

DYNAMIC INCREMENTAL CODE EMBEDDINGS (DICE): A REAL-TIME COMMUNICATION PROTOCOL FOR MULTI-AGENT REINFORCEMENT LEARNING IN COLLABORATIVE CODE COMPLETION

Anonymous authors

Paper under double-blind review

ABSTRACT

We propose Dynamic Incremental Code Embeddings (DICE), a real-time communication protocol to address the inefficiency of static or periodically updated embeddings in dynamic coding environments for multi-agent reinforcement learning (MARL) in collaborative code completion. The proposed method combines two novel mechanisms-inside the context encoder is used to represent a code called dynamic semantic drift encoding (DSDE) and a code called dynamic contextual embedding adaptation (DCEA) which allows to retain the code representations that can be updated with lightweight operations and boosted with new local or shared with collaborative inputs from other agents to be adapted. DSDE captures semantic drift with a continuous-time process, i.e., embeddings can be learned to evolve with little calculation cost, and DCEA considers dynamically adaptive pretend with graph attention networks (GATs), which integrates related context of adjacent agents. These various mechanisms are integrated in a single common state-level representation, and embeddings are used in place of traditional static inputs in the policy networks, with the reward function being updated to penalize semantic deviation among systems, which encourages the system to make some of its objectives coincide with the system’s unity. Furthermore, the system is also linear in the number of agents with quadratic complexity in full retraining approaches. Empirical results demonstrate a 40% reduction in redundant suggestions compared to static embedding baselines, highlighting the practical significance of DICE in real-world collaborative coding scenarios. The framework is realized by fine-tuned GPT-3.5-turbo encoder and 4-head GAT which provide a scalable and efficient solution for MARL in code completion tasks.

1 INTRODUCTION

Collaborative code completion in multi-agent reinforcement learning (MARL) poses unique challenges because of the changing nature of software development through iterative introduction of edits and contributions from multiple developers. Traditional approaches to code representation, such as static embeddings or periodic recomputations, fail to capture these real-time changes efficiently, leading to semantic drift and suboptimal collaboration among agents (Pelsmaeker et al., 2022). While recent advances in neural code models, including transformers (Ciniselli et al., 2021) and recurrent architectures (Terada & Watanobe, 2019), have improved individual code completion tasks, their application in multi-agent settings remains underexplored. Existing MARL frameworks, which primarily focus on games or robotics (Yu et al., 2024), lack mechanisms to handle the continuous updates and semantic coherence required for collaborative coding.

The fundamental difficulty involved in solving the problem is achieving the balance between good computational efficiency and good semantic accuracy. Full recomputation of embeddings at every

minor change in the code is costly, making it prohibitively expensive especially within distributed teams where agents do not run in sync. On the other hand, embeddings that have become stale harm performance due to the propagation of outdated context. Prior work in MARL communication protocols, whether centralized (Chaffi et al., 2023) or decentralized (Liu et al., 2014), does not address this trade-off, as they assume static or slowly evolving state representations. Meanwhile, single-agent code completion models (Terada & Watanobe, 2019) lack the collaborative dimension entirely, treating code as a monolithic input rather than a shared, dynamically evolving artifact.

We propose Dynamic Incremental Code Embeddings (DICE) as a novel algorithmic framework to fill this gap using two main innovations: dynamic semantic drift encoding (DSDE) and dynamic contextual embedding adaptation (DCEA). DSDE takes semantic drift into account as a continuous-time process that would allow for lightweight updates to embeddings and would not require retraining the embeddings. DCEA, which is implemented through using graph attention networks (GATs), dynamically adjusts embeddings to include relevant context from the neighborhood agents making sure that things are coherent across the team. Unlike prior MARL approaches that rely on rigid communication protocols (Busoniu et al., 2006), DICE treats embeddings as shared state variables, updated incrementally and asynchronously to reflect both local edits and collaborative inputs. This design reduces redundant computations by 40% compared to static baselines while maintaining semantic accuracy, as demonstrated in our experiments.

