer based on a unified framework of six core dimensions: **teachers preparation, safe study environment, requirements, teaching, teachers availability and methodology** (plus a "neutral" category). These dimensions are strategically designed to serve as the structural framework for the university's future standardized evaluation system. Given that the research corpus consists of legacy data with non-uniform qualitative framing, the necessity to reconcile historical feedback with these upcoming administrative requirements motivated the integration of this classification layer into our pipeline.

For the categorization and aspect-based sentiment analysis, we have compared three LLMs Llama-3-70B-Instruct, Qwen2-72B-Instruct, DeepSeek-llm-67b-chat [2, 11]. A critical feature of our pipeline is the integration of Aspect-Based Sentiment Analysis (ABSA). Rather than assigning a single global sentiment score to an entire response, the model identifies the specific polarity – **Positive, Neutral, or Negative** – associated with each identified category within the text.

We employed an in-context learning strategy with few-shot prompting, providing the model with category definitions and three representative examples for both classification and aspect-level sentiment detection[2]. To validate the system's performance, we established an evaluation protocol using a gold standard dataset of 200 manually annotated examples.

The vectorization, clustering steps were performed for each category separately at the review level.

2.3. Unsupervised latent topic discovery

As one of our main objectives is to discover new actionable topics, we have implemented the following steps into the pipeline.

2.3.1. Vectorization

To enable unsupervised clustering, the unstructured textual responses were transformed into high-dimensional dense vectors. In our experimental setup, we evaluated and compared two state-of-the-art multilingual transformer-based architec-

[1]Annotation guidelines: https://github.com/szegedai/StudendOpinionMining
[2]Prompt template: https://github.com/szegedai/StudendOpinionMining

tures: **Jina-v3**, drawing on the recent findings of Csányi et al. (2024), and the **Qwen3-Embedding-8B** model [4, 14].

These specific architectures were selected for their ability to handle the morphological complexity of the Hungarian language while providing a unified semantic space. This ensures a consistent and comparable representation across the multilingual (Hungarian, English, and French) corpus.

2.3.2. Topic discovery

For the discovery of latent topics, we employed Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) [7]. Unlike centroid-based methods such as K-means, HDBSCAN does not require a pre-defined number of clusters and is capable of identifying groups of varying shapes and densities. Crucially, the algorithm explicitly identifies "noise" – responses that do not align with any distinct thematic cluster – thereby ensuring that the resulting topics remain semantically coherent. To optimize the clustering results, we conducted a random search for the minimum cluster size parameter, defined within the range of $[2, \sqrt{N}]$, where N represents the sample size of the given aspect. The clustering quality was validated using a density-based validity index [9], which measures the stability and separation of the identified themes.

2.3.3. Automated topic labeling

To ensure the administrative utility of the clusters, we implemented an automated labeling procedure using a LLM. The model was tasked with generating concise, descriptive titles for each cluster by analyzing a subset of representative responses. During the development of this procedure, we compared two distinct sampling strategies for providing context to the LLM. Centroid-based sampling: Selecting the ten responses closest to the cluster centroid in Euclidean distance. Random sampling: Selecting ten responses at random from within the cluster.

3. Results and discussion

3.1. Categorization

The effectiveness of the few-shot prompting strategy was validated on a manually annotated gold standard dataset of 200 examples. As summarized in Table 2, the Llama-3 model outperformed its counterparts, achieving a weighted $F1$ -score of 0.72. A notable observation during the benchmarking process was the model's sensitivity to prompt length; while few-shot examples were critical for performance, an excessive number of examples in the prompt led to a gradual degradation in accuracy, particularly for shorter student responses.

Following validation, the model was deployed to classify the full corpus of 34,000 unique responses. The distribution of labels revealed a significant class imbalance, reflecting the primary concerns of the student body. The Methodology category

Table 2. Classification performance comparison across models ($N = 200$).

Model	Weighted F1
Llama-3-70B-Instruct	**0.72**
Qwen2-72B-Instruct	0.60
DeepSeek-llm-67b-chat	0.58

emerged as the most prominent dimension, particularly within Faculty 1, accounting for nearly half of the thematic assignments.

In order to further show the result of the classification Table 3 shows each classes F1 score and their support.

Table 3. LLama F1 scores across classes and their support.

Class	$F1$ score	Support
Methodology	0.78	108
Teachers preparation	0.73	90
Save Study Environment	0.60	19
Teacher Accesibility	0.0	5
Requirements	0.76	12
Teaching	0.66	6
Neutral	0.68	37

3.2. Aspect-based sentiment analysis

The evaluation of the sentiment analysis component focused on the model's ability to assign correct emotional valence to specific classes. As with the classification task, performance was measured against the gold standard dataset of 200 manually annotated examples. The results, summarized in Table 4, show that the **Llama-3-70B-Instruct** model achieved a $F1$-score of **0.82**, outperforming both Qwen2 and DeepSeek. Table 5 summerizes the F1 scores achived with the Llama model for each sentiment level.

Table 4. Model performance comparison on aspect-based sentiment analysis ($N = 200$).

Model	Sentiment ($F1$)
Llama-3-70B-Instruct	**0.82**
Qwen2-72B-Instruct	0.79
DeepSeek-llm-67b-chat	0.72

The higher $F1$-scores for sentiment analysis compared to classification suggest that identifying emotional polarity is a less ambiguous task for LLMs than thematic categorization within a specialized administrative framework.

By employing ABSA, the pipeline successfully identified "mixed" reviews that would be lost in a global sentiment approach. For instance, in cases where a student praised an instructor's preparedness but critiqued the course requirements, the model correctly assigned a *Positive* label to the "Teachers Preparation" aspect and a *Negative* label to the "Requirements" aspect. This granularity allows university management to differentiate between interpersonal success and structural curriculum issues, providing a more balanced view of faculty performance.

Qualitative review of the ABSA results indicated that the model's "world knowledge" was instrumental in handling ambiguous feedback. Llama-3-70B proved capable of identifying the correct category even when students used non-technical language or slang. For instance the text **"I loved the long classes on Monday mornings, with the teachers soporific voice!"**, which can be interpreted as a positive feedback about the lecturers class, but if we see the question for context: **"What would you suggest to change regarding the work of the instructor?"**, the word *loved* gets a sarcastic meaning and changes the answers polarity.

Table 5. F1 scores across classes for Aspect Based Sentiment Analysis.

Sentimen	***F*1 score**
Positive	0.89
Neutral	0.43
Negative	0.86

3.3. Vectorization performance and embedder selection

The selection of the **Qwen3-embedding-8B** model was driven by a comparative evaluation against the **Jina-v3** multilingual model. To assess the models' ability to capture the semantic nuances of the Hungarian language in an academic context, we performed a similarity-based retrieval test using reference student comments. Our primary selection criterion was to identify which model consistently placed semantically similar responses in closer proximity to the reference text within the vector space.

In the first comparative example, for the reference text **"There were few practical examples"**, the Qwen model's nearest neighbor was **"Sometimes it was hard to follow the pace; it would be good to have more practical examples"**. Conversely, Jina-v3 identified **"The subject might be useful in later studies; there were few examples"** as the most similar response. While both segments of the Qwen-selected response focus on pedagogical critique regarding the lecture's delivery, Jina's selection only aligns with the reference in its second half, introducing an irrelevant positive sentiment in the first.

In the second case, using the reference text **"Communication could have been better"**, Jina-v3 retrieved **"Overall, it was a positive experience"**, which is thematically distant from the specific concern raised. In contrast, the Qwen model placed **"Maybe a bit more explanation would have helped"** in closest proximity. This result is semantically much more relevant to the original intent, as it correctly identifies "explanation" as a core component of the instructional communication mentioned in the reference.

3.4. Latent topic discovery and clustering

By applying HDBSCAN, the pipeline successfully identified new latent subtopics that were previously invisible to both the quantitative Likert scales and the high-level administrative categories. We present the detailed latent topic discovery and sentiment mapping specifically for the **Methodology** category, which emerged as the most prominent dimension in our initial classification results. As illustrated in Figure 2, the semantic space is organized into well-defined "islands", representing specific student concerns or praises.

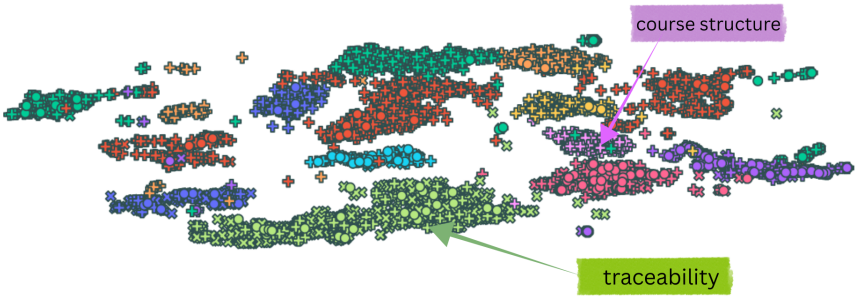


Figure 2. UMAP visualization of the identified latent student topics in the case of Methodology (Plus: positive, X: negative Circle: neutral sentiment.

The automated labeling procedure identified 11 distinct themes within the dataset, providing a comprehensive overview of the pedagogical environment. In order to get a better picture on the goodness of clustering 10 random example was picked and evaluated by hand how cohesive it was. These results are described in the following list with the clusters name and description.

- **Lecture Pace:** Feedback regarding the speed of delivery during classroom sessions. From the 10 checked answers there were only one where the text falls into another cathegory.
- **Clarity:** Comments on the intelligibility of the instructor's explanations. The same goes to this cluster as only one checked text mentioned compleately different topic.

- **Practicality:** Students' views on the hands-on applicability of the material. In this topic based on the examined answers there were three particular one that contained border line examples for the cluster. These answers were about how to improve the class, mainly with more practical examples.
- **Interactivity and Personal Opinions:** Feedback on the level of engagement and subjective student reflections. From the 10 evaluated examples there were no answer falling out of the topics scope.
- **Supplementary Materials:** The availability and quality of extra reading and study aids. The examined answere showed no outlier from the topic.
- **Traceability:** How easily students could maintain focus and follow the logic of the lecture. There were only one examined example that stated other then the one assigned. It particularly mentions lack of supporting material.
- **Course Structure:** Critiques regarding the logical organization and syllabus design. Each examined answer stated explicitly the course structure.
- **Instructor Knowledge:** Recognition of the teacher's expertise and professional command of the subject. From the 10 examined answers there were one falling out of the topics scope and could have been assigned into traceability.
- **Personal Experiences:** Anecdotal feedback regarding specific student–teacher interactions. Out of 10 answers 4 could have been assigned into practicality.
- **Material Accessibility and Slides:** Technical feedback regarding the distribution and quality of lecture slides. Each examines answers was correctly placed into the topic.
- **Real-world Examples:** The inclusion of industry-relevant or practical case studies. Out of the 10 examined answers 3 could have been assigned into practicality, 2 into Supplementary materials.

The discovery of these specific topics – such as the clear separation between "course structure" and "traceability" – demonstrates the system's ability to move beyond generic critiques. By mapping these topics to the previously analyzed sentiment scores, university management can identify not just that a course is viewed negatively, but specifically which aspect (e.g., the pace of the lecture or the lack of real-world examples) is driving the dissatisfaction.

The primary advantage of the proposed Aspect-Based Sentiment Analysis (ABSA) is the ability to map emotional valence directly onto the discovered latent topics. By projecting the sentiment scores onto the semantic manifold, we can identify not only the prevalence of specific themes but also their internal polarity distribution. As illustrated in Figure 3, the sentiment landscape reveals clear "pain points" and "success stories" within the institutional feedback.

The visualization demonstrates that certain topics, such as Course Structure is predominantly associated with positive (green) clusters, indicating it as consistent institutional strength. Conversely, topics related to Traceability show a higher density of negative (red) data points.

This granular mapping allows university management to move beyond global satisfaction metrics. For instance, while an instructor might receive an overall posi-

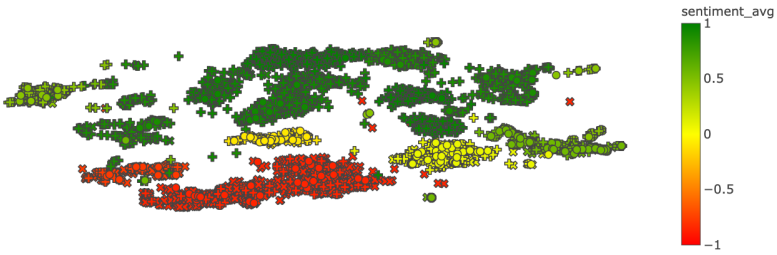


Figure 3. UMAP visualization of sentiment distribution across latent topics (within cluster sentiment average) (Green: Positive, Yellow: Neutral, Red: Negative).

tive rating, this analysis can pinpoint that the negative feedback is strictly isolated to the Traceability, rather than a lack of competence or clarity. Such precision is instrumental for targeted pedagogical interventions, as it provides a clear diagnostic tool for identifying which specific aspects of a course require immediate administrative or developmental attention.

4. Conclusion

Our study successfully established an automated pipeline for processing massive qualitative datasets within the Hungarian higher education framework. By leveraging the University of Szeged's 34,000 unique responses, the research proved that modern Large Language Models can effectively bridge the gap between vast unstructured data and precise administrative utility.

The research demonstrated the efficacy of high-parameter models like Llama-3-70B-Instruct in handling classification tasks with morphologically rich languages like Hungarian. We have also performed qualitative evaluation on the discovered topics and on their given name. Our proposed hybrid framework effectively pairs few-shot classification with unsupervised HDBSCAN clustering to capture both predefined institutional goals and emergent student concerns. The integration of Aspect-Based Sentiment Analysis allows the detection of nuanced, mixed feedback that global sentiment scores typically overlook.

The real-world application of this methodology offers immediate benefits to institutional quality assurance protocols. By automating a task that traditionally requires hundreds of manual labor hours allows university management to address pedagogical or infrastructural issues while they are still relevant to the current student cohort. Additionally, the objectivity provided by a standardized automated framework reduces the inherent bias of manual review, offering a reliable basis for faculty-level SWOT analyses and resource allocation. The unsupervised component of the pipeline is particularly valuable for discovering "hidden" systemic issues – such as specific technical failures or evolving student preferences – that remain invisible in standard quantitative Likert-scale ratings.

References

[1] A. Abdi, G. Sedrakyan, B. Veldkamp, J. van Hillegersberg, S. M. van den Berg: *Students feedback analysis model using deep learning-based method and linguistic knowledge for intelligent educational systems*, Soft Computing 27.19 (2023), pp. 14073–14094.

[2] AI@Meta: *Llama 3 Model Card* (2024), url: https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

[3] A. Baqach, A. Battou: *A new sentiment analysis model to classify students' reviews on MOOCs*, Education and Information Technologies 29.13 (2024), pp. 16813–16840.

[4] G. M. Csányi, D. Lakatos, I. Üveges, A. Megyeri, J. P. Vadász, D. Nagy, R. Vági: *From Fact Drafts to Operational Systems: Semantic Search in Legal Decisions Using Fact Drafts*, Big Data and Cognitive Computing 8.12 (2024), issn: 2504-2289, doi: 10.3390/bdcc8120185, url: https://www.mdpi.com/2504-2289/8/12/185.

[5] M. Hinek: *Témák és érzelmek–Szállodai vendégvélemények vizsgálata témamodellezéssel és szentimentelemzéssel* (2024).

[6] I. Keraghel, S. Morbieu, M. Nadif: *Beyond words: a comparative analysis of LLM embeddings for effective clustering*, in: International Symposium on Intelligent Data Analysis, Springer, 2024, pp. 205–216.

[7] L. McInnes, J. Healy, S. Astels, et al.: *hdbscan: Hierarchical density based clustering.* J. Open Source Softw. 2.11 (2017), p. 205.

[8] M. A. Mersha, J. Kalita, et al.: *Semantic-driven topic modeling using transformer-based embeddings and clustering algorithms*, Procedia Computer Science 244 (2024), pp. 121–132.

[9] D. Moulavi, P. A. Jaskowiak, R. J. Campello, A. Zimek, J. Sander: *Density-based clustering validation*, in: Proceedings of the 2014 SIAM international conference on data mining, SIAM, 2014, pp. 839–847.

[10] M. Osváth, G. Yang Zijian, K. Kósa: *Magyar páciensek narratív tapasztalatainak elemzése BERT témamodellezéssel és szentimentelemzéssel*, Berend Gábor–Gosztolya Gábor–Vincze Veronika (szerk.): XVIII. Magyar Számítógépes Nyelvészeti Konferencia. Szegedi Tudományegyetem TTIK Informatikai Intézet, Szeged (2022), pp. 311–324.

[11] *Qwen2 Technical Report* (2024).

[12] S. Srinivas, S. Rajendran: *Topic-based knowledge mining of online student reviews for strategic planning in universities*, Computers & Industrial Engineering 128 (2019), pp. 974–984.

[13] J. Sun, L. Yan: *Using topic modeling to understand comments in student evaluations of teaching*, Discover Education 2.1 (2023), p. 25.

[14] Y. Zhang, M. Li, D. Long, X. Zhang, H. Lin, B. Yang, P. Xie, A. Yang, D. Liu, J. Lin, F. Huang, J. Zhou: *Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models*, arXiv preprint arXiv:2506.05176 (2025).

Annales Mathematicae et Informaticae
63 (2026) pp. 55–64
DOI: 10.33039/ami.2026.06.008
URL: https://ami.uni-eszterhazy.hu

Attribute-based Bloom filter tabu search for the Flexible Flowshop Problem

Levente Áron Fazekas, Károly Nehéz

Institute of Information Technology, University of Miskolc
{levente.fazekas,karoly.nehez}@uni-miskolc.hu

Abstract. In realistic flexible flowshop scheduling, candidate solutions are evaluated through detailed simulation, making optimization fundamentally evaluation-limited. In this regime, classical tabu search becomes increasingly inefficient: exact memory structures grow over time, yet fail to prevent near-duplicate evaluations that induce almost identical simulation trajectories.

This paper reframes tabu memory as a constant-time rejection mechanism rather than an exact history structure. We first replace exact tabu memory with Bloom filters, yielding a fixed-size, negligible-cost filtering layer. We then extend this idea to attribute-based tabu, where multiple Bloom filters operate on structural features of solutions rather than their full representations.

The resulting multi-filter mechanism enables early rejection of both revisits and structurally similar candidates before simulation. Experimental results show that exact Bloom tabu closely reproduces classical behavior, while attribute-based filtering reduces the number of costly evaluations by over 20% on average. The results highlight a central design principle: in simulation-dominated optimization, efficiency is achieved not by storing more history, but by making evaluation rare.

Keywords: flexible flowshop scheduling, tabu search, Bloom filter, probabilistic memory, simulation-based optimization

2020 *Mathematics Subject Classification:* Primary 90B35; Secondary 68W20, 68P10

1. Introduction

The Flexible Flowshop Scheduling Problem (FFSP) extends the classical flowshop by allowing multiple parallel machines at each processing stage [5, 16, 18]. Realistic

Submitted: March 31, 2026
Accepted: June 30, 2026
Published online: July 17, 2026

variants incorporate release times, due dates, sequence-dependent setup times, machine eligibility, and multiple performance criteria such as makespan or tardiness [6, 12, 14, 15, 17]. Recent work shows that flexible and hybrid flowshop scheduling remains an active research area, including online batching variants, flexible-shop neighborhood search, multi-objective evolutionary methods, energy-aware blocking models, and parallel/distributed tabu-search variants [8, 10, 11, 13, 19].

In such settings, evaluating a candidate solution is no longer trivial. Instead, it requires a discrete-event simulation (DES) that models the dynamic behavior of the system. As a result, optimization becomes evaluation-limited: the dominant cost lies not in generating candidates, but in assessing them.

This shift has important consequences. Classical tabu search relies on maintaining an exact memory of visited solutions. However, when evaluation dominates runtime, this exact memory becomes problematic: it grows over time, incurs increasing overhead, and still fails to prevent near-duplicate candidates that produce almost identical simulation outcomes.

This paper proposes a different perspective. Instead of treating tabu memory as an exact record of the past, we treat it as a rejection layer placed in front of the simulator. From this viewpoint, the goal is not perfect recall, but efficient filtering.

The contributions of this work are threefold:

- we reinterpret tabu memory as a constant-time rejection mechanism;
- we show that Bloom filters provide an efficient fixed-size implementation of this idea; and
- we extend the approach to attribute-based tabu, enabling similarity-aware filtering before simulation.

2. Discrete-event simulation model

In the FFSP, each job must pass through S stages, where each stage contains a set of parallel machines. The evaluation of a candidate schedule is performed by a discrete-event simulation that models the flow of jobs through the system.

As illustrated in Figure 1, jobs flow through buffers between stages. At each stage, two decisions must be made: which job to process next (dispatching), and which machine to assign (machine selection). These decisions determine the resulting schedule and its performance.

The objective values are computed from completion times C_j and due dates d_j:

$$C_{\max} = \max_j C_j, \quad T_j = \max(0, C_j - d_j),$$
$$T_{\max} = \max_j T_j, \quad \sum_j T_j, \quad \sum_j U_j.$$

Because simulation captures dynamic interactions such as machine contention and setup dependencies, evaluating even a single candidate can be computationally expensive.

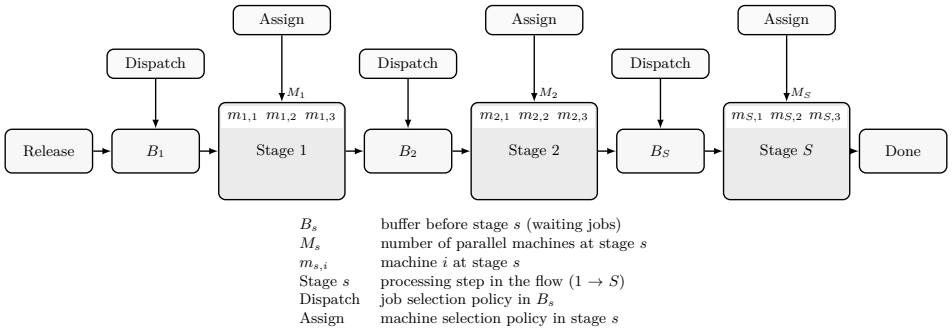


Figure 1. Discrete-event simulation model of the flexible flowshop. Jobs move through stage buffers, where dispatching and machine assignment decisions determine their progression.

3. Tabu search in the evaluation-limited regime

In a simulation-dominated setting, the role of tabu search must be reconsidered. This does not mean that tabu search itself is a weak baseline. On the contrary, tabu search is a long-established metaheuristic for combinatorial optimization and has repeatedly been compared with genetic algorithms, simulated annealing, iterated local search, GRASP, swarm-based methods, and other metaheuristics in scheduling and related optimization settings [2, 4, 7, 9, 15]. The aim of this paper is therefore more specific: instead of comparing tabu search as a whole against unrelated optimizers, we isolate the memory component of tabu search and ask whether its representation can be made cheaper and more informative in simulation-based scheduling.

Exact tabu memory prevents revisiting identical solutions, but this is often insufficient. The search space is vast, and the probability of generating exact duplicates is negligible. Instead, many distinct solutions produce nearly identical simulation trajectories, differing only in minor structural details.

From a computational perspective, these near-duplicates are the true source of inefficiency. They trigger expensive evaluations without contributing meaningful new information.

Thus, the fundamental question becomes: how can we prevent redundant evaluations, rather than exact revisits?

4. Bloom filters as tabu memory

Bloom filters provide a natural answer. They represent a set using a fixed-size bit array and multiple hash functions, enabling constant-time insertions and queries [1, 3].

Replacing exact tabu memory with a Bloom filter yields several advantages:

- constant-time operations independent of history size;
- fixed memory footprint;
- simple, cache-friendly implementation; and
- negligible overhead compared to simulation.

Bloom filters introduce false positives but no false negatives. In this context, a false positive simply skips a candidate early, which is often desirable.

The false-positive probability after n insertions is

$$p \approx \left(1 - e^{-kn/m}\right)^k,$$

with optimal $k \approx \frac{m}{n} \ln 2$.

For realistic parameter choices, even modest memory sizes yield extremely low error rates. More importantly, the filtering cost remains negligible compared to simulation.

This transforms tabu memory from a growing data structure into a lightweight rejection mechanism.

5. Attribute-based tabu filtering

The key extension of this work is to move beyond exact representations. Let $\phi_1(x), \ldots, \phi_q(x)$ denote structural features of a candidate solution, such as relative job orderings or stage-level patterns. Instead of storing full solutions, we maintain multiple Bloom filters over these attributes.

In the benchmark configuration, the multi-filter tabu memory consists of a single exact Bloom filter combined with a set of attribute-based filters. The exact component employs a 64-bit permutation hash and a Bloom filter of size 2^{15} with $k = 3$ hash probes.

The attribute layer uses the same Bloom filter parameters (2^{15} bits, $k = 3$) for each filter and encodes multiple structural views of the solution. These include stage-wise quantized processing-time sequence hashes, stage-wise position-bucketed processing-time histograms, an adjacent-pairs permutation signature, and a sampled inversion-order signature. For the five-stage instances considered, this results in a total of twelve attribute filters.

A candidate solution is rejected by the attribute layer if at least six of these filters indicate membership.

The tabu predicate becomes

$$\mathrm{Tabu}(x) = B_0(h(x)) \vee \big(B_1(\phi_1(x)) \wedge \cdots \wedge B_q(\phi_q(x))\big).$$

This formulation allows the search to reject not only exact revisits, but also structurally similar candidates. Importantly, all filtering occurs before simulation.

Conceptually, this represents a shift from representation-based memory to feature-based memory. Hashing becomes a mechanism for extracting structural signatures, and membership queries become similarity tests.

The effect is to reduce redundant evaluations, allowing computational effort to focus on genuinely new regions of the search space.

6. Why attribute-based filtering works

The effectiveness of attribute-based tabu filtering can be understood by examining the relationship between solution representations and their induced simulation trajectories.

Let x denote a candidate solution and let $\mathcal{S}(x)$ denote the outcome of its discrete-event simulation, including all relevant performance measures. In evaluation-limited settings, the optimization process operates not directly on x, but on $\mathcal{S}(x)$.

A key observation is that the mapping $x \mapsto \mathcal{S}(x)$ is many-to-one. Distinct candidate solutions often induce highly similar, or even identical, simulation trajectories. Formally, there exists an equivalence relation $\sim$ over the solution space such that

$$x \sim y \quad \Longrightarrow \quad \mathcal{S}(x) \approx \mathcal{S}(y),$$

where $\approx$ denotes equality or near-equality of objective values.

From this perspective, the search space is effectively partitioned into equivalence classes of solutions that behave similarly under simulation. Classical tabu search operates on exact representations and therefore treats all elements of an equivalence class as distinct, potentially exploring many representatives of the same class.

Attribute-based filtering provides a practical approximation of this equivalence structure. Let $\phi(x)$ denote a vector of structural features extracted from x. If the attribute mapping satisfies

$$\phi(x) = \phi(y) \quad \Longrightarrow \quad \mathcal{S}(x) \approx \mathcal{S}(y),$$

then ϕ acts as a coarse-grained representation of simulation behavior.

In this setting, tabu filtering over attributes does not merely prevent revisits in representation space, but instead suppresses redundant exploration of entire equivalence classes. The Bloom filters associated with attributes can therefore be interpreted as probabilistic caches over regions of the simulation space.

This interpretation clarifies both the strengths and limitations of the approach. When the attribute mapping aligns well with the structure of the simulation, redundant evaluations are effectively eliminated. However, if the mapping is too coarse, distinct high-quality solutions may be grouped together and prematurely excluded.

The central design problem is therefore to construct attribute mappings that preserve meaningful distinctions in simulation outcomes while collapsing irrelevant variation in representation space. In this sense, attribute-based tabu filtering can be viewed as a form of feature engineering for simulation-based optimization.

7. Experimental evaluation

The proposed methods were evaluated on a set of benchmark instances spanning multiple problem sizes and structural scenarios. Three tabu-memory variants were compared: classical exact memory, exact Bloom filtering, and attribute-based multi-filter tabu.

The results highlight a clear separation between implementation-level and policy-level effects. Exact Bloom filtering closely reproduces the behavior of classical tabu search, confirming that it can replace exact memory without loss of effectiveness while providing a fixed-size, constant-time alternative.

In contrast, the attribute-based approach alters the search dynamics. Across all tested instances, it consistently reduces the number of evaluated candidates, achieving an average reduction of 21.6%. This effect becomes more pronounced with increasing problem size, reflecting the growing prevalence of redundant evaluations in larger search spaces.

The impact on solution quality is more nuanced. In some cases, attribute filtering improves results by avoiding unproductive regions of the search space. In others, overly aggressive filtering excludes promising candidates. This behavior underscores the importance of attribute design and suggests the need for adaptive or problem-aware filtering strategies.

7.1. Evaluation-count reduction

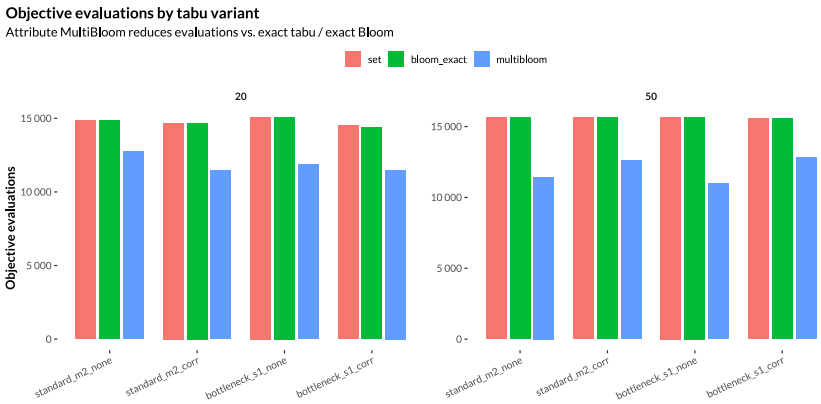
Table 1 reports the number of simulated candidates for each benchmark instance. The multi-filter attribute-based variant reduces the evaluation count in every comparison. The reduction ranges from 14.3% to 29.8%, with an overall average of 21.6%.

A clear scaling effect is observed: for $n = 20$, the average reduction is 19.6%, while for $n = 50$ it increases to 23.5%. This supports the interpretation that attribute-based filtering becomes increasingly effective as the density of simulation-equivalent candidates grows with problem size.

