

Automated Detection of AI-Obfuscated Plagiarism in Modeling Assignments

Timur Sağlam, Sebastian Hahner, Larissa Schmid, Erik Burger
firstname.lastname@kit.edu
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany

ABSTRACT

Plagiarism is a widespread problem in computer science education, exacerbated by the impracticability of manual inspection in large courses. Even worse, tools based on large language models like ChatGPT have made it easier than ever to obfuscate plagiarized solutions. Additionally, most plagiarism detectors only apply to code, and only a few approaches exist for modeling assignments, which lack broad resilience to obfuscation attacks. This paper presents a novel approach for automated plagiarism detection in modeling assignments that combines automated analysis with human inspection. We evaluate our approach with real-world assignments and plagiarism obfuscated by ChatGPT. Our results show that we achieve a significantly higher detection rate for AI-generated attacks and a broader resilience than the state-of-the-art.

CCS CONCEPTS

• **Information systems** → Near-duplicate and plagiarism detection; • **Software and its engineering** → Model-driven software engineering; • **Social and professional topics** → Computer science education; Software engineering education;

KEYWORDS

Plagiarism Detection, Obfuscation, ChatGPT, Artificial Intelligence

ACM Reference Format:

Timur Sağlam, Sebastian Hahner, Larissa Schmid, Erik Burger. 2024. Automated Detection of AI-Obfuscated Plagiarism in Modeling Assignments. In *46th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET '24)*, April 14–20, 2024, Lisbon, Portugal. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3639474.3640084>

1 INTRODUCTION

Plagiarism is a widespread phenomenon in today's computer science education [7, 15, 31, 35] as digital assignments can be easily duplicated and altered. Students still engage in plagiarism despite the threat of consequences [58]. Moreover, students are creative in *obfuscating* their plagiarism to conceal the relation to the original. In the case of code submissions, students commonly utilize techniques such as renaming, re-ordering, or restructuring [27, 50]. This is particularly common when assignments are mandatory, e.g., in beginners' courses [42]. As computer science courses are often

large [13], manual inspection is impractical [19], and the individual risk of detection is low [64]. In the past, obfuscating software plagiarism was considered more challenging than completing the assignment, as manual obfuscation was considered tedious and complicated [19]. However, with the rise of plagiarism generators [19], plagiarizing has become easier than ever. AI-powered tools can generate or alter source code and modeling artifacts [16, 18] with little manual effort and technical knowledge, making plagiarism obfuscation more accessible [29]. ChatGPT [39] combines the capabilities of large language models (LLMs) with the accessible interface of a chatbot, thus further exacerbating the problem [20].

Automated plagiarism detection was proposed over two decades ago [40, 43], but most state-of-the-art plagiarism detectors only apply to code [1, 8, 32, 37, 43, 44, 47]. However, computer science assignments and exams often include modeling assignments [14, 50], for example, UML assignments regarding class, use case, sequence, and activity models. Furthermore, modeling assignments become more common with the increasing adoption [10, 24] of model-driven techniques. These assignments are prone to plagiarism due to their complexity and their requirement of domain understanding and problem-solving skills [34]. While there is research on model differencing and model clone detection, these techniques alone are insufficient for plagiarism detection, as they are not resilient against obfuscation attempts [34, 49, 62]. There are two plagiarism detection approaches specifically targeting modeling assignments [34, 49]. However, both approaches are prone to *obfuscation attacks* [47], which try to conceal the relation between plagiarism and source. Martínez et al. [34] propose an approach based on *linear similarity hashing* (LSH). However, their approach is prone to obfuscation techniques such as renaming and insertion. In previous work [49], we introduced token-based plagiarism detection for metamodels. However, our initial approach is prone to re-ordering attacks [50] and has limited applicability as it can only be used for metamodels. Furthermore, it is unclear how well both approaches perform against the rising threat of AI-based plagiarism.

In this paper, we thus present a plagiarism detection approach for modeling assignments that outperforms previous approaches [34, 49], particularly regarding AI-obfuscated plagiarism. Our approach leverages the concept of token-based plagiarism detection but hardens obfuscation resilience by introducing a novel normalization technique for the token sequence. Furthermore, it applies to any EMF-based modeling artifact and can be generalized to any other MOF-conforming [38] artifacts with a tree-like structure. Our approach combines automated analysis with human judgment, thus ensuring an ethical process. It further scales well, thus allowing plagiarism detection even in large courses. We evaluate our approach using real-world data sets based on a modeling assignment.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

ICSE-SEET '24, April 14–20, 2024, Lisbon, Portugal

© 2024 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-0498-7/24/04.

<https://doi.org/10.1145/3639474.3640084>

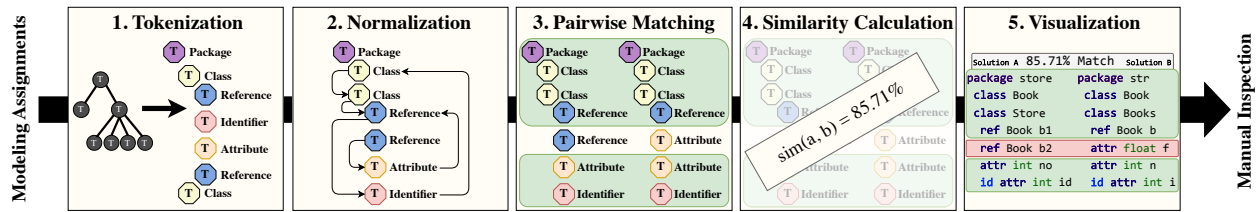


Figure 1: Overview of our modeling plagiarism detection approach and its steps, as detailed in sections 3.2, 3.3, 3.4, 3.5, and 3.6.

We cover three plagiarism types: AI-obfuscated plagiarism using ChatGPT, manual human plagiarism, and artificial plagiarism based on singular attack types. Our results demonstrate broad resilience against obfuscation attacks, particularly against the most prevalent [50] attacks based on re-ordering and renaming. Moreover, our approach reliably identifies AI-obfuscated plagiarism and significantly outperforms the state-of-the-art. Our contributions are:

- C1 A token-based plagiarism detection approach¹ for models, metamodels, and other modeling artifacts. It is obfuscation-resilient and significantly outperforms the state-of-the-art.
- C2 An examination of how well ChatGPT, with its natural language abilities, user-friendly interface, and wide popularity among students, can be exploited for modeling plagiarism.
- C3 A comprehensive three-stage evaluation using a real-world data set with both human and AI-obfuscated plagiarism.

In the following, Section 2 covers the status quo, Section 3 presents our detection approach (C1), Section 4 examines exploiting ChatGPT for plagiarism (C2), Section 5 provides the evaluation (C3), Section 6 discusses related work, and Section 7 concludes the paper.

2 STATE OF THE ART

Plagiarism detection for code is widely-researched [8, 37]. However, there is little work on modeling plagiarism detection. Software plagiarism detectors have a long history [40, 43]. Most approaches compare the structure of the code [36, 37], and among them, token-based approaches like MOSS [1], and JPlag [43], are the most widely employed in practice [37]. These approaches combine tokenization with pairwise comparison to identify code matches based on hashing and tiling [1, 44]. MOSS and Dolos [32] use winnowing [51], JPlag and Sherlock [25] use greedy string tiling with the running Karp-Rabin matching [60, 61]. Tokenization transforms the code into a parse tree, extracting a subset of nodes as tokens, thus linearizing the tree [47]. This abstraction ensures efficiency and resilience against obfuscation attacks like renaming, retyping, or obscuring constant values [44], and immunity against *lexical* [25] attacks.

To our knowledge, only two approaches for modeling plagiarism detection exist. Martínez et al. [34] use Locality Sensitive Hashing (LSH) by transforming models into context-aware fragments and then into integer vectors using *minhash*. The hamming distance between their LSH signatures determines the models' similarity. This approach combines contextual rather than structural information, making it vulnerable to lexical obfuscation attacks. The LSH signatures are based on names, which enables renaming-based attacks. The approach also lacks sufficient information on how

similarities are calculated, making manual inspection difficult. In previous work [49], we presented token-based plagiarism detection for metamodels using JPlag. While it is inherently immune to renaming attacks, it lacks broad resilience and is susceptible to re-ordering-based obfuscation. Despite demonstrating the feasibility of token-based plagiarism detection for metamodels, our initial approach has limited usability due to its restriction to metamodels. Both approaches are vulnerable to common [50] attack types, and how they handle AI-based plagiarism is unclear. In contrast, our approach combines the benefits of token-based plagiarism detection with a novel normalization approach, providing broad resilience against obfuscation attacks. It shows matching sections of the assignments, allowing auditability of the results. Last, we also consider usability aspects such as post-processing and visualization.

