Multi-Location Software Model Completion

Alisa Welter
welter@uni-saarland.de
Saarland University
Saarbrücken, Germany

Christof Tinnes
christof.tinnes@siemens.com
Siemens AG
Munich, Germany

Sven Apel
apel@uni-saarland.de
Saarland University
Saarbrücken, Germany

ABSTRACT

In model-driven engineering and beyond, software models are key development artifacts. In practice, they often grow to substantial size and complexity, undergoing thousands of modifications over time due to evolution, refactoring, and maintenance. The rise of AI has sparked interest in how software modeling activities can be automated. Recently, LLM-based approaches for software model completion have been proposed, however, the state of the art supports only single-location model completion by predicting changes at a specific location. Going beyond, we aim to bridge the gap toward handling coordinated changes that span multiple locations across large, complex models. Specifically, we propose a novel global embedding-based next focus predictor, NextFocus, which is capable of multi-location model completion for the first time. The predictor consists of a neural network with an attention mechanism that is trained on historical software model evolution data. Starting from an existing change, it predicts further model elements to change, potentially spanning multiple parts of the model. We evaluate our approach on multi-location model changes that have actually been performed by developers in real-world projects. NextFocus achieves promising results for multi-location model completion, even when changes are heavily spread across the model. It achieves an average Precision@k score of 0.98 for $k \leq 10$, significantly outperforming the three baseline approaches.

ACM Reference Format:
Alisa Welter, Christof Tinnes, and Sven Apel. 2026. Multi-Location Software Model Completion . In . ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In model-driven engineering and beyond, software models help bridge the gap between the problem domain and the implementation domain by offering multiple levels and types of abstraction, thereby reducing overall system complexity [35]. In practice, for example, in industrial automation and automotive engineering, where a substantial fraction of code is generated from models, these software models can become very large and complex [81]. For example, a single subsystem may undergo thousands of individual modifications when transitioning from the main development branch to customized versions [81]. In general, changes tend to quickly grow and cut across the model [14, 49, 70, 80]. Even a *single, local change* may require complex adjustments in other parts to preserve or correctly extend the system's functionality and semantics. This makes maintaining and evolving software models tedious, time-consuming, and error-prone [80, 82].

To address these challenges, initial steps have been taken to automate software model evolution, powered by the rise of AI. One area of focus is *software model completion*, where a (partial) software model is provided and a tool suggests changes to the software model. Before the advent of large language models (LLMs), previous work often relied on predefined sets of model completion patterns and used (semi-) automated, rule-based techniques to recommend completions for software models [27, 47, 48, 50, 51, 54, 68]. However, this approach is limited, as each new project-specific pattern requires defining additional edit rules. Specifying edit rules typically demands expertise in both the specification and domain-specific languages, and their evolution over time – such as through metamodel changes – adds further complexity.

Advancements in AI have opened up new possibilities for software modeling [18, 20, 21, 33, 82]. Recently, LLMs from the GPT family have been used successfully for model completion [20, 21, 82]. In particular, the general inference capabilities of LLMs are useful for handling domain concepts with few or no similar examples, which is common in the modeling domain. They have been shown to be effective at dealing with verbose and noisy textual components found in domain-specific modeling data in industry, making them valuable in scenarios where other approaches fall short [82].

Despite considerable progress, existing LLM-based approaches are typically limited to *single-location changes*. That is, they modify, extend, or add one or more elements that are directly connected to each other at a single location in the software model [82]. In practice, however, a single local change may require adjustments in other parts of the model. In general, bug fixes and feature additions may affect many different locations [8, 75]. We call these changes *multi-location software model changes*. Multi-location changes are particularly challenging to manage, as dependencies across the model can be easily overlooked, a problem that is well understood in the realm of code [6, 32, 46]. Applying them correctly is often error-prone, time-consuming and requires substantial domain knowledge to understand what needs to be changed and where – especially given the sheer size of real-world software models.

Addressing this problem, we propose an approach for *multi-location model completion*, that implicitly learns multi-location co-change patterns from data. Given a single-location model edit (by the user), a global embedding-based next focus predictor, NextFocus, suggests further locations anywhere in the model to be edited as well, based on similar patterns observed in the data. Technically, NextFocus rests on a neural network with an attention

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

layer that, given historical pairs of co-changed nodes as training data, ranks them and suggests them to the user.

For evaluation, we investigate the performance of NextFocus for multi-location model completion on a real-world dataset containing 32 projects with multi-location changes that were actually performed by modelers in a real-world scenario. For this purpose, we rely on standard recommendation metrics, in particular, Precision@k. We found that NextFocus achieves an average score of 0.98 over all $k \in \{1, \ldots, 10\}$, significantly outperforming three baselines. Notably, NextFocus performs well even when changes are spread across a software model. A manual investigation revealed patterns that worked well and those that did not: we observed high predictive performance, especially in structured, frequently recurring patterns – such as changes involving the renaming or replacement of existing types, but also the introduction of entirely new domain concepts. On the other hand, NextFocus (and baselines) struggle with some cases, e.g., when the hierarchy of modeling elements was changed. Finally, we evaluate NextFocus in an iterative, multi-location completion setting by combining it with state of the art, LLM-based, single-location model completion [82].

In summary, we make the following contributions:

- We define the notion of multi-location model completion based on single-location model completion [82].
- We propose a global embedding-based next focus predictor for *multi-location model completion*, NextFocus, that predicts new change locations based on historical data.
- We systematically evaluate our approach on 32 real-world modeling projects and compare it against baselines that suggest changes (i) randomly, (ii) based on historical co-change frequency, and (iii) based on semantic similarity.
- We analyze factors contributing to NextFocus predictive performance, including project size, multi-location change pattern size, and dispersion, pattern characteristics and the effect of available historical data (cross-project setting).
- We evaluate NextFocus in an iterative, multi-location completion setting by combining it with single-location model completion [82] and compare it to single-location completion, performed N times for next focus node prediction.

The dataset as well as the source code for NextFocus, and the experiments are provided in our Supplement [86].

2 RELATED WORK

In this section, we provide an overview of existing work on model completion and the relation to other modeling activities.

2.1 Model Completion (with LLMs)

Previous work explores recommending model completions using pattern catalogs, where partial models are completed by identifying matching changes through pattern or graph matching and then applying the missing parts accordingly [27, 47, 48, 50, 51, 54, 68]. These approaches typically rely on domain-specific pattern catalogs that must be manually created and maintained. As a result, they are tied to a specific domain and modeling language, requiring new catalogs to be created for each domain-specific context, and they struggle with the verbose and noisy textual components found in software models. While rule-based approaches are explicitly defined and typically complete, this completeness can become a limitation when facing complex or underspecified scenarios, such as those encountered in model completion tasks. As a consequence, efforts moving beyond rigid rule-based systems have been made. While these are more generalizable to broader applications, they focus on single-location model completion.

Initial steps from a natural language perspective have been taken by Agt-Rickauer et al. [2, 3], who use conceptual knowledge bases and semantic networks built from natural language data to suggest entity names of model elements. López et al. [59] train a skip-gram model to generate word embeddings specific to the modeling domain. They evaluate performance on meta-model classification, clustering, and an entity name recommendation task. Elkamel et al. [34] recommend UML classes using clustering over existing model repositories, based on word-level similarities in names, attributes, and operations. Burgueño et al. [16] propose word embedding similarity to recommend domain concepts.

More recently, deep-learning models have been adapted to modeling tasks [29]. For example, Di Rocco et al. [30] use an encoder–decoder network to suggest element types to add in change-based persistence (CBP) models. As CBP is less common in practice [90], we focus on state-based modeling instead. Weyssow et al. [87] trained a transformer-based model from scratch to suggest metamodel concepts, however, the effectiveness of such approaches is constrained by the limited availability of modeling data [20]. ModelMate [26] is a recommender system designed for textual DSLs based on fine-tuned language models. The approach has been evaluated on a modeling task (predicting `EStructuralFeature` names in Ecore meta-models) and compared against existing recommender systems [16, 28, 87]. Liu et al. [56] propose an approach for predicting connections between modeling elements.

Chaaben et al. [20, 21] use the few-shot capabilities of GPT-3 to suggest new model class names, attributes, and associations by providing examples from unrelated domains. The approach does not scale well to larger models, as it requires multiple queries depending on the model size and includes all model concepts in each prompt. Tinnes et al. [82] concentrate on the neighborhood of the most recently changed element, thereby addressing prompt size limitations by restricting the scope of the LLM to a localized area. Their method was shown to outperform the approach by Chaaben et al. [21] on industrial, real-world data. In addition, they incorporate domain-specific context through similarity-based few-shot retrieval from the software model repository.

In general, while various approaches for model completion have been proposed [29], the focus has been on *single-location changes*, with little attention to patterns that span multiple locations, possibly cutting across the entire software model.

2.2 Supporting Other Modeling Activities

Another line of research also uses models but does not consider model completion. For example, a related area concerns ChatGPT's model generation capabilities, either from natural language descriptions [18], requirements [33], or images of UML class diagrams [25]. López et al. [60] introduce a framework for generating model queries from natural language by fine-tuning open-source LLMs on a synthetic dataset created with ChatGPT. Other approaches provide similar examples of models through collaborative filtering [28] and similarity-based filtering [31], but ultimately rely on users to apply the final model completion based on the examples [4]. In the same vein, there is work on change impact analysis and trace link generation between different models, model types, and corresponding requirements artifacts, documentation, and code [5, 9, 36, 63].