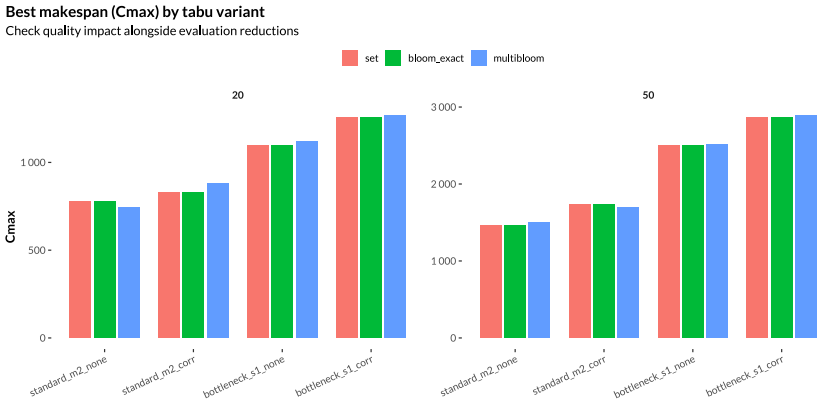
Table 1. Number of evaluated candidates. Reduction is computed relative to the classical set-based tabu memory.

n	Case	Set	Exact Bloom	Multi-Bloom	Reduction (%)
20	standard / no corr.	14866	14866	12745	14.3
20	standard / corr.	14684	14684	11472	21.9
20	bottleneck / no corr.	15089	15089	11850	21.5
20	bottleneck / corr.	14495	14361	11484	20.8
50	standard / no corr.	15661	15661	11438	27.0
50	standard / corr.	15664	15664	12610	19.5
50	bottleneck / no corr.	15671	15671	10995	29.8
50	bottleneck / corr.	15581	15581	12822	17.7

The same trend is visible in Figure 2, where the attribute-based method consistently lowers the number of expensive simulation calls. This reinforces the interpretation of the method as a pre-simulation filtering layer that removes redundant candidates before they incur computational cost.



(a) Evaluation count comparison.



(b) $C_{\max}$ comparison.

Figure 2. Benchmark plots illustrating evaluation counts and makespan across tabu-memory variants.

7.2. Solution-quality trade-off

The reduction in evaluation effort does not translate into uniform improvements in solution quality. Table 2 summarizes the relative change of the multi-filter method compared to the classical baseline, where negative values indicate improvement.

Two cases show clear improvements. For `standard_m2_none` with $n = 20$, the multi-filter method reduces evaluations by 14.3% while improving makespan

Table 2. Relative change of the multi-filter tabu memory against the classical set baseline. Entries marked *n/a* correspond to zero baseline tardiness values.

n	Case	$\Delta C_{\max}$	$\Delta \sum T$	$\Delta T_{\max}$	$\Delta \sum U$
20	standard / no corr.	−4.0%	n/a	n/a	+0
20	standard / corr.	+6.5%	+13.8%	+7.7%	+0
20	bottleneck / no corr.	+1.9%	n/a	n/a	+0
20	bottleneck / corr.	+0.6%	+0.5%	+2.1%	+0
50	standard / no corr.	+2.5%	+83.3%	+31.1%	+2
50	standard / corr.	−1.7%	−3.7%	−2.0%	+0
50	bottleneck / no corr.	+0.5%	+25.7%	+0.9%	+4
50	bottleneck / corr.	+0.6%	+1.2%	+0.6%	+0

by 4.0%. For `standard_m2_corr` with $n = 50$, it achieves a 19.5% reduction in evaluations while improving all reported time-based objectives.

In the remaining cases, however, the method exhibits a trade-off: fewer evaluations at the cost of slightly degraded performance. Across the eight matched instances, the multi-filter variant improves $C_{\max}$ in two cases and worsens it in six; it improves $\sum T$ in one case, worsens it in five, and leaves it unchanged in two.

These results reinforce the interpretation developed earlier. Exact Bloom filtering acts as an implementation-level improvement, preserving the behavior of classical tabu search while reducing overhead. In contrast, attribute-based multi-filter tabu modifies the effective search space by suppressing entire regions of simulation-equivalent solutions. Its benefit is most pronounced when evaluation cost dominates, but its impact on solution quality depends critically on how well the chosen attributes approximate meaningful equivalence classes in the simulation space.

This also clarifies the relationship between the proposed method and the broader tabu-search literature. Since tabu search has already been benchmarked extensively against other metaheuristics, the present comparison is deliberately orthogonal: it asks how much of the computational burden can be removed before the simulator is called. Exact Bloom tabu demonstrates that the classical tabu mechanism can be approximated by a fixed-size probabilistic memory, while attribute-based tabu shows that the same memory idea can be lifted from equality-based revisits to simulation-relevant structural similarity.

8. Discussion

The results reveal two distinct roles for Bloom filters in tabu search.

First, as an implementation tool, they provide a fixed-size, constant-time alternative to exact memory, eliminating bookkeeping overhead.

Second, as a modeling tool, they enable a new form of tabu logic based on structural similarity rather than exact equality.

In evaluation-limited optimization, the second role is particularly important. Reducing the number of simulations by even a modest percentage can translate into significantly deeper or broader exploration under a fixed time budget.

Future work should focus on adaptive attribute selection, dynamic filter management, and integration with aspiration criteria.

9. Conclusions

This paper proposed a reinterpretation of tabu search for simulation-based optimization.

Instead of maintaining exact history, tabu memory should act as a lightweight rejection layer that minimizes unnecessary evaluations. Bloom filters provide an efficient implementation of this idea, while attribute-based extensions enable similarity-aware filtering.

The experimental results demonstrate that this approach significantly reduces evaluation effort while preserving competitive solution quality.

The central insight is simple: in evaluation-dominated optimization, efficiency comes not from remembering everything, but from evaluating less.

References

[1] P. S. Almeida, C. Baquero, N. Preguiça, D. Hutchison: *Scalable Bloom Filters*, Information Processing Letters 101.6 (2007), pp. 255–261, ISSN: 0020-0190, DOI: 10.1016/j.ipl.2006.10.007, URL: https://www.sciencedirect.com/science/article/pii/S0020019006003127.

[2] K. A. G. Araújo, E. G. Birgin, D. P. Ronconi: *Local search and trajectory metaheuristics for the flexible job shop scheduling problem with sequencing flexibility and position-based learning effect*, 2024, arXiv: 2403.16787 [math.OC], URL: https://arxiv.org/abs/2403.16787.

[3] B. H. Bloom: *Space/time trade-offs in hash coding with allowable errors*, Commun. ACM 13.7 (July 1970), pp. 422–426, ISSN: 0001-0782, DOI: 10.1145/362686.362692.

[4] A. Bouketta: *Hybrid Metaheuristic Combining the Dragonfly Algorithm and Tabu Search for the Traveling Salesman Problem*, 2026, arXiv: 2606.09529 [cs.NE], URL: https://arxiv.org/abs/2606.09529.

[5] P. Brucker: *Scheduling algorithms*, 4th ed., Springer Berlin, Heidelberg, 1999, pp. XII, 367, ISBN: 978-3-540-24804-0, DOI: 10.1007/978-3-540-24804-0.

[6] L. A. Fazekas, K. Nehez: *Multi-objective genetic and memetic algorithms in flexible flow-shop scheduling*, in: Annales Mathematicae et Informaticae, vol. 61, 2025, pp. 94–107, DOI: 10.33039/ami.2025.10.011.

[7] F. Glover: *Tabu search: A tutorial*, Interfaces 20.4 (1990), pp. 74–94, DOI: 10.1287/inte.20.4.74.

[8] C. Hertrich, C. Weiss, H. Ackermann, S. Heydrich, S. O. Krumke: *Online Algorithms to Schedule a Proportionate Flexible Flow Shop of Batching Machines*, 2020, arXiv: 2005.03552 [cs.DS], URL: https://arxiv.org/abs/2005.03552.

[9] E. Hosseini: *Tabu Search and Simulated Annealing metaheuristic algorithms applied to the RoRo vessel stowage problem*, 2023, arXiv: 2301.04468 [cs.AI], URL: https://arxiv.org/abs/2301.04468.

[10] A. Janiak, D. Kowalczyk, M. Lichtenstein: *Parallel/Distributed Tabu Search for Scheduling Microprocessor Tasks in Hybrid Flowshop*, 2025, arXiv: 2509.11396 [cs.DC], url: https://arxiv.org/abs/2509.11396.

[11] F. Liu, X. Li, C. Lu, W. Gong: *Adaptive Knowledge-based Multi-Objective Evolutionary Algorithm for Hybrid Flow Shop Scheduling Problems with Multiple Parallel Batch Processing Stages*, 2024, arXiv: 2409.18524 [cs.NE], url: https://arxiv.org/abs/2409.18524.

[12] K. Mihaly, G. Kulcsar: *A new many-objective hybrid method to solve scheduling problems*, International Journal of Industrial Engineering and Management 14.4 (2023), pp. 326–335, doi: 10.24867/IJIEM-2023-4-342.

[13] A. Missaoui, C. Ozturk, B. O'Sullivan: *Refined Iterated Pareto Greedy for Energy-aware Hybrid Flowshop Scheduling with Blocking Constraints*, 2025, arXiv: 2510.03377 [cs.AI], url: https://arxiv.org/abs/2510.03377.

[14] B. Naderi, M. Zandieh, V. Roshanaei: *Scheduling hybrid flowshops with sequence dependent setup times to minimize makespan and maximum tardiness*, The International Journal of Advanced Manufacturing Technology 41 (2009), pp. 1186–1198, doi: 10.1007/s00170-008-1569-3.

[15] C. Oğuz, M. F. Ercan: *A Genetic Algorithm for Hybrid Flow-shop Scheduling with Multiprocessor Tasks*, Journal of Scheduling 8.4 (July 2005), pp. 323–351, issn: 1099-1425, doi: 10.1007/s10951-005-1640-y.

[16] M. L. Pinedo: *Scheduling, Theory, Algorithms, and Systems*, 6th ed., Springer Cham, 2012, pp. XVII, 698, isbn: 978-3-031-05920-9, doi: 10.1007/978-3-031-05921-6.

[17] I. Ribas, R. Leisten, J. M. Framiñan: *Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective*, Computers & Operations Research 37.8 (2010), Operations Research and Data Mining in Biological Systems, pp. 1439–1454, issn: 0305-0548, doi: 10.1016/j.cor.2009.11.001, url: https://www.sciencedirect.com/science/article/pii/S0305054809002883.

[18] R. Ruiz, J. A. Vázquez-Rodríguez: *The hybrid flow shop scheduling problem*, European Journal of Operational Research 205.1 (2010), pp. 1–18, issn: 0377-2217, doi: 10.1016/j.ejor.2009.09.024, url: https://www.sciencedirect.com/science/article/pii/S0377221709006390.

[19] J. C. Seck-Tuoh-Mora, N. J. Escamilla-Serna, J. Medina-Marin, N. Hernandez-Romero, I. Barragan-Vite, J. R. Corona-Armenta: *A global-local neighborhood search algorithm and tabu search for flexible job shop scheduling problem*, 2020, arXiv: 2010.12702 [cs.AI], url: https://arxiv.org/abs/2010.12702.

Annales Mathematicae et Informaticae
63 (2026) pp. 65–72
DOI: 10.33039/ami.2026.06.006
URL: https://ami.uni-eszterhazy.hu

Baseline review of Formal Methods and Foundations of Artificial Intelligence

Gábor Kusper

Eszterházy Károly Catholic University
kusper.gabor@uni-eszterhazy.hu

Abstract. Artificial Intelligence (AI) is advancing rapidly, yet many successful models remain opaque and provide limited assurance about reliability, safety, and failure modes. This motivates renewed interest in formal methods and foundational perspectives that can support trustworthy AI beyond empirical testing. This paper presents a baseline review of the selected papers volume of the inaugural International Conference on Formal Methods and Foundations of Artificial Intelligence (FMF-AI 2025), published as *Annales Mathematicae et Informaticae*, Vol. 61 (2025). The goal is twofold: (i) to map and summarize the first FMF-AI "snapshot" as a starting point for the Hungarian research ecosystem, and (ii) to define a reproducible baseline that can serve as a reference for measuring topical and methodological shifts in subsequent FMF-AI editions. The review clusters the twenty selected papers into five thematic groups and records their relative prevalence. In addition, it introduces simple baseline metrics that can be recomputed in future FMF-AI editions to observe structural changes in the research landscape. The main pattern is a clear imbalance between verification-oriented contributions and papers that primarily use AI methods in application or optimization contexts.

Keywords: FMF-AI, baseline review, Formal Methods and Foundations of Artificial Intelligence

2020 *Mathematics Subject Classification:* Primary 00A17, 68-02; Secondary 68T01

1. Introduction

Artificial Intelligence (AI) has achieved remarkable empirical success across a wide range of domains, yet this success is often accompanied by limited transparency,

Submitted: March 31, 2026
Accepted: June 30, 2026
Published online: July 17, 2026

weak interpretability, and uncertain failure modes. As AI systems become more pervasive, questions of reliability, accountability, and explanatory adequacy become increasingly central. In this setting, formal methods and foundational inquiry offer a complementary perspective: rather than relying exclusively on benchmark performance and empirical testing, they seek analyzable models, explicit assumptions, and, in some cases, guarantees about system behavior.

This perspective is particularly relevant for the International Conference on Formal Methods and Foundations of Artificial Intelligence (FMF-AI). The vision articulated in the conference foreword presents FMF-AI as a meeting point between theoreticians and practitioners, between formal reasoning and real-world AI applications, and between technical performance and the broader demand for responsible and explainable AI. It also frames the conference series as a growing community effort, rooted in collaboration, curiosity, and shared knowledge, and intended to develop into a recurring international forum.

FMF-AI 2025 produced two complementary publication outputs. In addition to the selected-papers special issue published in *Annales Mathematicae et Informaticae*, Vol. 61 (2025), the conference also resulted in a separate proceedings volume available on the conference website[1]. This dual publication context matters for the present study. The proceedings reflect the broader interdisciplinary profile of the event, spanning topics from mathematical foundations and large language models to educational and application-oriented themes, whereas the selected-papers volume offers a more compact and curated snapshot of contributions that are especially suitable for baseline comparison.

The purpose of this paper is not to rank the included papers or to evaluate them against a single technical standard. Instead, it provides a descriptive baseline review of the twenty selected papers published in the special issue. The central question is simple: when the first FMF-AI selected volume is viewed as a whole, what structural picture emerges from it, and what does that picture reveal about the balance between formal-methods-oriented work and the broader use of AI across domains?

More specifically, the review has two objectives. First, it maps and summarizes the thematic landscape of the FMF-AI 2025 selected papers. Second, it defines a small set of reproducible baseline descriptors that can later be recomputed for subsequent FMF-AI editions. The underlying assumption is that even simple counts and stable category assignments become informative when they are applied consistently over time.

The contribution of the paper is therefore threefold. It proposes a stable five-group thematic organization of the selected volume; it derives simple baseline metrics from this organization; and it offers an initial interpretation of the relationship between verification-oriented contributions and application-oriented AI research. The paper remains intentionally descriptive: its primary goal is to establish a usable reference point for later comparative work, particularly within the Hungarian research ecosystem to which several contributions in the volume are directly con-

[1] https://uni-eszterhazy.hu/fmf2025/proceedings

nected.

The remainder of the paper is organized as follows. Section 2 explains the scope and review design. Section 3 presents the five thematic groups. Section 4 summarizes the baseline metrics and their interpretation. Section 5 discusses limitations and transparency issues, and Section 6 concludes the paper.

2. Scope and review design

This review covers the twenty selected papers published in the FMF-AI 2025 volume of *Annales Mathematicae et Informaticae*. The paper adopts a descriptive, taxonomy-driven scoping-review mindset. Its goal is not exhaustive literature synthesis beyond the volume, but the creation of a reproducible baseline for this particular corpus.

For each paper, the review considers four descriptive dimensions: the addressed problem, the dominant method family, the main form of evidence, and the paper's explicit connection to trustworthy AI. The method family may be learning-based, optimization-oriented, symbolic or formal, or hybrid. The evidence may rely mainly on empirical evaluation, survey or user-study evidence, theoretical analysis, or formal guarantees. The trustworthy-AI connection may concern robustness, assurance, verification, safety-related reasoning, or a weaker indirect relation.

In the present baseline version, the primary emphasis is on thematic clustering and on a few structural metrics that can be computed in a straightforward and reproducible way. Since some papers could plausibly fit more than one perspective, each contribution is assigned to one primary thematic group. This choice sacrifices some nuance, but it enables stable paper counts and clear comparisons across later FMF-AI editions.

3. Thematic structure of the FMF-AI 2025 Volume

The selected papers can be organized into five thematic groups. These groups do not claim to be the only possible taxonomy; rather, they provide a stable descriptive structure that is broad enough to cover the whole volume while still highlighting meaningful differences among the contributions.

The number of groups was fixed at five for pragmatic methodological reasons. With only twenty papers, a finer taxonomy would easily produce several one-paper or two-paper categories, making the baseline unstable and difficult to compare across future editions. Conversely, a smaller taxonomy with only two or three groups would merge substantially different types of AI-related work, for example language technology, educational applications, and optimization-oriented contributions. The five-group structure therefore represents a compromise between interpretability, coverage, and future reusability.

3.1. Group I: Formal Verification of AI

Group I contains a single paper that explicitly combines empirical analysis with *formal verification* of robustness properties in neural network ensembles [14]. Within the selected volume, this contribution is the clearest realization of the conference's promise to connect formal methods with AI reliability. Its singular position is analytically important, because it marks the narrowest and most explicit core of what may be called verification-oriented FMF-AI research.

3.2. Group II: Testing and Reliability of AI

Group II gathers papers that address reliability concerns through empirical and engineering-oriented approaches rather than full formal verification. The covered topics include motion-enhanced video anomaly detection, AI-based interpretation of static human arm signals, comparative robustness and noise-sensitivity analysis in deep neural networks, and fault diagnosis based on textual system logs [2, 4, 11, 13]. These papers are closely related to dependability and trustworthy AI, but their contribution is primarily testing-, diagnosis-, or robustness-oriented.

3.3. Group III: Using AI for Text Analytics

Group III covers language- and text-related applications of AI. The papers in this group address toxic comment detection in Hungarian, syntactic comparison between human-written and AI-generated scientific texts, and the enhancement of a Hungarian conversational model [9, 19, 20]. Taken together, these contributions reflect both local-language priorities and broader questions raised by generative AI and language technologies.

3.4. Group IV: Using AI for Education

Group IV captures the strong educational and human-centered dimension of the volume. It includes automated fair team formation for STEAM activities using SMT, adaptive testing for programming proficiency, AI-based robotic applications in higher education, the impact of LLM use on algorithmic thinking, and a survey of AI-usage attitudes among Hungarian informatics students [1, 3, 6, 12, 15]. This group is especially important because it shows that FMF-AI 2025 is not restricted to technical verification questions; it also engages with pedagogy, human factors, and societal uptake.

3.5. Group V: Using AI for Optimization

Group V is the largest thematic cluster. It brings together papers in which AI, optimization, graph algorithms, or broader computational modeling play the central role. The topics include graph neural networks for refactoring P4 programs, the transition from rule-based to learned behaviors, multi-objective evolutionary scheduling, team coordination on graphs under risk, probabilistic models for sports

betting strategies, an improved Industry 4.0 reference architecture model, and community detection through overlapping label propagation [5, 7, 8, 10, 16–18]. Although these papers differ substantially in application area and methodological emphasis, they share the common characteristic that AI or computational intelligence is used as a constructive tool for solving domain problems.

4. Baseline metrics and discussion

The five-group taxonomy can be summarized with a simple paper-count distribution shown in Table 1. Even this compact representation already reveals an interpretable structural pattern.

Table 1. Baseline distribution of the FMF-AI 2025 selected papers.

Thematic group	Number of papers
Group I: Formal Verification of AI	1
Group II: Testing and Reliability of AI	4
Group III: Using AI for Text Analytics	3
Group IV: Using AI for Education	5
Group V: Using AI for Optimization	7
Total	20

A first aggregate metric emerges when Groups I and II are merged into a broader *verification-and-testing* side of the volume, while Groups III–V are merged into a broader *using-AI* side. Under this aggregation, the distribution becomes 5 versus 15, that is, a ratio of 1 : 3.

This ratio should be interpreted carefully. On the one hand, it indicates that the selected FMF-AI 2025 volume is dominated by papers that use AI methods in application, education, language, or optimization settings. On the other hand, it also shows that work directly addressing reliability is present, but concentrated mostly in empirical testing and robustness analysis rather than in explicit formal verification.

The most striking baseline observation is therefore that, despite the conference title, *explicit* formal verification appears in only one selected paper. This does not imply that the remaining papers are disconnected from trustworthy AI. Rather, it suggests that the strongest formal-methods interpretation of the conference theme is currently represented by a relatively small subset of the volume, while a much broader set of papers explores how AI can be applied, adapted, or evaluated in diverse contexts.

This imbalance should not necessarily be interpreted as a lack of relevance of formal methods. A more plausible interpretation is that rigorous verification techniques remain difficult to apply at scale in modern AI settings, whereas empirical testing, robustness studies, and application-driven AI methods are easier to deploy

across a broader variety of problems. In this sense, the present paper count distribution provides a useful baseline against which future FMF-AI volumes can be compared.

5. Limitations and transparency

The present review is intentionally narrow in scope. First, it examines only one selected-papers volume, so its observations should be read as a baseline description rather than as a mature characterization of the whole FMF-AI research direction. Second, the thematic grouping uses a single primary assignment for each paper, even though some contributions could reasonably belong to more than one category. Third, the current version focuses mainly on thematic distribution and a small number of structural metrics; later versions may add fuller coding of method families, evidence types, and trustworthy-AI dimensions.

A transparency issue should also be noted. The author is a co-author of two papers in the reviewed volume [6, 15]. For this reason, the present review deliberately avoids ranking, endorsement, or evaluative scoring of individual contributions. Its aim is descriptive organization rather than comparative judgment.

6. Conclusion and future work

This paper transforms the FMF-AI 2025 selected volume from a collection of individual papers into a structured baseline that can support later comparison. The main outcome is a stable five-group thematic map together with a small set of reproducible metrics, most notably the paper-count distribution across groups and the $1:3$ ratio between the verification-and-testing side and the broader using-AI side.

Beyond the descriptive contribution, the paper also highlights a substantive tension already visible in the first FMF-AI snapshot: the conference theme strongly invokes formal methods and foundations, yet the selected volume is much more strongly populated by papers that use AI in applied, educational, linguistic, and optimization-oriented contexts than by papers that perform explicit formal verification. Whether this pattern will persist or change in later editions is an open and interesting question.

Future work will extend the present baseline in at least two directions. First, the coding scheme can be refined by systematically analyzing method families, evidence types, and stronger or weaker forms of trustworthy-AI linkage. Second, the same taxonomy and baseline metrics can be recomputed for subsequent FMF-AI volumes, allowing the series to be studied longitudinally rather than only descriptively at a single point in time.

References

[1] A. A. Adil, G. Kovásznai, M. Khalid: *Automated fair team formation in STEAM activities using Satisfiability Modulo Theories (SMT)*, Annales Mathematicae et Informaticae 61 (2025), pp. 1–14, doi: 10.33039/ami.2025.10.001.

[2] M. I. Almurumudhe, O. Hornyák: *Motion enhanced video anomaly detection using masked autoencoder and hybrid loss functions*, Annales Mathematicae et Informaticae 61 (2025), pp. 15–30, doi: 10.33039/ami.2025.10.015.

[3] A. Apró, T. Tajti: *An adaptive testing system for programming proficiency using Item Response Theory*, Annales Mathematicae et Informaticae 61 (2025), pp. 31–42, doi: 10.33039/ami.2025.10.018.

[4] M. Z. Bagladi: *Artificial Intelligence for interpreting static human arm signals*, Annales Mathematicae et Informaticae 61 (2025), pp. 43–54, doi: 10.33039/ami.2025.10.005.

[5] B. S. Csüllög, M. Tejfel: *Using GNN for refactoring P4 programs*, Annales Mathematicae et Informaticae 61 (2025), pp. 55–67, doi: 10.33039/ami.2025.10.021.

[6] I. Drămnesc, E. Ábrahám, L. Antal, P. Csóka, N. Fachantidis, T. Jebelean, G. Kusper, K. Papadopoulos: *Artificial Intelligence based robotic applications for higher education*, Annales Mathematicae et Informaticae 61 (2025), pp. 68–79, doi: 10.33039/ami.2025.10.009.

[7] R. Erdész, E. Troll: *Making the boss smarter: A journey from rule-based to learned behaviors*, Annales Mathematicae et Informaticae 61 (2025), pp. 80–93, doi: 10.33039/ami.2025.10.017.

[8] L. Á. Fazekas, K. Nehéz: *Multi-objective genetic and memetic algorithms in flexible flow-shop scheduling*, Annales Mathematicae et Informaticae 61 (2025), pp. 94–107, doi: 10.33039/ami.2025.10.011.

[9] P. Hatvani, Z. G. Yang: *Automated detection of toxic comments in Hungarian*, Annales Mathematicae et Informaticae 61 (2025), pp. 108–117, doi: 10.33039/ami.2025.10.007.

[10] A. Izsó, I. Harmati: *Solving the Team Coordination on Graphs With Risky Edges problem for nonzero self-loop weights*, Annales Mathematicae et Informaticae 61 (2025), pp. 118–128, doi: 10.33039/ami.2025.10.008.

[11] M. A. Khudhair, A. Fazekas: *A comparative study on the noise sensitivity of binary classification based on robust deep neural networks*, Annales Mathematicae et Informaticae 61 (2025), pp. 129–140, doi: 10.33039/ami.2025.10.006.

[12] S. Király, E. Troll: *Algorithmic thinking at risk? Exploring LLM use in computer science education*, Annales Mathematicae et Informaticae 61 (2025), pp. 141–155, doi: 10.33039/ami.2025.10.020.

[13] Á. Kiss, K. Nehéz, O. Hornyák: *AI-driven fault diagnosis from textual system logs*, Annales Mathematicae et Informaticae 61 (2025), pp. 156–170, doi: 10.33039/ami.2025.10.003.

[14] Á. Kovács, R. Gunics, G. Kovásznai, T. Tajti: *Soft voting robustness in neural network ensembles with empirical analysis and formal verification*, Annales Mathematicae et Informaticae 61 (2025), pp. 171–185, doi: 10.33039/ami.2025.10.002.

[15] G. Kusper, G. I. Mátyás, T. Balla: *We are not afraid of the wolf! – AI usage attitudes among Hungarian informatics students*, Annales Mathematicae et Informaticae 61 (2025), pp. 186–201, doi: 10.33039/ami.2025.10.004.

[16] J. G. Pál, C. Bíró: *Evaluating profitability in sports betting using probabilistic models and betting strategies*, Annales Mathematicae et Informaticae 61 (2025), pp. 202–214, doi: 10.33039/ami.2025.10.016.

[17] I. Sima, D.-M. Cristea, E.-M. Ciortea, L. B. Iantovics: *cRAMI 4.0 an Improved Reference Architectural Model for Industrie 4.0 (RAMI 4.0) based on a three-dimensional cubic lattice model*, Annales Mathematicae et Informaticae 61 (2025), pp. 215–228, doi: 10.33039/ami.2025.10.023.

[18] S. P. Tahalea, M. Krész: *Betweenness-driven overlapping label propagation community detection*, Annales Mathematicae et Informaticae 61 (2025), pp. 229–247, doi: 10.33039/ami.2025.10.012.

[19] E. B. Varga, A. Baksa: *Syntactic comparison of human and AI-written scientific texts*, Annales Mathematicae et Informaticae 61 (2025), pp. 248–260, doi: 10.33039/ami.2025.10.013.

[20] Z. G. Yang, Á. Bánfi, R. Dodé, G. Ferenczi, F. Földesi, P. Hatvani, E. Héja, M. Lengyel, G. Madarász, M. Osváth, B. Sárossy, K. Varga, T. Váradi, G. Prószéky, N. Ligeti-Nagy: *ChatPULI: Enhancement to the first Hungarian conversational model*, Annales Mathematicae et Informaticae 61 (2025), pp. 261–274, doi: 10.33039/ami.2025.10.010.

Annales Mathematicae et Informaticae
63 (2026) pp. 73–87
DOI: 10.33039/ami.2026.06.005
URL: https://ami.uni-eszterhazy.hu

Introducing TAIPO, an AI-based product owner assistant for vibe coding*†

Gábor Kusper[a], Judit Szabó[a], Márk Szabó[a], Ádám Kovács[a], Tibor Tajti[a], Csaba Szabó[b], Ján Perháč[b], Marek Horváth[b], Branislav Sobota[b]

[a]Eszterházy Károly Catholic University
{kusper.gabor,szabo.judit,szabo.mark,kovacs2.adam,tajti.tibor}@uni-eszterhazy.hu

[b]Technical University of Košice
{csaba.szabo,jan.perhac,marek.horvath,branislav.sobota}@tuke.sk

Abstract. Vibe coding promises rapid progress by letting developers rely heavily on generative AI during implementation. In collaborative and educational settings, however, this speed also creates pressure on requirement traceability, prioritization, and change management. We introduce TAIPO (The AI-based Product Owner), an AI-supported product owner assistant that embeds generative AI into a Kanban workflow instead of treating it as a separate chat tool. TAIPO stores AI-generated clarifications directly on cards, generates prioritized backlog items from requirement text, decomposes stories into implementable tasks, supports refinement and acceptance-related feedback, and can proactively simulate product-owner interventions through contextual comments and unexpected change requests. The prototype is implemented as a web-based system with a PHP backend, a Vue 3 frontend, and Gemini-based AI services. It is grounded in a cleaned version of the TAWOS issue-tracking dataset in order to support more realistic project dynamics. We position TAIPO as an environment for structured vibe coding rather than as a pure code generator, with a particular focus on software engineering education where one instructor may need to support multiple student teams simultaneously.

*This project, i.e., Project no. 2024-1.2.5-TÉT-2024-00072 has been implemented with the support provided by the Ministry of Culture and Innovation of Hungary from the National Research, Development and Innovation Fund, financed under the 2024-1.2.5-TÉT funding scheme.

†This work was also supported by the Slovak Research and Development Agency under the Contract no. SK-HU-24-0037.