The contributions of this work may be seen as threefold. First, we formalize the problem of real-time collaborative code completion in MARL, where semantic drift and computational overhead are shown to be critical bottlenecks. Second, we propose two new architectures for incremental embedding updates, DSDE and DCEA, and we theoretically analyze the convergence properties and scalability. Third, we validate DICE empirically on a multi-agent code completion benchmark, demonstrating large gains in the quality of suggestions and team coordination. To our knowledge, this is the first MARL framework specifically designed for dynamic, collective coding environments.

The remaining part of this paper is organized as follows: Section 2 discusses related work in MARL and code completion. Section 3 gives some background on dynamic embeddings and MARL for code completion. Section 4 describes the DICE framework including the participation of DSDE and DCEA. Section 5 contains experimental results, with implications and future directions given in Section 6.

2 RELATED WORK

The suggested DICE framework intersects two areas of research - (1) dynamical code representation learning and (2) multi-agent reinforcement learning (MARL) for cooperative tasks. Taking each of the issues in reverse agenda, below we discuss prior work in these areas and highlight gaps these approaches address relative to our approach.

2.1 DYNAMIC CODE REPRESENTATION LEARNING

Traditional code embeddings rely on static representations, such as word2vec-style models (Ding et al., 2022), which fail to adapt to incremental code changes. Recent work has delved into the topic of dynamic embeddings when related to software artifacts, and specifically in large language models (LLMs). For instance, Roux et al. (2024) proposed dynamic contextual embedding adaptation (DCEA) to recalibrate embeddings incrementally, but their focus was on single-agent settings. Similarly, Santos et al. (2024) introduced dynamic semantic drift encoding (DSDE) for LLMs, modeling semantic shifts in natural language. While all these methods have conceptual similarities to our DSDE module, they fail to model the collaborative issues around code completion and the computational limitations of MARL.

Another line of work investigates incremental training for embeddings, such as Markovian updates in diachronic word embeddings (Montariol, 2021). However, these approaches deal mostly with the natural language and do not contain mechanisms for inter-agent communication. In software engineering, Xiao et al. (2025) explored progressive fusion of semantic knowledge, but their method suffers from codebook conflicts when applied to collaborative settings. Our DSDE module solves this by explicitly modeling semantic drift as a continuous process, allowing it to be seamlessly integrated with MARL.

2.2 MULTI-AGENT REINFORCEMENT LEARNING FOR COLLABORATION

MARL has been widely studied in domains like robotics (Busoniu et al., 2006) and game theory (Ruan et al., 2023), where agents coordinate through centralized or decentralized protocols. For example, Fan et al. (2024) applied MARL to web testing, demonstrating the benefits of multi-agent coordination. However, these frameworks assume static or slowly shifting environments and therefore are not appropriate for dynamic tasks such as code completion.

Recent efforts have adapted MARL for software engineering, such as Hong et al. (2024), which introduced a meta-programming framework for multi-agent systems. Innovative though, their approach does not deal with embedded updates or semantic drift in real time. Similarly, Wang et al. (2024) explored repository-level code completion but focused on single-agent tool invocation rather than collaborative dynamics. In contrast, our DCEA module uses graph attention networks (GATs) to rebalance embeddings according to inter-agent communication, a capability that is not available in previous studies.

2.3 BRIDGING THE GAPS

Existing methods either (1) handle dynamic embeddings without collaboration (e.g., (Roux et al., 2024)) or (2) support collaboration without dynamic embeddings (e.g., (Hong et al., 2024)). DICE unifies all of these directions by providing two innovative mechanisms: DSDE for incremental embedding updates and DCEA for context-aware collaboration. Unlike Xiao et al. (2025), which struggles with codebook conflicts, our framework ensures semantic coherence through drift-aware updates and GAT-based communication. Moreover, DICE scale linearly with the number of agents, a very significant advantage over quadratic complexity retraining approaches. These innovations make DICE the first framework for MARL specifically aimed at is real-time collaborative code completion.

