

Diff Diffusion: Predicting Edit Paths with Diffusion Models for Efficient Code Editing

Xiaolong Xu*, Hao Jiang*, Bo Jiang*[†], Zhiwen Deng,
Ningyuan Sun, Bingzhou Chen, Jin Chen, Chenxing Wei
ByteDance, Shenzhen, China

{xuxiaolong.06, jianghao.techxgenus, jiangbo.jacob, dengzhiwen.11,
sunningyuan, chenbingzhou, chenjin.723, weichenxing}@bytedance.com

Abstract

Diffusion large language models (dLLMs) are appealing for code generation because they can update many tokens in parallel, yet many are optimized for sequence synthesis rather than edit prediction. This setting is poorly matched to modern AI-assisted IDEs, where edit prediction must infer the next code change from recent edits, the current file, and cursor-local context. Such changes are usually sparse and preserve most source tokens, so regenerating a full code segment wastes computation. We propose **Diff Diffusion**, a diffusion-inspired LLM framework for code editing that predicts edit paths from an original snippet to its revised form. Diff Diffusion represents denoising as structured edit operations, and combines this formulation with multi-stage training and efficient parallel decoding. By focusing computation on changed regions, Diff Diffusion supports accurate low-latency edits and multi-position parallel modification. Across public edit-completion benchmarks, edit-prediction evaluations, and online validation, Diff Diffusion matches or leads autoregressive baselines while delivering $2.78\times$ faster inference on public edit-completion benchmarks and a 39.0% P50 model-latency reduction in production. This work provides a new modeling perspective for AI applications that require high accuracy and low latency.

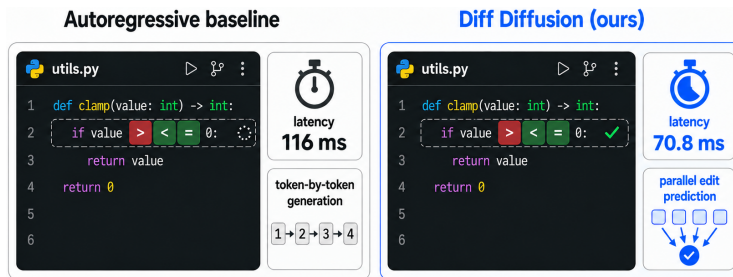


Figure 1: Diff Diffusion models real-time IDE code edit prediction.

1 Introduction

Large language models (LLMs) have achieved remarkable progress in natural language understanding and generation, and have gradually extended this success to code intelligence tasks. Beyond the

*The first three authors contributed equally.

[†]Corresponding author.

autoregressive (AR) paradigm, diffusion-based large language models (dLLMs) have become a promising alternative because they support parallel decoding and faster generation [2, 22]. By progressively denoising discrete noise tokens, dLLMs provide parallel generation capabilities and can make more efficient use of modern hardware computation. In terms of task performance, proprietary systems such as Mercury, Gemini Diffusion, and Seed Diffusion, as well as open-source projects such as DiffuCoder and Dream-Coder, have demonstrated competitive code-related capabilities [18, 13, 38, 12, 45].

However, current dLLMs mainly rely on masked diffusion to regenerate sequences from scratch, which does not match real programming workflows. In actual development, developers more often edit and refine existing code, and the changes are often small, local, and context-dependent. In IDE edit-prediction tasks, the system needs to predict one or more next code modifications in real time based on the user’s edit history, the current file, and the rewrite range around the cursor. This latency-sensitive workflow is illustrated in Figure 1. Autoregressive models must serially output many unchanged fragments token by token in this task, cannot go back to correct already generated content, and are not well suited to processing multiple edits in parallel; masked diffusion often needs to denoise an entire sequence even when only a small change is required. Over the past year, a number of works on discrete diffusion and discrete flow matching have discussed token insertion, modification, and deletion [38, 43, 23, 49, 15, 21, 11, 44, 35]. However, they do not formulate edit operations as source-conditioned edit-path diffusion for latency-critical editing tasks, where unchanged source tokens should be preserved and skipped while only a few regions are revised.

Therefore, we propose **Diff Diffusion**, a diffusion-inspired LLM framework for code edit prediction. In addition to modeling variable-length generation with inserted and deleted tokens, Diff Diffusion builds edit-path prediction from the initial state and optimizes around low-latency inference. Unlike existing masked diffusion methods, Diff Diffusion explicitly models edit paths and predicts the sequence of edit operations required to transform an original code snippet into the revised code. By incorporating structured edit representations into the diffusion process, the framework avoids redundant regeneration and instead focuses on local updates. We also design a multi-stage training paradigm to progressively strengthen edit-prediction ability, and propose parallel decoding and inference-system optimization strategies to accelerate inference while maintaining accuracy. We first validate public edit-prediction ability on the C3PO open edit-prediction benchmark, then construct a dedicated edit-prediction evaluation benchmark to analyze improvements on Stay, Act, Multi-line, Roll-back, Delete, and other specific metrics, and finally validate the method in production.

In summary, our contributions are:

- We propose **Diff Diffusion**, a source-conditioned diffusion-inspired framework that explicitly models and predicts code edit paths.
- We apply the framework to in-IDE edit prediction, using `same`, `delete`, `insert`, `replace`, and `mask-to-token` actions to reduce repeated decoding of unchanged code and support multi-position parallel modification.
- We design a multi-stage training paradigm and an efficient parallel decoding strategy to support accurate and fast inference.
- We evaluate the method on public benchmarks, edit-prediction evaluations, and production validation, demonstrating improvements in accuracy and latency.

2 Diff Diffusion

Diff Diffusion is designed for edit prediction in IDEs, where the model must recommend local code rewrites in real time from the current file, cursor-local context, and recent editing history. We initialize Diff Diffusion from Dream-Coder [45], which provided the strongest diffusion backbone for code at the time we began our experiments, combining competitive performance on code-related tasks with favorable latency. We keep its Transformer architecture, but change the denoising target from reconstructing a full sequence to predicting an edit path. This section presents the diff-based diffusion process and the multi-stage learning strategy, the overall framework is shown in Figure 2. Appendix A provides additional context on diffusion language models and code editing.