3 MODELING PLAGIARISM DETECTION

This section presents our first contribution (C1), a general approach for automated plagiarism detection for modeling assignments. The similarity between models and code can be leveraged for token-based approaches for modeling artifacts [49]. These approaches typically extract tokens from parse tree nodes, primarily focusing on the code's structure. In contrast, for modeling plagiarism, the focus must be on modeling patterns, structure, and relationships. A key challenge is the higher abstraction than code, where the lack of details impedes plagiarism detection. Furthermore, resilience against re-ordering-based obfuscation attacks is challenging. Re-ordering-based attacks are more common [50] as they are considerably easier for most models than for code, where the sequence of statements determines the code's behavior. In our approach, modeling assignment solutions undergo a comprehensive pairwise comparison process. To facilitate this comparison, an abstraction layer is extracted from each solution, on which the structures of these artifacts are compared. Ultimately, our approach produces a list of suspicious solutions, which are the subject of the instructor's final evaluation. Our five-step approach is illustrated in Figure 1:

- (1) **Tokenization:** We extract the token sequence by linearizing the model, abstracting from superfluous detail.
- (2) **Normalization:** We achieve resilience against element re-ordering by normalizing the token sequence.
- (3) **Pairwise Matching:** We efficiently compare pairs of token sequences to find all matching subsequences.
- (4) **Similarity Calculation:** We rank the pairs by leveraging different similarity metrics for the matches of a pair.
- (5) **Visualization:** We help humans to assess the results and decide what constitutes plagiarism effectively.

Our approach's novelty lies in steps 1, 2, and 5, while steps 3 and 4 are established techniques from token-based plagiarism detection.

¹We include the implementation of our approach in the supplementary material [48]. It is also integrated into the open-source software plagiarism detector JPlag [26].

3.1 Running Example

As a minimal running example, we consider a typical [14] undergraduate course where students are instructed in the elementary principles of modeling. In order to satisfy the course requirements, they need to complete an assignment that entails modeling a management system for bookstores, requiring the creation of an appropriate metamodel. The domain is described as follows:

Bookstore can have multiple locations that sell different books. Books are classified into different genres and have a unique identifier called an ISBN. Books have an author who may use a stage name. Stores have one owner.

This example is intentionally small to illustrate our approach and provide clarity for the reader; most modeling assignments are usually more comprehensive. Two potential solutions for this assignment are depicted in Figure 2. Although they exhibit some similarities, several features are modeled differently. While this is a small task, it is hard for a human to determine how similar these models are even for these two solutions. However, in a course, there may be 30 to 300 students [14]. Thus, manual checking for plagiarism becomes impractical.

3.2 Tokenization

Modeling assignments include diverse artifacts like metamodels, models, and model transformations [14], all vital in the modeling process. Most of these artifacts are typically systematized in a tree-like structure. Throughout this section, we will mainly refer to models and their elements. However, our approach applies to other tree-based modeling artifacts, such as metamodels, transformation rules, et cetera. Our approach transforms the modeling artifacts into an abstraction layer on which pairwise comparison can be performed. By omitting *some* details while including others, mainly structural information, this abstraction layer is resilient against certain obfuscation attacks such as renaming and re-typing. For example, we omit the names of elements and the exact types of

attributes, as they are usually the first to be changed for obfuscation purposes [49, 50]. In our approach, we tokenize as follows:

- (1) We iterate in a depth-first order over the tree structure of the model's containment references, starting from its root element.
- (2) For each element, we decide what tokens may be extracted. The extraction strategy can either be generic or domain-specific.
- (3) Token sequences from multiple files or artifacts are concatenated into one token sequence but separated via a pivot token.

Figure 3 illustrates this tokenization for EMF metamodels using our running example. The token tree (Figure 3a) is linearized into a token sequence (Figure 3b). Each token represents information from the metamodel's structure and is thus an abstract representation of the metamodel.

We discuss two kinds of strategies for selecting tokens: The first kind is *generic strategies*, where a token is extracted for each model element, and the identity of the token is the type of the element, e.g., the metaclass or meta-metaclass, respectively. Additionally, for elements that may contain additional child elements, a context end token is extracted after child elements, representing the context of the containment reference. However, the generic strategy has certain limitations. Some model elements are irrelevant and should not be extracted as tokens, as they increase the noise in the token sequence and can be used to obfuscate plagiarism. Additionally, the properties of elements are not represented in the token sequence at all. If these properties are semantically relevant, this leads to a lack of representation in the token sequence. As an example, for EMF metamodels, classes, and interfaces are differentiated by a property of the class. While this kind of strategy applies to any modeling assignment artifacts, it may perform worse for a distinct domain than a strategy specifically tailored to one domain. However, a generic strategy may perform well enough for many applications.

The second kind is *domain-specific strategies*, where the selection of tokens is designed for a single domain, metamodel, or transformation language. They allow more fine-tuning for selecting relevant tokens. For that, several rules apply. Two elements of a different type that can be used interchangeably without changing the semantics of the modeling artifact should map to a token with the same identity. Figure 3 illustrates such a domain-specific token selection. As seen in Figure 3b, attributes are extracted via the same token, independent of the attribute's data type. In contrast, elements of the same type that, through their properties, have different semantics should be mapped to different tokens. Furthermore, the use of end-of-context tokens can provide additional information for the detection of plagiarism. As seen in Figure 3b, attributes and identifier attributes are extracted as different tokens, even though they are the same metaclass. Similarly, this is done for references and containment references. Furthermore, properties themselves may lead to the extraction of tokens, e.g., for supertype references or similar (see Figure 3b).

In conclusion, selecting tokens for the abstraction layer in token-based plagiarism detection techniques requires careful consideration of the relevant elements and properties in the software artifacts. A domain-specific strategy allows for a more fine-tuned token selection and is thus generally preferred. The generic strategy is a good baseline for domains without a domain-specific strategy.

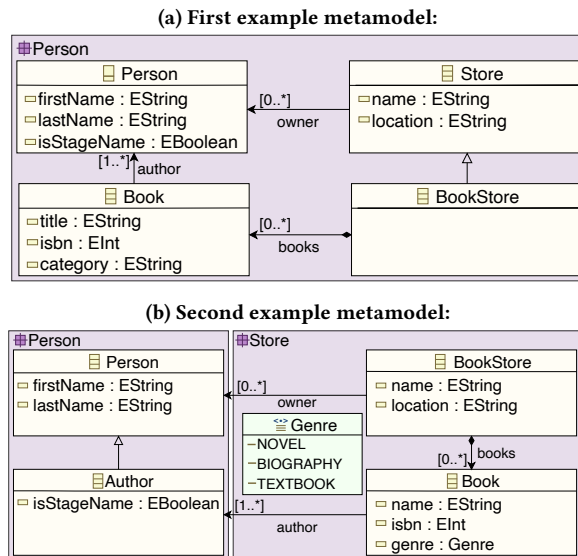
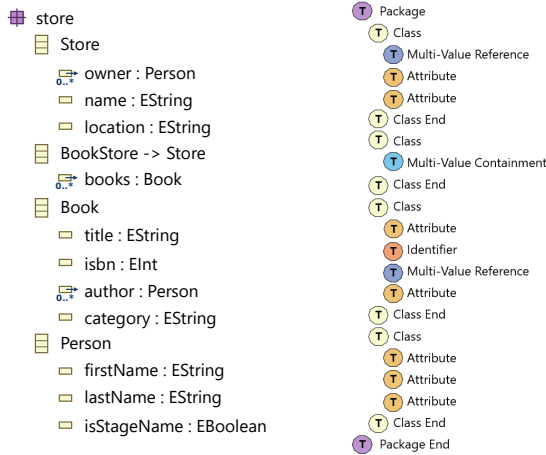


Figure 2: Two similar but not identical example metamodels modeling a bookstore.



(a) first example metamodel (b) extracted token sequence

Figure 3: Tokenization of metamodels to extract a linear token sequence as an abstraction layer for the comparison.

3.3 Normalization

While the token sequence is resilient against obfuscation attacks such as renaming or re-typing, it is prone to attacks based on moving and swapping elements [49] because most models allow variations in the order of containment references. Therefore, we normalize the order of containment references and, thus, the model tree while extracting tokens to ensure that the same patterns are represented identically across different modeling assignments. However, normalization of the token sequence is a non-trivial task. The properties used for normalization must be selected carefully to avoid introducing new attack vectors, and the process must minimize changes to the token sequence to avoid false positives. For example, if the normalization properties are names, identifiers, or aggregated properties like the number of contained elements, they can be easily changed with renaming or insertion attacks to affect the normalization and, thus, the token sequence. Normalization properties shall be stable, invariant, and meaningful to ensure accurate and effective plagiarism detection. Simple normalization approaches like sorting elements by their type or the lexical order of their children’s types are insufficient and vulnerable. The normalization is either too ineffective or can be affected via specific obfuscation attacks.

To address this, we propose a novel approach to normalize the order of the token sequence. We sort the elements in each containment reference first based on their token type and then based on the distribution of the tokens in their subtree of direct and indirect children. The normalization algorithm is depicted in Algorithm 1. We generate tokens for each element’s subtree and create a token type vector, showing how often each type occurs. For the class Book in Figure 3, this vector is $[2, 1, 1, 0, \dots, 0]$, as it contains two attributes, one identifier attribute, and one reference. To compare two of these vectors, we calculate the Euclidean norm $\|v\|_2 = \sqrt{\sum_{i=1}^n v_i^2}$ for each (to scale their length to 1) and interpret the resulting vectors as points in a multi-dimensional coordinate space, where each token type represents a dimension. We can calculate the nearest neighbor path between all points to compare, starting from the element’s point with the most tokens in the subtree. To measure

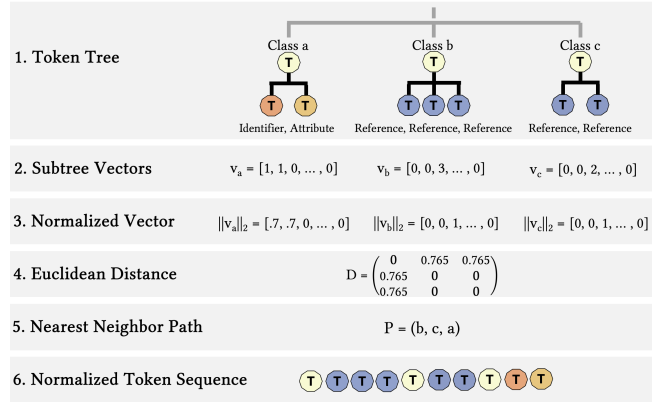


Figure 4: Token sequence normalization via element re-ordering of a containment reference.