Finally, there is the research area of meta-model co-evolution, where changes to the meta-model must be propagated to models and model transformations to maintain consistency [24, 38]. These approaches aim to ensure correctness according to meta-model constraints and synchronization across modeling artifacts. Closely related to meta-model co-evolution is model repair [13, 62], which focuses on automatically correcting inconsistencies in software models. Most approaches rely on graph transformation rules or OCL-like constraints to automatically repair models [52, 66, 69, 70], while others organize fixes (additionally) into repair trees [64, 76]. From a machine learning perspective, some model repair approaches use reinforcement learning, where rewards are based on achieving consistency and improving model quality [11, 12, 40]. In model completion, an oracle for checking consistency, is not available. In contrast to model repair and meta-model co evolution, we additionally do not focus on meta-model conformance, but instead on maintaining and extending the semantic and functional aspects of software models during software evolution.

2.3 Code Completion and Repair

Challenges similar to multi-location model completion have been explored for source code. Many code-centered approaches enhance single-location code completion by incorporating repository-level context into LLM prompts via static analysis [57, 72, 74].

Regarding code co-changes and change impact analysis, considerable work has been done in recent years [39, 42, 53, 91]. For multi-location code completion, CodePlan [10] converts a repository-level task into a plan graph of LLM-driven edit obligations discovered through incremental dependency and change-impact analyzes. It applies edits, recomputes affected dependencies, and iteratively extends the plan until all obligations are discharged. The resulting repository is then checked by an oracle; any failures become new input for the next cycle. A related but distinct area focuses on LLM-based code repair [85, 88, 89], where models iteratively refine code using feedback from an oracle [89].

It is important to note that the approaches that work for code are not (easily) transferable to software models. Unlike source code, software models are mostly non-executable artifacts that combine graphical structures with verbose textual annotations. This makes oracle-driven processes infeasible because candidate correctness cannot be validated automatically (e.g., via tests). In general, the field also suffers from a lack of publicly available datasets, which significantly hinders comprehensive comparisons between different approaches [17, 58, 67, 82]. Unlike source code, software models lack standard languages, formats, and evaluation metrics [41], making benchmarking difficult. In contrast, code completion benefits from many benchmarks [73, 84] like HumanEval [22].

3 PRELIMINARIES

3.1 Software Model Completion

We represent software models as graphs to establish a common ground across different formats and types of software models, as is common in the literature [45, 61, 80, 82].

Definition 3.1 (Abstract syntax graph)**.** An abstract syntax graph G_m of a software model m is an attributed graph, typed over an attributed type graph *TG* given by metamodel *TM*.

An attributed type graph *TG* specifies the typing for abstract syntax graphs, ensuring that all elements conform to the structural and semantic constraints specified by the metamodel *TM*. For our purpose, we use a simplified representation of abstract syntax graphs as labeled directed graphs, where node and edge labels correspond to the textual names of their respective types and relations in the abstract syntax graph.

Definition 3.2 (Labeled directed graph)**.** A labeled directed graph G over a label alphabet L is defined as the tuple (V, E, l), where V is a finite set of nodes, $E \subseteq V \times V$ is the set of directed edges, and $l : V \cup E \longrightarrow L$ assigns labels to nodes and edges [82].

In a directed graph G, direct successors of a node $v \in V$ are all nodes that are *directly* reachable from v via an outgoing edge.

Definition 3.3 (Direct successor set)**.** The direct successor set of a node v is defined as:

$$\mathrm{succ}(v) = \{ u \in V \mid (v, u) \in E \}, \tag{1}$$

where $(v, u) \in E$ denotes a directed edge from v to u.

We further define the software model difference between successive software model versions.

Definition 3.4 (Structural model difference)**.** A *structural model difference* Δ_{mn} of model versions m and n is obtained by matching corresponding model elements in the model graphs G_m and G_n.

The structural model difference Δ_{mn} contains changed elements $\Delta^{\cup}_{mn} = ((V_n \cup E_n) \setminus (V_m \cup E_m)) \cup ((V_m \cup E_m) \setminus (V_n \cup E_n))$ and preserved elements $\Delta^{=}_{mn} = (V_m \cup E_m) \cap (V_n \cup E_n)$[1].

Next, we introduce local and multi-location model completions.

Definition 3.5 (Model completion)**.** A software model completion $\gamma_{(c,s)}$ transforms a given source model m, represented by G_m, into a (partial) target model n, represented by G_n [82]:

$$m \xrightarrow{\gamma_{(c,s)}} n, \tag{2}$$

such that γ corresponds to the model difference Δ_{mn}. Here, c denotes the number of elements involved in the completion change, that is, $c = \left|\Delta^{\cup}_{mn}\right|$ and s is the maximum shortest-path distance between any pair of involved elements.

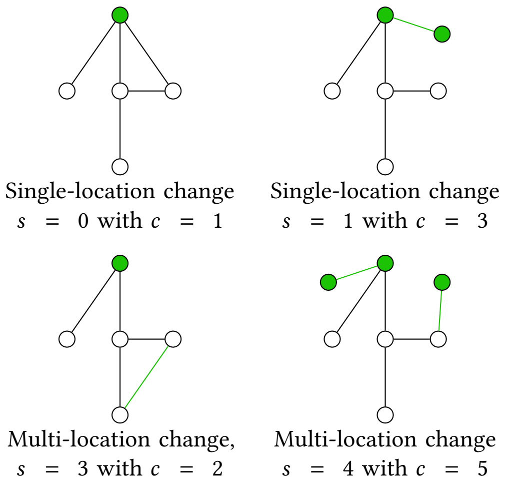
The parameter s gives an indication of how spread the involved elements of a software model completion pattern are across the model. A small s suggests that the change or pattern is locally confined, whereas a larger s implies that the completion affects distant parts of the model. Therefore, we can define single-location changes and multi-location changes as follows:

[1] For simplicity, we omit the explicit matching and assume that V_m and E_m are identified with their matched counterparts, where applicable.

Definition 3.6 (Single-location model completion)**.** A single-location software model completion is a model completion $\gamma_{(c,s)}$, where $s \leq 1$.

Definition 3.7 (Multi-location model completion)**.** A multi-location software model completion is a model completion $\gamma_{(c,s)}$, where $c > 1$ and $s > 1$.[2]

Examples for multi-location model completion and single-location model completion with different s and c are given in Figure 1.



Single-location change $s = 0$ with $c = 1$

Single-location change $s = 1$ with $c = 3$

Multi-location change, $s = 3$ with $c = 2$

Multi-location change $s = 4$ with $c = 5$

Figure 1: Examples of single-location and multi-location software model changes with different values of c and s, green element mark newly added elements

3.2 Machine Learning

Next, we outline the basic machine learning concepts required to understand our approach.

A *feedforward neural network* defines a mapping

$$y = f(x; \theta), \tag{3}$$

where x is the input and vector θ contains the learnable parameters. The parameters are learned by minimizing the difference between predicted and target values during training [37]. Neural networks are composed of layers, where each layer applies a transformation to its input before passing it to the next layer, forming a hierarchical representation of the data. In classification, they map an input x to an output category y. The learning process is guided by a loss function, which quantifies the error between the network's predictions and the actual target values [37].

An *embedding model* is a representation learning model that transforms natural language data or other structured data into a lower-dimensional continuous vector space and is defined as:

$$\phi : X \rightarrow \mathbb{R}^d, \tag{4}$$

where X is the input space (e.g., words, graph nodes, model elements), $\mathbb{R}^d$ is the d-dimensional vector space, and $\phi(x)$ is the embedding of x, capturing a subset of its properties in the lower-dimensional space.

4 APPROACH

This section is structured as follows: we first define the general concept of multi-location model completion, then provide an overview of the NextFocus's workflow, followed by detailed descriptions of the data preparation process and NextFocus's different phases.

4.1 Concepts

Since LLMs combined with retrieval-augmented generation have already demonstrated strong performance for model completion tasks, even on real-world industrial data [82], we decompose the problem of multi-location changes into an iterative approach between single-location model completion and finding the next focus nodes, as illustrated in Figure 2.

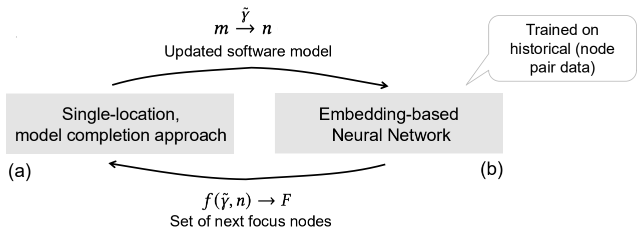
More specifically, a single-location, LLM-based model completion approach relying on an LLM (Figure 2, (a)) starts with a given software model m, a slicing criterion ς, and a set of relevant elements $C_m \subseteq V_m \cup E_m$[3]. A slicing criterion $\varsigma(C_m, m) \rightarrow C'_m$, is applied to extract the elements for the LLM context.[4] These elements serve as the context for the LLM that performs a single-location software model completion: $m \xrightarrow{\tilde{\gamma}} n$ (additional steps may be required depending on the approach [20, 21, 82]). As a next step, the global set of next focus nodes F needs to be predicted, enabling the overall approach to perform multi-location model completion (Figure 2, (b)), which is what we will explain next. First, we define the concept of a *focus function*.

Definition 4.1. Given a source model n and partial model completion $\tilde{\gamma} \subset \gamma_c$, with $c > 1$, the *focus function* is

$$f(\tilde{\gamma}, n) \rightarrow F \subseteq V_n \tag{5}$$

with the set of *focus nodes F* in V of n.

The *focus nodes* F are then given back again to the slicing criteria $\varsigma(C_n := F, G_n)$ for single-location model completion in a new iteration.



$m \xrightarrow{\tilde{\gamma}} n$
Updated software model
Trained on historical (node pair data)
Single-location, model completion approach
Embedding-based Neural Network
(a)
(b)
$f(\tilde{\gamma}, n) \rightarrow F$
Set of next focus nodes

Figure 2: Combined process of single-location model completion and next focus node prediction.

[2] Note that previous work [82] has focused on single-location model completion with $c \leq 2$ and $s \leq 1$; That is, at most one new node and one connection to an existing element are added.