Figure 2: Diff Diffusion framework.

2.1 Diff-Based Diffusion Process

Conventional masked diffusion randomly masks a subset of tokens and trains the model to recover them through iterative denoising. Edit prediction has a different objective: rather than recovering a full program fragment from a corrupted sequence, the model should transform an existing snippet into its revised form. We therefore define a diff-based diffusion process, in which the denoising trajectory is represented by edit operations. In addition to ordinary token prediction, the model predicts insertion, deletion, and replacement actions that explicitly define the edit path.

The denoising process from the source state to the target state can be viewed as diff prediction:

- During training, the supervised target is the diff that specifies how the current state should be modified.
- During decoding, given x_t , the model predicts a diff d_t such that applying it moves the sequence toward the previous denoising state, x_{t-1} .

Step-by-step formulation.

1. The initial state is the original code, denoted as $x_{\text{src}} = [t_0, t_1, \dots, t_n]$.
2. The model assigns an edit label to each token in the current state:
 - (a) keep the token unchanged by emitting its `token_id`;
 - (b) delete the token by emitting `del_token_id`;
 - (c) insert a mask token before the current position by emitting `insert_token_id`;
 - (d) replace the current token with a mask token by emitting `replace_token_id`;
 - (e) fill a mask token by predicting a code `token_id`.
3. Tokens marked for deletion are removed, and the requested mask tokens are inserted or used as replacements.
4. The process is repeated:
 - (a) the model predicts code tokens for mask positions;
 - (b) the predicted tokens are again assigned edit labels;
 - (c) deleted tokens are removed, and newly inserted mask tokens are introduced.

Advantages. This formulation retains the central benefit of masked diffusion: multiple positions can be updated in parallel. It also extends this benefit from generation to editing, allowing the model to apply edits at several locations in the same diffusion step. Because code edits usually preserve most source tokens and modify only a small region, the number of steps required to reach the target snippet is much smaller than in from-scratch masked diffusion or standard autoregressive generation. The insert and delete actions also make variable-length output a native part of the process, avoiding the fixed-canvas assumption of common masked-diffusion formulations.

Training Strategy. During training, we adjust the scheduler so that the initial state, target state, and intermediate diffusion states are sampled with different probabilities. This stabilizes learning across the trajectory. The intermediate states are constructed from the initial and target code through LCS alignment. Edit-prediction data is highly imbalanced, most tokens are unchanged, while insertion, deletion, and replacement labels are rare but determine the quality of the edit path. We therefore re-weight token types in the loss to increase the learning signal for edit actions.

Training Objective. Formally, let $x_t = [x_t^1, x_t^2, \dots, x_t^n]$ denote the sequence at diffusion step t , and let $d_t = [d_t^1, d_t^2, \dots, d_t^n]$ denote the corresponding diff labels. Each $d_t^i \in \mathcal{V}_{\text{diff}}$, where $\mathcal{V}_{\text{diff}}$ contains ordinary code-token ids as well as edit-control labels such as `delete`, `insert`, and `replace`. Re-emitting the current token id keeps a token unchanged, while predicting a token id at a mask position fills that mask. The model predicts a distribution $p_\theta(d_t^i | x_t)$ for each position.

The training objective is the weighted cross-entropy loss:

$$\mathcal{L}(\theta) = - \sum_{t=1}^T \sum_{i=1}^{|x_t|} w(d_t^i) \cdot \log p_\theta(d_t^i | x_t),$$

where $w(d_t^i)$ is a weight function that compensates for class imbalance:

$$w(d_t^i) = \begin{cases} \lambda_{\text{edit}}, & d_t^i \in \{\text{insert}, \text{delete}, \text{replace}\}, \\ 1, & \text{otherwise.} \end{cases}$$

At inference time, the model generates the revised snippet by iteratively sampling diffs and applying the update rule:

$$x_{t-1} = \text{ApplyDiff}(x_t, d_t),$$

until the edited content no longer changes.

2.2 Multi-Stage Diffusion Learning

We train Diff Diffusion from a masked-diffusion backbone because the proposed formulation still requires mask-to-token prediction. Starting from this backbone transfers denoising ability to the edit setting, after which we progressively specialize the model for edit-path prediction.

Stage I: Large-Scale Git Commit Continuous Pretraining (CT). We first train the model on large-scale code-editing data so that it learns the diff-based modeling format and output space. This matches other evidence that dLLMs can surpass autoregressive models after large-scale training [33, 30]. Commit diffs from open-source repositories provide natural supervision for source-to-target code transformations. In our setup, we use a multi-language subset of CommitPack [29] as the starting point.

Stage II: Code-Editing SFT. After Stage I, the model has learned to predict differences along commit-level edit paths. Stage II adapts this ability to the edit-prediction input-output format used in interactive IDE scenarios. Each training example consists of the original snippet, surrounding context, historical edit information, and the target snippet. During training, we sample intermediate diffusion states on the edit path and supervise the model to predict the diff from the current state toward the target. By combining the C3PO training set [7] with high-quality edit-prediction data, Stage II converts general commit-level editing ability into a policy suitable for interactive edit prediction.