Submitted: March 31, 2026
Accepted: June 30, 2026
Published online: July 17, 2026

Keywords: vibe coding, generative AI, product owner assistant, Kanban, software engineering education

2020 *Mathematics Subject Classification:* Primary 68T05, 68U35; Secondary 68M99

1. Introduction

Vibe coding is an emerging approach to programming that emphasizes creativity, flow, and collaboration with AI assistants. AI-assisted programming began to spread rapidly in 2022, as many developers started using Large Language Models (LLMs) to generate, explain, debug, and refactor code. However, the specific concept of *vibe coding* emerged later as a distinct way of describing a more fluid, AI-mediated style of software development.

The term "vibe coding" was introduced to broad public attention by Andrej Karpathy in an X post on February 2, 2025. After this post, the expression spread rapidly and became widely used to describe a style of development in which programmers rely heavily on generative AI to move quickly from idea to implementation while staying in a creative flow. Karpathy described his experience as follows [7]:

> There's a new kind of coding I call "vibe coding", where you fully give in to the vibes, embrace exponentials, and forget that the code even exists. It's possible because the LLMs (e.g. Cursor Composer w Sonnet) are getting too good. Also I just talk to Composer with SuperWhisper so I barely even touch the keyboard. I ask for the dumbest things like "decrease the padding on the sidebar by half" because I'm too lazy to find it. I "Accept All" always, I don't read the diffs anymore. When I get error messages I just copy paste them in with no comment, usually that fixes it. The code grows beyond my usual comprehension, I'd have to really read through it for a while. Sometimes the LLMs can't fix a bug so I just work around it or ask for random changes until it goes away. It's not too bad for throwaway weekend projects, but still quite amusing. I'm building a project or webapp, but it's not really coding - I just see stuff, say stuff, run stuff, and copy paste stuff, and it mostly works.

So, vibe coding is a development style in which programmers rely heavily on generative AI in order to maintain creative flow and move quickly from idea to implementation. For exploratory solo work this can be highly productive.

In team settings, however, the same speed introduces a different problem: the faster ideas and code are produced, the easier it becomes to lose track of why a decision was made, which requirement it satisfies, and how later changes should be propagated across the project. This problem becomes even sharper in education, where students need room for experimentation but instructors still need visibility, traceability, and process discipline.

The risks are not only organizational. Empirical studies show that AI coding assistants may suggest insecure solutions [11] and that code models may reproduce memorized training fragments, potentially raising intellectual-property and confidentiality concerns [22]. These observations do not imply that generative AI should be avoided in software engineering. They indicate that AI-based programming support needs stronger process integration than ad hoc prompting in isolated chat windows.

In agile development, the product owner (PO) is the role that most directly mediates between stakeholders, requirements, prioritization, and the development team [6]. Kanban, in turn, offers a lightweight but disciplined coordination structure through visualized work items, explicit workflow states, and work-in-progress (WIP) limits [1]. Our starting point is that these two ideas can be combined with generative AI: instead of treating AI as a detached coding companion, it can be embedded into the very artifacts through which agile coordination happens.

This paper introduces **TAIPO** (**T**he **AI**-based **P**roduct **O**wner), pronounced like *"typo"*. TAIPO is an AI-supported PO assistant that operates directly on a Kanban board. Developers can ask prompt-based questions on concrete cards, and the answers are stored on those cards as persistent project knowledge. TAIPO can generate and prioritize user stories from requirement text, decompose stories into implementable tasks, refine descriptions, support acceptance-related feedback, and proactively simulate PO activity by adding contextual comments or injecting new requirements and change requests.

The present work builds on an earlier line of research on the artificial implementation of the Product Owner (PO) role in educational agile project simulation. In that earlier work, the focus was on a simplified educational prototype that generated Minimum Viable Product (MVP) requirements and backlog items for predefined project topics [18]. TAIPO extends this direction toward a Kanban board centric environment in which AI interactions are persistent, card-bound, and embedded into an explicit workflow structure.

The work is situated in a Slovak–Hungarian joint research project on realistic project simulation and intelligent PO support for software engineering education. A major resource behind the project is the TAWOS dataset, a relational corpus of 508,963 issues mined from 44 Jira-based open-source agile projects [20]. TAWOS does not only contain issue titles and descriptions, but also linked comments, users, releases, sprints, versions, and change histories. In our associated data-preparation work, the processed TAWOS issue corpus underwent cleaning and validity scoring to remove noisy or low-value records; the selected filtering configuration flagged 16.2% of issues for removal [19]. This cleaned corpus provides a stronger empirical basis for simulating realistic project dynamics.

The contribution of the paper is threefold. First, it defines the design rationale of a card-centered AI product owner assistant for structured vibe coding. Second, it presents a publicly accessible prototype implemented as a PHP backend and a Vue 3 frontend with Gemini-based AI services. Third, it shows how cleaned issue-tracking data can support proactive simulation of PO interventions in project-based

education. Although TAIPO can also generate code, our main claim is not that it replaces software developers or product owners. Rather, TAIPO demonstrates that vibe coding can be embedded into an auditable agile workflow in which AI outputs remain linked to requirements, tasks, and later changes.

The remainder of the paper first positions TAIPO in the literature, then presents its design concept, and finally describes the first prototype.

2. Related work

TAIPO sits at the intersection of several research threads, but the combination is unusual: most prior work addresses only one piece of the problem, such as the interaction style of vibe coding, the educational use of AI programming assistants, backlog generation from requirements, or the quality of issue-tracking data. The closest literature can therefore be read as a set of adjacent building blocks rather than as direct precedents for a Kanban board-centered AI product owner assistant.

The public starting point of vibe coding already captures both its attraction and its danger: one can "forget that the code even exists" while an AI tool takes over more of the concrete implementation work [7]. Later conceptual work formalized this as a shift in intent mediation, where natural-language dialogue and probabilistic inference take over part of the path from human intent to executable artifacts [10]. Empirical observation of extended vibe-coding sessions refined this picture by describing iterative cycles of prompting, rapid inspection, application testing, and occasional manual editing rather than a simple one-shot generation model [16].

Current research also makes clear that vibe coding should not be understood as fully autonomous software production. The central distinction is that human judgment remains responsible for goal setting, trust calibration, and final acceptance even when a large share of the concrete code is AI-generated [15]. Qualitative studies further show that co-creation, communication, and flow are experienced as real benefits, but they coexist with recurring difficulties around specification, incomplete solutions, large review effort, debugging, and trust [12]. This combination of attraction and fragility is directly addressed to TAIPO. We suggest that better project memory and workflow structure may be at least as important as better code generation.

The need for such structure is reinforced by adjacent evidence on AI-generated code quality. Security-oriented evaluation of AI code assistants has shown that generated suggestions can reproduce insecure patterns in realistic scenarios [11]. Separate work on code-model memorization has shown that generated outputs may contain memorized training fragments, which brings confidentiality, licensing, and provenance concerns into the development process [22]. These results do not argue against AI support, they argue against leaving AI use disconnected from auditable project artifacts.

A second research stream concerns education. Vibe coding has already been framed as an educational setting in which AI can act as more than a code-emitting

utility and may be experienced as a teammate-like collaborator inside project work [9]. Observational evidence from student sessions shows that much of the interaction effort shifts toward testing and debugging, while direct code reading becomes relatively rare; the same work also indicates that more advanced learners tend to supply richer contextual prompts [4]. For us this is important because a PO-oriented assistant should not only accelerate work, but also help keep intent explicit when students do not continuously read every generated line.

A direct predecessor of the present work can also be found in earlier research on the artificial implementation of the product owner role in educational agile project simulation [18]. That work introduced a simplified educational prototype in which generative AI was used to produce MVP requirements and backlog items for predefined project topics, while student progress was tracked through a mobile application interface. The main contribution of that earlier approach was to show that AI-supported PO simulation can reduce teacher workload in project-based software engineering education. TAIPO builds on this foundation, but moves beyond it in several ways: it shifts from a mobile requirement-delivery tool toward a web-based Kanban environment, it makes AI interaction persistent at card level, and it frames the whole system explicitly as support for structured vibe coding rather than only as a requirement-generation assistant.

The broader computer science education literature points to the same shift from low-level syntax production toward problem formulation and evaluation. Prompt Problems were proposed as a new exercise type in which students solve programming tasks by crafting sufficiently effective natural-language prompts for code-generating models [3]. Experience reports from introductory courses suggest that prompt-based activities may tap a partly different skill profile than traditional programming assessments and may soften some syntax-related barriers [8]. At the same time, benefits for novices come with risks of over-reliance, weakened understanding, and uneven learning effects [13]. This literature supports the view that code generation AI tools should support specification, review, and reflection, not only faster code production.

A third body of related work comes from agile coordination. The product-owner role has been analyzed as a central mechanism for linking stakeholder value, backlog decisions, and day-to-day development work, rather than as a purely administrative role [6]. Kanban research emphasizes visualization of work, explicit workflow stages, and work-in-progress control as practical devices for improving flow and reducing overload [1]. TAIPO inherits both ideas by locating AI support inside the Kanban board that already governs the project and by storing the results on cards instead of in detached conversations.

A fourth research direction moves generative AI upward from source code toward backlog and design artifacts. Requirement documents have already been used as inputs for automated user-story generation, showing that LLMs can populate project-management artifacts directly rather than only emit code [14]. From a model-driven perspective, vibe modeling argues that AI-supported development needs stronger support for higher-level abstractions if reliability and system com-

plexity are to remain manageable [2]. TAIPO fits this direction, but with a more operational emphasis: the central artifact is the Kanban card, which carries prioritization, clarification, decomposition, and change over time.

The final direction is repository mining and issue-quality research, which matters because TAIPO's simulation layer is grounded in issue-tracking data rather than in hand-written scripts. TAWOS was introduced as a versatile relational dataset spanning 44 Jira-based agile open-source projects and more than half a million issues, explicitly intended to support multiple downstream analyses such as effort estimation, prioritization, and assignment [20]. This makes it a natural basis for realistic PO-style simulation, but only if the data are clean enough for reuse.

The importance of issue quality has long been recognized. Classical bug-report research found a persistent mismatch between what developers consider most useful and what reporters actually provide, with reproduction steps, stack traces, and test cases repeatedly treated as high-value information [24]. Later work showed that different bug-report elements do not contribute equally to debugging effectiveness and that their significance can be studied empirically across repositories [17]. At a broader level, bug-report analysis has been surveyed as a large family of mining tasks rather than a single classification problem [23]. More focused work has examined automated discrimination between bug and non-bug issues [5], while dedicated metrics have also been proposed for assessing the quality of issue-tracking data itself [21].

This background directly supports the data-preparation step behind TAIPO. Our associated TAWOS cleaning work treated issue validity scoring as a prerequisite for later reuse and found that 16.2% of the processed issue corpus should be filtered out before simulation or educational use [19]. This is why TAIPO is grounded in cleaned repository traces instead of raw issue text.

Our survey suggests a gap. Existing vibe coding work explains the interaction style and its tensions. Educational work explains why prompt-mediated development needs scaffolding. Agile work explains why product-owner decisions and board discipline matter. Requirements-oriented AI work shows that backlog artifacts themselves can be generated, and repository-mining research explains why realistic issue corpora require quality control. TAIPO addresses this gap by combining these directions in a single Kanban-board-centered solution.

3. Design concept

The immediate target environment of TAIPO is project-based software engineering education. In such courses, a single instructor often needs to act as product owner (PO) for several student teams at the same time. The main bottleneck is not only answering questions, but doing so quickly enough and consistently enough that each team can continue working without long waiting periods. Purely manual PO support scales poorly, while generic AI chat tools fail to preserve decisions as stable project artifacts.

Therefore, TAIPO should be designed around four main requirements. First,

persistence: AI-generated clarifications should remain attached to the relevant work item rather than disappearing in a detached chat history. Second, *controllability*: the system should respect workflow states, priorities, and WIP limits, and it should remain robust even when human users edit cards or reorganize work manually. Third, *realism*: the assistant should not only react to direct prompts, but also introduce the kind of asynchronous feedback and requirement volatility that real projects exhibit. Fourth, *scalability*: the overall interaction model should reduce instructor workload without turning the course into a fully automated black box.

The TAWOS resource is central to the realism requirement. The original dataset provides large-scale traces of agile issue management, see Section 2. TAIPO should use TAWOS as inspiration for simulation patterns, comment styles, and change-request (CR) dynamics, thereby linking repository mining with educational process support.

These design goals position TAIPO somewhere between a project management tool and an AI teaching assistant. It should remain closer to the former in the sense that the Kanban board should serve as the primary source of truth. At the same time, it should behave like the latter by continuously supplying explanations, suggestions, and project perturbations, for example by occasionally introducing a new CR that keeps teams active even when a human PO is not immediately available.

TAIPO should treat Kanban cards as the operational context of AI interactions. Each card should store a user story or task together with its description, priority, workflow state, and associated discussion. When a user invokes TAIPO from a concrete card, the prompt should be interpreted in the context of that card and of the surrounding board. The answer should then be written back to the board, typically as a comment or as a structured modification of the card fields. In this way, requirement interpretation should become part of the project state itself.

The Kanban board should support multiple workflow columns with explicit WIP limits, with the exception of the Backlog and Done columns. The Backlog should function as the source of user stories, whereas Done should serve as the sink for completed tasks.

Cards should be movable through drag-and-drop interactions so that users can reorganize work naturally while the system still preserves process constraints. These workflow restrictions should not suppress creativity; rather, they should provide a lightweight process envelope within which AI-supported backlog refinement and clarification can remain traceable and manageable.

TAIPO should include backlog population as a capability compatible with recent LLM-assisted user-story generation approaches, see Section 2, but it should treat generation as only the first step of a longer workflow. Because cards should persist as editable project artifacts, external human edits should not break the AI context, instead, they should become part of the next reasoning step.

TAIPO should support two complementary PO interaction modes: (1) a reactive mode and (2) a proactive mode.

The reactive mode should be initiated by users. In this mode, the assistant

should transform requirement text into prioritized user stories, refine existing cards, decompose user stories into tasks, and answer clarification questions attached to a concrete card.

The proactive mode should simulate the asynchronous behavior of a human PO. Its intervention frequencies should be configurable. For example, TAIPO should be able to add a contextual comment to a card every few hours during working days (between 8AM and 4PM). It should also be able to prompt users to update project progress when the board state suggests that the recorded status may no longer reflect the actual state of work. Less frequently, it should be able to introduce a new requirement, user story, or CR.

When a team reaches a milestone, or at the end of a sprint, which should be represented as an event generated by a configurable time-based service, TAIPO should be able to (1) accept or reject completed tasks, (2) provide *adaptive coaching*, i.e., explain the decisions of the PO, and (3) extend the requirement set with more specific follow-up expectations, thereby making the simulated project evolve in a less static and more realistic way.

TAIPO should also be able to ask the user proactively, through *notifications prompting*, whether the connected GitHub repository and the Kanban board are significantly out of sync.

The content of these interventions should be generated from the cleaned TAWOS-derived knowledge base together with the current board context. In classroom settings, this mechanism should help recreate the fact that real requirements rarely remain unchanged throughout a project.

The first prototype described in the next section implements a substantial subset of these design ideas.

4. The first prototype of TAIPO

The first prototype of TAIPO implements the core board-centered interaction model. It operationalizes the main ideas of the design concept. It should be understood as a first implementation, not as a fully functional release.

The prototype has been implemented as a web-based system that combines a PHP backend API with a Vue 3 single-page frontend. Board state is stored in a relational database. By default, the prototype uses SQLite, but the backend is configurable to work with other PDO-supported database systems as well. The frontend provides responsive drag-and-drop interaction, priority marking, and card editing, while the backend is responsible for persistence, workflow enforcement, configuration, and communication with external AI services. The current prototype uses the Gemini API for AI-supported content generation. The source code of the system is publicly available at `https://github.com/szabojuci/AIKanban.git`.

The interface is organized around a five-stage Kanban workflow that reflects the basic project lifecycle: Sprint Backlog, Implementation, Testing, Review, and Done. The in-progress columns use explicit WIP limits, so the board does not merely display tasks but also constrains parallel work. This is important for our

interpretation of TAIPO: the system is not intended to replace agile discipline with unrestricted AI automation, but to place AI-supported decisions inside a controlled process.

Figure 1 illustrates the current interface. The interface consists of five workflow columns, namely Sprint Backlog, Implementation, Testing, Review, and Done, with visible WIP counters where relevant. Cards contain titles, priorities, and action buttons. Opening a card exposes the "Ask TAIPO" dialog, where the user can enter a free prompt or select built-in Quick Prompts such as explanation, implementation advice, test-case generation, security-oriented checking, and code-snippet support. The generated response is stored with the selected card, so the AI interaction remains part of the board history.

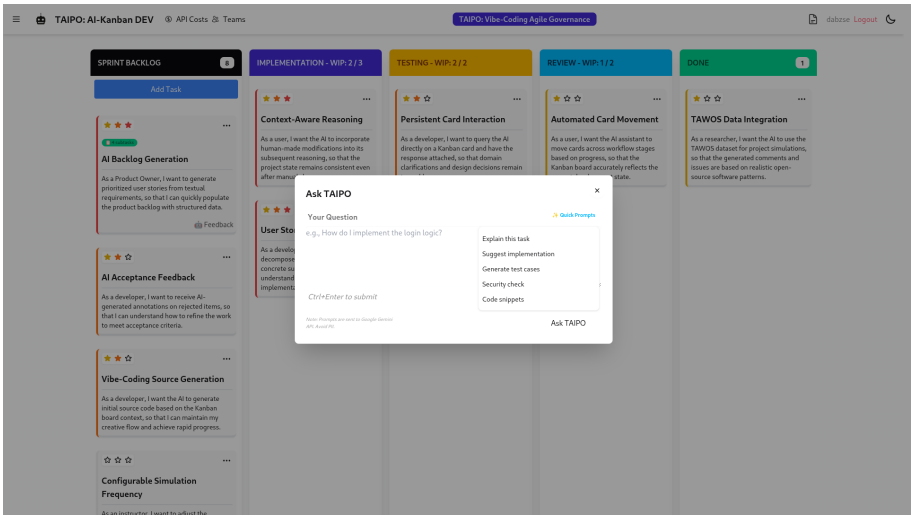


Figure 1. The TAIPO prototype with the "Ask TAIPO" dialog and built-in Quick Prompts.

The core interaction principle is that AI support is attached to concrete cards. When a user opens the "Ask TAIPO" dialog from a selected card, the prompt is interpreted in the context of that card and of the surrounding board state. To reduce prompting overhead in typical product-owner situations, the prototype also offers a small set of built-in Quick Prompts, including task explanation, implementation suggestions, test-case generation, security-oriented checking, and code-snippet support. The resulting answers are stored as persistent card-related artifacts rather than disappearing in an isolated chat session. If a card already contains earlier TAIPO feedback, the dialog can notify the user and offer regeneration of the answer. In this way, requirement clarification, design rationale, and implementation guidance remain inspectable later by both students and instructors.

The present prototype already supports the main interactions needed for a PO-oriented vibe-coding workflow. It can populate the backlog from textual re-

quirement descriptions, generate prioritized user stories, refine and reprioritize existing cards, and decompose a selected story into more concrete tasks. It can also support acceptance-related feedback by annotating non-accepted items and returning them to the backlog for further refinement. Optional code assistance is also available. This is auxiliary to the main idea: TAIPO is primarily a structured project-governance environment for AI-assisted development, not a stand-alone code generator. Repository integration is not yet implemented.

Table 1 summarizes the main capabilities currently available in the prototype.

Table 1. Main capabilities of the TAIPO prototype.

Capability	**Board-level effect**
Backlog population	Generates prioritized user stories from textual requirements and inserts them into the backlog.
Story decomposition	Breaks a selected story into concrete, implementable tasks.
Card-level clarification	Answers prompt-based questions and stores the resulting explanation on the relevant card.
Refinement and prioritization	Updates descriptions, priorities, and other card metadata to support backlog grooming.
PO simulation	Adds contextual comments and injects new requirements, user stories, or change requests at configurable intervals.
Acceptance handling	Supports returning rejected items to the backlog; richer AI-generated rejection feedback is ongoing work.
Optional code assistance	Can generate implementation snippets, but there is no repository integration.

Beyond direct user interaction, the prototype also supports proactive PO simulation. The intervention schedule is configurable. In the current default setup, TAIPO adds a contextual comment to a card every two hours during working days between 8AM and 4PM, based on a weekday-sensitive time-based mechanism. At present, this mechanism does not include calendar integration or public-holiday awareness. Less frequently, on average once every three days, it introduces a new requirement, a user story, or a change request. These interventions are generated from the current board context together with patterns derived from the cleaned TAWOS-based knowledge base. In educational settings, this makes the board behave less like a static assignment list and more like a small but evolving agile project.

To demonstrate the prototype, we use a weather-forecast web application containing a zoomable map view and a time-slider component, followed later by unforeseen requirement changes. This scenario is deliberately compact, yet rich enough to exercise the central functions of TAIPO: backlog creation from a short description, story decomposition, card-level clarification, reprioritization, and reaction to injected changes.

The example also makes the main design decision of TAIPO visible: project

memory is stored on the board itself. Clarifications about map behavior or time handling remain attached to the relevant cards, and later changes appear as new board artifacts instead of getting lost in side conversations.

A public demonstration instance is available at `https://dabzse.net/TAIPO/`. At the time of writing, it can be accessed with the demo credentials `testuser` / `password123`. This makes the prototype directly inspectable for reviewers and readers. Together with the public repository, the demo supports the practical reproducibility of the system and shows that TAIPO already exists as an operational prototype rather than as a purely conceptual proposal.

4.1. Exploratory classroom scenario

In addition to the weather-forecast example, we conducted a small exploratory classroom walkthrough using a compact *Mini-University* scenario. The log of the walkthrough can be accessed at: `https://docs.google.com/document/d/1d7S1YnEKmRrVrZbVZUDvb8k5ZMLROAdl/`.

The product brief described a single-file Streamlit application with JSON-based persistence and three roles: administrator, teacher, and student. The required functionality included role-based login, user and course administration, teacher-side grade entry, student-side course enrolment and drop, transcript and GPA views, and persistence after each data-changing operation. The purpose of the scenario was not to evaluate the Mini-University application itself, but to observe whether an initially unstructured AI-development brief could be transformed into a persistent, card-centered TAIPO workflow.

The walkthrough deliberately avoided a one-shot prompting style. During Sprint 0, the product brief was given to TAIPO as a requirement seed and decomposed into twelve backlog cards, including project skeleton, JSON persistence, authentication, administrator-side user management, course management, teacher-side course roster and grade entry, student-side enrolment, transcript calculation, role-based access-control tests, and later change-request handling. Four short mini sprints were then used to implement and review the cards: first the application skeleton, persistence, and login; then administrator functions and security constraints; then teacher and student workflows; and finally change management, review, and hardening.

At card level, TAIPO was used before implementation to generate acceptance criteria in Given/When/Then form, implementation notes, and smoke-test checklists. This made several rule-like requirements explicit before coding started. For example, administrators must not delete their own account or the last remaining administrator; teachers may only modify grades in their own courses; students may only enrol in or drop courses; and deletion operations must not leave inconsistent enrolment or grade records behind. These constraints are typical examples of product-owner knowledge that can easily disappear in a detached chat transcript, but can remain inspectable when attached to Kanban cards.

The most useful part of the walkthrough was the change-management step. In the last mini sprint, the team introduced a small change request: students should

not be able to enrol in full courses, and the user interface should explicitly mark such courses as full. TAIPO identified the affected earlier card, updated the relevant acceptance criteria, and proposed regression tests for the modified enrolment workflow. The resulting decision history remained attached to the change-request card and the affected feature card, illustrating how TAIPO can support traceability when requirements evolve after implementation has already started.

The walkthrough log recorded 28 card-level TAIPO interactions, 36 acceptance criteria, and 22 test items, including nine role-based negative tests. One change request was handled, and two defects or missing edge cases were identified during the process: self-deletion of an administrator account and stale course-related data after deletion operations. The final implementation was checked against the board history, and the reported features were linked back to TAIPO cards. These numbers should be interpreted as preliminary process observations, not as statistically valid evidence of productivity or learning gains.

This walkthrough is therefore not a controlled classroom evaluation. It does not measure learning outcomes, usability, developer productivity, or code quality against a baseline tool. Its value is narrower but still relevant: it shows that TAIPO can turn a long natural-language development brief into smaller work items, preserve acceptance criteria and tests as card-level artifacts, and make later changes auditable. A controlled study with student teams remains future work.

The proactive simulation layer of TAIPO should be interpreted in the same cautious way. It is derived from the cleaned TAWOS-based knowledge base in a pattern-based rather than replay-based manner. After filtering low-value issues, we use issue metadata and text traces, such as issue type, priority, comment density, status changes, and change-request-like comments, to derive intervention categories and plausible timing patterns. At runtime, the scheduler selects an intervention type, for example clarification, progress reminder, acceptance feedback, or a new change request, and conditions the generated text on the current board state. Thus, the realism claim means empirically inspired PO-like behavior grounded in issue-tracking traces, not a validated statistical reproduction of a specific Jira project.

Educational deployments also require explicit operational safeguards. The Mini-University walkthrough used synthetic project data, and the same principle should be followed in classroom use: students should not submit real personal data, secrets, private repository content, assessment-sensitive material, or confidential stakeholder information to an external model provider. TAIPO's code-oriented prompts are auxiliary; generated snippets should be treated as suggestions that require human review, testing, and security inspection before acceptance. The repository should not contain API keys, and deployments should provide keys through local configuration or environment variables. Instructors can also disable code-oriented prompts or restrict tasks to synthetic project descriptions when confidentiality or intellectual-property concerns are important.

Finally, the current prototype uses the Gemini API, which introduces practical dependencies such as cost, availability, data-transfer policies, and rate limits. This dependency is not central to the TAIPO concept: the board model, persistence

layer, and prompt construction can be separated from the concrete model backend. For classroom use, deployments should therefore include quota-aware settings, caching of repeated responses where appropriate, and a fallback mode in which proactive comments can be disabled or delayed. Future versions should make the model interface explicitly pluggable so that cloud models, institutional endpoints, or local models can be compared under the same board-centered workflow.

5. Conclusion and future work

Vibe coding is attractive because it accelerates exploration, but without process support it can weaken traceability and requirement control. TAIPO addresses this problem by embedding an AI-based product owner assistant directly into a Kanban workflow.

Instead of placing the model outside the process and leaving teams to negotiate how to remember or trust its output, TAIPO makes the Kanban board the shared memory of human and AI activity. This shift has several important consequences: (1) It improves traceability because AI-generated answers remain linked to cards. (2) It improves auditability because instructors or team leads can review not only code, but also the requirement interpretations and decisions that led to it. (3) It also improves resilience to change, since later prompts can reuse the updated board state rather than depend on vague chat context.

The educational value of this approach is particularly important. In many project-based software engineering courses, the scarcest resource is timely PO attention. TAIPO cannot replace a human instructor, but it can absorb a substantial part of repetitive clarification and backlog-maintenance work, while also making projects more realistic through controlled requirement changes. Because the assistant is grounded in cleaned issue-tracking data, its proactive behavior is closer to plausible project dynamics than a purely hand-written script would be.

At the same time, the current work has clear limitations. The Mini-University walkthrough is only a small exploratory scenario, not a controlled classroom study. We do not yet report quantitative learning outcomes, developer-productivity measurements, code-quality comparisons, or a formal comparison against alternative AI-assisted project-management tools. The present prototype relies on an external generative-model API, and some functions, especially acceptance-related rejection feedback and repository synchronization, are still evolving.

The next step is therefore to move beyond the first prototype and evaluate it in real educational settings at both the Technical University of Košice and Eszterházy Károly Catholic University. Based on these practical experiences, we plan to refine both the underlying concept and the implementation of TAIPO.

Since the source code is openly available, anyone can also try the system and experiment with it. Naturally, the repository does not include an API key, so users must provide their own. The authors warmly welcome feedback from all users, as such experience can directly support the further development of both the prototype and the broader idea of structured vibe coding.

References

[1] M. O. Ahmad, J. Markkula, M. Oivo: *Kanban in Software Development: A Systematic Literature Review*, in: Proceedings of the 2013 39th Euromicro Conference on Software Engineering and Advanced Applications, 2013, pp. 9–16, doi: 10.1109/SEAA.2013.28.

[2] J. Cabot: *Vibe Modeling: Challenges and Opportunities*, in: Advances in Conceptual Modeling, 2025, pp. 105–118, doi: 10.1007/978-3-032-08620-4_7.

[3] P. Denny, J. Leinonen, J. Prather, A. Luxton-Reilly, T. Amarouche, B. A. Becker, B. N. Reeves: *Prompt Problems: A New Programming Exercise for the Generative AI Era*, in: Proceedings of the 55th ACM Technical Symposium on Computer Science Education (SIGCSE 2024), 2024, pp. 296–302, doi: 10.1145/3626252.3630909.

[4] F. Geng, A. Shah, H. Li, N. Mulla, S. Swanson, G. Soosai Raj, D. Zingaro, L. Porter: *Exploring Student-AI Interactions in Vibe Coding*, in: Proceedings of the 28th Australasian Computing Education Conference, 2026, pp. 45–54, doi: 10.1145/3786228.3786236.

[5] S. Herbold, A. Trautsch, F. Trautsch: *On the Feasibility of Automated Prediction of Bug and Non-Bug Issues*, Empirical Software Engineering 25.6 (2020), pp. 5333–5369, doi: 10.1007/s10664-020-09885-w.

[6] M. D. Kadenic, D. A. de Jesus Pacheco, K. Koumaditis, G. Tjørnehøj, T. Tambo: *Investigating the role of Product Owner in Scrum teams: Differentiation between organisational and individual impacts and opportunities*, Journal of Systems and Software 206 (2023), pp. 1–17, doi: 10.1016/j.jss.2023.111841.

[7] A. Karpathy: *Vibe Coding*, X (formerly Twitter) post, Feb. 2025, url: https://x.com/karpathy/status/1886192184808149383 (visited on 01/14/2026).

[8] C. Kerslake, P. Denny, D. H. Smith IV, J. Prather, J. Leinonen, A. Luxton-Reilly, S. MacNeil: *Integrating Natural Language Prompting Tasks in Introductory Programming Courses*, in: Proceedings of the 2024 ACM Virtual Global Computing Education Conference, 2024, pp. 88–94, doi: 10.1145/3649165.3690125.

[9] G. Kusper, C. Szabó: *Vibe Coding in Education*, in: Proceedings of the 2025 International Conference on Emerging eLearning Technologies and Applications (ICETA), 2025, pp. 506–511, doi: 10.1109/ICETA67772.2025.11280285.