Algorithm 1 Normalize Token Sequence

Require: List of model elements E with length n

- 1: $V \leftarrow$ empty list of vectors
- 2: **for** $e_i \in E$ **do**
- 3: $v_i \leftarrow$ calculate token type vectors for subtree of e_i
- 4: $v_i \leftarrow$ euclidean norm $\|v_i\|_2$
- 5: **end for**
- 6: $D \leftarrow$ empty distance matrix with size $n \times n$
- 7: **for** $v_i, v_j \in V$ **do**
- 8: $d_{i,j} \leftarrow$ euclidean distance $d_E(v_i, v_j)$
- 9: **end for**
- 10: $v_{max} \leftarrow$ vector in V of element with largest subtree
- 11: $P \leftarrow$ nearest neighbor path for D starting from v_{max}
- 12: $S \leftarrow$ sort E by token type, if equal by position in P
- 13: **return** S

the distance between two n-dimensional points, p and q , we use the Euclidean distance metric $d_E(p, q) = \sqrt{\sum_{i=1}^n (q_i - p_i)^2}$. If, during the nearest neighbor path calculation, two or more points have the same distance to the last point in the path, we use the original order of the elements to ensure that normalization is stable. To normalize the order of the elements in a containment reference, we sort first based on their token type. If equal, we sort based on the order in the calculated path. Figure 4 shows the re-ordering of tokens for three elements of the same containment reference. The normed subtree vectors are used to calculate the elements’ distances, sorting the elements according to the computed nearest neighbor path. For our approach, we use the nearest neighbor path instead of the shortest path because modifying the subtrees affects the nearest neighbor path less. For the path, we chose the element with the most tokens in its subtree, as the resulting paths resulted in the most resilient normalization in our pre-study. Overall, our approach provides an effective solution to the normalization problem in modeling plagiarism detection.

3.4 Pairwise Matching

In this step, all $n * (n - 1) / 2$ token sequence pairs are compared, and matching subsequences are detected. Several highly efficient algorithms exist for this task, allowing thousands of comparisons in seconds. We thus use an adapted form of *Greedy String Tiling* [60]. We thus scan the token sequences iteratively, identifying the longest common subsequence between two token sequences. The found

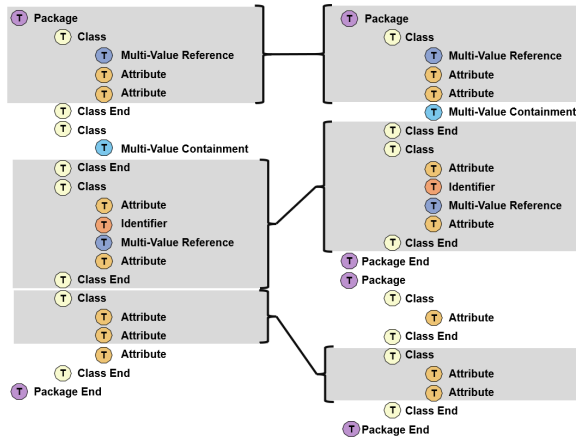


Figure 5: Subsequence matches for two token sequences of our running example with a minimal token match of three.

subsequence is marked as a match, and we then move on to the next smaller subsequence and repeat until no more matches can be found. We employ a rolling hash function for the subsequence search for improved performance, as detailed in [60] and [43]. Finally, to regulate the algorithms’ match sensitivity and to avoid false positives, we use a hyperparameter called *minimal token match* (MTM), which denotes the minimal length of matches, whereby shorter matches are ignored. Lowering the MTM value increases the sensitivity for plagiarism while also increasing the number of false positives. Our approach allows configuring the minimal token match to tweak the plagiarism detection to the datasets at hand. In Figure 5, the subsequence matches for our example are shown. The MTM value is three. Thus, subsequences of lengths below are not considered matches. These matches can be utilized for visualization purposes, aiding human inspection.

3.5 Similarity Calculation

The similarity of each modeling assignment pair can be calculated based on the subsequence matches. For an assignment pair (a, b) with the number of matched tokens $m(a, b)$ and the total number of tokens l_a and l_b , the similarity can be calculated with $sim(a, b) = \frac{2 \cdot m(a, b)}{l_a + l_b}$. For pairs of assignments of different sizes, the maximum similarity $maxsim(a, b) = \max(sim(a, b), sim(b, a))$ can be used. It is less resilient against false positives but better suited if many elements were inserted to obfuscate the plagiarism. We then use both metrics to rank pairs regarding their similarity. For our example in Figure 5, the similarities are calculated to be $sim = 68.3\%$ and $maxsim = 71.4\%$, respectively. Using these metrics, we apply three kinds of post-processing to enhance the results for human inspection.

- (1) Ranked Lists of Pairs: A list showing the assignment pairs in descending order by their similarity. This provides a prioritization in which order human inspection can be conducted.
- (2) Similarity Distribution: A histogram showing what similarities are common for most comparisons (probable true negatives) and if outliers exist (probable true positives).
- (3) Clustering of Assignments: We cluster assignments using hierarchical agglomerative and spectral clustering to detect group plagiarism. The clustering is configured via hyperparameters.

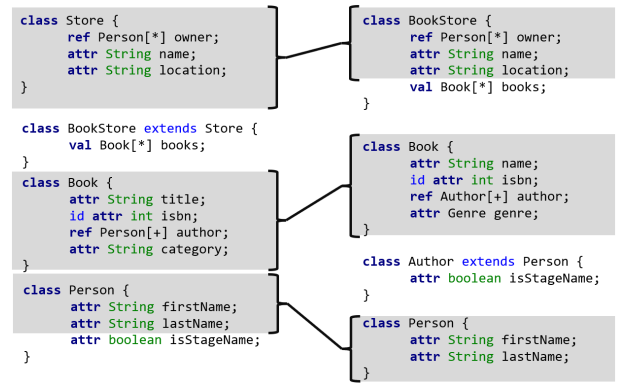


Figure 6: Linear representation of the matched metamodels of our running example via the Emfatic textual syntax [17].

3.6 Visualization

An essential benefit of token-based approaches is the explainability of the calculated similarities. Moss [1] and JPlag [43], for example, allow inspecting pairs of programs in a side-by-side code view, where the matched sections are highlighted. This facilitates the human assessment of the results and effective decision-making regarding what constitutes plagiarism. Providing this feature is crucial for the practical usage of plagiarism detectors [31]. For modeling artifacts, a graphical syntax or custom editors are commonly used to view them. However, a textual syntax has the benefit of a linear structure that allows visualizing matches just as for code. Nevertheless, a linear graphical syntax like a tree view also brings these benefits. We propose a side-by-side view based on textual syntaxes, thus effectively linearizing the modeling artifact. The choice of textual syntax has to be domain-specific. For example, for EMF metamodels, *Emfatic* [17] allows depicting them in a side-by-side view. We generate a simple tree-view-like view for domains without textual syntax based on the model elements and their containment structure. Figure 6 illustrates such a match visualization for the running example. Emfatic is used to visualize the similarities between the metamodels. During human inspection, it is comprehensible how the similarity scores were calculated, and which parts are similar.

4 MODELING PLAGIARISM WITH CHATGPT

As our second contribution (C2), we investigate whether ChatGPT can be effectively exploited for plagiarism. Unlike other plagiarism generators [19] and LLMs that have been less accessible, ChatGPT is widely available, easy to use, and popular among students [20]. We show the feasibility of leveraging it for modeling plagiarism, requiring only a minimal understanding of modeling concepts. Besides natural language capabilities, ChatGPT can “understand” and generate programming languages and other technical artifacts, such as XMI [18]. Thus, it can reason about EMF modeling artifacts and generate them. This can then be exploited for plagiarizing modeling assignments. There are generally two ways of using ChatGPT to cheat for modeling assignments:

- (1) Full-generation: The plagiarizer uses the assignment’s description to generate a complete solution using ChatGPT
- (2) Obfuscation: The plagiarizer provides a pre-existing solution and asks ChatGPT to generate an obfuscated version.

Whether the first technique constitutes plagiarism is still a subject of debate [2], while the latter closely aligns with human plagiarism practices [37]. To determine how viable these approaches are, we explored both options² for a typical [14] metamodeling assignment [49]. It tasks students with creating an EMF metamodel for designing component-based system architectures.