Stage III: Advanced Post-Training. The supervised objective in Stage II mainly learns the mapping from sampled intermediate states to the target state, whereas actual decoding repeatedly enters intermediate states produced by the model itself. Stage III therefore extends the training objective to learn more reliable edit directions. We apply this stage to a high-quality subset selected from the Stage-II SFT corpus. Inspired by discrete flow matching, we view the diff predicted from an intermediate diffusion state as a discrete modification direction from the current state toward the target state. This directional view supports our post-training schemes, Diff-DPO and AGRPO, both defined over denoising transitions. Let c denote the IDE context, including the source snippet, surrounding code, and recent edit history, and let y denote the target snippet. For a transition $x_t \rightarrow x_{t-1}$ induced by a realized diff d_t , we define the transition log-probability as

$$\ell_\theta(x_{t-1} | x_t, c) = \sum_{i=1}^{|d_t|} \log p_\theta(d_t^i | x_t, c), \quad x_{t-1} = \text{ApplyDiff}(x_t, d_t).$$

This transition-level view lets us apply preference learning and reinforcement learning directly to edit directions rather than to left-to-right token emissions. Diff-DPO uses paired final states, where

the positive state y^+ is the ground-truth target snippet and the negative state y^- is an erroneous edit sampled from the model. During policy decoding, we sample an intermediate state x_t ; from this same state, we construct x_{t-1}^+ and x_{t-1}^- as the next denoising states along shortest edit paths to y^+ and y^- , respectively. With a reference policy π_{ref} , we optimize a Diff Diffusion version of DPO [34, 39]:

$$\mathcal{L}_{\text{Diff-DPO}}(\theta) = -\mathbb{E}_{(c, x_t, x_{t-1}^+, x_{t-1}^-)} [\log \sigma(\beta(\Delta_\theta - \Delta_{\text{ref}}))],$$

where $\Delta_\pi = \ell_\pi(x_{t-1}^+ | x_t, c) - \ell_\pi(x_{t-1}^- | x_t, c)$ for $\pi \in \{\pi_\theta, \pi_{\text{ref}}\}$. This loss increases the model’s preference for the ground-truth edit direction over the erroneous model-generated direction, while the reference gap keeps the update anchored to the Stage-II policy.

For online reinforcement learning, we survey recent RL methods for diffusion and flow-matching models and adopt AGRPO because it gives the strongest performance with a simple implementation in our attempts [37, 6, 40, 24, 41, 48]. For each example $q = (c, y)$, the old policy samples a group of G complete edit trajectories $\{\tau_i\}_{i=1}^G$, where $\tau_i = \{(x_t^i, d_t^i, x_{t-1}^i)\}_{t=1}^{m_i}$ and the final state is $\hat{y}_i$. We use a terminal reward with a repetition penalty,

$$R_i = \mathbf{1}\{\hat{y}_i = y\} - \lambda_{\text{rep}} \text{Rep}(\hat{y}_i), \quad A_i = \frac{R_i - \frac{1}{G} \sum_{j=1}^G R_j}{\text{std}(\{R_j\}_{j=1}^G) + \delta},$$

which converts absolute rewards into group-relative advantages. We view diffusion generation as a multi-step Markov denoising process: instead of summing autoregressive output-token log-probabilities, AGRPO optimizes the policy objective over denoising transitions. For a sampled subset of diffusion steps $\mathcal{S}_i \subseteq \{1, \dots, m_i\}$, we maximize

$$\begin{aligned} \mathcal{J}_{\text{AGRPO}}(\theta) = \mathbb{E}_{q, \{\tau_i\} \sim \pi_{\text{old}}} \left[\frac{1}{G} \sum_{i=1}^G \frac{1}{|\mathcal{S}_i|} \sum_{t \in \mathcal{S}_i} \left(\min\{r_{i,t}(\theta) A_i, \text{clip}(r_{i,t}(\theta), 1 - \epsilon, 1 + \epsilon) A_i\} \right. \right. \\ \left. \left. - \beta_{\text{KL}} D_{\text{KL}}(\pi_\theta(\cdot | x_t^i, c) \| \pi_{\text{ref}}(\cdot | x_t^i, c)) \right) \right], \end{aligned}$$

where $r_{i,t}(\theta) = \exp(\ell_\theta(x_{t-1}^i | x_t^i, c) - \ell_{\text{old}}(x_{t-1}^i | x_t^i, c))$. Sampling only a small number of diffusion steps gives a low-cost Monte Carlo estimate of the expectation over diffusion steps, while retaining the trajectory-level reward that reflects whether the final edit is correct and optimizing complete edit trajectories with controllable compute cost.

3 Inference and Infrastructure

To support high-volume online edit-prediction requests, we build a dedicated inference engine for Diff Diffusion. Its system design follows recent work on accelerating diffusion language models through block-wise parallel decoding, approximate KV-Cache reuse, and system-level inference optimization [38, 43, 26]. In contrast to general-purpose generation settings, our target is an IDE code-editing task, where the inference engine must reduce the number of edit iterations and translate this reduction into end-to-end latency gains.

3.1 Block-wise Parallel Edit Decoding

AR models must generate the complete target snippet sequentially, whereas Diff Diffusion treats the current code snippet as an editable state. At each diffusion step, the model predicts `same`, `delete`, `insert`, `replace`, and `mask-to-token` actions in parallel. Non-mask positions are restricted to remain unchanged or special edit actions, so replacements must go through `replace` and subsequent mask filling. Block-wise decoding further partitions the rewrite range into local blocks, applies high-confidence edits within each block in parallel, and uses unchanged tokens from the original code as left and right anchors to constrain edit boundaries. In addition, insertion length is adaptive: when the inference engine assigns high confidence to an insertion region, a single insert action can emit several consecutive tokens, preventing long-completion cases from degenerating into token-by-token insertion.

3.2 Approximate KV-Cache in Editing Scenarios

We use approximate KV-Cache to reuse bidirectional cache during denoising. Diff Diffusion uses two levels of cache optimization. The first level targets stable context within a service request. Edit-prediction requests contain a system prompt, the current file, and historical edit information, so our inference engine reuses KV states for stable prefixes and converged blocks. The second level targets suffix reuse in long-sequence editing. When only a middle region needs to be modified, a large code suffix after the edit point remains unchanged. We therefore follow the idea of PIE and reuse suffix KV states after RoPE position correction [16]. This strategy is designed for local code edits with long unchanged suffixes, and in our tests it preserves accuracy while providing substantial end-to-end latency gains.

4 Experiments

We evaluate Diff Diffusion around the edit-prediction setting targeted by the method. The experiments cover four questions: whether the model learns public edit-completion patterns, whether post-training improves IDE-specific edit-prediction metrics, whether parallel edit decoding reduces inference latency, and whether the latency gains transfer to production traffic. Large-scale edit pretraining and supervised fine-tuning are run on an Ascend 910B cluster; reinforcement-learning training and evaluation are run on NVIDIA H20 GPUs. Our autoregressive baseline is the production edit-prediction model used before the diffusion cutover, trained from Seed-Coder [8]. It receives the same recent-edit history, file context, and edit range as Diff Diffusion, and generates the revised snippet from left to right. In the IDE edit-prediction evaluation, we also compare with Mercury-Coder, a diffusion-based model from Mercury [18], which provides an edit-prediction endpoint. Detailed training and inference settings are provided in Appendix C.