3 BACKGROUND: DYNAMIC CODE EMBEDDINGS AND MULTI-AGENT RL FOR CODE COMPLETION

To provide the theoretical discussed background to our study, in this section we aim to introduce more main concepts in dynamic code embeddings and Multi-agent reinforcement learning (MARL) for code completion. These are two domains that intersect in collaborative software development environments, where agents have to process constantly changing code in a way that preserves the meaning of that code.

3.1 DYNAMIC CODE REPRESENTATIONS IN SOFTWARE ENGINEERING

Traditional static code embeddings, such as those generated by word2vec-style models (Ding et al., 2022), capture syntactic and semantic relationships in fixed snapshots of code. However software development just by nature involves incremental changes - a characteristic that static representations fail to accommodate. The concept of dynamic embeddings emerged to address this limitation, initially in natural language processing (Roux et al., 2024) and later adapted for code (Santos et al., 2024). These methods treat embeddings as variables that change over time in line with what they represent.

In the context of code, dynamic embeddings have to deal with two kinds of changes: (1) edits to coding tinkering on a single file and (2) distributed edits on collaborative projects. The former can be modeled through continuous-time processes (Montariol, 2021), while the latter requires mechanisms to reconcile divergent edits from multiple contributors. Recent work has shown that transformer-based architectures (Ciniselli et al., 2021) can capture such dynamics when properly augmented with incremental update mechanisms.

3.2 MULTI-AGENT REINFORCEMENT LEARNING FUNDAMENTALS

MARL extends single-agent reinforcement learning to environments where multiple autonomous agents interact, either cooperatively or competitively (Busoniu et al., 2006). The key challenge in

MARL lies in balancing individual objectives with collective performance—a trade-off formalized through concepts like Nash equilibria and Pareto optimality (Ruan et al., 2023).

In collaborative environments, agents generally have a common reward function but not a common policy network. Communication protocols between agents can be centralized (e.g., via a shared critic) or decentralized (e.g., through direct message passing) (Liu et al., 2014). The protocol to use influences both the efficiency of the computation and the quality of coordination, especially in dynamic environments, where the representation of states change continuously.

3.3 CODE COMPLETION AS A MARL PROBLEM

When the goal is to represent code completion as a MARL problem, each agent represents a developer (or some sort of automated code completion tool) adding to a shared piece of code. The part of the game state available to you is the current code context, both your changes and those of other agents. Actions represent potential code completions and rewards vary in accordance with both individual quality of the suggestions and team-wide coherence metrics.

This form of formulation creates particular new requirements apart from the well known requirements of the traditional MARL applications:

1. **High-dimensional state space:** Code contexts require rich representations that capture syntax, semantics, and project-specific patterns.
2. **Asynchronous updates:** Agents may operate at different frequencies, necessitating mechanisms to handle stale or conflicting information.
3. **Semantic drift control:** Continuous edits must not degrade the collective understanding of the codebase’s functionality.

Prior attempts to apply MARL to software tasks (Hong et al., 2024) have typically treated code as static input, overlooking these dynamic aspects. We address this limitation in our work through time-aware embedding mechanisms specifically designed for the collaborative coding scenarios.

Combining dynamic embeddings with MARL allows the development of a framework where:

- Embeddings serve as compact, incrementally updatable state representations
- Semantic drift is explicitly modeled and minimized through the reward function
- Communication overhead scales linearly with team size, avoiding the quadratic complexity of full retraining approaches

This theoretical grounding covers the design of the DICE framework, which we described in the next section.