4.1 Fully-Generating a Solution

We tasked ChatGPT to generate a solution from scratch by providing the full assignment description. For prompt engineering, we approached ChatGPT with the mind of a novice student modeler [50]. We systematically tested multiple prompts where the most expedient was directly asking for an EMF metamodel that satisfies the description and is provided as syntactically correct XMI. Further inquiries, e.g., suggestions for improvement or correction, did not enhance the result quality³. Using this prompt, we conducted twenty sessions using two separate accounts. After each solution, we regenerated the next most likely response. We requested a single re-generation per session if it produced incoherent output, i.e., invalid XMI code. Although we were able to generate 36 EMF-compatible solutions using ChatGPT, none fulfilled the assignment. First, most of them contained a plethora of syntactical issues, including incorrect assignments of primitive types to attributes, improper assignment of reference types, duplicate names of structural features in types and super types, invalid lower bound cardinalities (e.g., -1), missing instance type names for enumerations, and packages lacking namespace URI and prefix. Second, all solutions were semantically insufficient, missing some essential elements. Incorrect modeling of relations between concepts and improper use of enumerations were common themes. Comparatively, the generated solutions contain about half as many classifiers and references as human solutions.

To evaluate the solutions regarding completeness and originality, we randomly selected a subset consisting of 7 ChatGPT-generated and 3 human solutions and asked the course instructor to review them. The instructor was unaware that some of the metamodels were generated. We asked the instructor to review the metamodels regarding their originality and whether they would accept the metamodels as valid solutions. We employed the *Think Aloud* method for the review, asking the instructor to verbalize their thoughts and actions. This process is similar to the experiment setup described in [50], where we conducted a similar experiment on human plagiarism. We asked a course instructor to check metamodels regarding their originality and validity as a solution. The instructor reviewed the solutions individually and arranged them side by side. They initially examined the overall structure and package partitioning. Within packages, they specifically checked for the presence of elements described in the task. They accepted only the three human solutions, as the other solutions contained errors and missed crucial concepts of the assignment. They noted that the generated solutions appeared similar, with small sizes and minimal structuring into packages.

The similarity can be attributed to the generated solutions' recurring patterns in correctly modeled concepts and the occurring syntactical and semantical issues. ChatGPT exhibits some degree

Table 1: Similarity of unrelated human originals compared to the similarity of the solutions generated with ChatGPT.

Type	Median	Mean	Q1	Q3	Maximum
Human	18.25	19.75	12.21	25.35	58.15
ChatGPT	35.37	36.97	26.23	47.60	86.84

of determinism in its outputs. To further examine this, we used our approach (C1) on both to compare the similarity of generated and human solutions. The results are shown in Table 1. The values indicate that, despite their small size and issues, generated solutions are notably more similar than unrelated human solutions. This shows that our approach can effectively detect many of these generated solutions. In summary, fully generating solutions works for small assignments, but the results are inadequate and easily detectable.

4.2 Obfuscating a Pre-Existing Solution

To obfuscate a pre-existing solution, we provide ChatGPT only with a solution in XMI form and instruct it to alter the structure of the solution to conceal the plagiarism while retaining semantics to fulfill the assignment. We applied this approach to solutions from the modeling course: a monolithic one with all elements in a single file and a fragmented one with elements distributed across five files in various packages. We generally observed a stronger obfuscation for the fragmented solution, which may relate to the semantic cohesion of the concepts modeled in the same package. We applied twelve different prompts in independent sessions for each solution, thus generating twelve plagiarized metamodels. The following is an example of such a prompt: *Change the following EMF metamodel in XMI format to look like a different one that models the same concepts. Show changed lines, including a description and the line number.*

ChatGPT employs various modifications to alter the solution, detailing what part of the XMI code changed. These modifications included inserting single classes, attributes, and references, deleting elements and their contained elements, re-ordering elements in containment references, and moving attributes or references to newly inserted superclasses. ChatGPT also moved classes and datatypes to different packages and renamed elements by abbreviating or removing abbreviations, using synonyms, or adding prefixes and suffixes based on the domain. Moreover, it changed the properties of existing elements, such as multiplicities, and added references to classes that referred to either existing or new classes. Vice versa, it added new classes containing multiple attributes and meaningful references to new and existing classes. It often made complex changes by adding sizable structures that had semantic relevance to the solution's domain. One example that arose multiple times included adding a new class named `Node` with a one-to-many relationship to the existing class `Links` and a many-to-one relationship to `Container`. Additionally, ChatGPT modified the `Links` class to depict a directed link between two nodes. It included two references, `source` and `target` to represent the endpoints. In one example, ChatGPT inserted a new subclass `SystemComponent` of an existing class `Component` and added a reference `partOf` which referenced an existing class `System`. In another one, a new abstract class called `NamedElement` with an attribute called `name` was inserted. Then, it introduced this class as a supertype for several existing classes that removed their `name` attribute, effectively duplicating the attributes.

²We use version 3.5 for the full generation, and version 3.0 and 3.5 for the obfuscation.

³We provide the method and all raw results as part of the supplementary material.

Most of these changes are not immediately apparent, as they fit the assignment’s domain. Even if modeling issues occurred, they were insignificant enough to be considered plausible human mistakes. Only rarely were the changes immediately conspicuous, and for those instances, it was due to chosen element names lacking logical coherence (such as renaming from `Deployment` to `DeploymentNew`). We found that ChatGPT occasionally generated minor syntactical issues, but for the most part, it produced correct metamodels that were well-obfuscated according to our instructions. In contrast to the technique of fully generating, ChatGPT seems to thrive on the given metamodel, thus avoiding most issues discussed in Subsection 4.1 by replicating what already exists.

4.3 Assessment

In summary, while ChatGPT can fully generate solutions for modeling assignments, they are inadequate, stand out to the human eye, and are likely to get flagged during tool-based inspection. Recent studies reached the same conclusion [16]. However, given a pre-existing solution, the usefulness of ChatGPT increases. It can “comprehend” the modeled domain and reason about the structure and concepts. Moreover, ChatGPT can perform complex changes, providing high flexibility in generating obfuscated models. While fully generating solutions might become feasible in the future, cheating via plagiarism by obfuscation currently seems the most feasible strategy, as it requires little modeling knowledge and produces well-obfuscated plagiarism that is inconspicuous for humans.

5 THREE-STAGE EVALUATION

This section evaluates our approach and compares it to the state-of-the-art. We show that our normalization technique achieves broad resilience against different plagiarism obfuscation attacks. Next, we demonstrate our approach’s effectiveness in detecting AI-based and human plagiarism (C3). The evaluation artifacts are available in the supplementary material [48]. We evaluate in three stages:

S1 Evaluation for isolated attack types.

S2 Evaluation for manual plagiarism by novice modelers.

S3 Evaluation for AI-obfuscated plagiarism by ChatGPT.

To measure the extent of the detection, we compare the differences between the plagiarism similarities and the similarities of unrelated originals (true negatives).

5.1 Dataset and Evaluation Design

We evaluate our approach on real student solutions and compare it to our previous token-based approach [49] and the LSH-based approach by Martínez et al. [34]. Note that our previous approach does not use our normalization technique. We implemented our approach based on JPlag [43]. For all compared approaches, we use the recommended hyperparameter. We did not conduct parameter tuning with the evaluation dataset for any of the three approaches to ensure a fair comparison. We also used the adapted version [49] of the implementation of Martínez et al. [34] that supports multi-file metamodels.

Data Set. We base our evaluation on a real dataset from previous work [49], which contains 210 real submission pairs from a typical [14] modeling assignment. The submissions stem from a model-driven software development practical course. It is a master’s level elective course. Students in groups between two and five

Table 2: Obfuscation attack types and their corresponding level [27] and type [3]. Adapted from [49].

Attack	Package	Class	Operation	Attribute	Reference	Supertype	Level	Type
Insert	✓	✓	✓	✓	✓	✓	L2.5, L3, L4	B, C
Delete	-	✓	-	✓	✓	-	-	B
Re-Order	✓	✓	-	✓	✓	-	L2.5, L3	B, C
Rename	✓	✓	-	✓	✓	-	L2, L2.5	C

were tasked to create metamodels for designing component-based system architectures. The metamodel allows the creation of models similar to UML component diagrams but also involves the aspect of software-to-hardware allocation. This assignment is loosely based on the Palladio component model (PCM) [4, 45]. This dataset is, to the best of our knowledge, free of plagiarism itself. On average, the metamodels contain five packages, 39 classifiers, 45 references, ten attributes, and one operation. While most metamodels were designed in a single file, some students fragmented them into several files. The dataset serves as a set of original solutions for all three stages of our evaluation. We then employ different kinds of plagiarism in each stage. The pairwise comparison of 21 submissions yields 210 pairs, i.e., $\binom{21}{2} = 210$, which are false positives when detected as plagiarism.

Metrics. To measure the detection rate for a plagiarism detector, we must compare the plagiarisms’ similarities to their sources and unrelated solutions’ similarities. For a human inspection to be feasible at scale, the plagiarism pairs must protrude from the rest of the solutions. Thus, for each of these approaches, we measure different similarity distributions:

- (1) *Plagiarism-To-Source Pairs:* the similarity distribution of the plagiarisms with their respective source solutions.
- (2) *Original Pairs:* the similarity distribution of the unrelated original solutions among themselves.
- (3) *Plagiarism-To-Plagiarism Pairs:* the similarity distribution of the plagiarisms of a single source with each other.