4.1 Data and Evaluation Setup

The training and evaluation data are summarized in Appendix B. CommitPack provides large-scale commit-level supervision for Stage I pretraining, the edit-prediction training data supports Stage II and Stage III post-training, C3PO provides an open edit-completion benchmark, and our edit-prediction evaluation set measures the IDE behaviors that matter for next-edit prediction.

4.2 Open Edit-Completion Benchmark

C3PO Edit Completion [7] asks a model to predict the remaining edit from contextual edits that have already occurred. This benchmark directly tests whether the model captures structured edit patterns rather than only token-level continuation. We report result accuracy, average decoding steps per sample, and average inference time per sample.

Table 1: Performance comparison on the C3PO edit-completion task.

Model	Result Accuracy (%)	Steps per Sample	Inference Time per Sample (ms)	Speedup Relative to AR
AR	83.18	14.62	403	—
Diff Diffusion	81.33	3.90	145	2.78×

Table 1 shows that Diff Diffusion reaches 81.33% result accuracy, close to the AR baseline, while reducing the average number of decoding steps from 14.62 to 3.90. This accuracy gap is consistent with prior findings that dLLMs benefit from larger-scale training and more optimization epochs before surpassing AR models [33, 30], while the C3PO training set is relatively small. The step reduction translates into 145 ms inference time per sample, compared with 403 ms for AR, yielding a 2.78× speedup on the public edit-completion task.

4.3 IDE Edit-Prediction Evaluation

We further evaluate on an edit-prediction set that mirrors next-edit prediction in IDEs. The examples are collected from diverse editing scenarios. Each example contains recent user edits, the current

file, and an edit range; the model must decide whether to recommend an edit and, when appropriate, generate the local code transformation. The evaluation set is organized into five subsets, each targeting a specific IDE behavior:

- **Act:** valid rewrite scenarios where recommendations should be triggered; this metric measures the proportion of correct local code transformations and reflects the model’s ability to identify and complete the next edit.
- **Stay:** quiet scenarios where no recommendation is needed; this metric measures whether the model keeps the original code unchanged and reflects its ability to suppress unnecessary suggestions.
- **Multi-line:** cross-line synchronized rewrite scenarios, such as consecutive changes to function parameters, call chains, or repeated patterns; this metric measures the model’s ability to complete multi-line edits.
- **Roll-back:** cases where the user has just made a local code change; this metric measures whether the model incorrectly reverts that recent modification and reflects its ability to preserve edit intent and history.
- **Delete:** local rewrite scenarios where surrounding context should be preserved; this metric measures unreasonable deletion of valid code and reflects the model’s ability to control edit boundaries.

Table 2: Results on the IDE edit-prediction evaluation set.

Model	Act (%) ↑	Stay (%) ↑	Multi-line (%) ↑	Roll-back (%) ↓	Delete (%) ↓
AR	45.90	91.73	68.42	9.0	1.0
Mercury-Coder	31.97	54.32	36.84	52.5	19.5
Diff Diffusion + CT	41.80	89.21	68.42	7.0	0.5
Diff Diffusion + CT + SFT	49.18	89.21	73.68	9.5	0.5
Diff Diffusion + CT + SFT + DPO	49.18	92.45	68.42	7.5	0.5
Diff Diffusion + CT + SFT + AGRPO	50.00	96.40	73.68	10.0	0.5

As shown in Table 2, the base Diff Diffusion model after CT substantially improves over Mercury-Coder on all five IDE edit-prediction metrics and already matches AR on Multi-line while reducing both Roll-back and Delete, reflecting the benefit of modeling edits as sparse transformations over mostly preserved code. Post-training then strengthens recommendation quality. SFT raises Act to 49.18% and Multi-line to 73.68%, DPO improves Stay while keeping Roll-back low, and AGRPO obtains the strongest Act and Stay scores, reaching 50.00% Act and 96.40% Stay while keeping Delete at 0.5%.

4.4 Inference Latency

We measure inference latency on the edit-prediction evaluation set. Before discussing the results, we first define the metrics reported in Table 3. Edit Range is the token span that the model may rewrite, Output Length is the average number of output tokens, Decoding Steps is the average number of decoding iterations needed for one edit, and Total Time is the average end-to-end time per sample.

For a 140-token rewrite range, diffusion decoding compresses generation from 126 AR steps to about five edit steps, reducing total time from 3964 ms to about 415–420 ms. PIE does not help this short case because both variants are memory-bound at this scale, and the extra suffix-cache bookkeeping and position correction slightly outweigh the saved computation. On the 1024-token long-rewrite setting, approximate suffix-cache reuse through PIE further lowers total time from 784 ms to 676 ms. These results show that the main efficiency gain comes from few-step parallel editing, while approximate suffix-cache reuse provides additional end-to-end savings when the unchanged suffix is long.

Table 3: Inference speed on the edit-prediction evaluation set.

Method	Edit Range (tokens)	Output Length	Decoding Steps	Total Time (ms)
AR	140	126	126	3964
Diff Diffusion	140	110	4.90	415
Diff Diffusion + PIE	140	110	5.00	420
Diff Diffusion	1024	1008	7.56	784
Diff Diffusion + PIE	1024	1012	8.78	676

4.5 Online Validation

Finally, we validate Diff Diffusion in a real IDE edit-prediction scenario with online A/B testing. Appendix D describes the relevant details. The online metrics used in this environment are as follows [52, 27]:

- **Acceptance rate:** the proportion of valid displayed edit suggestions that are accepted by users. Suggestions rejected after too short a display time and other invalid impressions are filtered out.
- **CPO (Characters per Opportunity):** the number of characters contributed by AI after each user input. We compute CPO from the edit distance before and after accepting an edit suggestion, measuring the effective rewrite amount produced by one user action.
- **Accepted lines:** the number of accepted code lines per user within the evaluation window, measuring the practical line-level code benefit obtained by users.
- **P50 model inference latency:** the median model inference latency for generating the model suggestion.