[3] Tinnes et al. [82] use recently changed elements, where Chaaben et al. [21] use a small number of related classes(nodes)

[4] For example, Tinnes et al. [82] use so called simple change graphs as a slicing criteria, but other options are also possible [21]

4.2 Workflow

An overview of the workflow of NextFocus is given in Figure 3. NextFocus rests on a neural network that learns from historical data which elements tend to change together. In the first step, the training data is constructed (Figure 3, Step 1–4), including that each software model's nodes are embedded using an embedding model (Figure 3, Step 3). Node pairs are then passed through a neural network (Figure 3, Step 5) for training. For inference (Figure 3, Step 6), the software model's nodes are put though the embedding model and the neural network to evaluate their probability of changing together. Afterwards the probabilities are ranked, and the nodes with the highest scores are suggested as the next focus nodes (Figure 3, Step 7). In what follows, we describe the key phases in detail.

4.3 Data Preparation

Following Tinnes et al. [82], we employ a model matcher – specifically EMFCompare [15] – to obtain the *structural model differences* Δ_{mn} between each pair of consecutive models[5] (Figure 3, Step 1). These differences highlight the elements that have changed as well as those that have been preserved. Our approach operates on a graph-based representation of models and is thus not limited to Ecore [82]. The extraction can always be applied to a model difference (i.e., as long as model matching and differencing can be performed), and is therefore adaptable to a wide range of modeling tools. Often, model elements carry unique identifiers, making matching straightforward. We transform the Ecore models into graphs with `networkx`, iterating over the detected changes to add them as nodes and connect them via their corresponding edges. This yields a general, notation-independent representation. The resulting historical change data, which consists of sequential model versions is then prepared: the dataset is split into training, validation, and test sets, where the first is used as historical context for training, and the last is used for testing. Details on the specific splitting are provided in Section 5.

Given a model difference, we construct a set of node pairs and label those that have been modified in the same commit as positive examples (Label 1). These labeled pairs serve as ground truth for both neural network training and evaluation (Figure 3, Step 2). Unchanged node pairs are labeled as 0:

$$\lambda_{\Delta_{mn}}\big((v_1, v_2) \in V \times V\big) = \begin{cases} 1, & \text{iff both } v_1 \text{ and a direct} \\ & \text{successor of } v_2 \in \Delta_{mn}^{\circlearrowright} \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

For clarity, we refer to recently changed elements in $\tilde{y}$, which were, for example, suggested by a single-location approach, as *anchor nodes*. So, in Equation 6, v_1 is the anchor node.

4.4 Training Phase

Given the training set consisting of model differences Δ_{mn}, we first apply a pre-processing step to balance the number of data points per software model. Specifically, we ensure that each model contributes an equal amount of training data, preventing the network from being biased towards larger software models with more data points. Then we embed each $v \in V_{\Delta_{mn}}$ according to Equation 4 (Figure 3, Step 3). For this purpose, we have explored various embedding models, aiming to balance computational efficiency with the ability to capture essential differences in the data. After evaluating different options in a pilot study, we selected "text-embedding-3-small" with an embedding size of $e = 1536$ from the OpenAI family. Then, we input the embedded node representations pairs with their respective ground truth value into the neural network (Figure 3, Step 4–5). We use the Adam Optimizer for training.

Neural Network Architecture. Regarding the neural network architecture, we have explored linear and non-linear networks, but ultimately decided for an attention-based model [83] . The overall architecture of our neural network is shown in Figure 4. The embedded node representation pairs, given as the input, are first processed by a single self-attention layer, followed by mean pooling. A final linear layer maps the pooled representation to a single logit value per sample. At inference time, the logit value is passed through a sigmoid function to obtain a probability, while during training, the raw logit values are used directly with the loss function, which applies the sigmoid function internally.

Loss function. The neural network is trained using the binary cross-entropy loss (BCE), l_i, which combines a sigmoid layer and the BCE loss for improved numerical stability. Given a minibatch $\{(z_i, y_i)\}_{i=1}^{N}$, where $z_i \in \mathbb{R}$ is the raw model output, $y_i \in \{0, 1\}$ is the ground-truth label, and $\hat{y}_i = \frac{1}{1+e^{-z_i}}$ is the predicted probability.

$$l_i = \max(0, z_i) - z_i y_i + \log\left(1 + e^{-|z_i|}\right) \quad (7)$$

To address extreme class imbalance, we apply a focal loss correction on top of the BCE formulation [55]. This imbalance arises from the sheer number of negative examples (i.e., nodes that do not change together), which are often well-classified and would otherwise dominate the total loss. We also add the focal loss weight to the loss term, which reshapes the loss function to down-weight easy examples.

$$w_i = 1 - (\hat{y}_i \cdot y_i + (1 - \hat{y}_i) \cdot (1 - y_i))^{\beta}, \quad (8)$$

where β controls up-weighting of misclassified individual data points. As a result, false negative examples – which may have been assigned high probabilities and are harder to classify using the standard BCE loss – contribute more to the training process, effectively pushing them out of the set of predictions with the highest probabilities, which will be later important for ranking.

While w focuses on individual data points, we additionally apply a class-level balancing factor a.

$$a_i = \alpha \cdot y_i + (1 - \alpha) \cdot (1 - y_i) \quad (9)$$

Additionally to the focal loss terms introduced by Lin et al. [55], to optimize for our recommendation task, we add a misclassification penalty for false negatives.

$$m_i = (1 - y_i) \cdot \hat{y}_i \cdot \lambda + 1, \quad (10)$$

where λ is the penalty scaling factor for incorrect high-probability negatives. Combining these components, the final focal loss function for our task of predicting next focus nodes is:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \alpha_i \cdot w_i \cdot l_i \cdot m_i \quad (11)$$

[5] In what follows, we discarded about 0.02% of nodes due to their non-parsability.

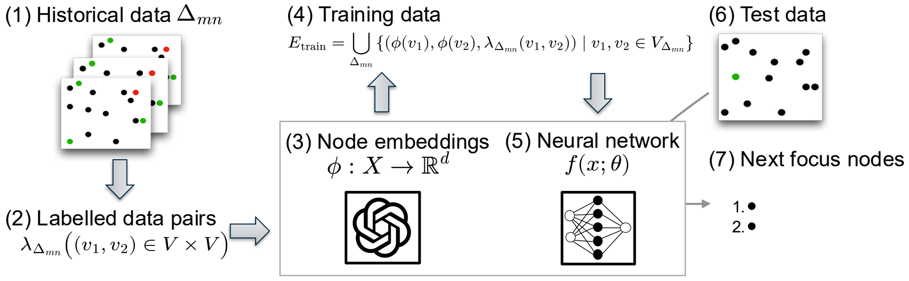

(1) Historical data Δ_{mn}
(2) Labelled data pairs
$\lambda_{\Delta_{mn}}\big((v_1, v_2) \in V \times V\big)$
(4) Training data
$E_{\text{train}} = \bigcup_{\Delta_{mn}} \{(\phi(v_1), \phi(v_2), \lambda_{\Delta_{mn}}(v_1, v_2)) \mid v_1, v_2 \in V_{\Delta_{mn}}\}$
(3) Node embeddings
$\phi : X \to \mathbb{R}^d$
(5) Neural network
$f(x; \theta)$
(6) Test data
(7) Next focus nodes
1.
2.

Figure 3: Overview of the global embedding-based next focus predictor (NextFocus) approach.

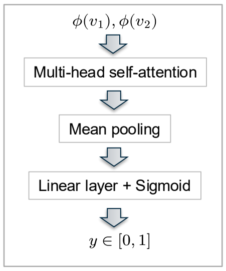


Figure 4: Neural Network architecture of NextFocus.

4.5 Inference Phase

Given a model represented by a labeled directed graph G and a partial model completion $\tilde{\gamma}$, for example, obtained from a single-location model completion approach (Figure 3, Step 6), NextFocus suggests the next *focus nodes* in the inference phase (Figure 3, Step 7). The anchor node $v_1 \in \tilde{\gamma}$ that has been changed is embedded, and NextFocus computes the probability of each other node $v \in V$, where $v \neq v_1$, changing together with v_1. According to Equation 6, a change is expected to occur at a (direct or indirect) successor of v, either through addition, deletion, or modification, which then can be suggested by a single-location model completion approach.

The node pairs (v_1, v) with $v \in V$ are passed to our trained neural network, which computes the probability $f(\phi(v_1), \phi(v))$ based on the historical evolution of the current software model and patterns learned from other models.

Finally, we rank all nodes $v \neq v_1$ based on the probability of changing together with v_1 and suggest the top-k candidates as the next focus nodes, which can then be presented to the user or be fed into the next iteration (Figure 2, Step (a)).

5 EVALUATION

We empirically evaluated NextFocus using historical real-world modeling data to assess its ability for multi-location model completion. Working with historical data allows us to separate the technical capabilities of our approach from other factors introduced by tools, such as the optimal number of recommendations shown, the layout and positioning of modeling elements, or tool-specific evaluation metrics. This facilitates a reproducible and comparable assessment of the capabilities of our approach.

In what follows, we outline our research questions, describe the evaluation setup and data used, and present our results.

5.1 Research Questions

We are interested in whether our NextFocus, given a change that has been applied (i.e., an anchor node), can effectively predict the next focus node(s) in a multi-location model completion task. Specifically, we examine whether a model trained on historical multi-location changes is able to generalize to new, unseen changes.

> **RQ 1:** *To what extent can NextFocus predict new focus nodes for multi-location software model completion?*

To better understand the NextFocus predictive performance, we investigate how its performance varies with the distance between the predicted focus node(s) and the originally changed (anchor) node, that is, how the performance of the model completion $\gamma_{(c,s)}$ depends on s. In particular, we examine whether the model is better at predicting single-location changes (close in terms of graph distance) or also performs well on more global changes.