The bigger the distance between plagiarism and non-plagiarism pairs, the easier it is to detect plagiarism effectively. Thus the similarity of the plagiarism pairs should be high, and the similarity of the original pairs should be low. If these pairs overlap, it is hardly possible to distinguish the plagiarisms from the originals. Thus, we measure whether and to what extent the considered approaches can find a noticeable difference between these pairs.

5.2 Resilience to Isolated Attack Types

Our first stage demonstrates broad resilience to different types of obfuscation attacks. We reused the artificially generated plagiarism from previous work [49]. There, we applied 20 obfuscation attacks of a single type based on existing classifications [3, 27]. We then randomly chose four original metamodels, thus generating 80 plagiarisms. Table 2 shows the complete attack set. We executed each attack for ten random model elements. For re-ordering-based attacks, ten random elements are moved or swapped. For the insertion of references, existing classes were referenced. For renames, we generate pseudo-realistic names based on fragments of existing elements’ names, which cannot be distinguished from real names at first glance.



Figure 7: Results for the attack type evaluation for our current approach, our previous one [49] (denoted as Sağlam et al.), and the LSH-based approach by Martínez et al. [34].

5.2.1 Results. Figure 7 shows the results for the different attack types. For deletion attacks, all three approaches can be affected by deleting elements and their children. However, this effect is limited for our current and previous approaches, as only very few plagiarisms drop below a similarity of 75% to their source metamodels. In contrast, for the LSH-based approach of Martínez et al. [34], this is the case for more than half of the plagiarism-to-source pairs. The median value drops below 75%. In contrast to deletion attacks, all approaches are only slightly affected by insertion. Our approaches perform very well, achieving median similarity values above 90%. The approach of Martínez et al. [34] performs slightly worse but still maintains a high median value. However, for insertions and deletions, their approach has some outliers that come close to the similarity values of the original pairs. For Re-ordering based attacks, our current approach outperforms the others, with the 25th percentile Q_1 being above 90% similarity, showing near-complete immunity. The approach of Martínez et al. [34] also performs well, showing next to no effect of the obfuscation attack. Our previous approach [49], however, can be affected by re-ordering attacks, showing only little resilience. For renaming-based obfuscation attacks, both our approaches show complete immunity, which is an inherent trait of token-based approaches. In contrast, the approach of Martínez et al. [34] is prone to renaming, showing a Q_1 at around 55%, thus having many pairs close the unrelated originals. This means their approach has next to no resilience to renaming.

5.2.2 Discussion. While deletion-based attacks seem effective on paper, they are not feasible in practice. Usually, very little can be removed from an assignment’s artifact without making the solution incomplete or even plain wrong. In our evaluation, we randomly removed elements along with their children, which sometimes resulted in removing large subtrees. With that in mind, the effects of these attacks were relatively mild, and our current approach, as well as our previous one [49], still allows differentiation between plagiarism and non-plagiarism. Regarding insertion attacks, our approach shows resilience, while for re-ordering and renaming, our approach demonstrates complete immunity. As expected, our previous approach [49] is prone to re-ordering-based attacks due to the lack of token sequence normalization. However, it is immune to renaming. In contrast, the approach from Martínez et al. [34] is

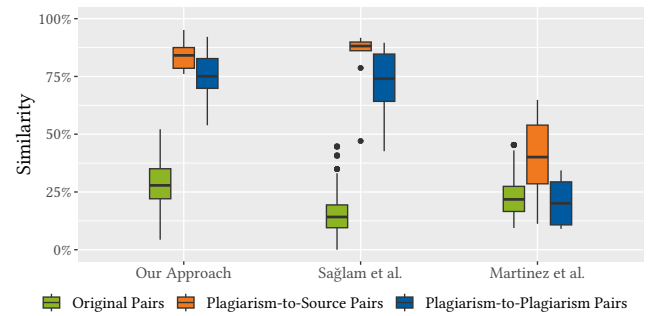


Figure 8: Results regarding human-generated plagiarism for our approach, our previous one [49] (denoted as Sağlam et al.), and the LSH-based approach by Martínez et al. [34].

strongly vulnerable to renaming-based attacks. It is worth noting that renaming and re-ordering are probably the most trivial attacks, both for humans and computers [50]. Overall, our approach performs well. The approach of Martínez et al. [34] performs the worst among the compared approaches. *While existing approaches are vulnerable to at least one attack type, ours demonstrates broad resilience while maintaining low similarity for pairs of unrelated metamodels.*

5.3 Detecting Human-Obfuscated Plagiarism

Our evaluation’s second stage evaluates the capabilities of plagiarism by novice modelers. We use the dataset introduced in [50] for this. It consists of 31 metamodels, thus producing 210 unrelated pairs, 10 direct and 10 indirect plagiarism pairs. These plagiarisms were created by novice modelers who were provided with the meta-modeling assignment outlined in Section 5.1 along with pre-existing solutions from the corresponding dataset. The participants made various types of changes, with an average of 43.7 changes per participant and a median of 31.5. On average, this corresponds to one change for every two model elements.

5.3.1 Results. The results of the manually obfuscated plagiarisms by novice modelers are illustrated in Figure 8. The results demonstrate that our approach can effectively distinguish the plagiarism-to-source pairs from the unrelated original pairs, as there is no overlap. Our previous approach [49] also demonstrates good separation except for one outlier falling below 50 percent similarity. However, for the approach of Martínez et al. [34], the plagiarism-to-source pairs still exhibit increased similarities but overlap with the unrelated pairs, making detection more challenging. Similar results are observed for the plagiarism-to-plagiarism pairs. Our approach exhibits a clear separation between plagiarism pairs and unrelated pairs. Our previous one [49] also provides a sufficient separation between them, although there is some overlap with the unrelated pairs, which drop down to 40 percent similarity. In contrast, for Martínez et al. [34], the plagiarism-to-plagiarism median similarity falls below the median of the unrelated pairs, making detection next to impossible.

5.3.2 Discussion. Some instances were sufficiently well-obfuscated to yield low similarities for our previous approach [49]. These instances accomplished this by heavily relying on re-ordering model elements, which is a vulnerability of this approach [50]. This observation aligns with the results of the first evaluation stage depicted

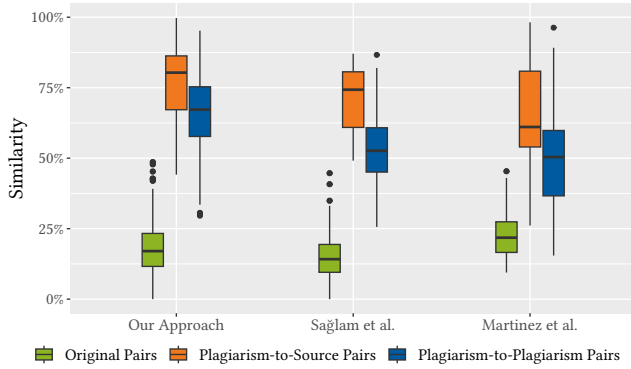


Figure 9: Results regarding ChatGPT-obfuscated plagiarism for our approach, our previous one [49] (denoted as Sağlam et al.), and the LSH-based approach by Martínez et al. [34].

in Figure 7. Our normalization technique is explicitly designed to address this type of attack. Nevertheless, both our current and previous approach yield overall high similarities for most plagiarism-to-source pairs. For these instances of plagiarism, we observed relatively weak obfuscation, which was not as strong as the obfuscation applied by ChatGPT. Thus, our normalization inherently has a limited impact on many of the comparisons. However, it still notably improves the detection of outliers. We can observe a more noticeable impact of our normalization in plagiarism-to-plagiarism pairs. Overall, our previous approach [49] exhibits a lower similarity for the unrelated pairs, which can be attributed to our normalization technique. The normalization reduces the impact of certain structural and syntactic variations between the unrelated pairs, thus slightly increasing their similarity. This effect, however, does not impact the separability of the original from the plagiarism pairs. In fact, the distinction between the stronger obfuscated instances and the original pairs is more considerable for our current approach than for our previous one. *In sum, both our current and previous approaches demonstrate strong performance in detecting human plagiarism, whereas Martínez et al. [34] perform worse.*

5.4 Detecting AI-obfuscated Plagiarism

In our third and main evaluation stage, we show the ability to detect comprehensively AI-obfuscated plagiarism. As discussed in Section 4, we investigated fully-generating solutions with ChatGPT. However, since this produced invalid and obviously-suspicious metamodels due to the currently limited modeling capabilities of ChatGPT [16], we chose solutions obfuscated by ChatGPT to evaluate our approach and compare it with the state-of-the-art. We let ChatGPT generate obfuscated versions of metamodels from the base dataset. This method requires little modeling knowledge and produces plagiarism that is well obfuscated and inconspicuous to the human eye. While ChatGPT occasionally generated minor syntactical issues, it mainly produced correct metamodels that were obfuscated according to our instructions. As discussed in Subsection 4.2, we provided ChatGPT with metamodels from the dataset and applied twelve different prompts for two metamodels to generate a total of 24 plagiarisms. This results in 24 Plagiarism-to-Source

Table 3: Similarity differences between the plagiarisms and the unrelated originals (higher means better detection) for the results in Figure 9.