Both AR and Diff Diffusion are served through dedicated inference engines with tensor parallel size 4 and FP8 quantization, while the AR model additionally enables NGram speculative sampling to reduce latency. Table 4 reports the production edit-prediction validation results.

Table 4: Production A/B results for IDE edit prediction.

Model	Acceptance rate	CPO	Accepted lines	P50 model latency
AR	23.8%	<u>1.06</u>	93.6	116 ms
Diff Diffusion	24.0%	1.05	102.6	70.8 ms

The online results confirm that the local decoding advantage transfers to the production serving path. Diff Diffusion keeps acceptance rate and CPO comparable to AR, increases accepted lines per user by 9.6%, and reduces P50 model inference latency by 39.0%. The edit-distance breakdown in Figure 3 further shows that the latency advantage holds across different edit sizes, establishing the practical value of edit-path diffusion for interactive coding systems where recommendation quality and response time must be optimized together.

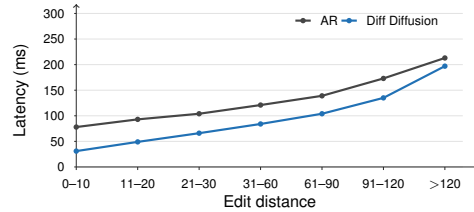


Figure 3: Latency by edit-distance bucket.

5 Discussion

This paper presents Diff Diffusion, a diffusion-inspired framework that explicitly models code edit paths. Our study shows that aligning diffusion models with the intrinsic properties of edit prediction, namely locality, incrementality, and context dependence, provides clear advantages over traditional

autoregressive methods that regenerate the entire sequence. For edit prediction, these advantages are concentrated in skipping unchanged tokens, editing multiple positions in parallel, and reducing inference latency through approximate KV-Cache reuse, PIE, and multi-token insertion. Our current limitations are that we have not yet developed a deeper theoretical analysis of edit-path diffusion, and our evaluation focuses on latency-critical IDE edit prediction. In the future, we will extend the framework to broader editing settings, including instruction-guided editing, repository-level changes, and agent workflows that combine planning, tool use, and human review.

Acknowledgments and Disclosure of Funding

This work was conducted at ByteDance and received no third-party funding or support. The authors declare no competing interests beyond their disclosed affiliation with ByteDance.

References

- [1] Aider-AI. Aider: AI pair programming in your terminal. Software repository, 2023. URL <https://github.com/Aider-AI/aider>. Accessed: 2026-09-02.
- [2] Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems*, volume 34, pages 17981–17993. Curran Associates, Inc., 2021. URL <https://proceedings.neurips.cc/paper/2021/hash/958c530554f78bcd8e97125b70e6973d-Abstract.html>.
- [3] Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, Chengxi Li, Chongxuan Li, Jianguo Li, Zehuan Li, Huabin Liu, Lin Liu, Guoshan Lu, Xiaocheng Lu, Yuxin Ma, Jianfeng Tan, Lanning Wei, Ji-Rong Wen, Yipeng Xing, Xiaolu Zhang, Junbo Zhao, Da Zheng, Jun Zhou, Junlin Zhou, Zhanchao Zhou, Liwang Zhu, and Yihong Zhuang. LLaDA2.0: Scaling up diffusion language models to 100B. *arXiv preprint arXiv: 2512.15745*, 2025.
- [4] Tiwei Bie, Jiansheng Cai, Maosong Cao, Xiang Cao, Binbin Chen, Fuyuan Chen, Kun Chen, Yuqi Ding, Lun Du, Zhuocheng Gong, Yanmei Gu, Kaiyuan Guan, Xiaoxu Guo, Kairong Han, Zenan Huang, Zijin Huang, Zhenzhong Lan, Chengxi Li, Jianguo Li, Zhe Li, Ruiqi Liang, Xinyao Lin, Huabin Liu, Lin Liu, Xinyi Liu, Xuejie Liu, Guoshan Lu, Yuan Lu, Yuxin Ma, Chenhui Mao, Xiaojing Qi, Kaida Qiu, Xiaoyu Shi, Yiding Tian, Huanyu Wang, Qi Wang, Rui Wang, Shuaidi Wang, Lanning Wei, Tao Wu, Siyang Xiao, Yipeng Xing, Chenkai Xu, Song Yan, Ying Yan, Jingyi Yang, Yichun Yang, Liangyu Zha, Tianze Zhang, Yifan Zhang, Junbo Zhao, Wenqian Zhao, Da Zheng, Jianyuan Zhong, Jiahao Zhou, Junlin Zhou, Tianyu Zhou, Liwang Zhu, and Yihong Zhuang. LLaDA2.2: Enabling agentic diffusion language models via Levenshtein editing. Technical report, 2026. URL https://github.com/inclusionAI/LLaDA2.X/blob/main/LLaDA2_2_tech_report.pdf. Released July 2026. Accessed: 2026-09-02.
- [5] Tiwei Bie, Maosong Cao, Xiang Cao, Bingsen Chen, Fuyuan Chen, Kun Chen, Lun Du, Daozhuo Feng, Haibo Feng, Mingliang Gong, Zhuocheng Gong, Yanmei Gu, Jian Guan, Kaiyuan Guan, Hongliang He, Zenan Huang, Juyong Jiang, Zhonghui Jiang, Zhenzhong Lan, Chengxi Li, Jianguo Li, Zehuan Li, Huabin Liu, Lin Liu, Guoshan Lu, Yuan Lu, Yuxin Ma, Xingyu Mou, Zhenxuan Pan, Kaida Qiu, Yuji Ren, Jianfeng Tan, Yiding Tian, Zian Wang, Lanning Wei, Tao Wu, Yipeng Xing, Wentao Ye, Liangyu Zha, Tianze Zhang, Xiaolu Zhang, Junbo Zhao, Da Zheng, Hao Zhong, Wanli Zhong, Jun Zhou, Junlin Zhou, Liwang Zhu, Muzhi Zhu, and Yihong Zhuang. LLaDA2.1: Speeding up text diffusion via token editing. *arXiv preprint arXiv: 2602.08676*, 2026.
- [6] Kevin Black, Michael Janner, Yilun Du, Ilya Kostrikov, and Sergey Levine. Training diffusion models with reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=YCWjhGrJFD>.
- [7] Shaked Brody, Uri Alon, and Eran Yahav. A structural model for contextual code changes. *Proceedings of the ACM on Programming Languages*, 4(OOPSLA):1–28, 2020. doi: 10.1145/3428283. Article 215.