> **RQ 2:** *How does the model's predictive performance of new focus nodes depend on the distance (in terms of graph radius) to the anchor node?*

We also investigate the conditions under which our NextFocus performs well and identify scenarios in which its predictive performance could be improved. Specifically, we examine which project specific properties influence the model's ability to correctly identify new focus nodes. These properties include, for example, the overall project size (i.e., the number of training data points), the proportion of positive instances (i.e., data points with a ground truth of one), and the kind and content of the change patterns. On the other hand, we study the performance of NextFocus in a cross-project setting, that is, whether NextFocus generalizes to previously unseen projects by transferring known project-specific characteristics. This aspect becomes particularly relevant in real-world scenarios, if no historical data are available for a given project.

> **RQ 3:** *Which project-specific or pattern-specific properties influence the predictive performance of our model and consequently how well does NextFocus perform in a cross-project setting?*

Finally, we investigate the performance of NextFocus in a complete multi-location model completion setting (not only next focus node prediction) that is obtained by iteratively combining it with a single-location model completion approach, as shown in Figure 2.

> **RQ 4:** *How effectively does NextFocus support iterative, multi-location model completion?*

5.2 Experiment Setup

We conducted six experiments to address the four research questions; Experiments 3 ,4 and 5 contribute to answering RQ3.

Data. For all experiments, we use a publicly available, real-world dataset, RepairVision [70, 71], which contains versioned modeling projects. This is essential for our study, as it provides us with ground-truth information on multi-location changes that have been *actually* performed by modelers in a real-world scenario[6].

In total, the data set contains 41 modeling projects, with 912 commits. On average, the models contain 1285.9 nodes, and there are 168.9 changes per commit. For our evaluation, we applied an additional filtering step (e.g., because we required projects to have, at least, three commits to allow for a valid train/validation/test split) resulting in 32 projects considered in total. Detailed information on each project and filtering is provided in our Supplement [86]. We use EMFCompare's model matching capabilities to compute structural model differences for all modeling projects.

Experiment 1. To answer RQ 1, we split the modeling dataset into training, validation, and test sets while respecting the historical timeline. More specifically, given the historically ordered structural model differences $\{\Delta_{m_1 m_2}, \Delta_{m_2 m_3}, \ldots, \Delta_{m_{n-1} m_n}\}$, where n is the number of structural model differences in a project, we split by commit, i.e., by structural model difference, to prevent data leakage between sets. We define the training set as $\{\Delta_{m_1 m_2}, \ldots, \Delta_{m_{n-3} m_{n-2}}\}$, the validation set as $\{\Delta_{m_{n-2} m_{n-1}}\}$, and the test set as $\{\Delta_{m_{n-1} m_n}\}$. Overall, this leads to a ratio of 71.88% train, 16.41% validation, and 11.70% test data points in the respective sets.[7] Using commit time for splitting, rather than random sampling, mirrors a real deployment: We train on what is known, the commit history, and expect the network to generalize to new, unknown software models in the test set.

We begin by preprocessing the data (see Section 4) and training the neural network on the training set while tuning hyperparameters on the validation set. During training, we explicitly over-sampled or under-sampled data points from each project to a fixed size, ensuring that the neural network treats each project equally rather than being biased toward larger datasets.

We tuned all hyperparameters using Bayesian optimization, more information is given in our Supplement [86]. The task is framed as a node-ranking problem: Given a recently changed (anchor) node, the model ranks other nodes based on the probability of changing with this anchor node.

For evaluation purposes, we take models from the test set, which include the latest changes in the modeling history $\{\Delta_{m_{n-1} m_n}\}$, specifically the transition from the second-to-last to the last model snapshot. Given a recently changed element in $\Delta^{\circlearrowright}_{m_{n-1} m_n}$, the anchor node, and the set of already existing elements $\Delta^{=}_{m_{n-1} m_n}$, we predict the next focus node(s), that is, the element whose successor is expected to be changed next (see Equation 6). That is, for evaluation, we remove the ground truth elements $\Delta^{\circlearrowright}_{m_{n-1} m_n}$ (changes that have been made by the modeler in a real-world scenario) from $\Delta_{m_{n-1} m_n}$ and investigate whether NextFocus is able to predict these correctly. We are particularly interested in the overall predictive performance of our NextFocus. Neural network performance is commonly evaluated using Precision@k on the test set [19, 65, 77, 79][8].

A prediction is considered correct if the suggested node(s) were indeed modified in the corresponding commit in the dataset.

Let $y_i \in \{0, 1\}$ be the binary ground-truth label for node i, where 1 indicates that i changed, we define Precision@k as the number of true positives among the top-k predictions, normalized by the minimum of k and the number of actual positives:

$$\text{Precision@}k = \frac{\#\text{true positives in top-}k}{\min(k,\ \#\text{actual positives})} \tag{12}$$

With regard to k, the number of recommendations, prior work consistently suggests keeping recommendation lists short and manageable for human users. Therefore, we limit k to a maximum of 10, but we report results for various values of $k \leq 10$, as well [1, 23, 51].

We compare NextFocus against three baselines: (i) *random selection* of focus nodes, (ii) *semantic similarity* based on pre-trained embeddings, and (iii) *historical co-change frequency*, which prioritizes nodes that have frequently changed together in the past. We selected these baselines to reflect fundamentally different strategies for focus node prediction: (i) *random selection* serves as a naive lower bound, illustrating how well the other approaches perform compared to uninformed guessing; (ii) *semantic similarity* builds on the assumption that semantically related elements tend to co-change, a concept also used in related work [2, 3, 16, 34, 44, 45, 59] , and (iii) *historical co-change frequency* builds on the assumption that elements which changed together in the past are likely to do so again [43, 44, 92]. Together, these baselines cover a broad range of factors that can influence performance.

For the *semantic similarity* baseline, we use the same embedding model as the one described in Section 4. Given the anchor node, we compute the cosine similarity between its embedding and those of all other nodes in the software model. The top-k most similar nodes are then recommended. While there is currently no multi-location model completion approach available that we could adopt as a baseline, we use *semantic similarity* as a reference point due to its significance in related domains. For instance, text-based similarity has been applied for change impact analysis [44] on source code, and in the UML model domain [45]. Prior efforts for single-location model completion also [2, 3, 16, 34, 59] focused on similarity.

For the *historical co-change frequency* baseline, we construct a co-change matrix that records how often each pair of nodes has changed together in past commits. During inference, we identify the top-k nodes with the highest co-change frequency with respect to the given anchor node and recommend those. We are interested in the overall performance, so we examine the overall distribution of Precision@k values. Historical co-change frequency has been frequently used on source code [43, 44, 92].

[6] Other datasets such as the ModelSet [58] contain only static snapshots, which would require synthetically constructing modeling histories. This does not reflect a real-world scenario and introduces confounding assumptions.

[7] To ensure realistic evaluation, we approximate a commonly used data ratio for train–validation–test splits (around 70–80% train, 10–15% validation/test). Since each structural model difference can contain a highly variable number of data points (node pairs, see Equation 6), especially, in later commits, where models tend to be larger, we had to restrict the number of structural model differences in the validation and test set. Otherwise, those sets would have ended up with more data points than the training set, despite covering fewer commits. On the other hand, the neural network is trained on individual data points rather than entire commits, which leads to the specific dataset split proportions used.

[8] We do not report recall, as the number of relevant items varies significantly across cases – from over 1000 to as few as 1–2, making recall highly sensitive to the denominator and thus difficult to interpret. Instead, we focus on top-k precision, which better aligns with our recommender system setting. The goal is to recommend the most likely next changes first – not to recover all possible changes. We additionally include a random baseline for comparison. Including a baseline that selects candidates randomly provides a meaningful lower bound and allows for relative performance assessment without relying on absolute metrics like recall.

Experiment 2. To investigate how NextFocus performs on multi-location change patterns of varying size, we limit the predicted focus nodes to a certain radius. That is, we only consider $\gamma_{(c,s)}$ with $s < \tau$, where s is the maximum shortest-path distance between any pair of involved elements in the multi-location change (Definition 3.5) and τ is a radius threshold. This setup allows us to analyze whether the model performs better on localized changes. By increasing τ, we study whether NextFocus maintains high Precision@k even as changes become more spread across the model. We train the neural network using the same setup as in Experiment 1.

Experiment 3. We investigate how specific project properties influence the overall performance of NextFocus. As a first step, we examine NextFocus's average performance across individual modeling projects. We also analyze the influence of the overall training set size per project and the number of positive examples included in each project's training set. While neural networks typically benefit from more data points seen during training, we aim to understand whether this correlates with higher average performance per project. Note that we explicitly over-sample or under-sample during training to normalize the number of data points per project. This ensures that NextFocus treats each project equally and avoids biasing towards datasets with more training examples. The neural network is trained as in Experiment 1.

Experiment 4. To answer RQ3, we manually analyze the graphs in our test set to examine which change patterns work well and which do not. We additionally summarize the change, determine whether the single-location changes truly belong together or occurred by coincidence, and identify the overall pattern. For additional support, we consulted OpenAI's GPT model (o3).

Experiment 5. Finally, to address the last aspect of RQ3, which is, whether NextFocus generalizes to a cross-project setting, we split the training and test sets by project rather than by historical data within a single project. The setup follows the idea of 10-fold cross-validation, where in each fold one project is used for testing and the remaining projects are used for training. Further details of the setup are given in our Supplement [86]. This setup allows us to evaluate how well the model, trained on certain projects, generalizes to previously unseen projects. We additionally compare it against a setup where historical data is available (Experiment 3).

Experiment 6. To answer RQ4, we first evaluate the performance gain of NextFocus by comparing it against a state of the art, single-location model completion approach. Specifically we compare NextFocus to the approach by Tinnes et al., called RaMc [82].