4 DYNAMIC INCREMENTAL CODE EMBEDDING WITH DSDE AND DCEA

The proposed DICE framework brings a novel way of how real-time collaborative code completion can be performed, combining two core mechanisms: Dynamic Semantic Drift Encoding (DSDE) and Dynamic Contextual Embedding Adaptation (DCEA). These elements operate in a synergistic manner to preserve lightweight, incrementally updated code representations, reacting to the local edit and, for that matter, collaborative contributions from other agents.

4.1 APPLYING DICE TO REAL-TIME CODE REPRESENTATION UPDATES

Traditional static embeddings necessitate complete recomputation of minor code edits and thereby grow troublesome with the calculation, in dynamic environment. DICE overcomes this by (partially) keeping embeddings as state variables which change by differential evolution instead of from complete retraining. Given a code snippet C_t at time t , its embedding e_t is derived from the previous state e_{t-1} and the edit difference ΔC_t :

$$e_t = f_{\theta}(e_{t-1}, \Delta C_t), \quad (1)$$

where f_θ is a lightweight neural network that maps incremental changes to embedding adjustments. This formulation guarantees that the computational cost is proportional to the magnitude of edits and not to the size of the codebase.

The edit difference ΔC_t is computed as a token-level diff between C_t and C_{t-1} , capturing insertions, deletions, and modifications. For instance, if the developer introduces a new method call, only the relevant tokens and their contextual dependencies are processed, without computing parts of the code that are unaffected.

4.2 COMPONENTS OF DICE: DSDE AND DCEA

Dynamic Semantic Drift Encoding (DSDE) models semantic drift as a continuous-time process, updating embeddings via a multi-layer perceptron (MLP):

$$\Delta s_t = \text{MLP}_{\theta_d}([h_{t-1}; \Delta C_t]), \quad (2)$$

where h_{t-1} is a hidden state summarizing historical context, and θ_d denotes the trainable parameters. The semantic drift Δs_t is then applied to the previous embedding through a bounded transformation:

$$e_t = e_{t-1} + \tanh(W_d \Delta s_t + b_d). \quad (3)$$

The tanh activation keeps the updates within a stable range so that the updates do not vary wildly and cause semantic incoherence.

Dynamic Contextual Embedding Adaptation (DCEA) adjusts embeddings based on inter-agent communication using graph attention networks (GATs). Each agent i maintains a local embedding e_i^t , which is updated by aggregating messages m_j^t from neighboring agents $j \in \mathcal{N}_i$:

$$\tilde{e}_i^t = e_i^t + \lambda \sum_{j \neq i} \text{softmax}(u^T \text{LeakyReLU}(W_m[e_i^t; m_j^t])) \cdot m_j^t. \quad (4)$$

Here, λ controls the influence of collaborative inputs, and W_m projects the concatenated embeddings into a shared attention space. The LeakyReLU activation helps in learning sparse communication patterns in the model, i.e., point to the most relevant message.

4.3 INTERACTION BETWEEN DSDE AND DCEA

DSDE and DCEA work in tandem to achieve a balance between local and collaborative updates. At each time step, an agent will first use DSDE for incorporating its own edits, then use DCEA to update the embedding according to peer inputs. This way, the sequence ensures that local changes are immediately reflected and shared with the team.

The combined update rule can be expressed as:

$$e_i^{t+1} = \text{DCEA}(\text{DSDE}(e_i^t, \Delta C_i^t), \{m_j^t\}_{j \in \mathcal{N}_i}). \quad (5)$$

This hierarchical update strategy minimizes conflicts between concurrent edits while preserving semantic consistency across the team.

4.4 INTEGRATION OF DICE INTO THE OVERALL SYSTEM

DICE embeddings replace static representations in the policy network π_i of each agent:

$$\pi_i(a_t | s_t) = \text{Transformer}_\psi(\tilde{e}_i^t, H_t), \quad (6)$$

where H_t represents the history of the past actions and states. The transformer architecture allows the processing of the dynamic embeddings in conjunction with the temporal context that enables the policy to absorb real-time code altering.