- [8] ByteDance Seed, Yuyu Zhang, Jing Su, Yifan Sun, Chenguang Xi, Xia Xiao, Shen Zheng, Anxiang Zhang, Kaibo Liu, Daoguang Zan, Tao Sun, Jinhua Zhu, Shulin Xin, Dong Huang, Yetao Bai, Lixin Dong, Chao Li, Jianchong Chen, Hanzhi Zhou, Yifan Huang, Guanghan Ning, Xierui Song, Jiase Chen, Siyao Liu, Kai Shen, Liang Xiang, and Yonghui Wu. Seed-Coder: Let the code model curate data for itself. *arXiv preprint arXiv: 2506.03524*, 2025.
- [9] Federico Cassano, Luisa Li, Akul Sethi, Noah Shinn, Abby Brennan-Jones, Jacob Ginesin, Edward Berman, George Chakrnashvili, Anton Lozhkov, Carolyn Jane Anderson, and Arjun Guha. Can it edit? evaluating the ability of large language models to follow code editing instructions. In *First Conference on Language Modeling*, 2024. URL <https://openreview.net/forum?id=D06yk3DBas>.
- [10] Cursor. Tab completion. Documentation, 2026. URL <https://cursor.com/help/ai-features/tab>. Accessed: 2026-09-02.
- [11] Fangyu Ding, Ding Ding, Sijin Chen, Kaibo Wang, Peng Xu, Zijin Feng, Haoli Bai, Kai Han, Youliang Yan, Binhang Yuan, and Jiacheng Sun. Beyond masks: Efficient, flexible diffusion language models via deletion-insertion processes. *arXiv preprint arXiv: 2603.23507*, 2026.
- [12] Shansan Gong, Ruixiang Zhang, Huangjie Zheng, Jiatao Gu, Navdeep Jaitly, Lingpeng Kong, and Yizhe Zhang. DiffuCoder: Understanding and improving masked diffusion models for code generation. *arXiv preprint arXiv: 2506.20639*, 2025.
- [13] Google DeepMind. Gemini Diffusion, 2025. URL <https://deepmind.google/models/gemini-diffusion/>. Official model page (experimental). Accessed: 2025-06-21.
- [14] Jiawei Guo, Ziming Li, Xueling Liu, Kaijing Ma, Tianyu Zheng, Zhouliang Yu, Ding Pan, Yizhi LI, Ruibo Liu, Yue Wang, Shuyue Guo, Xingwei Qu, Xiang Yue, Ge Zhang, Wenhui Chen, and Jie Fu. CodeEditorBench: Evaluating code editing capability of large language models. *arXiv preprint arXiv: 2404.03543*, 2024.
- [15] Marton Havasi, Brian Karrer, Itai Gat, and Ricky T. Q. Chen. Edit flows: Flow matching with edit operations. *arXiv preprint arXiv: 2506.09018*, 2025.
- [16] Zhenyu He, Jun Zhang, Shengjie Luo, Jingjing Xu, Zhi Zhang, and Di He. Let the code LLM edit itself when you edit the code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=zqzsZ5cXbB>.
- [17] Binyuan Hui, Jian Yang, Zeyu Cui, Jiayi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-Coder technical report. *arXiv preprint arXiv: 2409.12186*, 2024.
- [18] Inception Labs, Samar Khanna, Siddhant Kharbanda, Shufan Li, Harshit Varma, Eric Wang, Sawyer Birnbaum, Ziyang Luo, Yanis Miraoui, Akash Palrecha, Stefano Ermon, Aditya Grover, and Volodymyr Kuleshov. Mercury: Ultra-fast language models based on diffusion. *arXiv preprint arXiv: 2506.17298*, 2025.
- [19] Hao Jiang, Qi Liu, Rui Li, Shengyu Ye, and Shijin Wang. CursorCore: Assist programming through aligning anything. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 27586–27623. PMLR, 2025. URL <https://proceedings.mlr.press/v267/jiang25f.html>.
- [20] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL <https://openreview.net/forum?id=VTF8yNQm66>.
- [21] Jaeyeon Kim, Lee Cheuk-Kit, Carles Domingo-Enrich, Yilun Du, Sham Kakade, Timothy Ngatiaoco, Sitan Chen, and Michael Albergo. Any-order flexible length masked diffusion. *arXiv preprint arXiv: 2509.01025*, 2025.