We assess how RaMc performs when used iteratively for *next focus node prediction* on its own. For baselining, we use the same LLM, slicing procedure, prompt structure, linearization format, which represents the graph solely by edges, and the same database for few-shot retrieval from Tinnes et al. [82]. The only, but crucial difference lies in the procedure of next focus node prediction, which is either done by NextFocus (our approach) or indirectly by the LLM itself given via the source node of the suggested edge (approach of Tinnes et al. [82]). We perform the model completion for each data point, N times, set $N = 10$, and report the results for next focus node prediction. For NextFocus, we fix $k = 1$ to make the approaches comparable in this iterative scenario. Exact details on the methods are provided in our Supplement [86].

Second, we combine NextFocus with the single-location model completion, RaMc [82] and further improve RaMc for the multi-location setting, which we call RaMc′. This step is necessary because RaMc is not designed for iterative, multi-location completion. In particular, its edge-only linearization restricts the ability to represent jumps to new or isolated nodes in the model. As a result, when the next focus node is not yet structurally connected to previously completed regions, RaMc cannot represent this new node, leading to a drop in performance in multi-location settings.

The *multi-location model completion* is performed iteratively, interleaving the prediction of the next focus node (NextFocus), performing local model completion (RaMc or RaMc′), and updating the model under construction based on the completion result (see Figure 2). The updated model is then passed to the next iteration. We call the settings NextFocus+RaMc and NextFocus+RaMc′ and compare it against RaMc. RaMc′ is directly based on RaMc, but replaces the slicing procedure with radius-based slicing and switches the graph serialization from EdgeL to JSON. The prompt is slightly adapted accordingly, and we use a newer chat-based LLM version (no major performance impact and better forward comparability; see our Supplement [86]). The underlying database remains unchanged. As evaluation metrics, we both report the correctness of the next focus node and the resulting single-location model completion. We distinguish three levels of correctness for the model completion according to Tinnes et al. [82]: *format correctness*, *structural correctness* (valid graph structure and connections), *change structure correctness* (correct change types such as add, modify, or remove), and *type structure correctness* (exact type and change type). Exact details are on our Supplement [86].

5.3 Results

Experiment 1. We first focus on the overall Precision@k of NextFocus for multi-location software model completion, comparing it to the *random selection*, *historical co-change frequency*, and *semantic similarity* baselines.

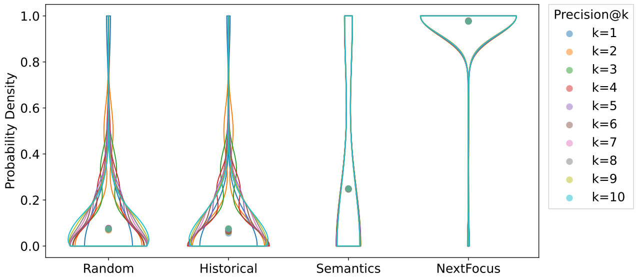


Figure 5: Precision@k distribution of *semantic similarity*, *historical co-change frequency*, *random selection*, and NextFocus.

Figure 5 presents a comparison of all approaches for values of $k \leq 10$. We calculate the overall mean of the precision@k values for each approach by averaging across all $k \in \{1, \ldots, 10\}$. Overall, NextFocus performs best, achieving an average of 0.98. It is followed by the *semantic similarity* baseline (0.25), the *historical co-change frequency* baseline (0.07), and the *random selection* baseline

(0.07). We conducted one-sided Mann-Whitney U tests to assess statistical significance. NextFocus significantly outperformed all baselines at every $k \in \{1, \ldots, 10\}$ ($p < 0.01$). Among the baselines, *semantic similarity* consistently outperformed both *historical co-change frequency* and *random selection* across all k ($p < 0.01$). Exact p-values are provided in our Supplement [86].

Summary Experiment 1: *NextFocus significantly outperforms all baselines in terms of Precision@k ($k \leq 10$), with the highest average precision of 0.98.*

Experiment 2. In Figure 6, we show the performance of NextFocus depending on the considered radius. We limit the radius to the maximum values observed. Some of our graphs are disconnected, hence the value infinity for the distance ($s = \infty$). We observe a generally negative monotonic relationship between radius and Precision@k, with a Spearman's correlation coefficient ρ ranging from −0.19 (Precision@3) to −0.14 (Precision@1). This indicates that absolute Precision@k slightly decreases with increasing radius.

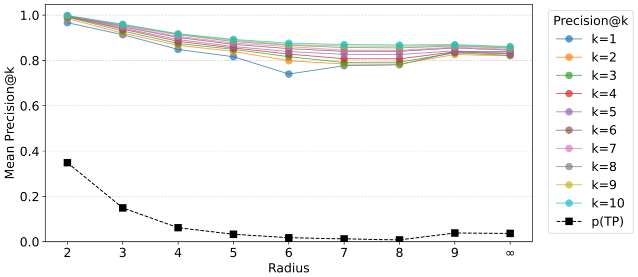


Figure 6: NextFocus's performance with regard to the maximum radius considered

We additionally plotted Precision@k for random guessing (Figure 6, p(TP)). For Precision@k, randomly selecting items yields an expectation equal to the overall prevalence of positives, independently of k. Since more nodes become candidates with increasing radius, making it harder for the model to identify relevant nodes, we additionally examined performance relative to random selection. NextFocus's performance improves relative to the *random selection* baseline, as indicated by a positive monotonic relationship between radius and the ratio of Precision@k to the prevalence of positives. Spearman's correlation coefficients for this ratio range from $\rho = 0.465$ (Precision@1) to $\rho = 0.625$ (Precision@10) with $p < 0.01$. Using the additive margin over chance (Precision@k − p(TP)), we again observe a positive monotonic relationship with the radius, Spearman's ρ ranges from 0.424 (Precision@2) to 0.515 (Precision@10) (all $p < 0.01$).

Summary Experiment 2: *We observe a slight negative monotonic relationship between maximum radius of the multi-location model completion and absolute Precision@k, but a positive monotonic trend for the ratio of Precision@k to positive prevalence.*

Experiment 3. We examine NextFocus 's average performance across individual modeling projects, as shown in Figure 7, which depicts the distribution of the predictive performance. While the overall performance remains higher for the NextFocus (0.58) than for the baselines, individual project outcomes vary, with some projects performing notably better. A Kruskal–Wallis test confirms that these differences are statistically significant across all k ($p < 0.01$).

We are particularly interested how the overall training set size and the number of positive examples in the training influence model performance. To visualize overall trends, we plot the relationship between dataset train size and the number of ground truth label equal to true and NextFocus's average project Precision@k over all $k \leq 10$, fitting a separate linear regression (Figure 8, 9). The green dots indicate the average predictive performance per project. To ensure a fair comparison across projects with varying candidate set sizes, we choose k dynamically as a small fraction of the total candidate count (e.g., $k = \lceil 0.01 \cdot \text{candidates} \rceil$). We find no statistically significant monotonic relationship between dataset train size and average project Precision@k for all approaches (Figure 8).

Analyzing the correlation between the number of positive examples and performance (Figure 9), we find no significant trend for the *historical co-change frequency*, NextFocus, and *random selection* ($p > 0.05$). Only *semantic similarity* shows a statistically significant weak positive correlation ($\rho = 0.35$, $p = 0.040$) [78].

Summary Experiment 3: *Performance significantly varies between different projects, but NextFocus still consistently outperforms the baselines across projects. No strong correlation is found between train dataset size or ground truth label count in the train dataset.*

Experiment 4. In contrast to the baselines, NextFocus showed particularly strong performance in several scenarios, especially where common patterns were applied. Notable cases of high predictive performance included changes that involved renaming or replacing existing types, as well as identifier updates. One example of such a case was replacing the enum `ValidationSetType` with a new one `AggregationType`.

Some software model extensions also yielded strong results, examples include additions of entirely new modeling concepts, such as the introduction of an `IfStatement` element to support if-then-else branching. In total, there were three projects in which all approaches performed well due to the high probability of selecting the correct target among the candidates. Examples with low precision (below 0.2) include changes in the modeling hierarchy. NextFocus did not perform well on all changes that introduced entirely new modeling concepts and performed worse on changes that shifted

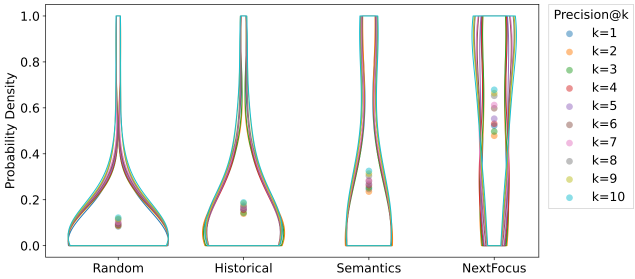


Figure 7: Distribution of average Precision@k per project for the *semantic similarity, historical co-change frequency, random selection* and NextFocus approach.

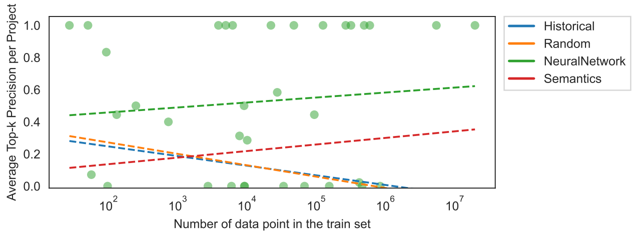


Figure 8: Average project Precision@k over k on the test set compared to the total number of training data points per project.