Plagiarism Type	Approach	Δ Mean	Δ Median	Δ IQR
Plagiarism-to-Source	Ours	59.77	64.39	45.37
	Sağlam et al.	56.87	60.22	40.23
	Martínez et al.	42.16	39.27	26.54
Plagiarism-to-Plagiarism	Ours	48.01	50.94	33.80
	Sağlam et al.	39.12	38.56	25.64
	Martínez et al.	27.43	28.58	9.20

Table 4: Significance of the results in Figure 9 assessed via one-sided Wilcoxon rank-sum tests (significance level of $\alpha = 0.05$, alternative hypothesis H_1 , p-value p , test statistic W , Cliff’s delta δ , sample size n) for Plagiarism-to-Source (P2S), Plagiarism-to-Plagiarism (P2P), and Original Pairs (OP).

Comparison	Pairs	H_1	p	W	δ	n
Ours to Sağlam et al.	P2S	greater	0.042	373	0.293	24
	P2P	greater	<0.001	13,024	0.495	132
	OP	less	1.000	232,839	0.173	630
Ours to Martínez et al.	P2S	greater	0.008	405	0.406	24
	P2P	greater	<0.001	13,291	0.526	132
	OP	less	<0.001	131,810	-0.336	630

pairs and 132 Plagiarism-to-plagiarism pairs, which must be distinguished from the 210 unrelated original pairs (true negatives).

5.4.1 Results. Figure 9 shows the results for plagiarism generated by ChatGPT. For the plagiarism-to-source pairs, our current and previous approaches achieve high median scores of 78% and 74%, respectively. The approach of Martínez et al. [34] has a significantly lower median score of 59%. We observe similar results when looking at the distance between the plagiarism-to-source pairs and the unrelated original pairs as listed in Table 3. Our approach achieves the highest values for all three difference metrics (Δ Mean, Δ Median, and Δ IQR). While our previous one [49] follows closely by between three to five percentage points, Martínez et al. [34] perform significantly worse, with 17 to 25 percentage points below our approach. The differences between the approaches are even more prominent when looking at the similarity scores of the plagiarism-to-plagiarism pairs. Our approach achieves the highest median score with 67%, our previous one achieves 53%, and Martínez et al. [34] achieve only 43%. Our approach achieves the highest values for the distances between the plagiarism-to-plagiarism pairs and the unrelated original pairs listed in Table 3. Our previous one [49] achieves between eight to twelve percentage points less, Martínez et al. [34] achieve between less 21 to 25 percentage points less.

To determine the statistical significance of the evaluation results presented in Figure 9, we conducted one-sided Wilcoxon rank-sum tests with a significance level of $\alpha = 0.05$. Ideally, to demonstrate a significantly better performance than the other approaches, ours has to show significantly higher scores for the plagiarism pairs (P2S and P2P) and significantly lower scores for the unrelated original pairs (OP). The results of the tests are summarized in Table 4. As the test statistic value (W) depends on the sample size and the sample

sizes of the pair types differ, we use Cliff’s delta (δ) as a normalized metric for the effect size. In the context of the Plagiarism-to-Source (P2S) and Plagiarism-to-Plagiarism (P2P) pairs, our approach achieves significantly higher similarity scores than both other approaches. While our approach achieves significantly lower scores for the original pairs (OP) compared to Martínez et al. [34], this is not the case for our previous approach [49]. However, the effect size is smaller than the effect sizes for the plagiarism pairs, which aligns with the similarity score differences in Table 3.

5.4.2 Discussion. Our evaluation found that our approach significantly outperforms the state-of-the-art regarding the detection of ChatGPT-generated metamodels. This is especially apparent when the obfuscation is strong, which is true for plagiarism-to-plagiarism pairs. While our previous approach [49] also performs well, it is less resilient to strong obfuscation attempts. This can be attributed to it being susceptible to reordering attacks, as shown in Subsection 5.2. Our current approach, however, is resilient to these attacks due to our normalization technique. Reordering and renaming are also likely the first steps a plagiarizer would take after using ChatGPT [50]. The difference between our approach and Martínez et al. [34] is even more considerable, as ChatGPT frequently employs renaming. The renaming attacks of ChatGPT are particularly effective, as ChatGPT considers the modeling domain and identifies suitable synonyms. Interestingly, as ChatGPT is partially deterministic, we observed recurring patterns for different prompts, such as inserting a “Node” and “Edge” class and adding references from specific existing classes to the inserted one. This shows that even with systematic prompt engineering and strong obfuscation, plagiarizers cannot be certain to avoid detection. *In sum, plagiarism generated with ChatGPT currently is predictable enough to be reliably detected by our approach. Our approach also significantly outperforms the state-of-the-art, especially for strongly obfuscated plagiarism.*

5.5 Threats to Validity

We now discuss how we addressed threats to validity according to [63] and [46]. Regarding *internal validity*, a potential threat is the impact of prompts on the strength of AI-generated plagiarism using ChatGPT. To mitigate this, we experimented with different prompts in a pre-study and asked for more changes after each prompt. The *external validity* and generalizability are threatened by the limited availability of datasets. There is only one published dataset [50] for modeling tasks, as students’ data is sensitive and rarely published. We addressed this limitation by carefully designing our evaluation of three different plagiarism types using a single dataset. EMF-based metamodels are widely-used, and metamodeling is a typical assignment [14]. Furthermore, the used dataset is from a real-world modeling assignment. To enhance *construct validity*, we closely oriented our evaluation design on existing work in this field [49]. To enhance *reliability*, we provide our approach in the supplementary material [48]. The datasets contain sensitive data and are only partially included. Finally, the detailed description of our evaluation design allows for re-applying it to other data sets. One *limitation* is that detecting plagiarism in small modeling assignments is challenging, as very little information is available. Minor similarities can lead to false positives. Additionally, small modeling tasks often use common patterns, thus challenging plagiarism detection. However, this problem is intrinsic and applies to all approaches.

6 RELATED WORK

In this section, we review related work that are not applicable to metamodel plagiarism detection due to their vulnerability to obfuscation attacks [49] and computational inefficiency for large courses [34]. Our focus is on MDE modeling artifacts, and image-based plagiarism detection [23, 41]. We loosely relate to automated UML grading [5, 6, 9, 22], where also the similarity of models needs to be computed. However, plagiarism detection tackles obfuscation attacks and the problem of separating plagiarism from the original.

Model Clone Detection. Clone detection [21, 52] deals with finding syntactically or semantically identical code fragments in a program. While the employed techniques are similar to plagiarism detection, clone detectors do not face the threat of obfuscation attacks. For clone detection, the size of the modification should be reflected in the similarity measures. In contrast, for plagiarism detection, it is even valid that some changes will not reduce the similarity at all to achieve resilience against obfuscation attacks. We relate to metamodel clone detection [53, 55–57], like the approach by Babur et al. [3]. However, these approaches are insufficient for plagiarism detection as they are prone to typical obfuscation attacks.

Model Differencing. Model Differencing is extensively researched [28, 54] and is commonly used for model versioning. Various approaches exist [30, 33, 65]. It is related to plagiarism detection, as both determine the similarity between models. Nevertheless, model differencing is vulnerable to obfuscation attacks. The de-facto standard is *EMF Compare* [12], which can be customized for specific matching strategies. EMF Compare’s identifier-based strategy fails to detect plagiarism, as changing identifiers is an easy obfuscation attack. Furthermore, its similarity-based strategy is susceptible to renaming and reordering-based attacks. Wittler et al. [62] demonstrate this in a different context, yet the same principle holds true. Since EMF Compare allows custom strategies, our approach could be implemented as one in order to use the visualization EMF Compare provides. Finally, we could employ related approaches for change visualization [11, 59].

7 CONCLUSION

This paper presented a novel, token-based approach for automated plagiarism detection of modeling assignments. Our approach is resilient against all common obfuscation attacks by combining model tokenization with a novel token sequence normalization. Nevertheless, our evaluation demonstrates that our approach consistently identifies AI-obfuscated plagiarism and significantly outperforms the state of the art. This paper contributes to academic integrity by providing critical insights into AI-based modeling plagiarism. Our approach benefits computer science education by providing a reliable means to check modeling assignments for plagiarism.

ACKNOWLEDGMENTS

This work is based on the project *SofDCar* (19S21002), which the German Federal Ministry for Economic Affairs and Climate Action funds. It is also supported by the Ministry of Science, Research and the Arts Baden-Württemberg (Az: 7712.14-0821-2), the pilot program Core Informatics of the Helmholtz Association (HGF), by the topic Engineering Secure Systems of the HGF and by KASTEL Security Research Labs.