- [22] Chengze Li, Yitong Zhang, Jia Li, Liyi Cai, and Ge Li. Beyond autoregression: An empirical study of diffusion large language models for code generation. *arXiv preprint arXiv: 2509.11252*, 2025.
- [23] Jinsong Li, Xiaoyi Dong, Yuhang Zang, Yuhang Cao, Jiaqi Wang, and Dahua Lin. Beyond fixed: Training-free variable-length denoising for diffusion large language models. *arXiv preprint arXiv: 2508.00819*, 2025.
- [24] Jie Liu, Gongye Liu, Jiajun Liang, Yangguang Li, Jiaheng Liu, Xintao Wang, Pengfei Wan, Di Zhang, and Wanli Ouyang. Flow-GRPO: Training flow matching models via online RL. *arXiv preprint arXiv: 2505.05470*, 2025.
- [25] Xiaoran Liu, Yuerong Song, Zhigeng Liu, Zengfeng Huang, Qipeng Guo, Ziwei He, and Xipeng Qiu. LongLLaDA: Unlocking long context capabilities in diffusion LLMs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(38):32186–32194, 2026. doi: 10.1609/aaai.v40i38.40491.
- [26] Yuxin Ma, Lun Du, Lanning Wei, Kun Chen, Qian Xu, Kangyu Wang, Guofeng Feng, Guoshan Lu, Lin Liu, Xiaojing Qi, Xinyuan Zhang, Zhen Tao, Haibo Feng, Ziyun Jiang, Ying Xu, Zenan Huang, Yihong Zhuang, Haokai Xu, Jiaqi Hu, Zhenzhong Lan, Junbo Zhao, Jianguo Li, and Da Zheng. dinfer: An efficient inference framework for diffusion language models. *arXiv preprint arXiv: 2510.08666*, 2025.
- [27] Chandra Maddila, Negar Ghorbani, Kosay Jabre, Vijayaraghavan Murali, Edwin Kim, Parth Thakkar, Nikolay Pavlovich Laptev, Olivia Harman, Diana Hsu, Rui Abreu, and Peter C. Rigby. AI-assisted SQL authoring at industry scale. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 148–158. IEEE, 2025. doi: 10.1109/ICSE-SEIP66354.2025.00019.
- [28] Microsoft. Inline suggestions from GitHub Copilot in VS Code. Documentation, 2021. URL <https://code.visualstudio.com/docs/editing/ai-powered-suggestions>. Accessed: 2026-09-02.
- [29] Niklas Muennighoff, Qian Liu, Armel Randy Zebaze, Qinkai Zheng, Binyuan Hui, Terry Yue Zhuo, Swamyam Singh, Xiangru Tang, Leandro von Werra, and Shayne Longpre. OctoPack: Instruction tuning code large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL <https://openreview.net/forum?id=mw1PWNSWZP>.
- [30] Jinjie Ni, Qian Liu, Longxu Dou, Chao Du, Zili Wang, Hang Yan, Tianyu Pang, and Michael Qizhe Shieh. Diffusion language models are super data learners. *arXiv preprint arXiv: 2511.03276*, 2025.
- [31] Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv: 2502.09992*, 2025.
- [32] Shen Nie, Qiyang Min, Shaoxuan Xu, Zihao Huang, Yuxuan Song, Yong Shan, Yankai Lin, Wayne Xin Zhao, Chongxuan Li, and Ji-Rong Wen. Improved large language diffusion models. *arXiv preprint arXiv: 2606.25331*, 2026.
- [33] Mihir Prabhudesai, Mengning Wu, Amir Zadeh, Katerina Fragkiadaki, and Deepak Pathak. Diffusion beats autoregressive in data-constrained settings. *arXiv preprint arXiv: 2507.15857*, 2025.
- [34] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pages 53728–53741, 2023. doi: 10.52202/075280-2338.
- [35] Houxing Ren, Mingjie Zhan, Zimu Lu, Ke Wang, Yunqiao Yang, Haotian Hou, Junting Pan, and Hongsheng Li. Edit-based refinement for parallel masked diffusion language models. *arXiv preprint arXiv: 2605.09603*, 2026.

- [36] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022. URL https://openaccess.thecvf.com/content/CVPR2022/html/Rombach_High-Resolution_Image_Synthesis_With_Latent_Diffusion_Models_CVPR_2022_paper.html.
- [37] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv: 2402.03300*, 2024.
- [38] Yuxuan Song, Zheng Zhang, Cheng Luo, Pengyang Gao, Fan Xia, Hao Luo, Zheng Li, Yuehang Yang, Hongli Yu, Xingwei Qu, Yuwei Fu, Jing Su, Ge Zhang, Wenhao Huang, Mingxuan Wang, Lin Yan, Xiaoying Jia, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Yonghui Wu, and Hao Zhou. Seed diffusion: A large-scale diffusion language model with high-speed inference. *arXiv preprint arXiv: 2508.02193*, 2025.
- [39] Bram Wallace, Meihua Dang, Rafael Rafailov, Linqi Zhou, Aaron Lou, Senthil Purushwalkam, Stefano Ermon, Caiming Xiong, Shafiq Joty, and Nikhil Naik. Diffusion model alignment using direct preference optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8228–8238, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Wallace_Diffusion_Model_Alignment_Using_Direct_Preference_Optimization_CVPR_2024_paper.html.
- [40] Yinjie Wang, Ling Yang, Bowen Li, Ye Tian, Ke Shen, and Mengdi Wang. Revolutionizing reinforcement learning framework for diffusion large language models. *arXiv preprint arXiv: 2509.06949*, 2025.
- [41] Chenxing Wei, Jiazhen Kang, Hong Wang, Jianqing Zhang, Hao Jiang, Xiaolong Xu, Ningyuan Sun, Ying He, F. Richard Yu, Yao Shu, and Bo Jiang. LFPO: Likelihood-free policy optimization for masked diffusion models. *arXiv preprint arXiv: 2603.01563*, 2026.
- [42] Jiayi Wei, Greg Durrett, and Isil Dillig. Coeditor: Leveraging repo-level diffs for code auto-editing. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=ALVwQjZRS8>.
- [43] Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dLLM: Training-free acceleration of diffusion LLM by enabling KV cache and parallel decoding. *arXiv preprint arXiv: 2505.22618*, 2025.
- [44] Zirui Wu, Lin Zheng, Zhihui Xie, Jiacheng Ye, Jiahui Gao, Shansan Gong, Yansong Feng, Zhenguo Li, Wei Bi, Guorui Zhou, and Lingpeng Kong. DreamOn: Diffusion language models for code infilling beyond fixed-size canvases. *arXiv preprint arXiv: 2602.01326*, 2026.
- [45] Zhihui Xie, Jiacheng Ye, Lin Zheng, Jiahui Gao, Jingwei Dong, Zirui Wu, Xueliang Zhao, Shansan Gong, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream-Coder 7B: An open diffusion language model for code. *arXiv preprint arXiv: 2509.01142*, 2025. URL <https://github.com/DreamLM/Dream-Coder>. Apache-2.0 License.
- [46] Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models. *arXiv preprint arXiv: 2508.15487*, 2025.
- [47] Zebin You, Shen Nie, Xiaolu Zhang, Jun Hu, Jun Zhou, Zhiwu Lu, Ji-Rong Wen, and Chongxuan Li. LLaDA-V: Large language diffusion models with visual instruction tuning. *arXiv preprint arXiv: 2505.16933*, 2025.
- [48] Anthony Zhan. Simple policy gradients for reasoning with diffusion language models. *arXiv preprint arXiv: 2510.04019*, 2025.
- [49] Andrew Zhang, Anushka Sivakumar, Chia-Wei Tang, and Chris Thomas. Flexible-length text infilling for discrete diffusion models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 31344–31359. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.1597. URL <https://aclanthology.org/2025.emnlp-main.1597/>.