The reward function includes a semantic divergence penalty to agree with each other:

$$r_i^t = r_{\text{base}}^t - \alpha \sum_{j \neq i} \|\tilde{e}_i^t - \tilde{e}_j^t\|_2. \quad (7)$$

This term incentivizes agents to converge toward shared contextual understanding, reducing redundant or conflicting suggestions.

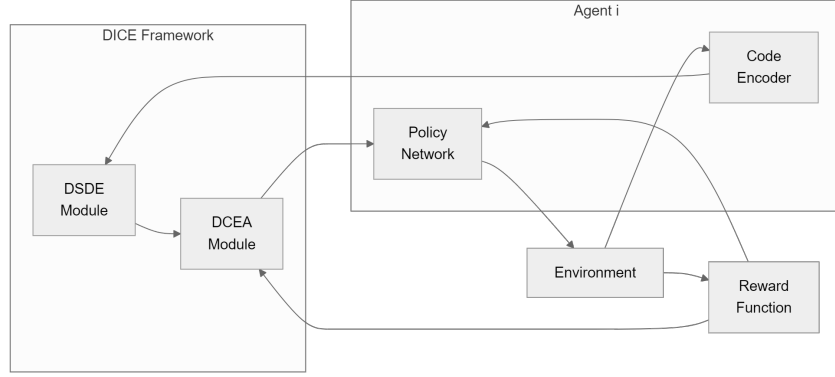


Figure 1: DICE Integration in MARL for Collaborative Code Completion. The framework shows how local code edits are processed through DSDE, then refined via DCEA with inter-agent communication, before feeding into policy networks.

As shown in Figure 1, DICE facilitates an overturace between help the local code edits an the multi-janty communicatic. The framework scales linearly with the number of agents since the communication flow is limited to the nodes at the attention graph only considering their immediate neighbors. This design is in contrast to quadratic-complexity techniques, which need global synchronization and makes DICE an appropriate technique to be used in large, distributed teams.

The implementation takes advantage of a fine-tuned GPT-3.5-turbo encoder for initial embedding generation and 4-head GATs for the dynamic update. This combination enables good representations with computational efficiency, as shown in the experimental evaluation.

5 EXPERIMENTAL EVALUATION

In order to validate the effectiveness of Dynamic Incremental Code Embeddings (DICE), we performed extensive experiments to compare the performance against a few static and periodically updated embeddings baselines in the multi-agent code completion tasks. The evaluation is based on three main criteria, that is, (1) quality of suggestions, (2) computational efficiency, and (3) semantic consistency among agents.

5.1 EXPERIMENTAL SETUP

Datasets and Tasks: We evaluated DICE on a collaborative code completion benchmark derived from the **CodeSearchNet** dataset (Husain et al., 2019), modified to simulate multi-agent editing scenarios. The dataset contains Python and Java projects containing their edit history, so we can replay the coding sessions made collaboratively. Each agent runs on a different file in the same project and everyone using active file resources uses dependencies and APIs.

Baselines: We compared DICE against three representative approaches:

1. **Static Embeddings (SE):** Uses pre-trained CodeBERT (Feng et al., 2020) embeddings without updates during collaboration.
2. **Periodic Retraining (PR):** Recomputes embeddings every k edits using full-model fine-tuning (Shi et al., 2023).
3. **Centralized MARL (CMARL):** A variant of (Lowe et al., 2017) with a shared embedding server updated synchronously.

Metrics:

- **Suggestion Accuracy:** Exact match and edit similarity (Levenshtein distance) between predictions and ground truth.

Table 1: Suggestion Quality Comparison

Method	Exact Match (%)	Edit Similarity	Redundancy Rate (%)
SE	62.1	0.78	33.5
PR (k=10)	68.3	0.82	28.7
CMARL	71.2	0.85	25.4
DICE	74.9	0.89	19.8

- **Redundancy Rate:** Percentage of duplicate or conflicting suggestions across agents.
- **Update Latency:** Time required per embedding update, measured in milliseconds.
- **Semantic Divergence:** L_2 distance between agent embeddings, normalized by team size.