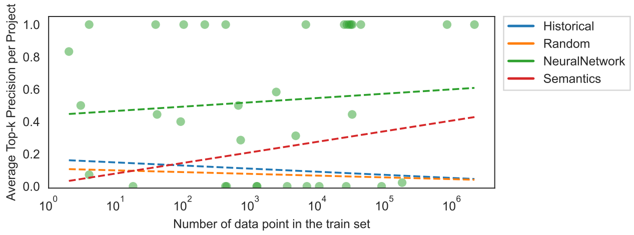


Figure 9: Average project Precision@k over k on the test set compared to the number of training positives (label=true) per project.

the underlying meaning of elements, such as adding new behavioral constructs or loosening attribute constraints, for example, the introduction of a `Trigger` concept for event-action logic and the removal of the uniqueness constraint from string-valued attributes.

Summary Experiment 4: *NextFocus excelles in scenarios with recurring patterns but also performed well on some model extensions. It is less effective for hierarchy-related changes. NextFocus outperformed the baseline approaches in almost all situations.*

Experiment 5. Next, we analyse NextFocus 's average performance in a cross-project setting. Figure 10 shows the distribution of predictive performance on test projects not seen during training. Not unexpectedly, the performance decreases compared to the intra-project setting – where training is performed on historical data of the projects – from 0.58 to 0.36, on average (with Precision@k ranging from 0.43 at $k = 10$ to 0.30 at $k = 3$).

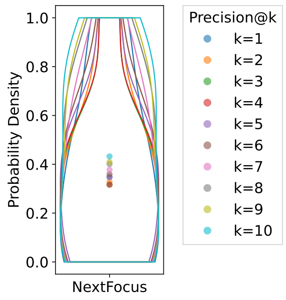


Figure 10: Distribution of average Precision@k per project in the cross-project setting.

Experiment 6. We compare NextFocus against a single-location, state of the art model completion approach, RaMc.

We report in Table 1 correctness across projects, averaged per project, following the same procedure as in Experiment 3 and 5, since results are paired across approaches. Notably, NextFocus achieves significant higher correctness for *next focus prediction* (63.94%) than RaMc (30.18%) (Wilcoxon signed-rank test, $p < 0.05$).

To isolate the contribution of NextFocus, we compare NextFocus+RaMc and RaMc: no significant difference is observed for *change structure correctness*, and *type structure correctness* ($p > 0.05$) but NextFocus already achieves significantly higher *structure correctness* and *format correctness* ($p < 0.05$). As outlined in the experimental setup (Section 5.2), RaMc was not originally designed for iterative multi-location completion, which limits its performance in NextFocus+RaMc.

As a result, the improved multi-location model completion approach, NextFocus+RaMc′, significantly outperforms the baseline in *next focus prediction*, *change structure*, and *structure correctness*.

Table 1: Average correctness across projects for next focus node prediction and single-location model completion (in %).

Approach	Next Focus	Format	Structure	Change Structure	Type Structure
RaMc	30.18	95.63	16.35	13.98	12.19
NextFocus+RaMc	63.94	99.67	22.85	10.48	8.48
NextFocus+RaMc′	60.33	96.12	40.68	21.31	16.11

5.4 Discussion

In a large, real-world software model, a local change can affect other (distant) parts of the model, even if it is well-structured [7, 8, 75]. To support modelers in finding the relevant locations to change, we propose NextFocus for multi-location model completion. NextFocus learns co-change patterns from historical data and suggests additional model locations to change.

RQ1. Our initial objective was to assess to what extent NextFocus can predict focus nodes for multi-location software model completion; to this end, we trained, evaluated, and compared NextFocus against three baselines, *semantic similarity*, *historical co-change frequency*, *random selection*. We found that NextFocus consistently outperformed the baselines, achieving an average score of 0.98 over all $k \leq 10$, and performed well independently of the number of recommendations. NextFocus successfully learns patterns from history, outperforming *historical co-change frequency* by better capturing contextual semantics. While *historical co-change frequency* alone is insufficient – since the same type of change can occur in other elements of the same or different models – semantic embeddings help identify such cases.

At the same time, learning from historical data proves to be effective, as witnessed by NextFocus superior performance compared to static *semantic similarity* alone. By including a random baseline, we verified that the performance is not due to chance. Given the different pattern characteristics – some involving a large number of changes, others only very few – NextFocus consistently predicted relevant nodes, even when only a small number of correct nodes needed to be identified from a large candidate set. Our approach reliably suggests relevant new focus nodes, matching patterns that were actually made by modelers in real-world settings.

RQ2. We investigated to what extent NextFocus's ability to predict new focus nodes depends on the distance between the predicted nodes and the anchor node, examining whether NextFocus can predict both local and global next focus nodes. We found that, while absolute performance slightly decreases with an increasing radius, this trend was to be expected, as more nodes become candidates and the probability of a node being a correct change node decreases. This illustrates how the task becomes harder for NextFocus in distinguishing relevant from irrelevant nodes. Nevertheless, NextFocus keeps predictive performance high, even at longer graph distances. The strong performance, independent of the distance, may be due to the model not relying on graph connections but instead focusing on semantic embeddings and historical co-changes.

RQ3. We were interested in the project-specific and pattern-specific properties that influence the performance of NextFocus. While machine learning performance often depends on factors like training set size and label distribution, we observed no correlation with dataset size – some large datasets performed poorly, and some small datasets yielded perfect predictions (Figure 8, green dots). This suggests that even small projects with a few examples may have benefited from other projects. The slight performance increase for *semantic similarity* may result from datasets with more positives in training also having more in testing, which raises the chance of correct focus nodes with similar semantic embeddings, despite the lack of training. Overall, performance varied more with the nature of the change pattern: NextFocus performs well on recurring patterns such as type replacements, likely because they appear frequently in training and exhibit clear semantic cues. However, NextFocus predictive performance was lower for uncommon patterns and hierarchy-related changes, though it still generally surpasses the baselines. This may be due to the fact that such cases require an understanding of deeper structural context than NextFocus provides. Applying NextFocus on projects that were unseen during training, accounting for cases where historical data may not always be available in real-world scenarios, we observe a drop in performance, which indicates that project-specific historical data indeed helps in making more accurate predictions. Nevertheless, NextFocus is still able to leverage information from other projects to make reasonable predictions on unseen data (e.g. 0.43 Precision@10).

RQ4. To assess how well NextFocus supports iterative, multi-location model completion in a realistic workflow, we first compare NextFocus' capabilities for next focus node prediction against state of the art single-location completion, which is iteratively executed. Notably, NextFocus achieves 63.94% correctness compared to 30.18% for RaMc (see Table 1). We additionally report correctness at multiple levels following the metrics by Tinnes et al. [82]. As RaMc does not support iteratively completing the model N times, a drop in model completion correctness occurs compared to the values stated in the work by Tinnes et al. [82]. More specifically, RaMc, does not support jumping to new focus nodes in the model due to the edge-only linearization (EdgeL format). The improved version, NextFocus+RaMc′, however, archives higher correctness, even in an iterative setting with $N = 10$ (e.g. 40.68% *structure correctness*).

5.5 Threats to Validity

With regard to internal validity, our evaluation relies on noisy historical commit data, performed by modelers in real-world scenarios, which, in some cases, may include unrelated or tool-generated changes from EMF; for example, our manual analysis revealed three commits for which it was unclear whether the multi-location changes were conceptually related or simply co-occurred in the same commit by coincidence. Additionally, the overall differences between modeling projects sizes can influence the overall performance, since bigger projects may dominate the performance. This is exactly why we conducted Experiment 3, which confirmed that NextFocus consistently outperforms the baselines across projects.

With regard to external validity, we cannot claim transferability to all domains. However, the inclusion of 32 real-world, diverse, open-source modeling projects, with multi-location changes that have been performed by modelers in the real-world, provides strong evidence for the generalizability of our findings. The lack of further publicly available datasets currently hinders the extension of our evaluation on more data [17, 58, 67, 82]. Due to the inherent structure of commit histories (see Section 5.2) and the need for manual semantic analysis, we limited the evaluation to one multi-location pattern per project – specifically, the most recent one. While this restriction was necessary for manual investigation, future work shall explore earlier versions of the model histories by shifting the train-test split toward older commits. For multi-location completion, we partially reimplemented the approach by Tinnes et al. [82] and acknowledge possible minor deviations from the original.

6 CONCLUSION AND FUTURE WORK

Software models often grow large and complex, undergoing thousands of changes through evolution, refactoring, and maintenance. With the rise of LLMs, new possibilities have opened up in the software modeling domain. While recent approaches support single-location model completion, we aim to extend this setting to multi-location model completion by proposing NextFocus. It consists of a node embedding mechanism, an attention-based neural network, and a ranking system. NextFocus achieves promising results for multi-location model completion, even when changes are largely spread across the model. NextFocus significantly outperforms the baselines: *random selection*, *semantic similarity*, *historical co-change frequency*, which reflect concepts common in similar domains [2, 3, 16, 34, 43–45, 59, 92]. NextFocus excelled in scenarios with recurring change patterns and also performed well on some model extensions. However, its performance was lower for less common patterns and hierarchy-related changes. NextFocus benefits from project-specific historical data; however, if such data is not available, it can still make use of information from other projects to perform reasonably in cross-project settings. Finally, combining NextFocus with single-location completion enables effective iterative, multi-location model completion, achieving 63.94% next focus node correctness.

7 DATA AVAILABILITY

We provide the data and Python code for NextFocus as well the baselines in our Supplement [86], including training and evaluation scripts to reproduce our analysis.

REFERENCES

[1] Bhisma Adhikari, Eric J Rapos, and Matthew Stephan. 2024. SimIMA: a virtual Simulink intelligent modeling assistant: Simulink intelligent modeling assistance through machine learning and model clones. *Software and Systems Modeling* 23, 1 (2024), 29–56.

[2] Henning Agt-Rickauer, Ralf-Detlef Kutsche, and Harald Sack. 2018. DoMoRe–a recommender system for domain modeling. In *Proceedings of the International Conference on Model-Driven Engineering and Software Development*, Vol. 1. Setúbal: SciTePress, 71–82.