REFERENCES

- [1] Alex Aiken. 2022. *MOSS Software Plagiarism Detector Website*. Stanford University. <http://theory.stanford.edu/~aikens/moss/> Accessed: 2024-01-12.
- [2] Brent A. Anders. 2023. Is using ChatGPT cheating, plagiarism, both, neither, or forward thinking? *Patterns* 4, 3 (3 2023), 100694. <https://doi.org/10.1016/j.patter.2023.100694>
- [3] Önder Babur, Loek Cleophas, and Mark van den Brand. 2019. Metamodel clone detection with SAMOS. *Journal of Computer Languages* 51 (apr 2019), 57–74. <https://doi.org/10.1016/j.cola.2018.12.002>
- [4] Steffen Becker, Heiko Kozirolek, and Ralf Reussner. 2009. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software* 82 (1 2009), 3–22. <https://doi.org/10.1016/j.jss.2008.03.066>
- [5] Weiyi Bian, Omar Alam, and Jörg Kienzle. 2019. Automated Grading of Class Diagrams, In 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), Loli Burgueño, Alexander Pretschner, Sebastian Voss, Michel Chaudron, Jörg Kienzle, Markus Völter, Sébastien Gérard, Mansooreh Zahedi, Erwan Bousse, Arend Rensink, Fiona Polack, Gregor Engels, and Gerti Kappel (Eds.). *MODELS 2019*, 700–709. <https://doi.org/10.1109/MODELS-C.2019.00106>
- [6] Weiyi Bian, Omar Alam, and Jörg Kienzle. 2020. Is Automated Grading of Models Effective? Assessing Automated Grading of Class Diagrams, In Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (Canada), Eugene Syriani, Houari A. Sahraoui, Juan de Lara, and Silvia Abrahão (Eds.). *MODELS 2020*, 365–376. <https://doi.org/10.1145/3365438.3410944>
- [7] Jess Bidgood and Jeremy B. Merrill. 2017. As computer coding classes swell, so does cheating. *The New York Times* (2017). <https://www.nytimes.com/2017/05/29/us/computer-science-cheating.html> Accessed: 2024-01-12.
- [8] Miguel A. Botto Tobar, Mark G.J. van den Brand, and Alexander Serebrenik. 2022. Cross-Language Plagiarism Detection: Methods, Tools, and Challenges: A Systematic Review. *International Journal on Advanced Science, Engineering and Information Technology* 12, 2 (20 May 2022), 589–599. <https://doi.org/10.18517/ijaseit.12.2.14711>
- [9] Younes Boubekeur, Gunter Mussbacher, and Shane McIntosh. 2020. Automatic Assessment of Students' Software Models Using a Simple Heuristic and Machine Learning, In Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (Canada), Esther Guerra and Ludovico Iovino (Eds.). *MODELS 2020*, Article 20, 10 pages. <https://doi.org/10.1145/3417990.3418741>
- [10] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2017. *Model-Driven Software Engineering in Practice*. Springer International Publishing. <https://doi.org/10.1007/978-3-031-02549-5>
- [11] M.G.J. Brand, van den, Z. Protic, and T. Verhoeff. 2011. RCVDiff - a stand-alone tool for representation, calculation and visualization of model differences. In *ME 2010 - International Workshop on Models and Evolution*. United States.
- [12] Cédric Brun and Alfonso Pierantonio. 2008. Model differences in the eclipse modeling framework. *UPGRADE, The European Journal for the Informatics Professional* 9, 2 (2008), 29–34.
- [13] Tracy Camp, W. Richards Adrion, Betsy Bizot, Susan Davidson, Mary Hall, Susanne Hambrusch, Ellen Walker, and Stuart Zweben. 2017. Generation CS: The Growth of Computer Science. *ACM Inroads* 8, 2 (may 2017), 44–50. <https://doi.org/10.1145/13084362>
- [14] Federico Ciccozzi, Michalis Famelis, Gerti Kappel, Leen Lambers, Sebastian Mosser, Richard F. Paige, Alfonso Pierantonio, Arend Rensink, Rick Salay, Gabi Taentzer, Antonio Vallecillo, and Manuel Wimmer. 2018. How Do We Teach Modelling and Model-Driven Engineering? A Survey, In Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (Copenhagen, Denmark), Önder Babur, Daniel Strüber 0001, Silvia Abrahão, Loli Burgueño, Martin Gogolla, Joel Greenyer, Sahar Kokaly, Dimitris S. Kolovos, Tanja Mayerhofer, and Mansooreh Zahedi (Eds.). *MODELS 2018*, 122–129. <https://doi.org/10.1145/3270112.3270129>
- [15] Georgina Cosma and Mike Joy. 2008. Towards a Definition of Source-Code Plagiarism. *IEEE Transactions on Education* 51, 2 (may 2008), 195–200. <https://doi.org/10.1109/te.2007.906776>
- [16] Javier Cámara, Javier Troya, Loli Burgueño, and Antonio Vallecillo. 2023. On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML. *Software and Systems Modeling* 22, 3 (May 2023), 781–793. <https://doi.org/10.1007/s10270-023-01105-5>
- [17] Chris Daly. 2004. *Eclipse Emfatic*. Eclipse Foundation. <https://www.eclipse.org/emfatic/> Accessed: 2024-01-12.
- [18] Marian Daun and Jennifer Brings. 2023. How ChatGPT Will Change Software Engineering Education, In Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 (Turku, Finland), Mikko-Jussi Laakso, Mattia Monga, Simon, and Judith Sheard (Eds.). *Annual Conference on Innovation and Technology in Computer Science Education*, 110–116. <https://doi.org/10.1145/3587102.3588815>
- [19] Breanna Devore-McDonald and Emery D. Berger. 2020. Mossad: Defeating Software Plagiarism Detection. *Proceedings of the ACM on Programming Languages* 4, OOPSLA, Article 138 (nov 2020), 28 pages. <https://doi.org/10.1145/3428206>
- [20] Henner Gimpel, Kristina Hall, Stefan Decker, Torsten Eymann, Luis Lämmermann, Alexander Mädche, Maximilian Röglinger, Caroline Ruiner, Manfred Schoch, Mareike Schoop, Nils Urbach, and Steffen Vandrik. 2023. Unlocking the power of generative AI models and systems such as GPT-4 and ChatGPT for higher education. (Mar 2023). opus.uni-hohenheim.de/volltexte/2023/2146/
- [21] Muhammad Hammad, Önder Babur, Hamid Abdul Basit, and Mark Van Den Brand. 2022. Clone-Seeker: Effective Code Clone Search Using Annotations. *IEEE Access* 10 (6 2022), 11696–11713. <https://doi.org/10.1109/ACCESS.2022.3145686>
- [22] Robert W. Hasker. 2011. UMLGrader: An Automated Class Diagram Grader. *J. Comput. Sci. Coll.* 27, 1 (oct 2011), 47–54.
- [23] Petr Hurtik and Petra Hodakova. 2015. FTIP: A tool for an image plagiarism detection, In 2015 7th International Conference of Soft Computing and Pattern Recognition (SoCPaR), Mario Köppen, Bing Xue, Hideyuki Takagi, Ajith Abraham, Azah Kamilah Muda, and Kun Ma (Eds.). *International Conference of Soft Computing and Pattern Recognition*, 42–47. <https://doi.org/10.1109/SOCPAR.2015.7492780>
- [24] John Hutchinson, Mark Rouncefield, and Jon Whittle. 2011. Model-Driven Engineering Practices in Industry, In Proceedings of the 33rd International Conference on Software Engineering (Waikiki, Honolulu, HI, USA), Richard N. Taylor, Harald Gall, and Nenad Medvidovic (Eds.). *International Conference on Software Engineering*, 633–642. <https://doi.org/10.1145/1985793.1985882>
- [25] Mike Joy and Micheal Luck. 1999. Plagiarism in programming assignments. *IEEE Transactions on Education* 42, 2 (may 1999), 129–133. <https://doi.org/10.1109/13.762946>
- [26] JPlag. 2023. *JPlag Repository*. GitHub. <https://github.com/jplag/JPlag> Accessed: 2024-01-12.
- [27] Oscar Karnalim. 2016. Detecting source code plagiarism on introductory programming course assignments using a bytecode approach. In *2016 International Conference on Information & Communication Technology and Systems (ICTS)*. IEEE, 63–68. <https://doi.org/10.1109/icts.2016.7910274>
- [28] Marouane Kessentini, Ali Ouni, Philip Langer, Manuel Wimmer, and Slim Bechikh. 2014. Search-based metamodel matching with structural and syntactic measures. *Journal of Systems and Software* 97 (11 2014), 1–14. <https://doi.org/10.1016/j.jss.2014.06.040>
- [29] Mohammad Khalil and Erkan Er. 2023. Will ChatGPT get you caught? Rethinking of Plagiarism Detection. *Interacción abs/2302.04335* (Feb 2023), 475–487. <https://doi.org/10.48550/arXiv.2302.04335>
- [30] Dimitrios Kolovos, Davide Di Ruscio, Alfonso Pierantonio, and Richard Paige. 2009. Different models for model matching: An analysis of approaches to support model differencing, In 2009 ICSE Workshop on Comparison and Versioning of Software Models. *2009 ICSE Workshop on Comparison and Versioning of Software Models*, 1–6. <https://doi.org/10.1109/CVSM.2009.5071714>
- [31] Tri Le, Angela Carbone, Judy Sheard, Margot Schuhmacher, Michael de Raath, and Chris Johnson. 2013. Educating Computer Programming Students about Plagiarism through Use of a Code Similarity Detection Tool, In 2013 Learning and Teaching in Computing and Engineering. *Learning and Teaching in Computing and Engineering*, 98–105. <https://doi.org/10.1109/LaTiCE.2013.37>
- [32] Rien Maertens, Charlottte Van Petegem, Niko Srijbol, Toon Baeyens, Arne Carla Jacobs, Peter Dawyndt, and Bart Mesuere. 2022. Dolos: Language-agnostic plagiarism detection in source code. *Journal of Computer Assisted Learning* 38, 4 (8 2022), 1046–1061. <https://doi.org/10.1111/jcal.12662>
- [33] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. 2011. A Manifesto for Semantic Model Differencing, In Models in Software Engineering, Jürgen Dingel and Arnor Solberg (Eds.). *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems* 6627, 194–203. https://doi.org/10.1007/978-3-642-21210-9_19
- [34] Salvador Martínez, Manuel Wimmer, and Jordi Cabot. 2020. Efficient plagiarism detection for software modeling assignments. *Computer Science Education* 30, 2 (jan 2020), 187–215. <https://doi.org/10.1080/08993408.2020.1711495>
- [35] William Murray. 2010. Cheating in Computer Science. *Ubiquity* 2010 (06 2010), 2. <https://doi.org/10.1145/1865907.1865908>
- [36] Lawton Nichols, Kyle Dewey, Mehmet Emre, Sitao Chen, and Ben Hardekopf. 2019. Syntax-Based Improvements to Plagiarism Detectors and Their Evaluations, In Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education (Aberdeen, Scotland UK), Bruce Scharlau, Roger McDermott, Arnold Pears, and Mihaela Sabin (Eds.). *ITICSE 2019*, 555–561. <https://doi.org/10.1145/3304221.3319789>
- [37] Matija Novak, Mike Joy, and Dragutin Kermek. 2019. Source-Code Similarity Detection and Detection Tools Used in Academia: A Systematic Review. *ACM Transactions on Computing Education* 19, 3, Article 27 (sep 2019), 37 pages. <https://doi.org/10.1145/3313290>
- [38] Object Management Group (OMG). 2016. *Meta Object Facility (MOF) Core Specification*. Version 2.5.1.
- [39] OpenAI. [n. d.]. *Introducing ChatGPT*. <https://openai.com/blog/chatgpt> Accessed: 2023-04-12.
- [40] K. J. Ottenstein. 1976. An Algorithmic Approach to the Detection and Prevention of Plagiarism. *ACM SIGCSE Bulletin* 8, 4 (dec 1976), 30–41. <https://doi.org/10.1145/382222.382462>