[10] C. Meske, T. Hermanns, E. von der Weiden, K.-U. Loser, T. Berger: *Vibe Coding as a Reconfiguration of Intent Mediation in Software Development: Definition, Implications, and Research Agenda*, IEEE Access 13 (2025), pp. 213242–213259, doi: 10.1109/ACCESS.2025.3645466.

[11] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, R. Karri: *Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions*, in: 2022 IEEE Symposium on Security and Privacy (SP), 2022, pp. 754–768, doi: 10.1109/SP46214.2022.9833571.

[12] V. Pimenova, S. Fakhoury, C. Bird, M.-A. Storey, M. Endres: *Good Vibrations? A Qualitative Study of Co-Creation, Communication, Flow, and Trust in Vibe Coding*, 2025, doi: 10.48550/arXiv.2509.12491, arXiv: 2509.12491.

[13] J. Prather, B. N. Reeves, J. Leinonen, S. MacNeil, A. S. Randrianasolo, B. A. Becker, B. Kimmel, J. Wright, B. Briggs: *The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers*, in: Proceedings of the ACM Conference on International Computing Education Research (ICER '24 Vol. 1), 2024, doi: 10.1145/3632620.3671116.

[14] T. Rahman, Y. Zhu: *Automated User Story Generation with Test Case Specification Using Large Language Model*, 2024, doi: 10.48550/arXiv.2404.01558, arXiv: 2404.01558 [cs.SE].

[15] R. Sapkota, K. I. Roumeliotis, M. Karkee: *Vibe Coding vs. Agentic Coding: Fundamentals and Practical Implications of Agentic AI*, 2025, doi: 10.48550/arXiv.2505.19443, arXiv: 2505.19443.

[16] A. Sarkar, I. Drosos: *Vibe Coding: Programming through Conversation with Artificial Intelligence*, in: Proceedings of the 36th Annual Conference of the Psychology of Programming Interest Group (PPIG 2025), 2025, pp. 87–118.

[17] M. Soltani, F. Hermans, T. Bäck: *The Significance of Bug Report Elements*, Empirical Software Engineering 25.6 (2020), pp. 5255–5294, doi: 10.1007/s10664-020-09882-z.

[18] C. Szabó, K. Lengyel: *Artificial Intelligence Based Implementation of the Product Owner Role in Educational Agile Project Simulation*, in: ICERI2025 Proceedings, 18th annual International Conference of Education, Research and Innovation, Seville, Spain: IATED, 2025, pp. 5910–5915, isbn: 978-84-09-78706-7, doi: 10.21125/iceri.2025.1622.

[19] M. Szabó, Á. Kovács, G. Kusper: *Issue Validity Scoring for Cleaning the TAWOS Database: OwnMetrics versus Local Small Language Models*, Manuscript for the Proceedings of the 13th International Conference on Applied Informatics (ICAI 2026), 2026.

[20] V. Tawosi, A. Al-Subaihin, R. Moussa, F. Sarro: *A Versatile Dataset of Agile Open Source Software Projects*, in: Proceedings of the 19th International Conference on Mining Software Repositories (MSR '22), 2022, pp. 707–711, doi: 10.1145/3524842.3528029.

[21] F. Tu, F. Zhang: *Measuring the Quality of Issue Tracking Data*, in: Proceedings of the 6th Asia-Pacific Symposium on Internetware, 2014, pp. 76–79, doi: 10.1145/2677832.2677844.

[22] Z. Yang, Z. Zhao, C. Wang, J. Shi, D. Kim, D. Han, D. Lo: *Unveiling Memorization in Code Models*, in: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, 2024, pp. 1–13, doi: 10.1145/3597503.3639074.

[23] J. Zhang, X. Wang, D. Hao, B. Xie, L. Zhang, H. Mei: *A Survey on Bug-Report Analysis*, Science China Information Sciences 58.2 (2015), pp. 1–24, doi: 10.1007/s11432-014-5241-2.

[24] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schröter, C. Weiss: *What Makes a Good Bug Report?*, IEEE Transactions on Software Engineering 36.5 (2010), pp. 618–643, doi: 10.1109/TSE.2010.63.

Annales Mathematicae et Informaticae
63 (2026) pp. 88–106
DOI: 10.33039/ami.2026.06.002
URL: https://ami.uni-eszterhazy.hu

Fully dynamic strong connectivity and reachability in digraphs*

Gregory Morse, Tamás Kozsik

Eötvös Loránd University
morse@inf.elte.hu
kto@elte.hu

Abstract. Computing strongly connected components (SCCs) and reachability in directed graphs is fundamental in compilers, static analysis, and many graph algorithms. While efficient offline algorithms are well known, maintaining this information dynamically under both edge insertions and deletions remains challenging.

This paper presents a deterministic fully dynamic algorithm that simultaneously maintains SCCs and reachability in directed graphs. The approach combines a union–find structure for efficient merging of SCCs during edge insertions with localized recomputation of SCCs using Nuutila's algorithm when deletions potentially split a component. Reachability information is maintained at the SCC level and propagated through the condensation DAG.

For a current graph with n vertices and m edges, the resulting algorithm supports $\mathcal{O}(1)$ reachability queries while updates have worst-case complexity $\mathcal{O}(m + n^2)$ due to reachability propagation. Although this does not improve the best known theoretical bounds for specialized dynamic algorithms, the method is simple, deterministic, and well suited to sparse graphs such as control flow graphs (CFGs). Experimental evaluation on random graphs and real program CFGs shows that the algorithm significantly outperforms repeated offline recomputation in practical scenarios.

Keywords: fully dynamic algorithm, strongly connected components, reachability, directed graphs, incremental algorithm, decremental algorithm, graph maintenance

2020 *Mathematics Subject Classification:* 68R10, 68W27

*The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013, Thematic Fundamental Research Collaborations Grounding Innovation in Informatics and Infocommunications).

Submitted: March 31, 2026
Accepted: June 30, 2026
Published online: July 17, 2026

1. Introduction

A fully dynamic algorithm maintains graph information under both edge additions and edge deletions. When a graph is being constructed or modified while queries and decisions about its continued construction are needed, a fully dynamic algorithm can perform better than repeated recomputation for the entire graph.

An unweighted directed graph (digraph) is an ordered pair $G = (V, E)$ with its vertex labels in V and ordered pairs of directed edges in E in the form (x, y) where $x \in V$ is the source or predecessor node and $y \in V$ is the destination or successor. Reachability of vertices based on the transitive closure of the edges can be defined as the existence of any path $s \xrightarrow{*} t$ for two nodes s and t. A strongly connected component (SCC) or region in the graph is defined as a maximal set of nodes S where all nodes are reachable from all other nodes in the set as $\{x \xrightarrow{*} y \| x \in S, y \in S\}$. A non-trivial SCC is one where there is strong connectivity, e.g., $|S| > 1$ or a single node x which has a self-loop and thus reach itself given $(x, x) \in E$. This leaves single node components without self-loops as trivial so a node x where $(x, x) \notin E \wedge \nexists y \in V(x \xrightarrow{*} y \wedge y \xrightarrow{*} x)$. If every SCC is collapsed into a single representative node with only its inter-SCC edges remaining, then the resulting graph is a directed acyclic graph (DAG). Any element of an SCC can be selected to be a root node for the SCC, though classically the root is chosen as the first node of an SCC which was visited during a depth first traversal of the graph. For convenience, we say the current number of vertices and edges after any update are $|V| = n$ and $|E| = m$ respectively. It is also easy to see the relationship between the number of edges and vertices as $m \leq n^2$.

Our focus is on deterministic algorithms that provide exact reachability information and constant-time $\mathcal{O}(1)$ queries. Although the worst-case amortized time complexity of the algorithm can seem less than ideal, in practice, the performance is sufficient when dealing with sparse graphs that contain sufficient strong connectivity. Furthermore, optimized variants which use batching so that a group of edges may be either added or deleted simultaneously can often significantly improve update throughput for fully dynamic algorithms. It should be noted that it is trivially shown that the more dense a graph becomes, then SCCs in the graph become more or larger, so the algorithm that will be presented would become very efficient in such types of graphs. However, in the usual contexts where SCCs are computed, these graphs are often control-flow graphs (CFGs) and tend to be sparse. SCCs have also been used as abstraction units in logical encodings, where SCC structure influences properties of SAT representations [14]. Regardless, for dense graphs, it is well known that SCCs are a performant method to compute transitive closure.

Contributions. This paper presents a deterministic fully dynamic algorithm that simultaneously maintains SCCs and all-pairs reachability in directed graphs. The algorithm uses union–find structures to efficiently merge SCCs during edge insertions and recomputes components locally during deletions using Nuutila's algorithm. Reachability information is propagated through the SCC DAG, yielding $\mathcal{O}(1)$ query time and worst-case $\mathcal{O}(m + n^2)$ update complexity.

2. Related work

Tarjan's algorithm is one of the most well-known for computing the SCCs of a digraph [23]. This algorithm uses two interleaved traversals of the graph to identify components as well as a representative node considered to be its entry. The first traversal is a depth first search (DFS) and the second is based on a stack which stores potential component elements that are removed when the component is identified. The time complexity is $\mathcal{O}(m + n)$, the same as for DFS, while the storage complexity is $n(2 + 5w)$ where w is the machine's word size.

Nuutila and Soisalon-Soininen modified Tarjan's algorithm to yield two new algorithms which reduce the number of nodes stored on the stack and thus a space complexity improvement to $n(1 + 4w)$ [17]. The first algorithm does not store component root nodes, while the second one stores only possible root nodes for non-trivial components. The interesting aspect of the second algorithm is that it leads to a simple extension to yield transitive closure information found with more detail in Nuutila's later paper [16]. However the addition of transitive closure has the worst case same complexity of its space and time which is $\mathcal{O}(n^2)$. Purdom had previously shown a similar method to compute SCCs and then perform topological sort of the SCC nodes and compute transitive closure based on the simplification provided by the SCC induced DAG to also run in $\mathcal{O}(n^2)$ [19]. The primary difference is that Purdom's algorithm computes the SCCs in advance and did not make use of a stack thus requiring an extra topological sort. Pearce improved the space efficiency of Tarjan's algorithm further to $n(1 + 3w)$ in [18].

More recently, Haeupler et al. show how to incrementally maintain topological ordering and SCCs in $\mathcal{O}(m^{3/2})$ time using bidirectional search in [8] with the Union-Find data structure. However, though decrementally maintaining topological ordering is possible under their algorithm, they merely discuss that decremental SCCs are a harder problem. Georgiadis et al. give efficient data structures for solving the incremental SCCs case in [6] by maintaining dominators and what they called the hyperloop nesting forest as it is based on the dominators not the depth first spanning tree. Georgiadis et al. also showed a way to compute SCCs upon potential deletion of every edge using the concept of classifying an edge as a strong bridge with $\mathcal{O}(n)$ time complexity in [7]. A strong bridge is an edge whose deletion would increase the number of SCCs. Bernstein et al. show a method to solve only the decremental SCC algorithm in $\mathcal{O}(m)$ time using ES-trees in [1]. This was improving on the previous best known decremental time of $\mathcal{O}(m\sqrt{n})$ in [2]. Karczmarz and Smulewicz give a randomized fully dynamic SCC data structure with conditionally tight $\mathcal{O}(n^{1.529})$ worst-case update time, and also reduce fully dynamic strong connectivity to dynamic reachability [12]. These specialized algorithms give stronger asymptotic bounds for individual dynamic SCC or reachability variants, while the algorithm here emphasizes a simple deterministic structure that maintains SCCs and all-pairs reachability together.

This paper is thereby motivated primarily to analyze not merely a fully dynamic algorithm for strong connectivity or reachability in their isolated sense, but

the simultaneous maintenance of both. Because of their relationship purely based on their mathematical definitions, it makes sense to do this. It should be noted that a DFS-based approach to their maintenance should also be considered and due to advances in algorithms for fully dynamic maintenance of the DFS-tree and its intervals [26], an ideal algorithm could likely look at the exact changes that occurred to the DFS-tree to recompute SCCs very efficiently. This would not only be an elegant algorithm that could be extended to loop nesting forest maintenance but could replace the use of the more expensive maintenance of all-pairs transitive closure. However, there are situations where reachability information is also needed. Thus the fact that SCCs can significantly reduce the amount of storage needed to make reachability queries makes their combination in dynamic algorithms appropriate.

As for transitive closure, the Floyd-Warshall algorithm is the classic algorithm for computing all-pairs shortest paths in a digraph however it is easily adapted to transitive closure but it has $\mathcal{O}(n^3)$ time complexity and $\mathcal{O}(n^2)$ space complexity due to using a matrix in [4]. This will be the worst-case space complexity for the reachability structure used in this paper when every vertex is an SCC. Since the cubic computation in Floyd-Warshall becomes impractically slow very quickly as the number of vertices of the graph increases, a better algorithm is the well known utilization of the DFS or breadth-first search (BFS) starting with every node in the graph which gives a time complexity of $\mathcal{O}(n^2 + nm)$ reducing it to quadratic. For decremental transitive closure maintenance, Even and Shiloach provided a single source $\mathcal{O}(mn)$ algorithm in [22] which uses a BFS structure maintaining depth information sometimes aptly referred to as an ES-tree to efficiently store the information and allow updates. It should be noted that maintaining a BFS tree in a fully dynamic setting is still an open problem and a semi-dynamic algorithm which works in both an isolated incremental or decremental setting is the best that is known [5]. In fact, not maintaining the BFS tree but only the depth or level information in a decremental setting is more straight-forward. However, edge addition still requires a BFS recomputation starting from the target vertex. Furthermore, this data structure solves the all-pairs shortest paths problem and transitive closure is, in fact, a subset of the information provided by shortest paths. It is also used in state-of-the-art decremental SCC as well as single source reachability (SSR) algorithms. Henzinger et al. reduce the time complexity for the single source problem to being sublinear but achieves it with randomization [9]. ES-trees will not be used in this fully dynamic context as no known efficient computation can be done incrementally.

As for fully dynamic transitive closure with $\mathcal{O}(1)$ queries, King presented an $\mathcal{O}(n^2 \log n)$ algorithm in [13]. Demetrescu and Italiano presented a quadratic $\mathcal{O}(n^2)$ algorithm in [3] using matrices with initialization time of $\mathcal{O}(n^3)$. Sankowski improved the $\mathcal{O}(n^2)$ to being worst-case in [21]. Roditty improves the initialization time to $\mathcal{O}(n^2)$ in [20].

It is thus understood that Union-Find has been the basis for all efficient incremental algorithms while ES-trees have been the basis for efficient decremental algorithms regarding SCCs and reachability.

As for connectivity of the digraph, it is not required for SCCs or transitive closure but for simplicity it can also be assumed a virtual root node of the graph which connects implicitly to all nodes is present, thereby reaching all nodes and being a trivial SCC wlog.

3. Algorithm

The Union-Find disjoint set data structure with the usual path compression but rank specified by order of arguments to the union operation will be used as it has been in many similar algorithms due to its efficiency in collapsing sub-graphs which is precisely what will be done with SCCs. However, it is reinitialized for vertices of an SCC when a deletion potentially splits the component; the new SCCs are recomputed using Nuutila's algorithm on the affected subgraph. This operation, referred to as `unmerge`, will require as input a complete disjoint set, and will effectively divide them back into individual set elements. This is needed when a SCC will be removed. As long as the entire set in the tree is passed, this is a very simple operation as it merely resets the parent of all entries in the given set to themselves. Obviously, this improves the time complexity of the incremental case at the expense of the decremental one. The simple justification for such a choice is that in compilers and static analysis, edge addition tends to be much more common than edge deletion, due to a typical initial construction of the CFG. However, certain refactorings and optimizations could still cause significant edge deletions.

Self-loops are considered as though they do not affect SCCs, they do change reachability but only in that very limited aspect. Other trivial cases are when an edge is added between two nodes in an SCC, or when an edge is removed between two nodes in an SCC without affecting the SCC. However, detecting that an edge removal does not affect the SCC requires an offline reachability check. Therefore, strictly adapting the offline algorithm to account for the trivial cases will not be particularly beneficial.

Four update cases are considered: (i) insertion merging SCCs, (ii) insertion without merge, (iii) deletion splitting an SCC, (iv) deletion without SCC change. In all of these cases, the reachability information will need to be updated by propagation. Only the predecessor adjacency lists for the graph are necessary for our part of the algorithm as the successor information is only used in Nuutila's algorithm for forward reachability.

When an edge addition induces a merging of SCCs, it merely requires a query that the destination edge reaching the source edge. By traversing all reachable SCCs of the destination node's SCC which can also reach the source, the new component can be readily identified and constructed. This will not lead to root node of the SCC being the same one that might be identified in an offline algorithm as by using reachability, because the DFS ordering is not maintained. However, the most difficult case is an edge removal within an SCC. The data structures do not contain any information to determine whether the SCC has remained intact, but if it has

not, the reachability information does not help break it into its smaller components with their new reachability information. Therefore, Nuutila's algorithm can be applied on the sub-graph of the SCC which had an edge removal. If the changes in the depth first spanning tree or the loop nesting forest information were available, a more detailed decremental strategy could be employed. In fact, Nuutila's algorithm identifies SCC root nodes according to the DFS order it computes; because it does not know the original DFS order for the sub-graph, these roots may still differ from those typically used in an offline algorithm. This is considered unimportant and the maintenance of DFS though possible would provide only this marginal benefit in the algorithm and certainly add unneeded complexity as maintaining entries of SCCs and deciding when the entry node DFS order has changed would further need to be managed.

The reachability information in the incremental case is propagated backwards through the DAG of SCCs by adding the newly reachable nodes whether the edge addition created a new SCC or merely added a path between SCCs. In the decremental case, the reachability is also backward propagated, however, before removal, reachability must be constructed for all nodes which reach the target node to make sure there is not another pathway to the source node and nodes which reach the source node that potentially need to be removed. Additionally it can only be done by visiting all predecessor nodes in an order such that all successors to a given predecessor have already been processed. The DFS reverse postordering or topological sort is well known to satisfy this property and is trivial to prove since the graph formed by SCCs is acyclic. Other methods to do the propagation would use successor reaching sets to directly rebuild the reaching set or compute the necessary set subtractions, however this leads to an $O(n^3)$ algorithm and would also be slower in practice.

The algorithm uses three data structures, summarized in Table 1. Successor reachability information is maintained only as this is sufficient for any query regarding any edge in a digraph.

Table 1. Maintained data structures.

Structure	Meaning
lp	Union-Find forest with unmerge capability and one tree for each SCC.
scc	Dictionary from SCC root elements to all elements in the SCC.
reach	Dictionary from SCC root elements to all vertices reached from that SCC.

These data structures allow $\mathcal{O}(1)$ queries for locating the SCC a given node x is part of (lp.`find`(x)), enumeration of all nodes in a given SCC for node x (scc[lp.`find`(x)]), as well as "x reaches y" (lp.`find`(y) $\in$ reach[lp.`find`(x)]) or "x reaches" queries (reach[lp.`find`(x)]).

Finally a few convenience dictionaries and functions are given:

- `nuutilaReachSCC` implements Nuutila's algorithm, returns SCCs and reachability information, and is used to recompute SCCs for sub-graphs. The Nuutila implementation is expected to be run only if the reachability between the two nodes in the component actually changed, for practical reasons.

- **pred** returns predecessors of a node such that $\text{pred}[v] \equiv \{x \| (x,y) \in E, y = v\}$, while **succ** returns successors such that $\text{succ}[v] \equiv \{y \| (x,y) \in E, x = v\}$. These are typically stored in arrays with adjacency lists or a matrix.

- `subGraph` returns the sub-graph of nodes and all edges among them:

$$\texttt{subGraph}(S) \equiv (S, \{(x,y) \mid (x,y) \in E, x \in S \wedge y \in S\}).$$

Algorithms 1 and 2 take a graph whose edge (x,y) was just added or removed respectively and update the SCC and reachability information based on that edge. In both cases, self-loops are dealt with first, then skipping of cases which have no effects, followed by merging or collapsing the SCCs if needed, and finally propagation of reachability information.

Algorithm 1 Incremental SCC and reachability update (AddEdgeReachSCC)

Require: Added edge (x,y) with source x and target y
Data: Union–Find lp with `find`/`union`; maps scc, reach, pred

if $x = y$ **then** ▷ self-loop
  if $x \in \text{scc}$ **then** $\text{reach}[x] \leftarrow \text{reach}[x] \cup \{y\}$
  return
$x_c \leftarrow \text{lp.}\texttt{find}(x)$; $y_c \leftarrow \text{lp.}\texttt{find}(y)$
if $x_c = y_c$ **then return** ▷ no SCC change
if $x_c \in \text{reach}[y_c]$ **then** ▷ new SCC discovered
  $C \leftarrow \text{scc}[x_c]$; $R \leftarrow \text{reach}[x_c] \cup \{x_c\}$
  for all $v \in \{\, \text{lp.}\texttt{find}(z) \mid z \in \text{reach}[y_c] \cup \{y_c\} \,\}$ **do**
    if $v \neq x_c \ \wedge \ x_c \in \text{reach}[v]$ **then** ▷ member of new SCC
      $C \leftarrow C \cup \text{scc}[v]$; $R \leftarrow R \cup \text{reach}[v] \cup \{v\}$
      $\text{lp.}\texttt{union}(x_c, v)$; $\text{scc.}\texttt{delete}(v)$; $\text{reach.}\texttt{delete}(v)$
  $\text{scc}[x_c] \leftarrow C$; $\text{reach}[x_c] \leftarrow R$; $y_c \leftarrow x_c$
$P \leftarrow \{x_c\}$; $V \leftarrow \emptyset$; $R \leftarrow \text{reach}[y_c] \cup \{y_c\}$ ▷ back-propagate reachability
while $P \neq \emptyset$ **do**
  remove some v from P
  $\text{reach}[v] \leftarrow \text{reach}[v] \cup R$; $V \leftarrow V \cup \{v\}$
  $P \leftarrow P \cup \{\, \text{lp.}\texttt{find}(z) \mid \exists\, w \in \text{scc}[v] \colon \ z \in \text{pred}[w] \ \wedge \ z \notin \text{scc}[v] \ \wedge \ z \notin V \,\}$

An example illustrating the algorithm is provided in Appendix A. Extensions to support batched edge updates are discussed in Appendix B.

Algorithm 2 Decremental SCC and reachability update (RemoveEdgeReachSCC)

Require: Removed edge (x, y) with source x and target y
Data: maps scc, reach, pred, succ; Union-Find lp; `nuutilaReachSCC`, `subGraph`

1: **if** $x = y$ **then** ▷ self-loop removal
2: **if** $x \in \text{scc} \ \wedge \ \{x\} = \text{scc}[x]$ **then** $\text{reach}[x] \leftarrow \text{reach}[x] \setminus \{y\}$
3: **return**
4: $x_c \leftarrow \text{lp.}\texttt{find}(x)$; $y_c \leftarrow \text{lp.}\texttt{find}(y)$
5: **if** $x_c = y_c$ **then** ▷ rebuild sub-components
6: $S \leftarrow \text{scc}[x_c]$
7: $(\text{subscc}, \text{subreach}) \leftarrow \texttt{nuutilaReachSCC}(\texttt{subGraph}(S))$
8: **if** $|\text{subscc}| = 1$ **then return** ▷ SCC did not change
9: $R \leftarrow \text{reach}[x_c] \cup \{x_c\}$
10: $\text{lp.}\texttt{unmerge}(S)$; $\text{scc.}\texttt{delete}(x_c)$; $\text{reach.}\texttt{delete}(x_c)$
11: $Q \leftarrow \{\, x \mid x \in \text{subscc} \,\}$; $P \leftarrow \emptyset$
12: **while** $Q \neq \emptyset$ **do** ▷ rebuild, propagate subgraph component reach
13: $v \leftarrow$ any of $\{\, w \mid w \in P,\ (\text{reach}[w] \setminus \{w\}) \cap Q = \emptyset \,\}$
14: $\text{scc}[v] \leftarrow \text{subscc}[v]$
15: **for all** $w \in \text{subscc}[v] \setminus \{v\}$ **do** $\text{lp.}\texttt{union}(v, w)$
16: $\text{reach}[v] \leftarrow \text{subreach}[v] \cup \bigcup_{c \in \{\text{lp.}\texttt{find}(z) \mid w \in \text{scc}[v],\ z \in \text{succ}[w]\},\, c \notin S} (\text{reach}[c] \cup \{c\})$
17: $S \leftarrow S \setminus \text{subscc}[v]$; $Q \leftarrow Q \setminus \{v\}$
18: **else**
19: $P \leftarrow \{x_c\}$; $R \leftarrow \text{reach}[y_c] \cup \{y_c\}$
20: $V \leftarrow \emptyset$ ▷ back-propagate reachability with successor checking
21: **while** $P \neq \emptyset$ **do**
22: $v \leftarrow$ any of $\{\, w \mid w \in P,\ (\text{reach}[w] \setminus \{w\}) \cap P = \emptyset \,\}$
23: $\text{reach}[v] \leftarrow \text{reach}[v] \setminus \Big(R \setminus \bigcup_{w \in \text{scc}[v], c \in \{\text{lp.}\texttt{find}(z) \mid z \in \text{succ}[w]\}, c \neq v} (\text{reach}[c] \cup \{c\})\Big)$
24: $P \leftarrow (P \setminus \{v\}) \cup \{\, \text{lp.}\texttt{find}(z) \mid w \in \text{scc}[v],\ z \in \text{pred}[w],\ z \notin \text{scc}[v] \wedge z \notin V \,\}$
25: $V \leftarrow V \cup \{v\}$

4. Correctness and complexity

Firstly self-loops are dealt with as trivial cases. Edge addition which induces a self-loop to a node in a non-trivial component requires no update. For a trivial node, it merely is able to reach itself without having any other effects. Similarly upon removal of a self-loop edge in a non-trivial component causes no changes. In a non-trivial single node component, it merely removes its ability to reach itself. The proof of this is straight-forward from the definition of reachability in SCCs. Self-loop handling is obviously $\mathcal{O}(1)$ for both the incremental and decremental case.

The reachability information always represents a DAG given that any cycle has

been collapsed to a single node via SCCs. Both the strong connectivity and the transitive closure part of the algorithms involves straight-forward formal proofs that can be expressed by the transitive relationship between the edges. The propagation order being correct is proven by the fact that the DFS provides a topological ordering for a DAG [25]. The proofs are straight-forward based on these well-known properties and omitted for brevity. Graph nodes are visited only once when computing SCC adjustments and SCC root nodes are visited a maximum of one time when propagating reachability information in the incremental algorithm. The decremental or batching algorithm can require two visits when determining the new reachability information before adjusting the affected nodes.

Any Union-Find data structure operation takes $\mathcal{O}(m\alpha(n))$ where m here refers to the number of union or find operations and α is the inverse Ackermann function which makes any operation essentially $\mathcal{O}(1)$ [24].

In the incremental case, merging SCCs requires $\mathcal{O}(n)$ for enumerating and combining the SCCs where n is the number of SCCs being merged and worst case $\mathcal{O}(n^2)$ for the reachability set union operations. While for the final step of propagating reachability, the complexity is $\mathcal{O}(m+n)$ for the predecessor traversal and worst case $\mathcal{O}(n^2)$ again for set union operations. This means that given all considerations, the worst case total update time is $\mathcal{O}(m+n^2)$. The algorithm will be expected to perform favorably when edge insertions are expected to be frequent as there is no fixed $\mathcal{O}(m+n)$ DFS traversal.

As for decremental case, determining if an SCC needs to be decomposed requires the worst case of DFS, e.g., $\mathcal{O}(m+n)$ operations. When decomposing an SCC into two or more new SCCs, it requires the time complexity of Nuutila's algorithm. Afterwards, it takes $\mathcal{O}(n)$ time where n in this case is the number of vertices in the SCC being decomposed and the expected worst case $\mathcal{O}(n^2)$ union operations when building the reachFrom dictionary as well as $\mathcal{O}(n^2)$ for finally doing the set subtraction operations. As mentioned Nuutila's algorithm runs only on the sub-graph of the SCC being decomposed. This could be substituted for Tarjan's algorithm as the reachability information obtained only provides an optimization, but there will be a significant performance penalty. Naturally, there are practical time issues due to the set union and subtraction operations used for the reachability in our proposed algorithm and the implementation details of Nuutila. Instead of checking Nuutila to determine that only a single SCC still exists, running a $\mathcal{O}(m+n)$ topological search with early termination is considered an important optimization especially as the graph grows more dense. Nuutila's algorithm with a single SCC has the same complexity but it cannot terminate early and due to all the data structures and reachability construction, it increases the time from doing a topological search by some constant factor of n depending on implementation details. Finally, the reachability propagation in the decremental case is $\mathcal{O}(m+n)$ to traverse the predecessors and compute their DFS postorder. Then enumerating them requires $\mathcal{O}(n)$ yet since we must rebuild the reachability information, it takes worst case $\mathcal{O}(m+n^2)$ to traverse all the predecessors and make appropriate set subtraction operations. This still makes the worst case total update time $\mathcal{O}(m+n^2)$.

This is why it can be worth checking reachability with a DFS before doing the computation, perhaps based on a fixed heuristic based on the density of the graph, or number or size of the components.

When considering the set operations used for reachability, the complexity is mostly dependent on the cost of set union and intersection operations. The classical algorithms and the proposed algorithm are generally intended for sparse graphs. However, dense graphs tend to have more and larger SCCs so in practice it will be very efficient in this context as well. In fact, as far as the SCCs are concerned, the incremental algorithm is reasonably efficient, as we perform only a single propagation, and especially for dense graphs given that m is bounded by n^2 as only unweighted graphs need to be considered. However, the use of Nuutila's algorithm and reachability propagation is the primary drawback. Even though reachability allows for easy incremental computation of SCCs, its own maintenance requires a backwards traversal of the SCC DAG with set union operations in the incremental case and both union and subtraction operations in the decremental case. The worst case will appear when adding an edge from a node which is reached by half of nodes to a node which reaches the other half of the nodes requiring $\frac{n^2}{4}$ set union operations. Deleting that same edge requires $\frac{n^2}{4}$ set subtraction operations.