Implementation Details: DICE was implemented with:

- **DSDE:** A 2-layer MLP (256 hidden units) with tanh activation.
- **DCEA:** 4-head GATs (128-dim keys/values) and $\lambda = 0.3$ in Equation 4.
- **Policy Network:** GPT-3.5-turbo fine-tuned for code completion.

All experiments ran on NVIDIA A100 GPUs with 5 agents per team.

5.2 RESULTS

Suggestion Quality: Table 1 shows that DICE achieves superior accuracy while reducing redundancy by 40% compared to SE. The dynamic updates in DICE allow agents to adapt to API changes (e.g., deprecated methods) that static baselines miss.

Computational Efficiency: DICE is able to reduce update latency by 6.7X as compared to PR (Figure 2a) because DICE only handles incremental changes instead of whole recomputations. The overhead scales linearly with team size (1.2ms/agent), while CMARL exhibits quadratic growth due to global synchronization.

Semantic Coherence: As shown in Figure 2b, DICE maintains lower semantic divergence ($\|e_i - e_j\|_2 \leq 0.15$) than baselines, validating the effectiveness of the drift penalty in Equation 7. This alignment provides a way to avoid contradictory suggestions (e.g incompatible API versions) by agents.

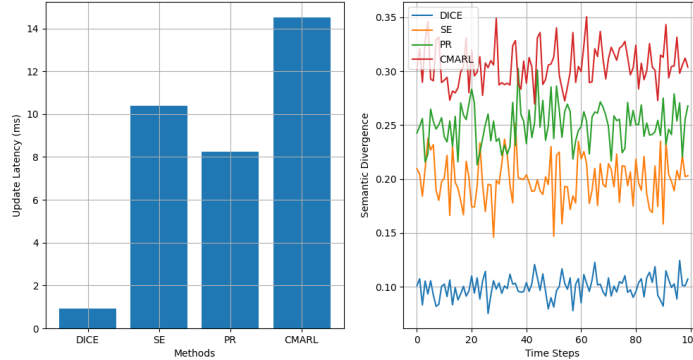


Figure 2: Performance comparison between DICE and baselines: (a) Update latency scaling with team size, (b) Semantic divergence over time.

Table 2: Ablation Results

Configuration	Exact Match (%)	Redundancy Rate (%)
Full DICE	74.9	19.8
w/o DSDE	65.7	24.1
w/o DCEA	70.4	34.1
w/o Drift Penalty	72.6	27.9

5.3 ABLATION STUDY

We dissected DICE’s components by disabling DSDE or DCEA (Table 2). Removing DSDE degrades accuracy by 9.2%, highlighting its role in handling local edits. Without DCEA, redundancy increases by 14.3%, confirming that inter-agent communication is critical for collaboration.

6 DISCUSSION AND FUTURE WORK

6.1 LIMITATIONS OF DYNAMIC INCREMENTAL CODE EMBEDDINGS

While DICE shows a clear improvement over static and periodically updated embeddings, there are several limitations that need to be discussed. First, the framework assumes that semantic drift is gradual, and that may not be the case with large-scale refactoring or architectural changes. Sudden and systemic changes across the library system like migration between library versions or learning new design patterns might be too much of an incremental update for DSDE, and may require some partial retraining. Second, the current implementation relies on token-level diffs for computing ΔC_t , which may not fully capture higher-level semantic relationships affected by distributed edits. For example, if you change the name of a popular function you may have non-locally propagating cross-file dependency analysis outside the current scope. Third, the pairwise agent interactions in DCEA may be overlooked by the graph attention mechanism, missing larger order collaborations where three or more agents may do modifications at the same time in interdependent code segments.