[3] Henning Agt-Rickauer, Ralf-Detlef Kutsche, and Harald Sack. 2019. Automated recommendation of related model elements for domain models. In *Model-Driven Engineering and Software Development: 6th International Conference, MODELSWARD 2018, Funchal, Madeira, Portugal, January 22-24, 2018, Revised Selected Papers 6*. Springer, 134–158.

[4] Lissette Almonte, Esther Guerra, Iván Cantador, and Juan de Lara. 2024. Engineering recommender systems for modelling languages: concept, tool and evaluation. *Empirical Software Engineering* 29, 4 (2024), 102.

[5] Sajid Anwer, Lian Wen, Shaoyang Zhang, Zhe Wang, and Yong Sun. 2024. BECIA: a behaviour engineering-based approach for change impact analysis. *International Journal of Information Technology* 16, 1 (2024), 159–168.

[6] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-oriented software product lines*. Springer.

[7] Sven Apel and DeLesley Hutchins. 2010. A Calculus for Uniform Feature Composition. *ACM Transactions on Programming Languages and Systems* 32, 5 (2010), 19.

[8] Sven Apel, Christian Kästner, and Christian Lengauer. 2011. Language-independent and automated software composition: The FeatureHouse experience. *IEEE Transactions on Software Engineering* 39, 1 (2011), 63–79.

[9] Afef Awadid and Rémi Boyer. 2023. Supporting Change Impact Analysis in System Architecture Design: Towards a Domain-Specific Modeling Method. In *11th International Conference on Model-Based Software and Systems Engineering (MODELSWARD)*.

[10] Ramakrishna Bairi, Atharv Sonwane, Aditya Kanade, Arun Iyer, Suresh Parthasarathy, Sriram Rajamani, B Ashok, and Shashank Shet. 2024. Codeplan: Repository-level coding using llms and planning. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 675–698.

[11] Angela Barriga, Rogardt Heldal, Adrian Rutle, and Ludovico Iovino. 2022. PARMOREL: a framework for customizable model repair. *Software and Systems Modeling* 21, 5 (2022), 1739–1762.

[12] Angela Barriga, Adrian Rutle, and Rogardt Heldal. 2019. Personalized and automatic model repairing using reinforcement learning. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 175–181.

[13] Angela Barriga, Adrian Rutle, and Rogardt Heldal. 2022. AI-powered model repair: an experience report—lessons learned, challenges, and opportunities. *Software and Systems Modeling* 21, 3 (2022), 1135–1157.

[14] Lionel C Briand, Yvan Labiche, and Leeshawn O'Sullivan. 2003. Impact analysis and change management of UML models. In *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.* IEEE, 256–265.

[15] 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.

[16] Loli Burgueño, Robert Clarisó, Sébastien Gérard, Shuai Li, and Jordi Cabot. 2021. An NLP-based architecture for the autocompletion of partial domain models. In *Proceedings of the International Conference on Advanced Information Systems Engineering*. Springer, 91–106. doi:10.1007/978-3-030-79382-1_6

[17] Lola Burgueño, Davide Di Ruscio, Houari Sahraoui, and Manuel Wimmer. 2025. Automation in Model-Driven Engineering: A look back, and ahead. *ACM Transactions on Software Engineering and Methodology* (2025).

[18] Javier Cámara, Javier Troya, Lola 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 (2023), 781–793.

[19] Thibaut Capuano, Houari A Sahraoui, Benoit Frenay, and Benoit Vanderose. 2022. Learning from Code Repositories to Recommend Model Classes. *J. Object Technol.* 21, 3 (2022), 3–1.

[20] Meriem Ben Chaaben, Lola Burgueño, Istvan David, and Houari Sahraoui. 2024. On the Utility of Domain Modeling Assistance with Large Language Models. *arXiv preprint arXiv:2410.12577* (2024).

[21] Meriem Ben Chaaben, Lola Burgueño, and Houari Sahraoui. 2023. Towards using few-shot prompt learning for automating model completion. In *Proceedings of the International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE, 7–12.

[22] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint* (2021). doi:10.48550/arXiv.2107.03374

[23] Valerie Chen, Alan Zhu, Sebastian Zhao, Hussein Mozannar, David Sontag, and Ameet Talwalkar. 2025. Need Help? Designing Proactive AI Assistants for Programming. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–18.

[24] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. 2008. Meta-model differences for supporting model co-evolution. In *Proceedings of the 2nd Workshop on Model-Driven Software Evolution-MODSE*, Vol. 1.

[25] Aaron Conrardy and Jordi Cabot. 2024. From image to uml: first results of image based uml diagram generation using llms. *arXiv preprint arXiv:2404.11376* (2024).

[26] Carlos Durá Costa, José Antonio Hernández López, and Jesús Sánchez Cuadrado. 2024. ModelMate: A recommender for textual modeling languages based on pre-trained language models. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*. 183–194.

[27] Shuiguang Deng, Dongjing Wang, Ying Li, Bin Cao, Jianwei Yin, Zhaohui Wu, and Mengchu Zhou. 2016. A recommendation system to facilitate business process modeling. *IEEE transactions on cybernetics* 47, 6 (2016), 1380–1394.

[28] Juri Di Rocco, Davide Di Ruscio, Claudio Di Sipio, Phuong T Nguyen, and Alfonso Pierantonio. 2023. MemoRec: a recommender system for assisting modelers in specifying metamodels. *Software and Systems Modeling* 22, 1 (2023), 203–223.

[29] Juri Di Rocco, Davide Di Ruscio, Claudio Di Sipio, Phuong T Nguyen, and Riccardo Rubei. 2025. On the use of large language models in model-driven engineering: J. Di Rocco et al. *Software and Systems Modeling* 24, 3 (2025), 923–948.

[30] Juri Di Rocco, Claudio Di Sipio, Phuong T Nguyen, Davide Di Ruscio, and Alfonso Pierantonio. 2022. Finding with nemo: a recommender system to forecast the next modeling operations. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*. 154–164.

[31] Claudio Di Sipio, Juri Di Rocco, Davide Di Ruscio, and Phuong T Nguyen. 2023. MORGAN: a modeling recommender system based on graph kernel. *Software and Systems Modeling* (2023), 1–23.

[32] Marc Eaddy, Thomas Zimmermann, Kaitlin D Sherwood, Vibhav Garg, Gail C Murphy, Nachiappan Nagappan, and Alfred V Aho. 2008. Do crosscutting concerns cause defects? *IEEE transactions on Software Engineering* 34, 4 (2008), 497–515.

[33] Tobias Eisenreich, Sandro Speth, and Stefan Wagner. 2024. From requirements to architecture: An ai-based journey to semi-automatically generate software architectures. In *Proceedings of the 1st International Workshop on Designing Software*. 52–55.

[34] Akil Elkamel, Mariem Gzara, and Hanêne Ben-Abdallah. 2016. An UML class recommender system for software design. In *Proceedings of the International Conference of Computer Systems and Applications (AICCSA)*. IEEE, 1–8.

[35] Robert France and Bernhard Rumpe. 2007. Model-driven development of complex software: A research roadmap. In *Future of Software Engineering (FOSE'07)*. IEEE, 37–54.

[36] Dominik Fuchß, Tobias Hey, Jan Keim, Haoyu Liu, Niklas Ewald, Tobias Thirolf, and Anne Koziolek. 2025. LiSSA: toward generic traceability link recovery through retrieval-augmented generation. In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering. ICSE*, Vol. 25.

[37] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[38] Markus Herrmannsdoerfer, Sebastian Benz, and Elmar Juergens. 2008. Automatability of coupled evolution of metamodels and models in practice. In *International conference on model driven engineering languages and systems*. Springer, 645–659.

[39] Yang Hong, Chakkrit Tantithamthavorn, Patanamon Thongtanunam, and Aldeida Aleti. 2024. Don't forget to change these functions! recommending co-changed functions in modern code review. *Information and Software Technology* 176 (2024), 107547.

[40] Ludovico Iovino, Angela Barriga, Adrian Rutle, Rogardt Heldal, et al. 2020. Model repair with quality-based reinforcement learning. *Journal of Object Technology* 19, 2 (2020).

[41] Maliheh Izadi and Matin Nili Ahmadabadi. 2022. On the evaluation of NLP-based models for software engineering. In *Proceedings of the 1st International Workshop on Natural Language-based Software Engineering*. 48–50.

[42] Fehmi Jaafar, Yann-Gaël Guéhéneuc, Sylvie Hamel, and Giuliano Antoniol. 2011. An exploratory study of macro co-changes. In *2011 18th Working Conference on Reverse Engineering*. IEEE, 325–334.

[43] Zijian Jiang, Hao Zhong, and Na Meng. 2021. Investigating and recommending co-changed entities for JavaScript programs. *Journal of Systems and Software* 180 (2021), 111027.

[44] Huzefa Kagdi, Malcom Gethers, and Denys Poshyvanyk. 2013. Integrating conceptual and logical couplings for change impact analysis in software. *Empirical Software Engineering* 18 (2013), 933–969.

[45] Dhikra Kchaou, Nadia Bouassida, and Hanêne Ben-Abdallah. 2017. UML models change impact analysis using a text similarity technique. *IET Software* 11, 1 (2017), 27–37.

[46] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. 1997. Aspect-oriented programming. In *ECOOP'97—Object-Oriented Programming: 11th European Conference Jyväskylä, Finland, June 9–13, 1997 Proceedings 11*. Springer, 220–242.

[47] Stefan Kögel. 2017. Recommender system for model driven software development. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 1026–1029.

[48] Stefan Kögel, Raffaela Groner, and Matthias Tichy. 2016. Automatic Change Recommendation of Models and Meta Models Based on Change Histories.. In *ME@ MoDELS*. 14–19.