The worst-case space complexity is $\mathcal{O}(n)$ for the Union-Find and SCC dictionary, while being $\mathcal{O}(n^2)$ for the reachability information. Although for dense graphs, matrices are of course preferred over adjacency lists or sets for storing predecessors and successors as well as reachability, when SCCs are used, in fact, the proposed dictionary would be better for maintaining reachability information to avoid complications from re-labeling. The worst case storage and time complexity thereby occurs in a DAG where given the nodes labeled in ascending numerical order, each node connects to every other node with a larger label.

The algorithm is less effective on nearly acyclic graphs where the problem reduces to transitive closure. It performs best when SCCs are large or numerous, which is typical for control flow graphs. The performance of the algorithm depends on a lot of implementation details, for example, such as that sets or dictionaries might be implemented with hash tables or balanced trees yielding $\mathcal{O}(1)$ vs. $\mathcal{O}(\log n)$ time complexity for additions or look-ups respectively. The time complexity for set union and subtraction operations also could be considered as operations such as simple set operation equivalences e.g., when A is a set and B is a set of sets allows $A - \bigcup_{S \in B} (S) \equiv A - \bigcup_{S \in B} (A \cap S)$ to reduce sizes for union operations.

There is also the idea of maintaining the reverse reachability information so instead of fast lookup for "x reaches" queries, fast look-ups could also occur for "x is reached by" queries. There is insignificant additional time complexity to maintaining the reverse data structure, it is simply updated any time reach is however since the dictionary is maintained as a mapping of components to all reachable nodes and not components to reachable components, some simple reduction of nodes to components and expansion from components to nodes is needed giving nonetheless worst case $\mathcal{O}(n)$.

It should be noted that the worst-case complexity of many fully dynamic algo-

rithms will often be similar to the average-case of the offline algorithm. In other words, any online algorithm should never be worse than linear in terms of nodes and edges for SCC maintenance or quadratic relative to the nodes for transitive closure. Constructing scenarios where the worst case occurs generally relies on pathological cases though and does not actually happen in practice, e.g., a complete DAG whose vertices are totally ordered and whose $\frac{n(n-1)}{2}$ edges all go from earlier to later vertices. The initial construction of a large CFG would still ideally be done via an offline algorithm in the compilation context because of syntax and semantic guarantees of programming languages, however, in the context of decompilation where self-modifying or unreachable code is an issue, the penalty of incremental construction would be a necessity. The average-case complexity is more useful however it depends on a lot of averages of graph properties, such as the average number of successors or predecessors, the graph density, the average size of SCCs, the number of SCCs in proportion to the number of nodes, etc.

5. Experimental results

The testing program was written in carefully optimized Python 3 code using dictionary and set support from the language and executed on a Intel i7-9750H CPU. This is compared with an optimized Nuutila implementation which supports self-loops and also uses the suggested component stack duplicate and sorting modifications. The online and offline algorithms were both modified to utilize a stack data structure and avoid use of recursion which due to interpreter stack depth limitations would limit the number of vertices to around 1000 otherwise. The public Git repository contains the Python 3 implementation, including sccreach.py, the test.py benchmarking harness, CFG extraction support such as gengraphs.idc, benchmark inputs in cfgs/, and rerun instructions [15]. The released CFG inputs are intended to make the reported experiments rerunnable; exact package versions and Windows build numbers for the original binaries were not recorded, and proprietary disassembly has been removed from the public sanitized data.

We first tested on random graphs and these results, provided in Appendix C, are not included for brevity yet they reflect the realistic experiments. We then tested these results by using realistic CFGs extracted with Hex-Rays IDA Freeware 7.0. IDA provides IDC scripting and can export function flow graphs in textual Graph Description Language (GDL) format, which was used by a short extraction script before parsing the graph edges [10, 11]. The corpus contains Linux x86-64 ELF binaries sendmail (1,599 functions), smbd (799), and vsftpd (705), and Windows 10 x64 PE binaries explorer.exe (11,703), kernel32.dll (2,533), and user32.dll (2,651). The private extraction artifacts contain disassembly, so the public repository redistributes only sanitized graph topology and labels sufficient to rerun the benchmark. The results appear in Figure 1.

The results exactly confirm the suspicion that if the graphs have a lot of non-trivial components or large components as in the first experiment, the incremental algorithm will perform exceptionally well. In fact, it is only about a constant a_1

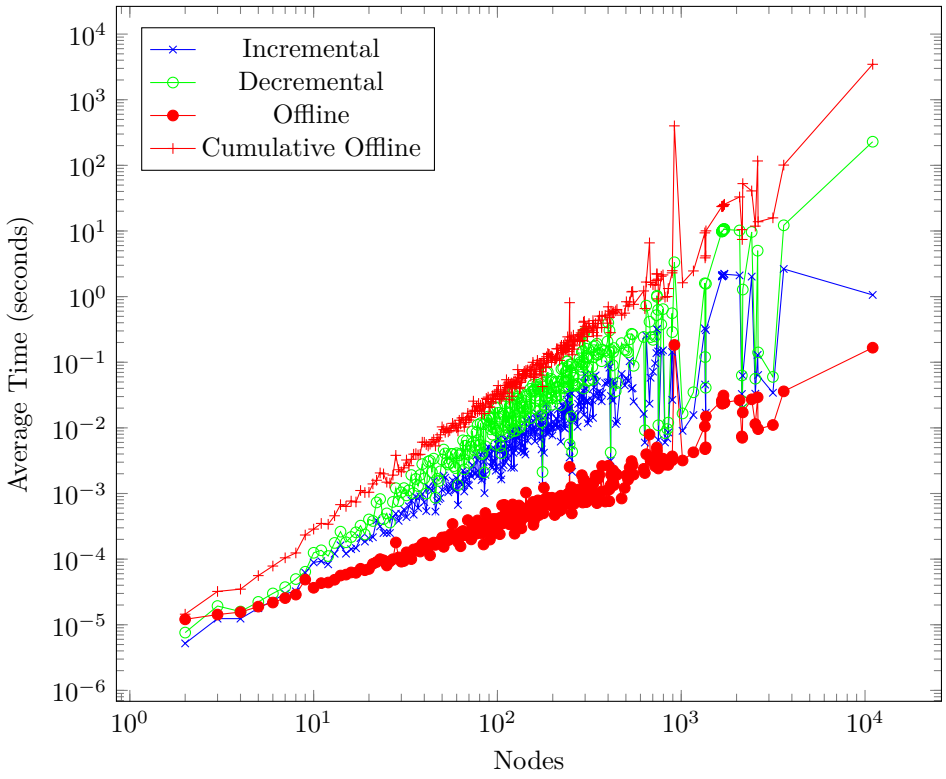


Figure 1. Real Control Flow Graph Edge Addition and Edge Deletion Execution Time.

time slower than doing a single offline computation after all the edges are added. In this context, the deletion does not perform as well but it is only a factor slower based on the number of nodes, not edges or roughly $a_2 n$ times slower than a single offline computation where a_2 is a constant. On the other hand, with the locality and average out-degree causing fewer non-trivial SCCs and closer to a DAG, the quadratic propagation of reachability information becomes the predominant factor. However, realistic CFGs tended to perform more closely to the locality model than the first model, largely because CFGs tend to be far more sparse than the first model, and they are constructed in a way that by the time the SCC is identified, the inter-component edges would have already been added. Unsurprisingly in all the cases, both algorithms outperform cumulative offline computation, the decremental was always slower than the incremental as by design, and the incremental was slower than a single offline computation.

6. Conclusion

This represents only a step towards finding a highly efficient and deterministic fully dynamic algorithm in this area. The challenge of finding the right data structure has left it open so that although worst-case bounds are known for such an algorithm, there is more to be done to make this practical for very large graphs. Specifically a better data structure to maximize time and space efficiency of this algorithm is needed.

As well, most dynamic graph algorithms focus on maintaining only a single type of graph information, often only incrementally or decrementally based on very specialized data structures. This paper has explored the opportunity to combine maintenance of two types of graph information simultaneously where both strong connectivity and reachability are used to lower the complexity and assist with computing the other.

In the future, more complex fully dynamic algorithms which can maintain even more simultaneous types of information are to be expected. This would be a potential basis for a shift in how compilers are constructed if the control flow graph never required explicit re-computation.

Furthermore, fully dynamic algorithms which can compute loop nesting forests for reducible and irreducible graphs are a logical extension to SCC maintenance though requiring techniques based on DFS, dominators or a recursive application of SCCs. Lastly, incremental algorithms for maintaining reducible graphs based on irreducible graphs or single-entry single-exit regions in graphs from arbitrary graphs would be a step forward for representing arbitrary code in high-level programming languages and would benefit from already having efficient reachability queries.

References

[1] A. Bernstein, M. Probst, C. Wulff-Nilsen: *Decremental Strongly-Connected Components and Single-Source Reachability in near-Linear Time*, in: Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA: Association for Computing Machinery, 2019, pp. 365–376, isbn: 9781450367059, doi: 10.1145/3313276.3316335.

[2] S. Chechik, T. D. Hansen, G. F. Italiano, J. Łącki, N. Parotsidis: *Decremental Single-Source Reachability and Strongly Connected Components in $\tilde{O}(m\sqrt{n})$ Total Update Time*, in: 2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS), 2016, pp. 315–324, doi: 10.1109/FOCS.2016.42.

[3] C. Demetrescu, G. F. Italiano: *Fully dynamic transitive closure: breaking through the $O(n^2)$ barrier*, in: Proceedings 41st Annual Symposium on Foundations of Computer Science, 2000, pp. 381–389, doi: 10.1109/SFCS.2000.892126.

[4] R. W. Floyd: *Algorithm 97: Shortest Path*, Commun. ACM 5.6 (June 1962), p. 345, issn: 0001-0782, doi: 10.1145/367766.368168.

[5] P. G. Franciosa, D. Frigioni, R. Giaccio: *Semi-dynamic breadth-first search in digraphs*, Theoretical Computer Science 250.1 (2001), pp. 201–217, issn: 0304-3975, doi: 10.1016/S0304-3975(99)00132-2.

[6] L. Georgiadis, G. F. Italiano, N. Parotsidis: *Incremental Strong Connectivity and 2-Connectivity in Directed Graphs*, in: LATIN 2018: Theoretical Informatics, ed. by M. A. Bender, M. Farach-Colton, M. A. Mosteiro, Cham: Springer International Publishing, 2018, pp. 529–543, isbn: 978-3-319-77404-6, doi: 10.1007/978-3-319-77404-6_39.

[7] L. Georgiadis, G. F. Italiano, N. Parotsidis: *Strong Connectivity in Directed Graphs under Failures, with Applications*, in: Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '17, Barcelona, Spain: Society for Industrial and Applied Mathematics, 2017, pp. 1880–1899, doi: 10.1137/1.9781611974782.123.

[8] B. Haeupler, T. Kavitha, R. Mathew, S. Sen, R. E. Tarjan: *Incremental Cycle Detection, Topological Ordering, and Strong Component Maintenance*, ACM Trans. Algorithms 8.1 (Jan. 2012), issn: 1549-6325, doi: 10.1145/2071379.2071382.

[9] M. Henzinger, S. Krinninger, D. Nanongkai: *Sublinear-Time Decremental Algorithms for Single-Source Reachability and Shortest Paths on Directed Graphs*, in: Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing, STOC '14, New York, New York: Association for Computing Machinery, 2014, pp. 674–683, isbn: 9781450327107, doi: 10.1145/2591796.2591869.

[10] Hex-Rays: *Announcing version 7.6 for IDA Freeware*, url: https://hex-rays.com/blog/announcing-version-7-6-for-ida-freeware (visited on 06/09/2026).

[11] Hex-Rays: *gen_flow_graph*, url: https://docs.hex-rays.com/9.0/developer-guide/idc/idc-api-reference/alphabetical-list-of-idc-functions/1253 (visited on 06/09/2026).

[12] A. Karczmarz, M. Smulewicz: *On Fully Dynamic Strongly Connected Components*, in: 31st Annual European Symposium on Algorithms (ESA 2023), ed. by I. L. Gørtz, M. Farach-Colton, S. J. Puglisi, G. Herman, vol. 274, Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 68:1–68:15, isbn: 978-3-95977-295-2, doi: 10.4230/LIPIcs.ESA.2023.68.

[13] V. King: *Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs*, in: 40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039), 1999, pp. 81–89, doi: 10.1109/SFFCS.1999.814580.

[14] G. Kusper, Y. Zijian Győző, B. Nagy: *Using extended resolution to represent strongly connected components of directed graphs*, Annales Mathematicae et Informaticae Accepted manuscript (2023), issn: 1787-6117, doi: 10.33039/ami.2023.08.011.

[15] G. Morse: *dyngraph: Dynamic graph maintenance source code and experiment scripts*, url: https://github.com/GregoryMorse/dyngraph (visited on 06/08/2026).

[16] E. Nuutila: *An Efficient Transitive Closure Algorithm for Cyclic Digraphs*, Information Processing Letters 52 (1994), doi: 10.1016/0020-0190(94)90128-7.

[17] E. Nuutila, E. Soisalon-Soininen: *On finding the strongly connected components in a directed graph*, Information Processing Letters 49.1 (1994), pp. 9–14, issn: 0020-0190, doi: 10.1016/0020-0190(94)90047-7.

[18] D. J. Pearce: *A space-efficient algorithm for finding strongly connected components*, Information Processing Letters 116.1 (2016), pp. 47–52, issn: 0020-0190, doi: 10.1016/j.ipl.2015.08.010.

[19] P. Purdom: *A transitive closure algorithm*, BIT Numerical Mathematics 10.1 (Mar. 1970), pp. 76–94, issn: 1572-9125, doi: 10.1007/BF01940892.

[20] L. Roditty: *A Faster and Simpler Fully Dynamic Transitive Closure*, ACM Trans. Algorithms 4.1 (Mar. 2008), issn: 1549-6325, doi: 10.1145/1328911.1328917.

[21] P. Sankowski: *Dynamic transitive closure via dynamic matrix inverse: extended abstract*, in: 45th Annual IEEE Symposium on Foundations of Computer Science, 2004, pp. 509–517, doi: 10.1109/FOCS.2004.25.

[22] Y. Shiloach, S. Even: *An On-Line Edge-Deletion Problem*, J. ACM 28.1 (Jan. 1981), pp. 1–4, issn: 0004-5411, doi: 10.1145/322234.322235.

[23] R. Tarjan: *Depth first search and linear graph algorithms*, SIAM JOURNAL ON COMPUTING 1.2 (1972), DOI: 10.1137/0201010.

[24] R. E. Tarjan: *Data Structures and Network Algorithms, 2. Disjoint Sets*, in: Society for Industrial and Applied Mathematics, 1983, pp. 23–31, DOI: 10.1137/1.9781611970265.ch2.

[25] R. E. Tarjan: *Edge-disjoint spanning trees and depth-first search*, Acta Informatica 6.2 (June 1976), pp. 171–185, ISSN: 1432-0525, DOI: 10.1007/BF00268499.

[26] B. Yang, D. Wen, L. Qin, Y. Zhang, X. Wang, X. Lin: *Fully Dynamic Depth-First Search in Directed Graphs*, Proc. VLDB Endow. 13.2 (Oct. 2019), pp. 142–154, ISSN: 2150-8097, DOI: 10.14778/3364324.3364329.

Appendix A: Example

Consider the graph in Figure 2. When the edge $(5, 3)$ is added, it creates a new SCC as seen in Figure 3. By traversing forward through SCCs reachable from the destination node's SCC, the component represented by 4 traverses to component 7 which is the source node component and finally components 8, 9 and 10 are ignored since they do not reach component 4. The two components are collapsed into a new component and their reachability merged by a union operation. Finally the reachability for the predecessors of the new component is propagated backwards as it would be for an edge addition which does not change the SCCs. Likewise, if the edge $(5, 3)$ is then deleted, Nuutila's algorithm computes the SCCs for the sub-graph of the SCC being decomposed as shown in Figure 4. As it can be seen, component 8, 9 and 10 are not in the sub-graph so as the SCCs are rebuilt all of them are included in the new reachability sets. Then the SCCs which reach the target as well as all SCCs the target reaches is computed in a single topological ordered predecessor-based traversal of nodes. Afterwards, predecessor-based propagation starts from the new SCCs, to allow precise subtractive operations to recompute reachability sets affected by the deletion, which is identical to the propagation that occurs for an edge deletion that does not change the components.

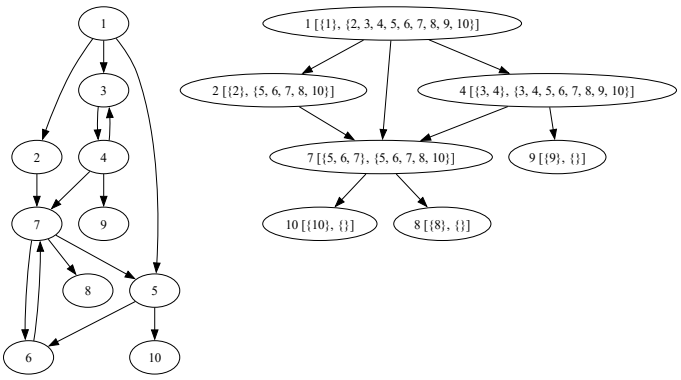


Figure 2. A digraph and its corresponding SCC DAG with its SCCs and reachability shown.

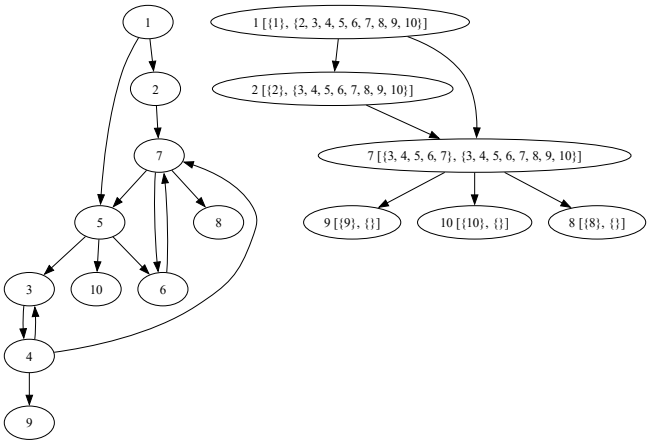


Figure 3. The digraph with edge (5, 3) added.

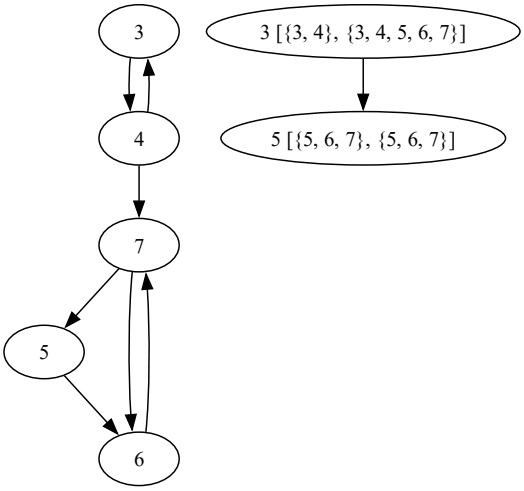


Figure 4. The sub-graph and its corresponding SCC DAG with its SCC and reachability when edge (5, 3) is removed.

Appendix B: Batching

Batching can have special optimizations for for multiple edges which all come from a single source node or go to a single target node in many circumstances including this one. However, we consider only arbitrary edges, rather than various special batching cases. The basic idea is that the time complexity for processing a batch of edges must have the same worst case as the non-batching dynamic algorithms making the number of edges in the batch an additive term and not multiplicative term.

In the incremental context, self-loops and edge additions which are already within the same component should be filtered out. Additionally if this filtering yields only a single edge then it should be processed with the non-batching algorithm, which is a standard optimization. In the decremental context, self-loops should be filtered out and optionally traversal reachability checks can filter out edges within a component where the source still reaches the target via a longer path. The single edge case also applies. After this basic processing, two options are then possible when converting the standard incremental and decremental algorithms to a batching algorithm for both the incremental and decremental case. It should be noted that Nuutila's algorithm will need to be applied regardless in all situations with batching but in different ways.

The first method for the incremental context simply uses Nuutila to recompute the SCCs and reachability for the graph assumed to have the new edges already present but having its existing SCCs collapsed. The results are then directly merged. The second method uses Nuutila on the subgraph induced only by the filtered added edges and any components reachable from source or target nodes of these edges, followed by merging and propagating reachability for these nodes.

For the decremental case, the first method is equivalent in that Nuutila can recompute the SCCs and reachability for the graph assumed to have the deleted edges already removed but having SCCs collapsed except the ones which are being decomposed. The second method only uses Nuutila for the components to be decomposed, and then re-merges the resulting components followed by propagating reachability. In fact, the algorithm as written is very easily adapted to the batching setting. It is the incremental case which requires a different SCC strategy as the detection of merging components via reachability is possible only if the reachability information is continuously updated. One strategy could be to process the edges which are detected as forming an SCC in sub-batches and propagating their reachability, and then repeatedly doing this until no more SCC merges can be done. Yet this would lead to a worst case time complexity equivalent to using the non-batching incremental algorithm and hence inappropriate for batching.

Unfortunately practically speaking there are cases where the proposed batching algorithms would perform slower than the non-batching algorithms, unless the batch sizes are large. In the event that current graph is small and the number of batched edges is large enough, an offline computation would achieve better performance without batching, with a reasonable threshold being a constant factor for the edge insertion case and a factor of n for the edge deletion scenario. Even with carefully optimized batching code, in practice the best performance may occur when small batches are done with non-batching algorithms, large batches are done with offline computation and the batching algorithms are used for a carefully determined range in between.

The most elegant batching algorithm would handle intermixed edge additions and deletions, not just one or the other. However, this can be done efficiently by using a general principle for intermixed batching. Namely, filtering out additions and deletions of identical edges, followed by dividing it into separate sets of edge

additions and edge deletions and using the respective incremental or decremental batching algorithms only once each.

Appendix C: Random graph experiments

Here we build random connectivity maintained graphs with $10 - 200$ nodes by adding $\frac{n^2}{2}$ random edges until the point that the graph becomes dense. This number of edges guarantees at least one non-trivial SCC will be present. Then we delete those same edges in the reverse order to maintain connectivity which gives the results in Figure 5.

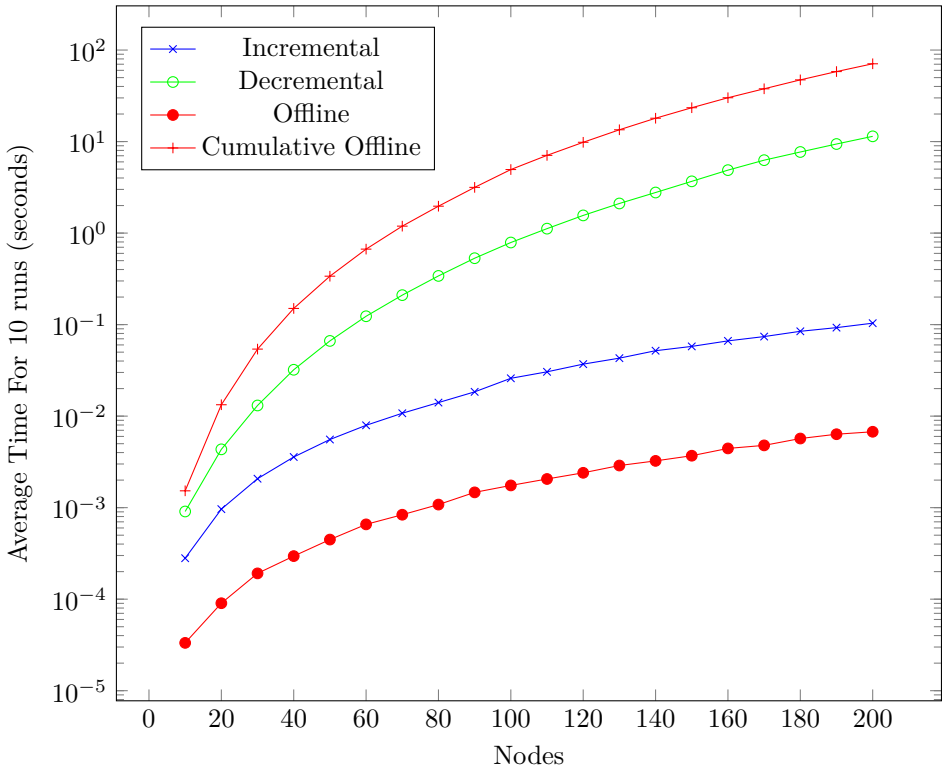


Figure 5. Random Connected Graph Edge Addition and Deletion Performance Execution Time Averaged Over 10 Runs.

The problem with random graphs is that eventually a single large component will emerge and does not necessarily provide a good understanding of how it would perform on actual CFGs. Instead, a better method is to use an out-degree and locality parameter so that there will be many smaller SCCs. However, this is still unrealistic as large loops can and do occur in real programs. So instead we opt for

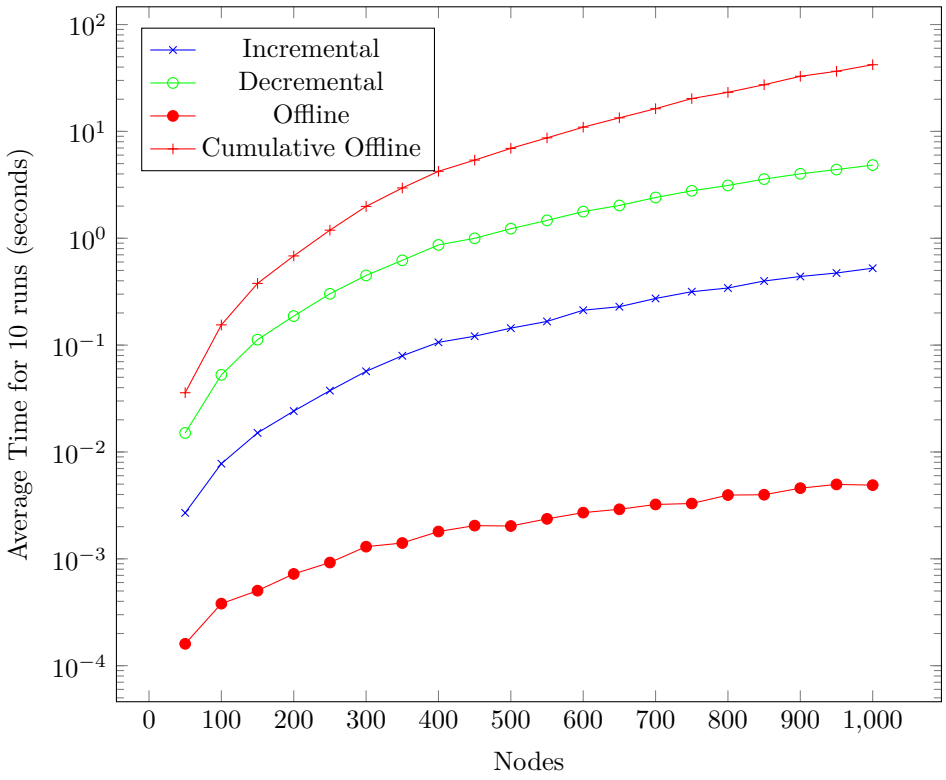


Figure 6. Random Fixed Locality=5 and Average Out-Degree=5 Connected Graph Edge Addition and Deletion Execution Time Averaged Over 10 Runs.

testing with both methods to evaluate the performance in different circumstances. The results for a fixed locality and average out-degree are presented in Figure 6.

Since maintaining connectivity of the graph skews the randomness towards nodes which became part of the graph earlier in the edge addition choices, it should be noted that mathematically the results here are not trying to achieve traditional standards of randomness where there is typically a fixed expected probability for each edge. Given that this algorithm has use cases which are practical in nature, it made more sense to sacrifice the simple probabilities offered by random graphs that do not necessarily have connectivity, for a more context sensitive variant with biases which are understood. A graph is said to be connected if there exists one or more nodes which can reach every other node, and one of these nodes in the CFG context is the entry and termed the root.

Annales Mathematicae et Informaticae
63 (2026) pp. 107–121
DOI: 10.33039/ami.2026.07.003
URL: https://ami.uni-eszterhazy.hu

Case study: Refactoring with design and architectural patterns

Bálint Dominik Orosz, Tamás Kozsik

ELTE Eötvös Loránd University, Budapest, Hungary
balint.orosz@inf.elte.hu, tamas.kozsik@elte.hu

Abstract. Many software quality issues can be attributed to the fact that developers fail to adhere to architectural decisions, or that during the life cycle of the software the original design decisions fade away, and violations of the architecture emerge. In this paper, we present a case study on refactoring with design and architectural patterns. For describing the architecture of software tools, we propose a methodology, which relies on a language that allows software architects to natively express, and enforce the application of design and architectural patterns. The subject of the case study is an industrial software tool, which have been refactored by respecifying it using the introduced language. From this new specification, the transpiler of the language was able to generate a code base with correctly implemented design and architectural patterns, which could be compared to the original. Our methods were validated through expert reviews, and the Chidamber and Kemerer metrics suite. These showed that with our approach the quality of the refactored software tool can be increased, while preserving functionalities, and also satisfying originally neglected non-functional requirements.

Keywords: architecture, design pattern, refactoring, domain-specific language

2020 *Mathematics Subject Classification:* Primary 68N15, 68N19, 68N20; Secondary 68M99

1. Introduction

With its ever-changing requirements, modifications are inevitable during a software's lifecycle. Many studies have identified the structural issues of the implementation as the main cause for the majority of future software changes [2, 10]. For this reason, several automated refactoring approaches have been developed to

Submitted: March 31, 2026
Accepted: July 6, 2026
Published online: July 17, 2026

transform code bases from a structural point of view, so that they would apply possible design and architectural patterns, in hopes of increasing code quality [1, 11, 16, 18, 21]. However, we argue that the outputs of these refactorings do not necessarily provide satisfactory guarantees for the avoidance of future modifications, as there is still a possibility that developers reintroduce faulty design decisions.

In this paper, we present a case study, along with a language supporting explicit design and architectural pattern applications, to present an approach which could be used to improve the quality of industrial code bases. The language allows software architects to better express design decisions, and provides support for advanced code generation with forced application of selected architectural and design patterns. The approach manifests itself in a language combining architecture specification, first-order formal logic based description of functionality, and concrete programming language code. To make our approach more focused and more effective, we restrict ourselves to a subset of potential applications: the approach is designed to support the construction and restructuring of software development tools, ones which are often used in Continuous Integration and Continuous Delivery (CI/CD) pipelines, or desktop applications used by software developers. This is also in line with our primary industrial motivation, namely the improvement of a substantial code base used daily by a software development company.