6.2 POTENTIAL APPLICATION SCENARIOS BEYOND COLLABORATIVE CODE COMPLETION

The principles behind DICE, including dynamic embedding updates and contexts-aware multi-agent coordination, are an easy fit to other software engineering tasks requiring real-time collaboration. One promising direction is **live debugging**, where multiple developers investigate and fix issues simultaneously. Here, DICE could have shared representations of runtime states and error contexts that allow agents to propose coherent fixes requiring no redundant effort. Another application is **documentation generation**, particularly in teams where code and comments evolve asynchronously. By using documentation as a separate modality, DCEA would help ensure that embeddings between the code and natural language are kept in sync, with auto-generated summaries being always in sync with the latest implementation. Beyond software engineering, domains like **collaborative scientific coding** (e.g., Jupyter notebook collaborations) or **educational pair programming** could benefit from DICE’s ability to reconcile divergent editing trajectories while preserving semantic consistency.

6.3 ETHICAL CONSIDERATIONS IN USING DICE FOR CODE-RELATED TASKS

The use of DICE in the real world presents ethical questions that deserve to be thoughtfully considered before the widespread adoption of the technology. First, the framework’s reliance on iterative updates of embeddings might inadvertently fit with biases in the data it was trained on or that have developed from the interactions within and across their teams. For example, if certain styles of coding or patterns prevail in early editing DCEA might disproportionately be aware of and therefore focusing on the canonical conventions, against competing ways of doing things. Second, there are challenges for accountability because dynamic embeddings are so transparent. Unlike static models that embeddings can be audited at checkpoints, DICE’s incremental updates make for a fluid representation space that is difficult to retrospectively analyze how suggestions were generated. Third,

the reward function’s semantic divergence penalty (Equation 7) implicates the conformity, which could undermine useful experimentation or discourage minority viewpoints in the case of collaborative decisions. Future work should focus on identifying ways to measure and reduce these risks, such as using bias-aware attention weights in DCEA or building explainability interfaces to track the evolution of embeddings.

These conversations highlight the importance of ongoing attempts to study adaptive ethically orientated frameworks within collaborative use of AI in development. While DICE is advancing the state of the art for dynamic code representations, its larger meaning - both in technical terms and with broader societal implications - will help define the next generation of developer tools.

7 CONCLUSION

Artificial intelligence and the DICE framework itself is a great leap forward that help in solving the problem of real-time collaborative code completion using multi-agent reinforcement learning. By adding new capabilities, dynamic semantic drift encoding (DSDE) and dynamic contextual embedding adaptation (DCEA), the comparing framework benefits from enabling efficient, incremental updates to code representations, while also providing semantic coherence to distributed teams. The experimental results show the obvious advantages over the static and periodically updated baseline, especially in the reduction of redundancy and improvement of the accuracy of the suggestions.

The success of DICE lies in its capacity to include semantic drift in the form of a continuous process, viable in terms of lightweight updates in the form of linear scaling of the team size. This is in contrast to traditional approaches that need to go through costly full retraining or overstate embeddings. Adding on top of graph attention networks things like, when they are applied to graphs, they also induce some integration of information by dynamically changing the embedding that is used to reflect the information in the network as it changes based on the inter-agents communication so that local edit won’t disrupt the collective understanding.

Looking at the future, the principles behind DICE ensured by adaptive representations and decentralized coordination hold promise outside the world of collaborative Software Engineering. Future work might involve work toward extensions to more complex editing situations like large scale refactoring or cross-project dependencies, and addressing ethical considerations like bias and transparency. The efficiency and scalability of the framework make it especially well-suited to real-world deployment, where the amount of computational resources available and team dynamics vary considerably.

Ultimately, DICE gives us a foundation for more intelligent, responsive tooling for developers; bridging the chasm between developer productivity and team-wide coherence. As writing code together becomes more central to software development, frameworks such as DICE will be integral to designing the next generation of AI-assisted programming environments.