- [41] Prajakta Ovhal. 2015. Detecting plagiarism in images. In *2015 International Conference on Information Processing (ICIP)*. IEEE, 85–89. <https://doi.org/10.1109/INFOP.2015.7489356>
- [42] Chris Park. 2003. In Other (People's) Words: Plagiarism by university students—literature and lessons. *Assessment & Evaluation in Higher Education* 28, 5 (oct 2003), 471–488. <https://doi.org/10.1080/02602930301677arXiv:https://doi.org/10.1080/02602930301677>
- [43] Lutz Prechelt, Guido Malpohl, and Michael Philippsen. 2000. *JPlag: Finding plagiarisms among a set of programs*. Karlsruhe Institute of Technology. <https://doi.org/10.5445/ir/542000> Technical Report.
- [44] Lutz Prechelt, Guido Malpohl, Michael Philippsen, et al. 2002. Finding plagiarisms among a set of programs with JPlag. *Journal of Universal Computer Science* 8, 11 (2002), 1016. <https://doi.org/10.3217/jucs-008-11-1016>
- [45] Ralf H. Reussner, Steffen Becker, Jens Happe, Robert Heinrich, Anne Koziolk, Heiko Koziolk, Max Kramer, and Klaus Krogmann. 2016. *Modeling and Simulating Software Architectures – The Palladio Approach*. MIT Press, Cambridge, MA.
- [46] Per Runeson and Martin Höst. 2008. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering* 14, 2 (dec 2008), 131–164. <https://doi.org/10.1007/s10664-008-9102-8>
- [47] Timur Sağlam, Moritz Brödel, Larissa Schmid, and Sebastian Hahner. 2024. Detecting Automatic Software Plagiarism via Token Sequence Normalization. In *Proceedings of the 46th International Conference on Software Engineering (Lisbon, Portugal) (ICSE '24)*. IEEE Press, 13 pages. <https://doi.org/10.1145/3597503.3639192>
- [48] Timur Sağlam, Sebastian Hahner, Larissa Schmid, and Erik Burger. 2023. *Supplementary Material for "Automated Detection of AI-Obfuscated Plagiarism in Modeling Assignments"*. Zenodo. <https://doi.org/10.5281/zenodo.10442139>
- [49] Timur Sağlam, Sebastian Hahner, Jan Willem Wittler, and Thomas Kühn. 2022. Token-Based Plagiarism Detection for Metamodels. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (Montreal, Quebec, Canada), Thomas Kühn 0001 and Vasco Sousa (Eds.)*. *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 138–141. <https://doi.org/10.1145/3550356.3556508>
- [50] Timur Sağlam, Larissa Schmid, Sebastian Hahner, and Erik Burger. 2023. How Students Plagiarize Modeling Assignments. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C) (Västerås, Sweden)*. *ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 98–101. <https://doi.org/10.1109/MODELS-C59198.2023.00032>
- [51] Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. 2003. Winnowing: Local Algorithms for Document Fingerprinting. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (San Diego, California), Alon Y. Halevy, Zachary G. Ives, and AnHai Doan (Eds.)*. *ACM SIGMOD Conference*, 76–85. <https://doi.org/10.1145/872757.872770>
- [52] G. Shobha, Ajay Rana, Vineet Kansal, and Sarvesh Tanwar. 2021. Code Clone Detection—A Systematic Review. In *Emerging Technologies in Data Mining and Information Security*. *Advances in Intelligent Systems and Computing*, 645–655. https://doi.org/10.1007/978-981-33-4367-2_61
- [53] G Shobha, Ajay Rana, Vineet Kansal, and Sarvesh Tanwar. 2021. Comparison between Code Clone Detection and Model Clone Detection. In *2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*. IEEE, 1–5. <https://doi.org/10.1109/icrito51393.2021.9596454>
- [54] Matthew Stephan and James R. Cordy. 2013. A Survey of Model Comparison Approaches and Applications. In *Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development - Volume 1: MODEL-SWARD, Slimane Hammoudi, Luís Ferreira Pires, Joaquim Filipe, and Rui César das Neves (Eds.)*. *International Conference on Model-Driven Engineering and Software Development* (2013), 265–277. <https://doi.org/10.5220/0004311102650277>
- [55] Daniel Strüber, Jennifer Plöger, and Vlad Acretoae. 2016. Clone Detection for Graph-Based Model Transformation Languages. In *Theory and Practice of Model Transformations, Pieter Van Gorp and Gregor Engels (Eds.)*. *International Conference on Model Transformation* 9765, 191–206. https://doi.org/10.1007/978-3-319-42064-6_13
- [56] Harald Störrle. 2013. Towards clone detection in UML domain models. *Software & Systems Modeling* 12, 2 (01 May 2013), 307–329. <https://doi.org/10.1007/s10270-011-0217-9>
- [57] Harald Störrle. 2015. *Effective and Efficient Model Clone Detection*. Vol. 8950. Springer International Publishing, Cham, 440–457. https://doi.org/10.1007/978-3-319-15545-6_25
- [58] Anna Sutton, David Taylor, and Carol Johnston. 2014. A model for exploring student understandings of plagiarism. *Journal of Further and Higher Education* 38, 1 (1 2014), 129–146. <https://doi.org/10.1080/0309877X.2012.706807arXiv:https://doi.org/10.1080/0309877X.2012.706807>
- [59] Mark van den Brand, Zvezdan Protić, and Tom Verhoeff. 2010. Generic Tool for Visualization of Model Differences. In *Proceedings of the 1st International Workshop on Model Comparison in Practice (Malaga, Spain), Davide Di Ruscio and Dimitris S. Kolovos (Eds.)*. *IWMCP '10*, 66–75. <https://doi.org/10.1145/1826147.1826160>
- [60] Michael Wise. 1993. String Similarity via Greedy String Tiling and Running Karp-Rabin Matching. *Unpublished Basser Department of Computer Science Report* (01 1993).
- [61] Michael J. Wise. 1995. Neweyes: a system for comparing biological sequences using the running Karp-Rabin Greedy String-Tiling algorithm. *Proc Int Conf Intell Syst Mol Biol* 3 (1995), 393–401.
- [62] Jan Willem Wittler, Timur Sağlam, and Thomas Kühn. 2023. Evaluating Model Differencing for the Consistency Preservation of State-based Views. *Journal of Object Technology* 22, 2 (July 2023), 2:1–14. <https://doi.org/10.5381/jot.2023.22.2.a4> The 19th European Conference on Modelling Foundations and Applications (ECMFA 2023).
- [63] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in Software Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg. I-XXIII, 1–236 pages. <https://doi.org/10.1007/978-3-642-29044-2>
- [64] Lisa Yan, Nick McKeown, Mehran Sahami, and Chris Piech. 2018. TMOSS: Using Intermediate Assignment Work to Understand Excessive Collaboration in Large Classes. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (Baltimore, Maryland, USA), Tiffany Barnes, Daniel D. Garcia, Elizabeth K. Hawthorne, and Manuel A. Pérez-Quinones (Eds.)*. *Technical Symposium on Computer Science Education*, 110–115. <https://doi.org/10.1145/3159450.3159490>
- [65] Manouchehr Zadahmad, Eugene Syriani, Omar Alam, Esther Guerra, and Juan de Lara. 2019. Domain-specific model differencing in visual concrete syntax. In *Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering (Athens, Greece), Oscar Nierstrasz, Jeff Gray, and Bruno C. D. S. Oliveira (Eds.)*. *SLE 2019*, 100–112. <https://doi.org/10.1145/3357766.3359537>