During our study, since the targeted industrial code base has been implemented in C#, the language proposed in this paper can be seen as a direct extension of C#: an extension which raises C# to the end user level to that of a software architect. However, the provided approach in this paper is independent of any languages, it is fit for most modern object-oriented programming languages also (for instance Java).

The main contribution of this paper is a case study, during which we demonstrate that the use of a specific language describing pattern applications would lead to a robust design and refactoring framework. This would allow, albeit in a restricted domain, software companies to develop better quality code by allowing architects to make architectural and design pattern applications explicit on a high abstraction level, and force implementations to follow requirements imposed by architectural decisions. The applicability of the approach is demonstrated on a case study, evaluated using a set of standard code quality metrics.

Although, the supporting language and its code generation process can be seen as essential accessories of the presented case study, their formal specification is outside the scope of this paper. Rather, its higher level constructs are presented along with key examples, providing a better understanding at this level.

The paper is structured as follows: Section 2 reviews related work, Section 3 presents the titular language, its main concepts, and its layered structure through key examples. Subsection 3.1 explains the supporting code generation process of the language. Section 4 presents the case study on the refactoring of an industrial software tool, the results of which are analyzed and evaluated in Subsection 4.3. Section 5 summarizes the achievements, addresses the limitations of the proposed approach, and discusses future work opportunities.

2. Related work

In the past decades, there have been quite a few studies related to design patterns, their applications and possibilities, and automated code refactoring attempts. Recent systematic literature reviews on these topics [1, 22] have revealed useful insights.

Baqais, Alshayeb published a systematic literature review on automatic software refactorings [1]. They analyzed and compared 41 papers from several aspects. They concluded that 78% of the papers applied refactorings on the code level, while 10% of them on the model level. 5% and 5% used the levels of package and UML, while 2% applied transformations to a graph representing the original software version. Also, for these refactorings, only 29% of the published papers provided some tool for supporting their methodologies, while the majority of them only proposed their techniques. As far as the applied techniques are concerned, most of them used some genetic algorithms or their variants, while applications of hill climbing, simulated annealing, and particle swarm optimizations were also used partially. 54% of these techniques were especially created for automation purposes. 56% of the papers used statistics, as their algorithms are usually non-deterministic.

Tokuda, Batory studied automated code refactorings [21]. Their studies were closely related and narrowed down to object-oriented designs, as they believed that code changes might be automatically applied through visual modifications done on a Graphical User Interface (GUI). This way, architects could visually modify software design and specification through types and their connections.

Jeon, Lee, Bae defined an automated refactoring approach with design pattern-based code transformations for Java applications [11]. Their approach primarily converted Java applications to Prolog facts based on specific rules, and matched these with pre-defined Prolog rules corresponding to each input design pattern. Through these Prolog queries, a number of candidate spots could be identified as potential design pattern application areas in the examined code base.

Shahir, Kouroshfar, Ramsin also investigated the opportunities of applying design patterns to refactor real-world models. In [18], they showcased a case study, in which they refactored an already existing system from their previous studies. It is notable that they already applied design patterns during the initial requirements engineering and analysis phases of the software development lifecycle. Their results showed an increased code cohesion and reduced coupling properties, while the output model became more comprehensible for stakeholders.

Elish, Alshayeb attempted to classify *refactoring to pattern* techniques as their main argument was that choosing between these techniques is not as straightforward as they should be [6]. When it came to the priorization of the techniques, their approach was to measure through different metrics how the quality attributes of the modified system changes when design patterns are introduced. They use the Chidamber and Kemerer metrics suite [3] for internal quality attributes, and the ISO/IEC 9126 standard for external ones. Although, their approach was useful overall, their methodology presumed that design patterns are only introduced to

the system with low-level refactorings. Therefore, their methodology is to be tested against higher level code bases and also higher level refactorings.

Mahouachi, Kessentini, Ghedira noted that in previous works, the detection of design-level problems and definition of possible solutions solving these problems were sharply separated. Therefore, they experimented with handling these two steps as one, and accordingly, they specified a genetic algorithm [13]. The input of this algorithm was a set of examples, containing faulty code segments and manually created possible solutions for their corrections, and also the set of possible refactoring rules was provided as input. The algorithm then applied the available refactoring rules to the examples, and then the output was compared to the expected manual solutions by a fitness function.

Zeinab et al. investigated how design patterns and anti-patterns mutate with the ever-changing development requirements, and how the fault-proneness properties of these designs change with modifications [23]. They specifically examined whether the presumed mutations across design patterns and anti-patterns are possible, and also the possibility of these mutations (for this reason, they used Markov models).

A noteworthy example is the use of aspect-oriented programming through the AspectJ extension of the Java language in the works of Hannemann, Kiczales. In [8], they used aspects to extract the repetitive implementation of certain design patterns from the source code. This way, design pattern implementations were separated from their specific applications.

Another notable example can be found in the works of Dévai et al., in which they introduce a system for executable UML specifications [4]. Their system introduces special description extensions to the UML set of tools, which later can be used to specify applications exclusively on the design level. This approach tries to address and tackle the scalability issues of UML, which makes its use in an industrial environment difficult.

In our previous works [15], we have already experimented with error detection related to design and architectural patterns, and the conformance of code bases to them.

Based on the previous related works, a similarity is that our approach also aims to eliminate boilerplate code to reduce the possibility of errors, by using patterns as templates, with automated code generation.

A notable difference is that most works have a clear distinction between design and architectural patterns, whereas we handle architectural patterns as design patterns applied to higher level design constructs. Also, most works mainly focus on specific examples of design patterns, rather than covering the whole spectrum of design and architectural patterns. Furthermore, most works use large open-source code bases for validation, while our primary target is a specific software tool, an independent subsystem of a larger industrial code base.

Therefore, during our studies, we focused on these deficiencies, and integrated these insights into our approach. Regarding the validation techniques, along with expert reviews, we also use the Chidamber and Kemerer metrics suite, similarly to other works.

3. LADY: Language for architectural design

LADY is a language for describing software architecture, in particular the architecture of software tools. Hence, the language is domain-specific in the sense that it is best suited to formulate the design of command-line applications, which can be used in CI/CD pipelines, or desktop applications with a graphical user interface – although it can be used to describe general standalone software components as well.

LADY especially targets software architects (with a more senior understanding regarding design decisions) as end users, and enables them to specify software tools efficiently, with a clear description of design decisions. It allows architects to natively express applications of patterns relevant in this domain, such as Model-View-ViewModel, or pipe-and-filter. Furthermore, LADY prohibits certain language features if they interfere with the application of the applied patterns, significantly complicating developers to accidentally deviate from the original design, although it remains technically possible to weaken architectural guarantees through plugin codes.

According to these principles, LADY has two sets of language constructs, the first being the general programming constructs suitable to define software tools, the other being the set of pattern application constructs. Naturally, elements of the former set are used to determine the functionalities of the specified application, while elements of the latter set are going to serve as higher level structuring guidelines, by influencing the architectural setup of the specified application.

In this regard, LADY should not be considered merely as an architecture description language, as its main purpose is not only to describe and govern architectural setups (it also handles lower-level constructs, such as design patterns). Rather it is closer to being a model-driven code generation language, as it also aims to guide the whole design process. Although, it is notable that the language is planned to be part of an automatic refactoring framework in future works.

3.1. Code generation process

Instead of a traditional compiler, LADY is accompanied with a code-to-code compiler, a transpiler, which outputs C# code (instead of executable binaries or some intermediary virtual machine code). This is especially useful when the language is used in refactoring (this being one of the primary motivations behind it), because in this way it becomes possible to observe and compare the refactored output code to the original. Naturally, from this setup, the compilation of executable binaries does not require further modification of the methodology.

The code generation in the transpiler ensures that the output code meets the general requirements for the application of the supported patterns. Moreover, when the currently used patterns do not prevent it (or other factors do not interfere with it), the transpiler will generate concurrently executing code by default. For this purpose, certain static code analysis rules (like the immutability check of user-defined types or the inference-check of sequential expressions) are applied by

the transpiler to collect necessary information, which influences the transpilation process and its output.

Design patterns are implemented by the transpiler as expected [9], the only notable exception being the *singleton* design pattern, which is often regarded as an anti-pattern [17]. In LADY, if a type is marked singleton, then the type is not going to know of itself whether it is a singleton type, rather, a dependency injection framework is used in the context of the whole application, and the annotated type is registered into it as a singleton implementation. This results in the type effectively being a singleton throughout the whole life-cycle of the application, while still maintaining code quality standards and best practice principles.

3.2. Constructs for describing software tools

At the architecture design level, the main emphasis is on the specification of functionalities, and the placement of these functionalities in special structures (as provided by the first set of constructs of LADY), which are going to be guided and safeguarded by the other set of language constructs: the architectural and design patterns.

The structural constructs of LADY are designed with a layered approach. The bottom layer defines the layer of object types. LADY provides two kinds of such types: *entities* and *records*. The former is mainly used for persistence purposes (like data transfer objects—DTOs), while the latter is closer to the traditional concept of objects (i.e., types with functionalities).

The next layer is concerned with the specification of functionalities in so-called *actions*. Ideally, these *actions* are described with formal logic. For practical reasons, if something cannot be, or is too complex to be, formalized, C# code can also be used.

One layer higher, types and *actions* are encapsulated by *components*. These are the main guiding structures within the language, and an application is basically built on their collaboration.

In the top layer, the structure of the main *application* needs to be defined. An application can either be a console or a desktop application. Within these, the possible *commands* of the *application* need to be placed, consisting of their input parameters and validators (according to the usual best practices). Also, each *command* has to be connected to some functionality through an *action* of a *component*.

Key examples

Listing 1. A main entry point for a console application with a specific command.

```
1 ConsoleApplication
2 {
3     commands: [
4         {
```

```
            name: "analyzer",
            input: {
                scopePath: {
                    type: String,
                    validators: [REQUIRED]
                }
            }
            action: NamespaceAnomalyAnalyzer.Main(scopePath)
        }
    ]
}
```

Listing 1 shows an example for a possible main entry point of a console application with a specific command. This code will ensure that during the transpilation process, a C# console application will be generated, with only one possible invokable command, with the name *analyzer*, which has a required *String* input parameter. Besides, the action of this command is referring to the *Main* action of the *NamespaceAnomalyAnalyzer* component. This example gives an abstract specification of this top-level application description, which is free of any concrete implementation details.

Listing 2. A component description containing an action.

```
component NamespaceAnomalyAnalyzer
{
    record AnalyzerRecommendation
    {
        Path: String;
        Namespace: String;
        SupposedPath: String;
        SupposedNamespace: String;
    }

    action Main
    {
        input: { scopePath: String; }
        output: { result: Sequence<AnalyzerRecommendation>; }

        logic: {
            projectFiles := <code lang="csharp">...</code>
            AND
            ...
            AND
            result := map ...
        }
    }
}
```

Listing 2 depicts a *component* and an *action* specification. Notable cases include the possibility to define *records* within the context of a *component* (and therefore making it component-specific), the abandonment of concrete collection types and the use of an abstract *Sequence* type (to make sure that the most suitable alternative is used according to the specific example), and the possibility to use mixed specification styles (formal and plugin code) during the specification of an *action*'s functionality.

3.3. Pattern-related language constructs

LADY provides a way to describe the application of architectural and design patterns natively. These patterns are present as higher organizing principles, and they ensure a more robust architectural setup within the specification.

Regarding the pattern-related language constructs, the following architectural patterns [20] have been selected to be part of the language: the *Model-View* (MV), the *Model-View-ViewModel* (MVVM), and the *pipe-and-filter* architectures. Besides these, LADY supports the application of a number of design patterns [7], namely: *factory*, *builder*, *singleton*, *proxy*, *composite*, *iterator*, *mediator*, *command*, and *chain-of-responsibility*.

Pattern applications in the language work with an underlying *context* definition, which is mostly not noted explicitly in the source code. That is, pattern applications have a limited and well-defined boundary, in which their applications take effect and ensure their purposes. For instance, an MV or MVVM architecture might determine the whole setup of an application specification, while they might also be used within the context of a single component of the application. Notably, the *pipe-and-filter* architecture might be used with multiple contexts, as its use on the level of higher-level commands (for a whole command) and actions (for a smaller set of logical steps) also seem rational and justified.

Key examples

Listing 3. An action with a pipe-and-filter modifier causing the action's functionality to execute its steps in a sequential manner.

```
pipe-and-filter action Main
{
    input: { scopePath: String; }
    output: { result: Sequence<AnalyzerRecommendation>; }

    logic: {
        projectFiles :=
            <code lang="csharp">
                Directory.GetFiles( scopePath, "*.csproj",
                                    SearchOption.AllDirectories )
            </code>
        AND
        projects := map projectFiles (projectFile) ->
                      create ExtendedProject(scopePath, projectFile)
        AND
        sourceFiles := flatten projects (project) ->
                         project.SourceFiles
        AND
        result := forall sf in sourceFiles:
                    ((sf.Namespace != sf.SupposedNamespace)
                    OR
                    (sf.Path != sf.SupposedPath))
    }
}
```

Listing 3 gives an example of an *action* with the *pipe-and-filter* modifier, forcing the underlying steps to be evaluated sequentially, one after another, excluding any concurrent execution opportunities. Moreover, according to the rules of this architectural pattern, the steps (separated by the conjunctions) are going to be limited to use only the output result of the immediately preceding step, instead of any results of any previous steps. This example also illustrates that the definition of an action can be provided as a mixture of logical descriptions of functionalities and C# code fragments.

Listing 4. *Composite* design pattern application for representing the setup of a C# solution.

```
entity Solution { Path: String; }

entity Project { Path: String; }

entity SourceFile
{
    Path: String;

    lazy Namespace: String
    {
        <code lang="csharp">...</code>
    }
}

composite
{
    parts: [Solution, Project, SourceFile],
    type: 1-N
}
```

Listing 4 showcases the use of the *composite* design pattern. The example follows the abstract description of a C# solution, which contains projects, with their contained source files. This is the classical setup for a composite design pattern application with a one-to-many type, causing the transpiler to generate the required navigational properties to the entities. The example also features the use of the *lazy* keyword, which delays the evaluation of the field, while it can also be seen that the functionality of a field might be specified through the use of C# plugin codes as well.

Listing 5. An example for the *factory* design pattern application.

```
factory component for Project
{
    action Load
    {
        input: { scope: String; path: String; }
        logic: { ... }
    }
}
```

Listing 5 features an example for the use of the *factory* design pattern. The *factory* keyword introducing this declaration will ensure that within the context of

the whole application, the *Project* type can only be instantiated through actions of this component.

4. Case study

To demonstrate the capabilities and expressive power of LADY, we summarize the lessons learned during the specification and recreation of an already existing software tool, a desktop application which is in use in an industrial environment. Regarding its functionalities, it aims to modify the structure of C# projects and solutions, adjusting the namespaces of C# source files to the name of their containing folders – and vice versa.

Although, other software tools and plugins performing the same operations had already existed (except for the manual restructuring of projects based on custom rulesets) [5, 12, 14], the creation of this tool was justified, as the sheer size of the input C# solutions requires special processing algorithms not present in other, commonly used tools and plugins. However, the refactoring of the tool became necessary finally, since its original implementation was not capable of handling sufficiently large input data – the tool was violating critical non-functional requirements related to input size, runtime, and memory-handling.

4.1. Research questions

As part of the preparation for the case study, the following research questions have been formulated:

RQ1 Is LADY expressive enough to describe the architecture of an industrial software tool?

RQ2 Can quality improvements be seen after the respecification process?

RQ3 What is the state of non-functional requirements after the respecification process?

4.2. Respecification process

Using LADY, we specified, and – relying on the transpiler – refactored the selected industrial application. The new specification includes a more precise, formal logic based description of the tool's functionalities, as well as the enforcement of the desired design and architectural patterns. After the process, the new implementation of the tool, which was generated from the improved specification was able to fulfill all functional and non-functional requirements.

During the respecification process, most of the previous specification was omitted and rewritten in the form of formal logic rules. The original specification was reverse engineered from the original code base, and requirements were collected from preceding design documents.

Although, special C# codes were necessarily added in specific cases, like OS-level file and directory handling, and unique file modification operations. However, this only happened in a negligible number of cases. Naturally, these plugin C# codes were explicitly made part of the output solution, while the remaining specification was transpiled.

4.3. Results

The output of our case study, the refactored code base was validated through three steps.

At first, system tests were run to ensure the preservation of functionalities, as naturally required for any sound refactoring process. In this case, the use of unit and component tests were not an option, as during the specification and code generation processes, the signature of the methods are likely to change, which invalidates any previous unit tests.

Testing was followed by a code review performed by software architects, the owners of the original code base. The new code base was especially reviewed for clean code measurements, and it was inspected whether the implementation conforms to the applied design and architectural patterns.

Finally, the Chidamber and Kemerer metrics suite was selected as a ground to compare the original and refactored code bases on a metric-level.

Table 1. Chidamber and Kemerer metrics suite results for the original and the refactored code bases.

Metric	Original code base	Refactored code base
WMC	5.875	4.074
DIT	1	1.142
NOC	0	0
CBO	1.75	0.481
RFC	9.75	7.962
LCOM_1	-	0.62

Table 1 shows the results of the selected metrics. Regarding this kind of validation, a clear improvement can be seen in half of the metrics. Namely, the WMC (Weighted Methods per Class), CBO (Coupling between Object Classes), and RFC (Response for a Class) metrics have shown a decrease in their values, which indicates a positive change.

Regarding the remaining metrics, the following observations can be made. The NOC (Number of Children) metric shows the number of immediate subclasses of a class. Since neither the original, nor the refactored code base had any subclass behavior besides the built-in inheritance from the *Object* class, this metric is not relevant in our case.

As far as the DIT (Depth of Inheritance Tree) metric is concerned, an increase can be seen in the measured value. This is due to the fact that each of the generated service classes received an extra implemented interface during our transpilation process, but these corresponding interfaces are not present in the original code base. The DIT value for the original code base is 1, indicating that all the classes have the *Object* class as their base class, while the DIT value for the new code base is slightly increased, as some of the classes now possess interface base-types as well. Since from an object-oriented perspective this can be regarded as a sign of increased code quality, the slight increase in DIT is not just acceptable, but even desirable.

A similar reasoning applies for the last metric. It can be seen that the LCOM (Lack of Cohesion of Methods) metric could not be measured in the original code base, due to its overuse of static classes and methods. LCOM became meaningful in the new code base, and again, the measured value is justified and accepted, since from OOP point of view the transition from static methods to instance methods can be observed as quality improvement.

Finally, it is worth considering threshold values for the metrics published in previous works, such as the ones available for CBO, RFC, and WMC [19]. We can conclude that our related values for all of those metrics are well in the desired ranges.

4.4. Non-functional requirements

Furthermore, the case of non-functional requirements must also be examined. Before the respecification process, the runtime of the original tool could be measured in hours, while after changing the underlying architectural setup, this runtime on the same inputs could be measured in just a couple of minutes, which indicates a large performance optimization. Similarly, the originally present memory-handling problems disappeared after the process.

The former can be attributed to the analysis related to immutable types and the concurrent code generation, while the latter to the specific control flow inherited from the use of the *pipe-and-filter* architecture, which results in the dropping of unnecessary objects of previous steps from memory.

5. Conclusion and discussion

The results presented in this paper contribute to the analysis and refactoring of large-scale industrial code bases. As our initial examinations revealed, many problems in code bases are related to some structural issues which might be solved by introducing some rigorous implementations of architectural and design patterns. In the existing literature, we have seen that many works are focusing on the violation detection of pattern implementations. Some other relevant works propose annotations to eliminate the repetitive, boilerplate code of the patterns. We have concluded that the root cause of our problems can be traced back to the fact

that programming languages often do not provide sufficient support for the native description of patterns at the location of their applications in source codes.

Therefore, we have defined a domain-specific language, characterised by the fact that it allows architectural and design pattern applications on the level of native language elements. Also, through a well-controlled code generation process, we are able to ensure that the code output by the transpiler of the language contains a correct and efficient implementation of these patterns.

To validate our methods, we have performed a case study, during which we have specified and refactored an already existing industrial software tool. As a result, code quality issues of the original implementation were eliminated, and the problem of not meeting non-functional requirements was also solved. Comparing the original and refactored code bases through the Chidamber and Kemerer metrics suite, our method proved to produce a clear improvement in half of the metrics. Furthermore, even when improvements were not shown directly by the metric values, enhancements on the code quality were justified. This shows that our language is suitable for describing software architecture, and can also serve as a fundamental tool for a large-scale automated refactoring process.

It is notable, however, that the mentioned metrics suite is mainly for capturing object-oriented and structural properties, while architecture-based improvements were mainly assessed through expert reviews by code architects. Moreover, the improvement of the architectural properties can be indirectly discovered by the satisfaction of non-functional requirements, which were previously neglected.

To address the research questions, LADY has proven to be expressive enough to describe the design structures of the respecified industrial software tool (*RQ1*). As explained above, clear code quality improvements could be seen after the respecification process: improvements on structure (*RQ2*) and on the side of non-functional requirements also (*RQ3*).

Regarding future research possibilities, we plan to extend the scope of our language from specifying software tools to supporting the architectural description of general software applications. Ideally, by being able to handle general software applications, future research must focus on strengthening the results of this paper with a broader set of open-source applications. Besides, the formal specification of the language is planned to be published in a standalone paper as well.

Another research direction can improve the scalability of the approach by the addition of "outer components": this would allow code bases to be refactored by smaller units (by components or a set of components), while keeping existing component connections intact with outer components not being refactored. Also, the mentioned immutability checks for user-defined types, and the inference checks for sequential expressions will be discussed in another paper.

Additionally, as we agree that the plugin C# codes may weaken currently existing architectural guarantees, we are planning on extending the analysis capabilities of the transpiler architecture to pre-analyse these codes during the compilation process.

Acknowledgement

SUPPORTED BY THE EKÖP-KDP-24 UNIVERSITY EXCELLENCE SCHOLARSHIP PROGRAM COOPERATIVE DOCTORAL PROGRAM OF THE MINISTRY FOR CULTURE AND INNOVATION FROM THE SOURCE OF THE NATIONAL RESEARCH, DEVELOPMENT AND INNOVATION FUND.

T. K. was supported by project no. TKP2021-NVA-29 under the Ministry of Innovation and Technology of Hungary from the National Research, Development, and Innovation Fund and financed under the TKP2021-NVA funding scheme.

References

[1] A. A. B. Baqais, M. Alshayeb: *Automatic software refactoring: a systematic literature review*, Software Quality Journal 28.2 (2020), pp. 459–502, DOI: 10.1007/s11219-019-09477-y.

[2] K. H. Bennett, V. T. Rajlich: *Software maintenance and evolution: a roadmap*, in: Proceedings of the Conference on The Future of Software Engineering, ICSE '00, Limerick, Ireland: Association for Computing Machinery, 2000, pp. 73–87, ISBN: 1581132530, DOI: 10.1145/336512.336534.

[3] S. Chidamber, C. Kemerer: *A metrics suite for object oriented design*, IEEE Transactions on Software Engineering 20.6 (June 1994), pp. 476–493, ISSN: 1939-3520, DOI: 10.1109/32.295895.

[4] G. Dévai, T. Gregorics, M. Tóth, D. Asztalos, D. J. Németh, G. F. Kovács, B. Németh, Z. Gera, A. Dobreff, B. Gregorics, A. Nagy, M. Budai, Z. Kulik, K. Kanyó: *txtUML*, in: Proceedings of the MoDELS 2016 Demo and Poster Sessions (MoDELS 2016), Saint-Malo, France, 2016, pp. 8–15.

[5] DevExpress: *Rename Namespace to Match Folder Structure*, https://docs.devexpress.com/CodeRushForRoslyn/117359/refactoring-assistance/rename-namespace-to-match-folder-structure, 2021.

[6] K. Elish, M. Alshayeb: *Using Software Quality Attributes to Classify Refactoring to Patterns*, Journal of Software 7 (Feb. 2012), DOI: 10.4304/jsw.7.2.408-419.

[7] E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Design patterns: elements of reusable object-oriented software*, USA: Addison-Wesley Longman Publishing Co., Inc., 1995, ISBN: 0201633612.

[8] J. Hannemann, G. Kiczales: *Design pattern implementation in Java and aspectJ*, ACM SIGPLAN Notices 37.11 (Nov. 2002), pp. 161–173, ISSN: 0362-1340, DOI: 10.1145/583854.582436, URL: https://doi.org/10.1145/583854.582436.

[9] J. Hunt: *Gang of Four Design Patterns*, in: Scala Design Patterns: Patterns for Practical Reuse and Design, Cham: Springer International Publishing, 2013, pp. 135–, ISBN: 978-3-319-02192-8, DOI: 10.1007/978-3-319-02192-8_16, URL: https://doi.org/10.1007/978-3-319-02192-8_16.

[10] C. B. Jaktman, J. Leaney, M. Liu: *Structural Analysis of the Software Architecture — A Maintenance Assessment Case Study*, in: Software Architecture: TC2 First Working IFIP Conference on Software Architecture (WICSA1) 22–24 February 1999, San Antonio, Texas, USA, ed. by P. Donohoe, Boston, MA: Springer US, 1999, pp. 455–470, ISBN: 978-0-387-35563-4, DOI: 10.1007/978-0-387-35563-4_26, URL: https://doi.org/10.1007/978-0-387-35563-4_26.

[11] S.-U. Jeon, J.-S. Lee, D.-H. Bae: *An automated refactoring approach to design pattern-based program transformations in Java programs*, in: Ninth Asia-Pacific Software Engineering Conference, 2002. Dec. 2002, pp. 337–345, doi: 10.1109/APSEC.2002.1183003.

[12] JetBrains: *Adjust Namespaces*, https://www.jetbrains.com/help/resharper/Refactorings__Adjust_Namespaces.html, 2025.

[13] R. Mahouachi, M. Kessentini, K. Ghedira: *A New Design Defects Classification: Marrying Detection and Correction*, in: Fundamental Approaches to Software Engineering, ed. by J. de Lara, A. Zisman, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 455–470, isbn: 978-3-642-28872-2.

[14] Microsoft: *Sync Namespace and Folder Name*, https://learn.microsoft.com/en-us/visualstudio/ide/reference/sync-namespace-and-folder-name, 2024.

[15] B. D. Orosz, J. Szücs, M. Cserép: *Architecture-Aware Static Analysis and Violation Detection of C# Student Submissions*, Computers 15.7 (2026), issn: 2073-431X, doi: 10.3390/computers15070404, url: https://www.mdpi.com/2073-431X/15/7/404.

[16] M. Paixão, A. Uchôa, A. C. Bibiano, D. Oliveira, A. Garcia, J. Krinke, E. Arvonio: *Behind the Intents: An In-depth Empirical Study on Software Refactoring in Modern Code Review*, in: 2020 IEEE/ACM 17th International Conference on Mining Software Repositories (MSR), May 2020, pp. 125–136, doi: 10.1145/3379597.3387475.

[17] M. Radford: *SINGLETON – the Anti-Pattern!*, ACCU Overload 57 (2003), url: https://accu.org/journals/overload/11/57/radford_337.

[18] H. Y. Shahir, E. Kouroshfar, R. Ramsin: *Using Design Patterns for Refactoring Real-World Models*, in: 2009 35th Euromicro Conference on Software Engineering and Advanced Applications, Aug. 2009, pp. 436–441, doi: 10.1109/SEAA.2009.56.

[19] R. Shatnawi: *A quantitative investigation of the acceptable risk levels of object-oriented metrics in open-source systems*, IEEE Trans. Softw. Eng. 36.2 (Mar. 2010), pp. 216–225.

[20] I. Sommerville: *Software Engineering*, 10th, Pearson, 2015, isbn: 0133943038.

[21] L. Tokuda, D. Batory: *Evolving Object-Oriented Designs with Refactorings*, Automated Software Engineering 8.1 (2001), pp. 89–120, doi: 10.1023/A:1008715808855.

[22] F. Wedyan, S. Abufakher: *Impact of design patterns on software quality: a systematic literature review*, IET Software 14.1 (2020), pp. 1–17, doi: https://doi.org/10.1049/iet-sen.2018.5446.

[23] Zeinab, Kermansaravi, M. S. Rahman, F. Khomh, F. Jaafar, Y.-G. Gueheneuc: *Investigating Design Anti-pattern and Design Pattern Mutations and Their Change- and Fault-proneness*, 2021, arXiv: 2104.00058 [cs.SE], url: https://arxiv.org/abs/2104.00058.

Annales Mathematicae et Informaticae
63 (2026) pp. 122–131
DOI: 10.33039/ami.2026.06.007
URL: https://ami.uni-eszterhazy.hu

Extensions of the C++ Parallel STL library

Zoltán Porkoláb, Gábor Tóthvári

ELTE Eötvös Loránd University
Faculty of Informatics
Department of Programming Languages and Compilers
Budapest, Hungary
gsd@inf.elte.hu
eyjqmy@inf.elte.hu

Abstract. C++17 introduced Parallel STL to simplify writing parallel code for developers already comfortable with the long-term used Standard Template Library. However, while Parallel STL can be powerful, it also presents risks: as many experts have noted, issues like non-commutative or non-associative algorithms can lead to unexpected results, and the standard offers plenty of other ways to "shoot yourself in the foot". In this paper we discuss some of these traps, including the set of problems, where the answer must preserve the original order of the elements.We propose a generic *filter-reduce* algorithm for solving such parallel tasks. The algorithm is implemented on top of the Parallel STL itself, therefore it utilizes all of its benefits. Tests prove that filter-reduce is scalable and performs well on the specific set of problems.

Keywords: C++, ParSTL, parallel algorithms

2020 *Mathematics Subject Classification:* Primary 68W01, 68W10

1. Introduction

Since the early years of the new millennia, the importance of parallel programming has been well-known for programmers. Earlier hardware development provided faster and faster chips and programmers were able to utilize this speed-up with single threaded programs. However, by the early years of the new millennia hardware development reached strong barriers and the generic processor speed-up

Submitted: March 31, 2026
Accepted: June 30, 2026
Published online: July 17, 2026

has stopped. As the Moore's Law does not apply anymore for the speed of the processors, software performance could be increased mostly via enabling parallel execution [11].

Unfortunately, writing efficient, scalable, and correct parallel programs is inherently challenging. Most programming languages provide low-level multi-threading elements: thread-management classes, mutex objects, locks, conditional variables, etc. and it is the programmers' task to build efficient higher level algorithms from these components. Meanwhile, these complex programs should avoid various traps and pitfalls like possible race conditions, deadlocks, or resource starvation.