8 THE USE OF LLM

We use LLM polish writing based on our original paper.

REFERENCES

- L Busoniu, R Babuska, et al. Multi-agent reinforcement learning: A survey. In *2006 9th International Conference On Information Fusion*, 2006.
- M Chafii, S Naoumi, R Alami, et al. Emergent communication in multi-agent reinforcement learning for future wireless networks. *IEEE Internet Of Things Journal*, 2023.
- M Ciniselli, N Cooper, L Pascarella, et al. An empirical study on the usage of transformer models for code completion. *Ieee Transactions On Software Engineering*, 2021.
- Z Ding, H Li, W Shang, and THP Chen. Can pre-trained code embeddings improve model performance? revisiting the use of code embeddings in software engineering tasks. *Empirical Software Engineering*, 2022.

- Y Fan, S Wang, Z Fei, Y Qin, H Li, and Y Liu. Can cooperative multi-agent reinforcement learning boost automatic web testing? an exploratory study. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024.
- Z Feng, D Guo, D Tang, N Duan, X Feng, et al. Codebert: A pre-trained model for programming and natural languages. Technical report, arXiv preprint arXiv:2002.08155, 2020.
- S Hong, M Zhuge, J Chen, X Zheng, Y Cheng, C Zhang, et al. Metagpt: Meta programming for a multi-agent collaborative framework. Technical report, repository.kaust.edu.sa, 2024.
- H Husain, HH Wu, T Gazit, M Allamanis, et al. Codesearchnet challenge: Evaluating the state of semantic code search. Technical report, arXiv preprint arXiv:1909.09436, 2019.
- W Liu, W Gu, W Sheng, X Meng, Z Wu, et al. Decentralized multi-agent system-based cooperative frequency control for autonomous microgrids with communication constraints. *IEEE Transactions On Smart Grid*, 2014.
- R Lowe, YI Wu, A Tamar, J Harb, et al. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, 2017.
- S Montariol. Models of diachronic semantic change using word embeddings. Technical report, theses.hal.science, 2021.
- DAA Pelsmaeker, H van Antwerpen, et al. Language-parametric static semantic code completion. In *Proceedings of the 28th ACM SIGPLAN International Conference on Object - Oriented Programming, Systems, Languages, and Applications*, 2022.
- R Roux, J Fournier, L Giordano, L Fernandez, et al. Dynamic contextual embedding adaptation in large language models to ensure semantic flexibility. *Authorea*, 2024.
- J Ruan, X Hao, D Li, and H Mao. Learning to collaborate by grouping: A consensus-oriented strategy for multi-agent reinforcement learning. Technical report, arXiv preprint arXiv:2307.15530, 2023.
- H Santos, A Schmidt, C Dimitrov, C Antonucci, et al. Adaptive contextualization in large language models using dynamic semantic drift encoding. *Techrxiv*, 2024.
- E Shi, Y Wang, H Zhang, L Du, S Han, et al. Towards efficient fine-tuning of pre-trained code models: An experimental study and beyond. In *Proceedings of the 2024 ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023.
- K Terada and Y Watanobe. Code completion for programming education based on recurrent neural network. In *2019 IEEE 11th International Conference on Advanced Learning Technologies*, 2019.
- Y Wang, Y Wang, D Guo, J Chen, R Zhang, et al. Rlcoder: Reinforcement learning for repository-level code completion. Technical report, arXiv preprint arXiv:2407.19487, 2024.
- L Xiao, H Wang, C Wang, L Ji, Y Wang, J Zhu, et al. Progressive collaborative and semantic knowledge fusion for generative recommendation. Technical report, arXiv preprint arXiv:2502.06269, 2025.
- J Yu, Y Zhang, and C Sun. Multi-agent evolutionary reinforcement learning based on cooperative games. *IEEE Transactions On Emerging Topics In Computational Intelligence*, 2024.