[49] Roland Kretschmer, Djamel Eddine Khelladi, Roberto Erick Lopez-Herrejon, and Alexander Egyed. 2021. Consistent change propagation within models. *Software and Systems Modeling* 20, 2 (2021), 539–555.

[50] Tobias Kuschke and Patrick Mäder. 2017. RapMOD—In Situ Auto-Completion for Graphical Models. In *Proceedings of the International Conference on Software Engineering (ICSE): Companion Proceedings*. IEEE, 303–304.

[51] Tobias Kuschke, Patrick Mäder, and Patrick Rempel. 2013. Recommending auto-completions for software modeling activities. In *International conference on model driven engineering languages and systems*. Springer, 170–186.

[52] Alexander Lauer, Jens Kosiol, and Gabriele Taentzer. 2023. Empowering model repair: a rule-based approach to graph repair without side effects. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 831–840.

[53] Bixin Li, Xiaobing Sun, Hareton Leung, and Sai Zhang. 2013. A survey of code-based change impact analysis techniques. *Software Testing, Verification and Reliability* 23, 8 (2013), 613–646.

[54] Ying Li, Bin Cao, Lida Xu, Jianwei Yin, Shuiguang Deng, Yuyu Yin, and Zhaohui Wu. 2013. An efficient recommendation method for improving business process modeling. *IEEE Transactions on Industrial Informatics* 10, 1 (2013), 502–513.

[55] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*. 2980–2988.

[56] Haoyu Liu, Yunwei Dong, Qiao Ke, and Zhiyang Zhou. 2024. ReCo: A Modular Neural Framework for Automatically Recommending Connections in Software Models. In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 637–648.

[57] Junwei Liu, Yixuan Chen, Mingwei Liu, Xin Peng, and Yiling Lou. 2024. Stall+: Boosting llm-based repository-level code completion with static analysis. *arXiv preprint arXiv:2406.10018* (2024).

[58] José Antonio Hernández López, Javier Luis Cánovas Izquierdo, and Jesús Sánchez Cuadrado. 2022. Modelset: a dataset for machine learning in model-driven engineering. *Software and Systems Modeling* (2022), 1–20.

[59] José Antonio Hernández López, Carlos Durá, and Jesús Sánchez Cuadrado. 2023. Word embeddings for model-driven engineering. In *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 151–161.

[60] José Antonio Hernandez López, Máté Földiák, and Dániel Varró. 2024. Text2vql: teaching a model query language to open-source language models with ChatGPT. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*. 13–24.

[61] José Antonio Hernández López, Riccardo Rubei, Jesús Sánchez Cuadrado, and Davide Di Ruscio. 2022. Machine learning methods for model classification: a comparative study. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*. 165–175.

[62] Nuno Macedo, Tiago Jorge, and Alcino Cunha. 2016. A feature-based classification of model repair approaches. *IEEE Transactions on Software Engineering* 43, 7 (2016), 615–640.

[63] Bennett Mackenzie, Vera Pantelic, Gordon Marks, Stephen Wynn-Williams, Gehan Selim, Mark Lawford, Alan Wassyng, Moustapha Diab, and Feisel Weslati. 2020. Change impact analysis in simulink designs of embedded systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1274–1284.

[64] Luciano Marchezan, Roland Kretschmer, Wesley KG Assunção, Alexander Reder, and Alexander Egyed. 2023. Generating repairs for inconsistent models. *Software and Systems Modeling* 22, 1 (2023), 297–329.

[65] Duncan McElfresh, Sujay Khandagale, Jonathan Valverde, John Dickerson, and Colin White. 2022. On the generalizability and predictability of recommender systems. *Advances in Neural Information Processing Systems* 35 (2022), 4416–4432.

[66] Tom Mens, Ragnhild Van Der Straeten, and Maja D'Hondt. 2006. Detecting and resolving model inconsistencies using transformation dependency analysis. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 200–214.

[67] Vittoriano Muttillo, Claudio Di Sipio, Riccardo Rubei, Luca Berardinelli, and MohammadHadi Dehghani. 2024. Towards Synthetic Trace Generation of Modeling Operations using In-Context Learning Approach. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 619–630.

[68] Patrick Mäder, Tobias Kuschke, and Mario Janke. 2021. Reactive Auto-Completion of Modeling Activities. *Transactions on Software Engineering* 47, 7 (2021), 1431–1451. doi:10.1109/TSE.2019.2924886

[69] Nebras Nassar, Hendrik Radke, and Thorsten Arendt. 2017. Rule-based repair of EMF models: An automated interactive approach. In *International conference on theory and practice of model transformations*. Springer, 171–181.

[70] Manuel Ohrndorf, Christopher Pietsch, Udo Kelter, Lars Grunske, and Timo Kehrer. 2021. History-Based Model Repair Recommendations. *Transactions of Software Engineering Methodology* 30, 2, Article 15 (2021). doi:10.1145/3419017

[71] Manuel Ohrndorf, Christopher Pietsch, Udo Kelter, and Timo Kehrer. 2018. ReVision: A tool for history-based model repair recommendations. In *Proceedings of the International Conference on Software Engineering (ICSE): Companion Proceedings*. ACM, 105–108. doi:10.1145/3419017

[72] Siru Ouyang, Wenhao Yu, Kaixin Ma, Zilin Xiao, Zhihan Zhang, Mengzhao Jia, Jiawei Han, Hongming Zhang, and Dong Yu. 2024. RepoGraph: Enhancing AI Software Engineering with Repository-level Code Graph. *arXiv preprint arXiv:2410.14684* (2024).

[73] Debalina Ghosh Paul, Hong Zhu, and Ian Bayley. 2024. Benchmarks and Metrics for Evaluations of Code Generation: A Critical Review. In *2024 IEEE International Conference on Artificial Intelligence Testing (AITest)*. IEEE, 87–94.

[74] Huy N Phan, Hoang N Phan, Tien N Nguyen, and Nghi DQ Bui. 2024. RepoHyper: Search-Expand-Refine on Semantic Graphs for Repository-Level Code Completion. *arXiv preprint arXiv:2403.06095* (2024).

[75] Awais Rashid, Jean-Claude Royer, and Andreas Rummler. 2011. *Aspect-oriented, model-driven software product lines: The AMPLE way*. Cambridge University Press.

[76] Alexander Reder and Alexander Egyed. 2012. Computing repair trees for resolving inconsistencies in design models. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. 220–229.

[77] Deepjyoti Roy and Mala Dutta. 2022. A systematic review and research perspective on recommender systems. *Journal of Big Data* 9, 1 (2022), 59.

[78] Patrick Schober, Christa Boer, and Lothar A Schwarte. 2018. Correlation coefficients: appropriate use and interpretation. *Anesthesia & analgesia* 126, 5 (2018), 1763–1768.

[79] Yan-Martin Tamm, Rinchin Damdinov, and Alexey Vasilev. 2021. Quality metrics in recommender systems: Do we calculate metrics consistently?. In *Proceedings of the 15th ACM conference on recommender systems*. 708–713.

[80] Christof Tinnes, Timo Kehrer, Mitchell Joblin, Uwe Hohenstein, Andreas Biesdorf, and Sven Apel. 2023. Mining domain-specific edit operations from model repositories with applications to semantic lifting of model differences and change profiling. *Automated Software Engineering* 30, 2 (2023), 17.

[81] Christof Tinnes, Wolfgang Rössler, Uwe Hohenstein, Torsten Kühn, Andreas Biesdorf, and Sven Apel. 2022. Sometimes you have to treat the symptoms: tackling model drift in an industrial clone-and-own software product line. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1355–1366.

[82] Christof Tinnes, Alisa Welter, and Sven Apel. 2025. Software Model Evolution with Large Language Models: Experiments on Simulated, Public, and Industrial Datasets. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. 950–962. doi:10.1109/ICSE55347.2025.00112

[83] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[84] Kaixin Wang, Tianlin Li, Xiaoyu Zhang, Chong Wang, Weisong Sun, Yang Liu, and Bin Shi. 2025. Software Development Life Cycle Perspective: A Survey of Benchmarks for Code Large Language Models and Agents. *arXiv preprint arXiv:2505.05283* (2025).

[85] Yuxiang Wei, Chunqiu Steven Xia, and Lingming Zhang. 2023. Copiloting the copilots: Fusing large language models with completion engines for automated program repair. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 172–184.

[86] Welter, Alisa and Tinnes, Christof and Apel, Sven. 2026. Supplementary Website. https://github.com/se-sic/modelcompletion_multilocations. Accessed: 5 January 2026.

[87] Martin Weyssow, Houari Sahraoui, and Eugene Syriani. 2022. Recommending metamodel concepts during modeling activities with pre-trained language models. *Software and Systems Modeling* 21, 3 (2022), 1071–1089. doi:10.1007/s10270-022-00975-5

[88] Boyang Yang, Haoye Tian, Jiadong Ren, Hongyu Zhang, Jacques Klein, Tegawendé F Bissyandé, Claire Le Goues, and Shunfu Jin. 2024. Multi-objective fine-tuning for enhanced program repair with llms. *arXiv preprint arXiv:2404.12636* (2024).

[89] He Ye and Martin Monperrus. 2024. Iter: Iterative neural repair for multi-location patches. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–13.

[90] Alfa Yohannis. 2020. *Change-Based Model Differencing and Conflict Detection*. Ph. D. Dissertation. University of York.

[91] Daihong Zhou, Yijian Wu, Xin Peng, Jiyue Zhang, and Ziliang Li. 2024. Revealing code change propagation channels by evolution history mining. *Journal of Systems and Software* 208 (2024), 111912.

[92] Thomas Zimmermann, Andreas Zeller, Peter Weissgerber, and Stephan Diehl. 2005. Mining version histories to guide software changes. *IEEE Transactions on software engineering* 31, 6 (2005), 429–445.