At the same time writing multi-threaded programs usually does not bring efficiency and scalability automatically. Often the programmer not only has to understand effective parallel constructs but also should be aware of platform-specific features, sometimes even down to the hardware level. What is the cost of starting a new thread on the actual platform? How much resources (processors, memory) are available without overcommitting? How expensive is the context switch between threads? These questions become even more critical when implementing a portable software running on various platforms.

The C++ programming language is still very popular amongst developers who are responsible to implement high performance systems, many times utilize modern multicore hardware writing highly parallel applications. Since the C++11 standard, the C++ programming language provides the necessary building blocks for writing portable parallel programs: the thread class, various mutexes, lock guards, conditional variables and even atomic operations. However, implementing *effective* platform independent multithreaded programs are far from trivial in C++11.

Since the C++17 version of the Standard [3, 14], the standard library provides the *Parallel STL*, a parallel version of the Standard Template Library. The Standard Template Library (STL) is part of the C++ language since the first standard, known and widely used by the developer community [6]. In C++17 most of the STL algorithms received new overloads that take a first *execution policy* parameter, which specifies the algorithm's – possibly parallel – execution [5]. The C++ community highly appreciated the Parallel STL library as it promised a safe and effective way to implement business logic on a (relatively) higher abstraction level; while it still promised efficient platform-specific concurrent implementation. As the original STL library is well known for everybody coding in C++, Parallel STL provides an easy entry level to the parallel world for the average programmer.

Unfortunately, this promise can be misleading in some cases. Some algorithms require special conditions that are less familiar to programmers. Others work deceptively and the results differ from a sequentially executed version of the same algorithms. In this article, we will discuss the latter cases of tasks, specifically where we want to obtain the result in sequential order after a filtering that can be performed in parallel. We implemented a new *filter-reduce* algorithm, where the filter step can be effectively executed in a parallel way, but the reduction step requires keeping the order of the resulting elements. We implemented a working prototype of the algorithm in two variants built entirely on top of the Parallel STL,

therefore preserving its portability and efficiency. The two variants using either a mutex or a lock-free solution for the only critical part of the algorithm. We tested the prototype on several test cases to prove its scalability.

This paper is organized as follows. In Section 2 we overview the basics of the Parallel STL library. We discuss the known issues regarding the library in Section 3. Our filter-reduce algorithm and their implementation details are described in Section 4. Then in Section 5 we evaluate our measurement results. The paper concludes in Section 6.

2. The Parallel STL library

The *Standard Template Library (STL)* is a central element of the C++ programming language since the original C++98 standard [3, 14]. The STL provides various data structures – sequential, associative and (since C++11) unsorted containers and fundamental algorithms for searching, copying, and modifying data. The connection between the containers and the algorithms is provided by *iterators* – an abstraction of the pointer type [6]. The essence of the STL is to generalise as much as possible without a performance penalty [9] using parametric polymorphism – *templates* in C++ [12]. The algorithms often accept *functor* parameters as customisation points: specific classes that provide public `operator()` methods and thus behave as if they were functions. Such functors can have a state and are the higher level generalisation of the basic operations and functions. Since C++11, one can also use *lambda functions* instead of named functor classes.

Writing parallel algorithms for the STL is not trivial. One should decide how to split the containers into partitions for parallel execution, how many threads can be run efficiently on a specific target platform, and whether the cost overhead of parallelisation is less than the single-threaded execution. Such decisions are well-observable in Listing 1, an example from Bjarne Stroustrup [10] for summing the elements of a `std::vector`. Most of these questions depend on the size of the container, the task we execute, and on specifics of the target platform.

Parallel STL became part of the C++ Standard in 2017 to simplify the execution of parallel algorithms. The new library introduced overloaded versions for most of the original STL algorithms by extending them with a new first parameter, the *execution policy* which specifies how the algorithm is supposed to be executed. Possible values of the execution policy specify sequential or parallel execution with further specifying possible SIMD execution. The parameter's value describes the *requested* execution policy, but this is just a suggestion: the implementation may decide a different strategy, e.g., when the cost of starting new threads would be greater than simple sequential execution. The Standard on execution policy is *open*: additional execution policies may be provided by a standard library implementation, e.g., for the benefit of a specific platform.

The main advantage of the Parallel STL is to reduce the entry level to start using parallel algorithm. Programmers can utilize their knowledge of the classical STL while they do not need to know platform specific information. As an example,

observe Listing 2 as the Parallel STL based solution of summarization of the vector elements we have seen on Listing 1 implemented without the Parallel STL.

Listing 1. Manual parallelisation of STL based on Bjarne Stroustrup.

```
// Simple accumulator function object.
template <class T, class V>
struct Acc {
  T* b;
  T* e;
  V val;
  Acc(T* bb, T* ee, const V& vv) : b{bb}, e{ee}, val{vv} { }
  V operator()() { return std::accumulate(b, e, val); }
};
double sum(const std::vector<double>& v) {
  if (v.size() < 10000) // execute non-parallel for small vectors
    return std::accumulate(v.begin(), v.end(), 0.0);

  // spawn to 4 tasks if v is large enough.
  auto f0{async(Acc{&v[0], &v[v.size() / 4], 0.0})};
  auto f1{async(Acc{&v[v.size() / 4], &v[v.size() / 2], 0.0})};
  auto f2{async(Acc{&v[v.size() / 2], &v[v.size() * 3/4], 0.0})};
  auto f3{async(Acc{&v[v.size() * 3/4], &v[v.size()], 0.0})};

  return f0.get() + f1.get() + f2.get() + f3.get();
}
```

Listing 2. Summarizing the elements of a vector using Parallel STL.

```
// Using the Parallel STL
double sum(const vector<double>& v) {
  double res = std::reduce(std::execution::par, v.begin(), v.end(), 0.0);
  return res;
}
```

Parallel STL algorithms are usually implemented on the top of a lower level library.In most UNIX systems implementation uses Intel's *Threading Building Blocks* (TBB) library [2, 8]. In the core of the implementation we usually find a thread pool which works on the container divided into small slices. Contrary to the example on Listing 1 there are much more slices than working threads, so each thread in the pool will work on many slices. This helps to balance the workload between threads, as fast-finishing threads may process more slices. In the same time there is no guarantee that the earlier completed slice will contain the earlier elements in the container.

3. Problem statement

Since the introduction of the Parallel STL, practice identified several issues, mostly related to non-commutative and non-associative algorithms, that can cause serious

errors for inattentive or inexperienced developers [1, 4, 7]. While these problems are widely discussed in the C++ community, the naïve use of Parallel STL may lead to surprising results in other, less known areas as well. In this work we will discuss some of these problems, namely algorithms, which requires returning the results in some well-defined order.

Many old-style STL algorithms, like `find` and `find_if` returns the first element that matches the search criteria inside a container. However, when we apply the corresponding Parallel STL algorithm to the very same data, the outcome may differ, returning a different result or not to find any result at all. The root cause of this phenomenon is the naïve implementation of the parallel version of the algorithm, which usually just applies the same criterion on multiple slices of the container and then tries to fold the results. However, the parallel execution on the slices most cases does not guarantee a stable behavior; the results on the slices return in undefined order. Moreover, if the criteria of the search is non-local, e.g., we are looking the Nth element fulfilling the predicate, that "count" will be local for the slice a thread in working on.

Listing 3. Naïve use of parallel find_if to find the 3rd element greater than 50.

```
#include <vector>
#include <iostream>
#include <algorithm>
#include <execution>
int main() {
 std::vector<int> v(100'000, 2); // fill a large vector with small elements
 // set a few random elements greater than 50
 v[5001] = 123;
 v[11001] = 999;
 v[22222] = 1233; // <--- 3rd
 v[33333] = 546464;
 v[55555] = 555;
 v[77777] = 7657;
 v[77778] = 7658;
 v[77779] = 7659;
 v[88888] = 312231;

 const auto res = std::find_if( std::execution::par, v.begin(), v.end(),
           [counter = 0](int i) mutable { if ( i>50 ) return ++counter == 3;
                                          else return false; });
 if ( res != v.end() )  // we found it
   std::cout << *res << '\n';
 return 0;
}

```

Consider the example on Listing 3. We initialized a large vector with small elements (almost all are smaller than 50), except a few random elements which value is greater than 50. We are looking for the 3rd element which has a value greater than 50. We use the well-known `find_if` algorithm from STL, which takes the first parameter as the execution policy, then two parameters as the specification

of the search interval, and the last parameter as a predicate functor to specify the element we are searching for. If we run this algorithm sequentially we correctly find the 3rd element greater than 50 at index 22222 (line 10). However, when we run the algorithm with parameter `std::execution::par`, it can happen that we find the result e.g., at index 77779 (at line 15) or we do not find the third element greater than 50 at all.

The reason is that when running in parallel mode, the algorithm applies the functor *for all slices independently*. Even if there are many elements greater than 50, it can happen that neither of the slices contain 3 elements fulfill the search criteria. Moreover, if we are "lucky" to find 3 elements greater than 50 in a certain slice we do not know whether we have similar elements at any of the previous indexes.

One can suggest to use a shared state to count the number of elements found. A possible implementation could be a shared pointer from the functors pointing to a single atomic integer variable. Apart from the fact that it would completely destroy the efficiency of parallel execution, another problem is that we do not know the order in which each slice is processed. It may happen that a slice containing higher indexes is finished to process before the slice(s) of lower indexes. Therefore our shared counter would be totally unusable.

4. A filter-reduce algorithm

We designed a generic filter-reduce algorithm to solve the problem described in the previous section. The algorithm executes a parallel *filter* step to search elements with a given criteria in a container then executes a sequential *reduce* step to identify the element(s) we are looking for. The filter criteria is local, i.e., it does not depend on other elements in the container, while the reduce step keeps the order of the elements. The algorithm is specifically effective in cases where the result set of the filter step is significantly smaller than the original data set, therefore the sequential reduce step is executed on relatively few elements.

A possible usage scenario can be seen of Listing 4. Similar to the basic idea of the `transform_reduce` algorithm [1, 4], we also split the search to be performed into a locally and parallel executed filter step and a sequential reduce step performed on the filter result.

We implemented the algorithm as a working prototype using standard C++17 features and on the bases of the Parallel STL itself. That way we can utilize the effective portable implementation of the Parallel STL while providing a flexible way to parameterize the algorithm using C++ functors.

The filter step is executed by a Parallel STL `for_each` algorithm, which detects when a new slice starts to be processed and creates a new vector of iterators for each slices to collect the filter results. As these vectors are created by slice, no data race or even false sharing occurs during the filter step between the execution threads. However, as we do not know in which order the slices are processed we need to collect these result vectors into one single container – called *Collector* – preserving

the relative ordering of the results. That is the only step when we have to be concerned with possible data race. We came up with two possible implementations: one uses ordinary mutex-based locking and the other is implemented with a lock-free data structure. The mutexed version uses `std::mutex` for synchronization, while the lock-free one is based on an implementation by Williams [13]. It is a linked list stack, using the `compare_exchange_weak` operation from the C++ standard library. The difference when compared with mutexes is when a thread has to wait before entering the critical section it is not put to sleep, but rather it can immediately retry the operation.

After the parallel filter step has finished the reduce step is running in a sequential way. This reduce step walks thru the filter results stored in the Collector and returns whatever return type is specified in the reduce parameter.

Listing 4. Usage example for the filter-reduce algorithm.

```
#include <vector>
#include <iostream>
#include <algorithm>
#include <execution>
int main() {
 std::vector<int> v(100'000, 2); // fill a large vector with small elements
 // set a few random elements greater than 50
 v[5001] = 123;
 v[11001] = 999;
 v[22222] = 1233; // <--- 3rd
 v[33333] = 546464;
 v[55555] = 555;
 v[77777] = 7657;
 v[77778] = 7658;
 v[77779] = 7659;
 v[88888] = 312231;

 const auto res = filter_reduce(std::execution::par, v.begin(), v.end(),
                  [](int i) { return  i > 50; },  // filter
                  [cnt = 0] (int) mutable { return 3 == ++cnt; }); // reduce
 if ( res != v.end() )  // we found it
   std::cout << *res << '\n';
 return 0;
}
```

5. Evaluation

We have benchmarked the speed of the implementations and compiled the results into Table 1 and Table 2. As testing the correctness of the algorithm, the prototype always has produced the same results as the classical sequential STL algorithms. As expected, parallel execution has greatly reduced the running times. For small amounts of data, the lock-free implementation enjoyed a noticeable lead over the

mutexed version. As the size of the dataset increased, however, the difference was much smaller.

The benchmarks tested two kinds of workloads, depending on the cost of the filter (search condition) function. There was an "easy" workload, with the filter condition being the greater than operator (see Table 1). The operator has a minimal cost of execution, therefore the parallel filter step required less effort. Furthermore, a hard workload was also tested where the condition was a prime number testing function (see Table 2). Here, the parallel filter step required much more effort.

When we present the results on a logarithmic scale on Figure 1, it is observable that the algorithm scales well when the input size increases.

Table 1. Timing results of easy workload benchmark in milliseconds.

Amount of numbers	10^5	10^6	10^7	10^8
Maximum number greater than `RAND_MAX`				
Singlethreaded	1.11	7.23	78.65	736.06
Mutexed	1.69	2.69	16.63	146.69
Lockfree	0.38	2.13	15.41	143.95
Nth number greater than `RAND_MAX`				
Singlethreaded	0.11	0.80	6.75	75.35
Mutexed	1.02	2.34	2.04	10.85
Lockfree	0.23	0.33	0.94	8.17

Table 2. Timing results of hard workload benchmark in milliseconds.

Amount of numbers	10^5	10^6	10^7	10^8
Maximum prime number search				
Singlethreaded	128.43	1246.65	12 360.50	124706.00
Mutexed	24.94	225.53	2237.01	22391.90
Lockfree	23.70	225.46	2231.61	22375.30
Nth prime number search				
Singlethreaded	251.10	2474.89	24 640.80	255497.00
Mutexed	45.95	446.85	4441.81	50154.80
Lockfree	44.91	446.55	4445.92	61809.60

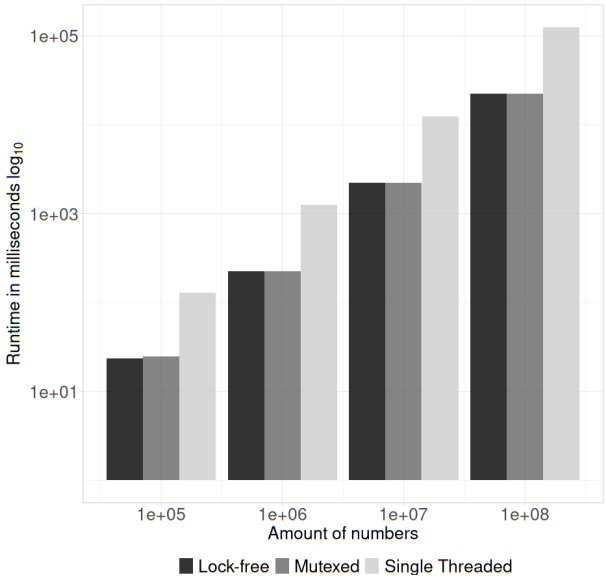


Figure 1. Benchmark results of the maximum prime number search.

6. Conclusion

Writing effective and portable parallel code is a challenging task which requires deep understanding of the language elements and the platforms. Since C++17, the C++ programmers can use the Parallel STL, an easy-to-learn extension of the customary Standard Template Library. However, naïve use of some Parallel STL algorithms may produce puzzling results. There is also lack of algorithms which obtain the result in sequential order after a filtering executed in parallel way. To fill this gap, we introduced a filter_reduce algorithm, which applies a filter criteria executed in parallel way on the elements, then a reduce step provides a well-order result sequentially.

We implemented a working prototype with two locking mechanism: one using a classical mutex and the other implemented as a lock-free algorithm. We tested the prototype on various configurations and proved that the algorithm is useful, especially when the filtering criteria is true on a significantly smaller number of elements.

References

[1] B. Barth, R. Szalay, Z. Porkoláb: *Towards Safer Parallel STL Usage*, in: 2022 IEEE 16th International Scientific Conference on Informatics (Informatics), 2022, pp. 39–44, doi: 10.1109/Informatics57926.2022.10083416.

[2] Intel Corporation: *Intel Threading Building Blocks*, visited 2022-05-07, URL: http://intel.com/content/www/us/en/developer/tools/oneapi/onetbb.html.

[3] *ISO/IEC 14882:2020 – Programming languages C++*, visited 2022-05-20, URL: https://www.iso.org/standard/79358.html (visited on 05/20/2022).

[4] N. M. Josuttis: *C++17: The Biggest Traps – [C++ on Sea 2019]*, visited 2022-05-07, URL: http://youtube.com/watch?v=mAZyaAo3M70&t=3975.

[5] D. Kühl: *CppCon 2017: C++17 Parallel Algorithms*, visited 2022-10-14, URL: http://youtube.com/watch?v=Ve8cHE9LNfk&t=1784.

[6] D. R. Musser, A. A. Stepanov: *Algorithm-oriented generic libraries*, Software: Practice and Experience 24.7 (1994), pp. 623–642, URL: http://stepanovpapers.com/musser94algorithmoriented.pdf.

[7] N. Pataki: *Safe iterator framework for the C++ Standard Template Library*, Acta Electrotechnica et Informatica 12.1 (2012), p. 17, URL: http://aei.tuke.sk/papers/2012/1/03_Pataki.pdf.

[8] J. Reinders: *Intel Threading Building Blocks - outfitting C++ for multi-core processor parallelism*, Jan. 2007, ISBN: 978-0-596-51480-8, URL: http://oreilly.com/library/view/intel-threading-building/9780596514808.

[9] A. Stepanov, D. E. Rose: *From Mathematics to Generic Programming*, 1st ed., New York: Springer, 2009, ISBN: 978-0387952902.

[10] B. Stroustrup: *C++11 - the new ISO C++ standard*, visited 2022-05-07, URL: http://stroustrup.com/C++11FAQ.html (visited on 05/07/2022).

[11] H. Sutter et al.: *The free lunch is over: A fundamental turn toward concurrency in software*, Dr. Dobb's journal 30.3 (2005), visited 2022-10-14, pp. 202–210, URL: http://gotw.ca/publications/concurrency-ddj.htm.

[12] D. van de Voorde, N. M. Josuttis, D. Gregor: *C++ Templates: The Complete Guide (2nd Edition)*, 2nd, Addison-Wesley Professional, Sept. 2017, ISBN: 978-0-321-71412-1, URL: http://tmplbook.com.

[13] A. Williams: *C++ Concurrency in Action*, 1st ed., Manning Publications, Feb. 2012, ISBN: 9781933988771.

[14] *Working Draft – Standard for Programming language C++*, visited 2022-05-20, URL: http://eel.is/c++draft (visited on 05/20/2022).

Annales Mathematicae et Informaticae
63 (2026) pp. 132–148
DOI: 10.33039/ami.2026.06.004
URL: https://ami.uni-eszterhazy.hu

Data cleaning and spam filtering methods in the TAWOS database

Márk Szabó[a], **Ádám Kovács**[ab], **Gábor Kusper**[a]

[a]Eszterházy Károly Catholic University
szabo.mark@uni-eszterhazy.hu, kovacs2.adam@uni-eszterhazy.hu,
kusper.gabor@uni-eszterhazy.hu

[b]University of Debrecen, Doctoral School of Informatics

Abstract. The TAWOS dataset is a large relational database of issue-tracking data mined from open-source projects. While it is valuable for empirical software engineering, the issue texts also contain spam, placeholders, and off-topic noise that can distort downstream analytics and fine-tuning tasks. We present a four-stage filtering pipeline for cleaning TAWOS and focus on a validity scoring step that assigns each issue a 0–100 score from its title and description.

We compare a deterministic rule-based classifier, OwnMetrics, against four local small language models (Llama-3.1-8B, Mistral-7B, Phi-3.5-mini, and Gemma-3-4B) prompted for JSON scores. Evaluation first uses a near-balanced labeled benchmark of 947 GitHub issues (481 spam/noise and 466 legitimate issues), collected from moderator-locked spam issues and legitimate issues from popular repositories.

On this GitHub-based threshold-selection benchmark, OwnMetrics obtains the highest accuracy, 91.0%, at threshold 75, while the best LLM configurations reach 89.2% (Gemma-3-4B) and 89.1% (Mistral-7B). A separate manually labeled in-domain validation on 300 Jira/TAWOS issues (39 spam/noise and 261 non-spam) provides an in-domain sanity check consistent with the benchmark results, with OwnMetrics giving the strongest spam-class F1 among the selected configurations.

On a 10,000-item sample, OwnMetrics yields zero parsing failures and an average execution time of 2.28 ms per item, whereas the LLMs require seconds per item and Meta-Llama-3.1-8B exhibits failure rates above 54%. Applying the full filtering pipeline to the processed TAWOS issue corpus flags 16.2% of records for filtering. The results show that a lightweight domain-tailored scorer can be both accurate and robust for large-scale issue cleaning.

Submitted: March 31, 2026
Accepted: June 30, 2026
Published online: July 17, 2026

Keywords: software repositories, issue tracking, data cleaning, spam filtering, large language models

2020 *Mathematics Subject Classification:* Primary 68N30; Secondary 68P15, 68T50

1. Introduction

Issue tracking systems are central to modern software development, because they collect bug reports, feature requests, status updates, discussions, and historical traces of the development process. Large public corpora built from issue trackers therefore provide a useful basis for empirical software engineering research. The TAWOS dataset is one such resource: it contains 508,963 issues from 44 open-source Agile projects and stores them in a relational schema designed for broad reuse [21].

However, large issue datasets are only as reliable as the textual and structural quality of the underlying reports. Prior work has shown that the quality of bug reports strongly affects downstream development activity and that issue misclassification can distort later modeling steps [12, 19, 22, 25]. More generally, issue-report analysis has long emphasized that software repositories contain heterogeneous, noisy, and inconsistently labeled artifacts [2, 11, 24]. For this reason, cleaning and validation are not only preprocessing conveniences: they are methodological requirements for trustworthy repository mining.

Our work addresses this need for the TAWOS database. The immediate motivation is practical: the cleaned data should provide a better basis for later teaching material, statistical analysis, simulation of development processes, and future fine-tuning workflows. We therefore design a filtering pipeline that scores issue validity on a 0–100 scale and removes records that fall below a selected threshold. The pipeline combines a hand-crafted rule-based method tailored to issue text with local LLM-based zero-shot scoring.

The main contributions of this paper are as follows. First, we formulate a practical issue-validity scoring task for TAWOS and construct a curated labeled benchmark of 947 issues. Second, we present OwnMetrics, a deterministic rule-based scoring method based on six interpretable components and an explicit early spam-detection stage. Third, we benchmark OwnMetrics against four local small language models across several temperature settings. Fourth, we analyze not only predictive quality but also output robustness, score distributions, cross-model agreement, spam detection rate, and execution time. Finally, we report the effect of applying the complete filtering pipeline to the processed TAWOS issue corpus.

2. Related work

2.1. Bug-report quality and issue-report structure

Bug and issue reports have long been studied as textual artifacts whose quality affects maintenance work. Zimmermann et al. analyzed what makes a bug report

useful for developers and emphasized the role of concrete observations, reproduction information, and contextual detail [25]. Soltani et al. later studied the significance of individual bug-report elements and showed that report usefulness depends not only on the presence of a report in an issue tracker, but also on the information carried by its textual fields [19]. These findings motivate validity signals such as title clarity, description detail, coherence, and technical vocabulary in our scorer.

A complementary line of work studies how issue trackers can improve report structure at submission time. Sülün et al. analyzed issue-template usage in large-scale GitHub projects and showed that structured templates and forms are widely used in popular repositories and are associated with improved issue-management outcomes [20]. Nikeghbal et al. proposed GIRT-Model for automatically generating issue-report templates, aiming to help maintainers define structured issue forms [17]. Our problem is different: we clean an already existing, historical issue corpus. Therefore, we cannot assume that the original reports followed a template; the method must evaluate noisy title–description pairs after the fact.

2.2. Issue report classification and label quality

Automated issue report classification is closely related to our work, but its usual target variable differs. Classical studies classify change requests or issue reports as bugs, enhancements, questions, documentation tasks, or other issue types [2, 11, 12]. Herzig et al. showed that bug/non-bug misclassification can distort downstream defect-prediction results [12], while Herbold et al. investigated the feasibility of predicting bug and non-bug issues automatically [11]. Recent work has extended this line with pre-trained and transformer-based models, and the systematic mapping study of Laiq and Dobslaw summarizes traditional machine learning, deep learning, transformer, and emerging LLM-based approaches for issue report classification [16].

This literature is useful for our task because it shows that issue title and description are standard inputs for text-based classification. At the same time, our output is not an issue type such as bug or enhancement. We estimate whether an issue should be retained for downstream corpus use. This makes explainability, scalability, and failure-free execution more central than in many interactive triage settings. Label quality is also important. Colavito et al. studied the impact of data quality for automatic issue classification using pre-trained language models and identified construct-validity problems in issue labeling [4]. Wu et al. similarly showed, in the context of security bug report prediction, that label correctness can substantially affect conclusions drawn from mined datasets [23]. These observations support treating our benchmark as a controlled spam/noise benchmark rather than as a complete ground truth for all levels of TAWOS issue quality.

2.3. Noise, spam, and validity threats in repository mining

Repository-mining studies have repeatedly warned that public repository data can be misleading without explicit validity checks. Kalliamvakou et al. discuss the

promises and perils of mining GitHub and emphasize that repository datasets require careful filtering before they are used for empirical claims [15]. Tu and Zhang also treat issue-tracking data quality as a measurable concern rather than as a guaranteed property of the dataset [22]. This perspective matches our motivation: cleaning is not only a preprocessing convenience, but a requirement for trustworthy repository mining.

Spam and off-topic issue reports are less studied than bug-versus-feature classification, but recent work on locked GitHub issues is relevant. Ferreira et al. analyzed GitHub issues locked as too heated and showed that moderation labels and actual discussion content do not always align perfectly [7]. Ehsani et al. released an annotated dataset of locked GitHub issue threads, demonstrating that locked issue data can support analyses of problematic repository interactions [6]. These works justify using moderation signals as useful proxies, while also showing why they should not be interpreted as perfect ground truth.

2.4. LLM-based and rule-based filtering methods

Large language models are increasingly used in software-engineering tasks. Hou et al. provide a systematic literature review of LLMs for software engineering and highlight both the promise of these models and the methodological challenges involved in evaluating them [13]. In issue-report classification, Rejithkumar et al. formulated the task as text-to-text generation [18], and Colavito et al. benchmarked LLMs for automated issue labeling, including trade-offs between performance, response time, hardware requirements, and response quality [3]. These studies motivate comparing against language-model baselines, but they also show why accuracy alone is insufficient for deployment-oriented filtering.

Rule-based methods remain useful when the target signal is dominated by stable and interpretable textual patterns. In the present task, obvious spam, placeholder reports, malformed descriptions, link flooding, missing software context, and very low-effort titles are often recognizable without open-ended reasoning. OwnMetrics therefore uses deterministic scoring rather than supervised training or unconstrained generation. Compared with supervised issue-type classifiers, it does not require a project-specific training set. Compared with LLM prompting, it has no parsing uncertainty and can be run cheaply over a large exported issue corpus.

2.5. Positioning of the present work

The present work is positioned at the intersection of bug-report quality assessment, issue report classification, repository-noise analysis, and LLM-based text filtering. Its specific goal is narrower than general bug triage: it estimates whether issue texts are suitable for downstream use in a cleaned TAWOS corpus. The main contribution is therefore not a new general-purpose classifier, but a deterministic, transparent, locally executable validity scorer evaluated against local zero-shot language-model baselines under accuracy, robustness, and runtime constraints.

3. TAWOS and the evaluation data

3.1. The TAWOS database

TAWOS (Tawosi Agile Web-based Open-Source) is a large issue-tracking dataset introduced by Tawosi et al. [21]. It was mined from 44 public Jira-based projects and stored as a relational database. The original dataset includes not only issue texts but also related entities such as comments, users, change logs, sprints, versions, components, and links between issues [21]. In the present work, however, we deliberately focus on the central `issues` table, because it contains the title, description, status, type, and time-related fields that are most directly relevant for first-pass validity scoring. Figure 1 shows the broader relational context of this table in the TAWOS database.

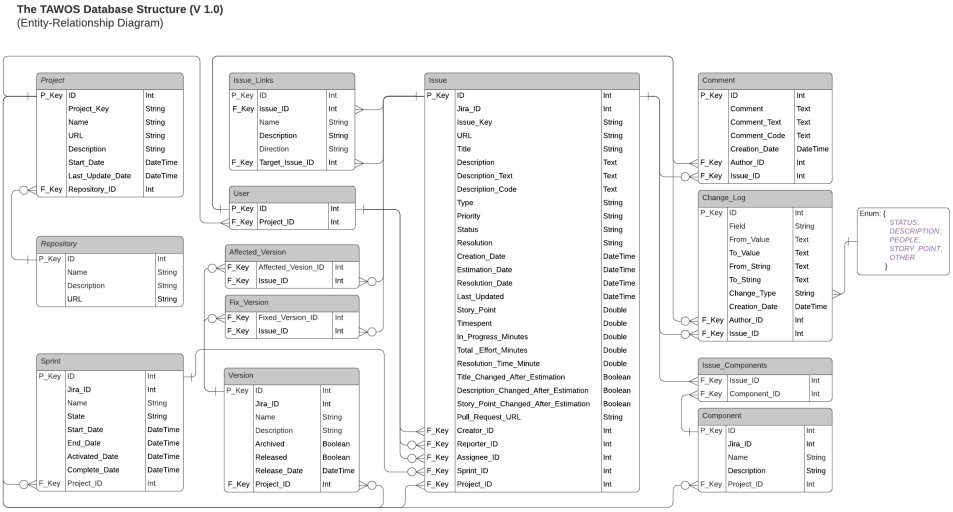


Figure 1. Structure of the TAWOS database.

This design choice keeps the present study focused and computationally lightweight. At the same time, it preserves a clear path for future work, where other data could be integrated into a richer spam/noise detector.

3.2. Cleaning pipeline

The overall workflow consists of four stages, summarized in Figure 2. First, the relevant issue fields are extracted from the original MySQL database and stored as CSV snapshots for the experiments. We used CSV because it is easy to inspect manually, convenient for debugging individual records, and simple to reuse across repeated scoring runs. Some field selection and structural filtering operations could also be expressed directly as SQL queries before export; future implementations

may push more of this structural filtering into SQL. We did not benchmark SQL-level processing against file-based processing, and no performance claim is made about this implementation choice. Second, a structural cleaning step removes irrelevant tables or columns for the present task. Third, each remaining issue receives a validity score in the range 0–100. Fourth, issues that fall below a selected threshold are filtered out together with their related data. The core of this paper is the third stage, i.e., how validity is defined and how reliably it can be estimated.

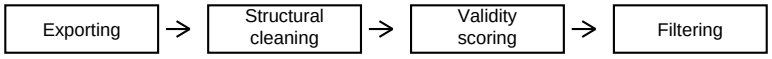


Figure 2. TAWOS issue-cleaning pipeline used in the present study.

3.3. Labeled benchmark

To evaluate the scoring methods, we assembled a curated labeled benchmark of 947 issues. The benchmark was near-balanced between 481 spam/noise and 466 non-spam examples. Spam examples were collected from GitHub issues in the 2023–2025 period by using the GH Archive event stream [9] and selecting issues from active, highly starred repositories that had been locked by moderators for spam-related reasons. Legitimate non-spam issues were retrieved with the GitHub REST API from popular and actively moderated repositories [10].

Although the target corpus is Jira-based and the benchmark is GitHub-based, both sources contain issue titles and descriptions written in the same general genre of software-development communication. This makes the benchmark suitable for a first controlled spam/noise comparison under identical labeled conditions. However, it should not be interpreted as a complete in-domain gold standard for TAWOS or as an unbiased estimate of Jira-specific generalization performance; this platform-shift threat is revisited in Section 7.

3.4. In-domain Jira/TAWOS validation set

To directly check this platform shift, we additionally created a small manually labeled in-domain validation set from Jira-based TAWOS issues. The set contains 300 issues, including 39 spam/noise cases and 261 non-spam cases. The issues were selected partly by random sampling from the processed TAWOS issue corpus and partly by manual search for suspicious records, so that the validation set would contain enough spam/noise cases for meaningful diagnostic metrics. Consequently, this set should not be used to estimate the natural prevalence of spam/noise in TAWOS. It was not used for prompt design, model selection, threshold selection, or tuning. Instead, it is used only as an in-domain sanity check for the configurations selected before this validation. Because the set is imbalanced, with 13.0% spam/noise cases, we report balanced accuracy and spam-class precision, recall, and F1 in addition to overall accuracy.

4. Validity scoring methods

4.1. Common scoring interpretation

All methods assign a validity score between 0 and 100, where lower values indicate spam, non-software-related, or very poor-quality reports, and higher values indicate clear and informative issue descriptions. The threshold-based binary decision is obtained afterward; the score itself is intended to remain useful even beyond binary filtering, for example when ranking issues for manual inspection.

4.2. OwnMetrics: a rule-based validity scorer

OwnMetrics is a deterministic rule-based classifier that uses only the issue title and description. Before scoring, the text is lightly normalized: URLs are removed, markdown links are reduced to their anchor text, markdown headers and HTML comments are stripped, and common template markers are ignored. This preprocessing is intended to reveal the actual human-authored content rather than its surrounding formatting.

The method then performs an early spam check. Extremely short placeholders, one-character content, template-only reports, and clearly promotional or abusive patterns are assigned a fixed low score immediately, without running the full scoring pipeline. In practice this catches obvious spam such as promotional messages, gambling or dating patterns, pharmacy-style advertisements, simple placeholders, and other trivial cases.

If the issue passes this early filter, OwnMetrics computes six partial scores. Table 1 summarizes their ranges and semantics.

The title score rewards detailed and structured titles, while penalizing very short, vague, or meaningless titles. The description score rewards longer and better formatted descriptions, and penalizes empty, joke-like, or template-only content. The coherence score checks whether the description actually elaborates on the same topic as the title. The style and professionalism scores down-rank visually spam-like, aggressive, or socially inappropriate reports. Finally, the technical-content score rewards domain-specific vocabulary, version strings, stack traces, code fragments, and other signals of genuine software discussion.

The raw score is the sum of the six partial scores and lies in the interval $[-130, 140]$. It is then normalized to the final 0–100 scale:

$$\text{score} = \text{clip}_{[0,100]}\left(\left\lfloor 100 \cdot \frac{r + 130}{270} \right\rfloor\right),$$

where r denotes the raw score and $\text{clip}_{[0,100]}$ truncates the result to the valid score range.

The main strength of OwnMetrics is that every component is interpretable and directly grounded in observable textual properties. This makes the method attractive when large-scale processing, debugging, or predictable behavior are more important than open-ended reasoning.

Table 1. OwnMetrics score components and their ranges.

Component	Range	Main signals
Title quality	$[-30, 25]$	Length, word count, gibberish detection, low-effort titles, structured prefixes, technical keywords
Description quality	$[-30, 35]$	Length, word count, gibberish, template-only structure, joke markers, multi-line formatting, lists
Content coherence	$[-20, 15]$	Overlap between title and description after stopword removal; penalties when the description is unrelated to the title
Style	$[-15, 20]$	Special-character ratio, repeated punctuation, repeated characters, excessive capitals, emoticons, sentence structure
Professionalism	$[-25, 20]$	Severe spam phrases, slang, insults, placeholders, begging patterns, newsletter-like non-issue text
Technical content	$[-10, 25]$	Software-engineering terms, code fragments, stack traces, file paths, version numbers, error markers

4.3. Local LLM-based scoring

Alongside OwnMetrics, we evaluated local small language models used in a zero-shot manner as issue-quality scorers. An initial screening phase discarded weaker candidates, after which the following four models were retained for detailed evaluation:

- Llama-3.1-8B-Instruct-Q4_1 [5],
- Mistral-7B-Instruct-v0.3.Q5_K_M [14],
- Phi-3.5-mini-instruct.Q8_0 [1],
- gemma-3-4b-it-qat-Q8_0 [8].

Each model was tested with temperatures 0.20, 0.30, 0.40, and 0.50. The models received only the issue title and description, and were instructed to return a JSON object of the form `{"p": <integer>}`. Early pilot experiments used a shorter prompt that emphasized output format but did not define the meaning of the score intervals precisely. This resulted in both more malformed outputs and lower accuracy. The final prompt therefore made the scoring semantics explicit.

This prompt design is important for two reasons. First, it reduces ambiguity between neighboring score regions. Second, it makes thresholding easier later, because the models are explicitly told what low, medium, and high scores represent.

Table 2. Score semantics communicated to the local LLMs.

Score band	Meaning given to the LLMs
0–20	Spam, advertisements, offensive content, empty reports, or gibberish
21–40	Not software-related, off-topic, or personal/social discussion
41–60	Vague or unclear issue with missing context
61–80	Acceptable issue that is understandable but lacks detail
81–100	High-quality issue with a clear problem statement, reproduction information, or a specific feature request

5. Experimental setup

An issue is classified as spam/noise when its validity score is below a threshold τ. We tested multiple thresholds and selected the best-performing value for each model configuration on the labeled benchmark. The best threshold was 75 for OwnMetrics and for most non-Llama configurations, whereas the best Meta-Llama-3.1-8B results were obtained at threshold 50. The full-benchmark results in Table 3 should therefore be interpreted as exploratory threshold-selection results rather than as unbiased estimates of generalization performance. To check the stability of this selection without additional model inference, we also performed a stratified five-fold threshold-selection validation over the already generated validity-score outputs: in each fold, the model configuration and threshold were selected on the training folds by accuracy, using F1 as a tie-breaker, and evaluated on the held-out fold.

The evaluation has three parts. First, we measure accuracy and F1 on the labeled benchmark. Second, we evaluate the selected configurations on the manually labeled 300-item Jira-based TAWOS validation set described above; this set is not used for threshold or model selection. Third, we examine deployment-oriented behavior on a 10,000-item sample drawn from the processed TAWOS issue corpus. The sample was created by the export script before validity filtering: issues were selected from the TAWOS `Issue` table using MySQL-level random ordering (`ORDER BY RAND()`) and the configured export limit was set to 10,000. The resulting CSV snapshot was then kept fixed and reused for all evaluated methods. No class labels, model scores, or threshold decisions were used when constructing this sample, and it was used only for robustness, score-distribution, agreement, spam-rate, and runtime analyses, not for training or threshold selection. Since runtime depends on the hardware and local software stack, the reported times should be interpreted comparatively rather than as universal constants.

For the broad comparison beyond the best-accuracy table, we use one representative configuration from each method family: OwnMetrics, Gemma-3-4B at temperature 0.50, Meta-Llama-3.1-8B at temperature 0.50, Mistral-7B at temperature 0.20, and Phi-3.5-mini at temperature 0.20. These configurations are used for the in-domain Jira/TAWOS validation and to compare deployment-oriented

behavior on the same fixed 10,000-item snapshot.

6. Results

6.1. Results on the labeled benchmark

Table 3 shows the best accuracy obtained by each model family over the tested temperature settings and thresholds in this exploratory threshold-selection benchmark.

Table 3. Best accuracy per model family on the labeled threshold-selection benchmark.

Model family	Temperature	Best accuracy	Best threshold
OwnMetrics	–	91.0%	75
Gemma-3-4B	0.20 / 0.30	89.2%	75
Mistral-7B-Instruct-v0.3	0.20	89.1%	75
Phi-3.5-mini-instruct	0.20	83.9%	75
Meta-Llama-3.1-8B-Instruct	0.20 / 0.50	66.2%	50

Figure 3 provides a visual summary of the same best-accuracy comparison across the evaluated model families.

OwnMetrics achieved the highest overall accuracy at 91.0%. Among the LLMs, Gemma-3-4B and Mistral-7B were the strongest, reaching 89.2% and 89.1%, respectively. Phi-3.5-mini was clearly usable but weaker, while Meta-Llama-3.1-8B lagged far behind the other methods.

Table 4 reports the stratified five-fold threshold-selection validation. The held-out results preserve the same ordering as the full-benchmark selection: OwnMetrics remains the strongest method, while Gemma-3-4B and Mistral-7B form the closest LLM baselines. The cross-validation also selected the same threshold pattern as the full benchmark in all folds: $\tau = 75$ for OwnMetrics, Gemma, Mistral, and Phi, and $\tau = 50$ for Meta-Llama.

The confusion-matrix values of the selected configurations are summarized in Table 5. OwnMetrics produced the strongest balance between false positives and false negatives. The selected Gemma and Mistral configurations were close to it, whereas Meta-Llama-3.1-8B missed a very large number of spam items.

6.2. OwnMetrics ablation

To better understand the contribution of the OwnMetrics components, we performed a small ablation study on the labeled benchmark. Since OwnMetrics is deterministic, this analysis does not require retraining or additional model inference. Removed components were excluded from the raw score, and the remaining component range was re-normalized to the 0–100 validity scale. For comparability

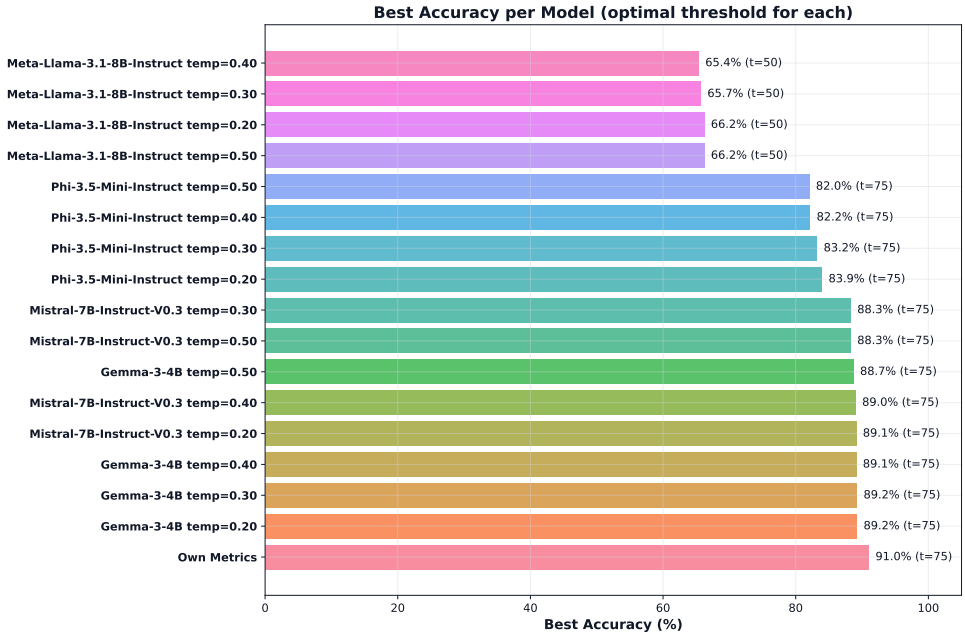


Figure 3. Best accuracy per model on the labeled threshold-selection benchmark.

Table 4. Stratified five-fold threshold-selection validation on the labeled benchmark. In each fold, the model configuration and threshold are selected on the training folds and evaluated on the held-out fold. Spam/noise is treated as the positive class. Accuracy and F1 are mean ± standard deviation; precision and recall are fold means. All values are percentages.

Model family	Accuracy	F1	Precision	Recall
OwnMetrics	91.0 ± 2.0	91.0 ± 2.1	92.1	90.0
Gemma-3-4B	88.8 ± 3.0	89.0 ± 2.8	89.1	89.0
Mistral-7B-Instruct-v0.3	88.8 ± 1.6	88.0 ± 1.6	96.3	81.1
Phi-3.5-mini-instruct	83.9 ± 2.1	83.9 ± 2.0	85.7	82.3
Meta-Llama-3.1-8B-Instruct	65.8 ± 2.3	49.4 ± 5.6	98.8	33.1

with the main filtering setting, all variants were evaluated using the fixed operational threshold $\tau = 75$.

The ablation results show that the early spam detector is the most influential component: removing it substantially reduces recall and F1. The technical-content and professionalism components also contribute to the final performance, while a variant using only title and description quality components is clearly insufficient.

Table 5. Confusion-matrix summary for selected model configurations on the labeled benchmark. Spam is treated as the positive class.

Method	TN	FP	FN	TP	Accuracy	F1
OwnMetrics	429	37	48	433	91.0%	91.1%
Gemma-3-4B ($t = 0.50$)	415	51	56	425	88.7%	88.8%
Mistral-7B ($t = 0.20$)	451	15	88	393	89.1%	88.4%
Phi-3.5-mini ($t = 0.20$)	399	67	85	396	83.9%	83.9%
Meta-Llama-3.1-8B ($t = 0.50$)	449	17	308	173	65.7%	51.6%

Table 6. Ablation analysis of OwnMetrics on the labeled benchmark at the fixed operational threshold $\tau = 75$.

Configuration	Accuracy	Precision	Recall	F1
Full OwnMetrics	91.0%	92.1%	90.0%	91.1%
Without early spam detector	79.7%	94.5%	63.8%	76.2%
Without technical-content component	90.1%	92.9%	87.1%	89.9%
Without professionalism component	89.7%	90.1%	89.4%	89.8%
Title and description components only	73.5%	94.6%	50.7%	66.0%

This suggests that the reported performance is not explained by a single length-based heuristic, but by the combination of early spam detection and multiple issue-quality signals.

6.3. In-domain Jira/TAWOS validation

The additional in-domain validation results are shown in Table 7. The set consists of 300 manually labeled Jira-based TAWOS issues. The selected thresholds and model configurations were fixed before this validation and were not tuned on the Jira/TAWOS set. Spam/noise is treated as the positive class. Since the set contains only 39 spam/noise cases and 261 non-spam cases, accuracy alone would be misleading; a majority-class baseline would already reach 87.0% accuracy while detecting no spam/noise issues. We therefore emphasize balanced accuracy, spam-class precision, recall, and F1. Invalid or unparsable outputs were treated as keep/non-spam decisions for the binary metrics and are also reported separately as failures.

The in-domain validation is consistent with the main benchmark conclusion at the level of a small Jira/TAWOS sanity check. OwnMetrics achieves the highest overall accuracy and the strongest spam-class F1, while producing no parsing failures. Gemma-3-4B obtains the highest spam recall, but it also produces more false positives. Mistral-7B remains close to OwnMetrics. Meta-Llama-3.1-8B has a seemingly high overall accuracy because most records are non-spam, but its bal-

Table 7. In-domain validation on 300 manually labeled Jira/TAWOS issues. The set contains 39 spam/noise and 261 non-spam issues. Thresholds and model configurations were selected before this validation and were not tuned on the Jira/TAWOS validation set. Spam/noise is the positive class, and all metric values are percentages except the confusion-matrix counts.

Method	τ	TP	FP	FN	TN	Acc.	Bal. acc.	Prec.	Rec.	F1	Fail.
OwnMetrics	75	33	28	6	233	88.7	86.9	54.1	84.6	66.0	0.0
Gemma-3-4B ($t = 0.50$)	75	36	48	3	213	83.0	87.0	42.9	92.3	58.5	0.0
Meta-Llama-3.1-8B ($t = 0.50$)	50	7	8	32	253	86.7	57.4	46.7	17.9	25.9	51.0
Mistral-7B ($t = 0.20$)	75	32	29	7	232	88.0	85.5	52.5	82.1	64.0	0.0
Phi-3.5-mini ($t = 0.20$)	75	34	48	5	213	82.3	84.4	41.5	87.2	56.2	1.7

anced accuracy and spam recall are low, and it is strongly affected by parsing failures.

6.4. Robustness on a 10,000-item sample

Accuracy alone is not sufficient for large-scale cleaning. Since the LLMs had to return a strict JSON format, we also measured output failures. OwnMetrics produced no failures at all. Gemma and Mistral were highly reliable, remaining around 0.1% failure. Phi-3.5-mini stayed below 1.2%. Meta-Llama-3.1-8B, however, failed on more than half of the 10,000-sample experiment, with failure rates between 54.0% and 59.2% depending on temperature.

Table 8 combines score statistics, failure rates, and runtime for the selected representative configurations. The median validity scores are consistently higher than the means, which indicates that most issues are rated as rather valid while a smaller set of heavily penalized spam items pulls the average downward. The LLMs also tend to have lower dispersion than OwnMetrics, suggesting more compressed internal scoring behavior.

Table 8. Summary statistics for selected representative configurations on a 10,000-item sample.

Method	Temp	Mean	Median	Std.	Min	Max	Failure	Time
OwnMetrics	-	78.8	85.0	21.1	5.0	96.0	0.0%	2.28 ms
Gemma-3-4B	0.50	78.6	85.0	13.4	0.0	100.0	0.1%	2.86 s
Meta-Llama-3.1-8B	0.50	79.8	81.0	19.5	0.0	100.0	54.0%	4.29 s
Mistral-7B	0.20	76.1	81.0	14.6	0.0	100.0	0.1%	5.77 s
Phi-3.5-mini	0.20	80.0	81.0	11.2	0.0	85.0	0.5%	4.78 s

6.5. Score distributions and spam agreement

The score-distribution statistics reveal an interesting trade-off. OwnMetrics is more spread out than the LLMs, which is consistent with its explicit penalties and rewards for many different textual phenomena. The LLMs compress more of their predictions into the upper mid-range, even when they separate spam from non-spam reasonably well. This can be useful if one wants stable relative ranking, but it may be less informative when strong score contrast is desired.

Pairwise agreement between the selected methods is also high. On the 10,000-item sample, the pairwise spam-decision agreement values lie between 0.83 and 0.87. This suggests that the models, despite their architectural differences, recognize a substantial common core of spam/noise cases. At the same time, the agreement is not perfect, which leaves room for ensemble strategies or manual review near the threshold boundary.

6.6. Spam detection rate and full-corpus filtering

At threshold 75, the selected configurations mark noticeably different proportions of the 10,000-item sample as spam. OwnMetrics flags 16.6% of items, Gemma around 19.2%, Mistral around 13.7–14.2%, Phi around 19.7–20.9%, and Meta-Llama only about 3.3–5.3%. This shows that even when two methods achieve similar benchmark accuracy, they may still define the operational spam boundary somewhat differently.

When the complete filtering pipeline was applied to the processed TAWOS issue corpus, 16.2% of records were flagged as spam-like, noisy, or otherwise unsuitable for further use. The resulting filtered corpus is intended to provide a cleaner basis for later extraction of statistically meaningful development-process properties and for realistic simulation studies.

7. Discussion and threats to validity

The empirical picture is consistent across the evaluation dimensions. OwnMetrics achieved the best exploratory benchmark accuracy, the strongest result in the five-fold threshold-selection validation, the best spam-class F1 on the small in-domain Jira-based TAWOS validation set, zero parsing failures, and by far the lowest runtime. The ablation study further indicates that this performance does not come from a single length-based heuristic: the early spam detector has the largest effect, but the technical-content and professionalism components also contribute. This suggests that for the present task, a hand-crafted scorer can outperform small local general-purpose LLMs when the target signal is dominated by surface-level and medium-depth textual regularities such as placeholders, promotional phrases, malformed content, missing coherence, and recognizable software-engineering vocabulary.

The LLMs nevertheless remain interesting baselines. Gemma-3-4B and Mistral-7B came surprisingly close to OwnMetrics in accuracy, and their pairwise agreement

with the rule-based method remained high. This indicates that local LLMs can be viable for issue validation even without task-specific training, provided that the prompt clearly defines the scoring semantics and the expected output format. However, the large gap in runtime and the non-negligible risk of malformed outputs remain practical disadvantages.

A first threat to validity comes from the labeled benchmark itself. The benchmark was built from GitHub issues, while the target corpus is Jira-based TAWOS data. Although the basic task – distinguishing software-related, meaningful issue descriptions from spam or noise – is similar across the two platforms, style differences, issue-management workflows, community norms, and moderation practices may not transfer perfectly. Consequently, the benchmark results should be treated as controlled evidence for spam/noise discrimination rather than as a direct estimate of in-domain TAWOS performance. The five-fold threshold-selection validation reduces the risk that the reported ranking is only a result of choosing a threshold on the full benchmark, but it does not remove this platform-shift limitation because the folds still come from the same GitHub-based benchmark. The additional 300-item Jira-based TAWOS validation set partially addresses this concern and provides direct in-domain evidence for the selected configurations. However, because it is small, imbalanced, and partly enriched with manually searched suspicious records, it should be interpreted as a diagnostic sanity check rather than as a full TAWOS gold standard or a prevalence estimate. A larger in-domain validation set would be needed for stronger external validation. A second threat is that only title and description were used. TAWOS also contains comments, change logs, issue links, users, versions, and sprint information [21]; these may provide useful extra evidence in ambiguous cases. A third threat is that runtime measurements are environment-specific. Finally, moderator-locked GitHub spam is a pragmatic but incomplete proxy for the broader notion of "noise": some low-quality issues are not spam, and some spam-like reports may escape moderation.

These threats also point to natural future work. The most promising extensions are to incorporate metadata beyond the title and description, to use hybrid decision rules where OwnMetrics handles obvious cases and an LLM reviews uncertain ones, and to calibrate the threshold based on the downstream task. For example, a training corpus for fine-tuning may justify a stricter threshold than a corpus intended for exploratory repository mining.

8. Conclusion

We presented a practical data-cleaning pipeline for the TAWOS issue database and focused on a validity-scoring stage that assigns a score from 0 to 100 to each issue text. The proposed OwnMetrics rule-based method combines an early spam detector with six interpretable scoring dimensions and achieved the strongest overall performance in our experiments. The added ablation analysis shows that the early spam detector is especially important, while the other scoring components also contribute to the final result. The small Jira-based TAWOS validation set provides

an in-domain sanity check consistent with the benchmark results. Among the local LLMs, Gemma-3-4B and Mistral-7B came closest in accuracy, but neither matched OwnMetrics in robustness or efficiency.

From a practical perspective, the main result is simple: for this type of large-scale issue cleaning, a domain-tailored deterministic scorer is already strong enough to serve as the backbone of an operational filtering pipeline. At the same time, the LLM experiments show that zero-shot local models are competitive enough to remain useful as baselines, secondary reviewers, or ensemble components. Applying the complete pipeline to the processed TAWOS corpus flagged a substantial share of records as potential noise, yielding a cleaner starting point for later educational, analytical, and modeling tasks.

References

[1] M. Abdin et al.: *Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone*, arXiv preprint arXiv:2404.14219 (2024), arXiv: 2404.14219 [cs.CL].

[2] G. Antoniol, K. Ayari, M. D. Penta, F. Khomh, Y.-G. Guéhéneuc: *Is It a Bug or an Enhancement? A Text-Based Approach to Classify Change Requests*, in: Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds, IBM Corp., 2008, pp. 304–318, doi: 10.1145/1463788.1463819.

[3] G. Colavito, F. Lanubile, N. Novielli: *Benchmarking Large Language Models for Automated Labeling: The Case of Issue Report Classification*, Information and Software Technology 184 (2025), p. 107758, doi: 10.1016/j.infsof.2025.107758.

[4] G. Colavito, F. Lanubile, N. Novielli, L. Quaranta: *Impact of Data Quality for Automatic Issue Classification Using Pre-Trained Language Models*, Journal of Systems and Software 210 (2024), p. 111838, doi: 10.1016/j.jss.2023.111838.

[5] A. Dubey et al.: *The Llama 3 Herd of Models*, arXiv preprint arXiv:2407.21783 (2024), arXiv: 2407.21783 [cs.LG].

[6] R. Ehsani, M. M. Imran, R. Zita, K. Damevski, P. Chatterjee: *Incivility in Open Source Projects: A Comprehensive Annotated Dataset of Locked GitHub Issue Threads*, in: Proceedings of the 21st IEEE/ACM International Conference on Mining Software Repositories, Association for Computing Machinery, 2024, pp. 515–519, doi: 10.1145/3643991.3644887.

[7] I. Ferreira, B. Adams, J. Cheng: *How Heated Is It?: Understanding GitHub Locked Issues*, in: Proceedings of the 19th International Conference on Mining Software Repositories, Association for Computing Machinery, 2022, pp. 309–320, doi: 10.1145/3524842.3527957.

[8] Gemma Team: *Gemma 3 Technical Report*, arXiv preprint arXiv:2503.19786 (2025), arXiv: 2503.19786 [cs.CL].

[9] GH Archive: *GH Archive*, 2026, url: https://www.gharchive.org/ (visited on 03/16/2026).

[10] GitHub: *REST API Endpoints for Issues*, 2026, url: https://docs.github.com/rest/issues/issues (visited on 03/16/2026).

[11] S. Herbold, A. Trautsch, F. Trautsch: *On the Feasibility of Automated Prediction of Bug and Non-Bug Issues*, Empirical Software Engineering 25.6 (2020), pp. 5333–5369, doi: 10.1007/s10664-020-09885-w.

[12] K. Herzig, S. Just, A. Zeller: *It's Not a Bug, It's a Feature: How Misclassification Impacts Bug Prediction*, in: Proceedings of the 2013 International Conference on Software Engineering, IEEE Press, 2013, pp. 392–401, doi: 10.1109/ICSE.2013.6606585.

[13] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, H. Wang: *Large Language Models for Software Engineering: A Systematic Literature Review*, ACM Transactions on Software Engineering and Methodology 33.8, 220 (2024), pp. 1–79, DOI: 10.1145/3695988.

[14] A. Q. Jiang et al.: *Mistral 7B*, arXiv preprint arXiv:2310.06825 (2023), arXiv: 2310.06825 [cs.CL].

[15] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, D. Damian: *An In-Depth Study of the Promises and Perils of Mining GitHub*, Empirical Software Engineering 21.5 (2016), pp. 2035–2071, DOI: 10.1007/s10664-015-9393-5.

[16] M. Laiq, F. Dobslaw: *Automatic Techniques for Issue Report Classification: A Systematic Mapping Study*, Automated Software Engineering 33, 72 (2026), DOI: 10.1007/s10515-026-00616-x.

[17] N. Nikeghbal, A. H. Kargaran, A. Heydarnoori: *GIRT-Model: Automated Generation of Issue Report Templates*, in: Proceedings of the 21st IEEE/ACM International Conference on Mining Software Repositories, Association for Computing Machinery, 2024, pp. 407–418, DOI: 10.1145/3643991.3644906.

[18] G. Rejithkumar, P. R. Anish, S. Ghaisas: *Text-To-Text Generation for Issue Report Classification*, in: Proceedings of the 3rd ACM/IEEE International Workshop on Natural Language-based Software Engineering, Association for Computing Machinery, 2024, pp. 53–56, DOI: 10.1145/3643787.3648042.

[19] M. Soltani, F. Hermans, T. Bäck: *The Significance of Bug Report Elements*, Empirical Software Engineering 25.6 (2020), pp. 5255–5294, DOI: 10.1007/s10664-020-09882-z.

[20] E. Sülün, M. Saçakçı, E. Tüzün: *An Empirical Analysis of Issue Templates Usage in Large-Scale Projects on GitHub*, ACM Transactions on Software Engineering and Methodology 33.5, 117 (2024), pp. 1–28, DOI: 10.1145/3643673.

[21] V. Tawosi, A. Al-Subaihin, R. Moussa, F. Sarro: *A Versatile Dataset of Agile Open Source Software Projects*, in: Proceedings of the 19th International Conference on Mining Software Repositories, New York, NY, USA: Association for Computing Machinery, 2022, pp. 707–711, DOI: 10.1145/3524842.3528029.

[22] F. Tu, F. Zhang: *Measuring the Quality of Issue Tracking Data*, in: Proceedings of the 6th Asia-Pacific Symposium on Internetware, Association for Computing Machinery, 2014, pp. 76–79, DOI: 10.1145/2677832.2677844.

[23] X. Wu, W. Zheng, X. Xia, D. Lo: *Data Quality Matters: A Case Study on Data Label Correctness for Security Bug Report Prediction*, IEEE Transactions on Software Engineering 48.7 (2022), pp. 2541–2556, DOI: 10.1109/TSE.2021.3063727.

[24] J. Zhang, X. Wang, D. Hao, B. Xie, L. Zhang, H. Mei: *A Survey on Bug-Report Analysis*, Science China Information Sciences 58.2 (2015), pp. 1–24, DOI: 10.1007/s11432-014-5241-2.

[25] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schröter, C. Weiss: *What Makes a Good Bug Report?*, IEEE Transactions on Software Engineering 36.5 (2010), pp. 618–643, DOI: 10.1109/TSE.2010.63.