- [50] Fengqi Zhu, Rongzhen Wang, Shen Nie, Xiaolu Zhang, Chunwei Wu, Jun Hu, Jun Zhou, Jianfei Chen, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. LLaDA 1.5: Variance-reduced preference optimization for large language diffusion models. *arXiv preprint arXiv: 2505.19223*, 2025.
- [51] Fengqi Zhu, Zebin You, Yipeng Xing, Zenan Huang, Lin Liu, Yihong Zhuang, Guoshan Lu, Kangyu Wang, Xudong Wang, Lanning Wei, Hongrui Guo, Jiaqi Hu, Wentao Ye, Tiejuan Chen, Chenchen Li, Chengfu Tang, Haibo Feng, Jun Hu, Jun Zhou, Xiaolu Zhang, Zhenzhong Lan, Junbo Zhao, Da Zheng, Chongxuan Li, Jianguo Li, and Ji-Rong Wen. LLaDA-MoE: A sparse MoE diffusion language model. *arXiv preprint arXiv: 2509.24389*, 2025.
- [52] Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. Productivity assessment of neural code completion. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*, MAPS ’22, pages 21–29. ACM, 2022. doi: 10.1145/3520312.3534864.

A Related Work

Diffusion Language Models (dLLMs). Diffusion models have been widely adopted in continuous domains such as image generation [36]. Recently, discrete diffusion models have emerged as an alternative paradigm for large language models because their denoising process can update multiple positions in parallel rather than following a strict left-to-right order [2]. Masked diffusion is the dominant formulation, LLaDA and Dream scale this approach to 7B–8B parameters with performance comparable to autoregressive models, and subsequent LLaDA-family work extends it to preference alignment, long-context modeling, multimodal instruction tuning, sparse MoE, and 100B-scale training [31, 46, 50, 25, 47, 51, 3, 5, 32]. More recent work further studies variable-length denoising, insertion/deletion operations, edit-based refinement, and code infilling [43, 26, 23, 49, 15, 21, 11, 4, 44, 35]. These works show that insertion and deletion operations have become an important direction in dLLM modeling. Diff Diffusion focuses on edit-path modeling from the original snippet to the target snippet, as well as the system-level benefits of this modeling for low-latency tasks. The specific differences are shown in Table 5.

Table 5: Representative dLLMs with edit operations.

Method	Initial State	Edit Actions	Edit Task Focused	Main Goal	Production Grade
Seed Diffusion [38]	Masked sequence	Mask-to-token	Partial	Quality + latency	No
LLaDA2.1 [5]	Masked sequence	Mask-to-token, token replacement	No	Latency + quality	No
LLaDA2.2 [4]	Masked sequence	Keep, substitute, delete, insert	Partial	Self-correction	No
Edit Flows [15]	Random sequence	Insert, delete, substitute	No	Variable length	No
DID [11]	Masked sequence	Delete, insert	No	Variable length	No
DreamOn [44]	Masked span	Expand, delete, fill mask	Partial	Variable-length infilling	No
Diff Diffusion	Source span	Same, delete, insert, replace, fill mask	Yes	Edit latency + quality	Yes

Code LLMs. LLMs have become a central model family for code intelligence. Proprietary models such as Claude, GPT, and Gemini exhibit strong coding capabilities, while open-source models including Qwen2.5-Coder and Seed-Coder advance code modeling through pretraining on large-scale code corpora, instruction tuning with self-supervised data, commit histories, and evolutionary instructions, and alignment methods such as DPO and GRPO [17, 8]. Diffusion-based code models are now emerging alongside these autoregressive systems: proprietary models such as Mercury, Gemini

Diffusion, and Seed Diffusion, as well as open-source efforts such as DiffuCoder and Dream-Coder, have shown competitive performance on code-related tasks [18, 13, 38, 12, 45].

Code Editing. Code editing is an important task in AI-assisted programming, extending the classical program synthesis paradigm to enable fine-grained code modifications. It is widely integrated into popular tools such as Continue, Aider, Copilot, Cursor, and Trae and supports well-known next-edit prediction features, including Cursor Tab and Copilot NES [28, 10]. Code editing leverages various forms of software engineering knowledge, including historical edit records, user instructions, and compiler error messages. Numerous benchmarks have been proposed to evaluate model performance. Function-level and file-level benchmarks include C3PO, APEval, CanItEdit, CodeEditorBench, Aider, and HumanEvalFix [7, 19, 9, 14, 1, 29], while repository-level evaluations include CoEditor and SWE-bench [42, 20].

B Data Statistics

Table 6: Training and evaluation data.

Dataset / Benchmark	Role	Scope	Scale
CommitPack subset	Stage I edit-path pretraining	Git commit diffs in Python, C++, Java, JavaScript, Go, and Rust	87B tokens
Edit prediction training data	Stage II / III IDE post-training	High-quality next-edit examples collected from multiple scenarios	30M examples
C3PO Edit Completion	Public benchmark and SFT data	C# code edit examples from 53 GitHub projects	39K train / 5K eval examples
Edit prediction evaluation set	In-house IDE benchmark	Act, Stay, Multi-line, Roll-back, and Delete cases	122 / 278 / 19 / 200 / 200 examples each subset

Asset Licenses and Terms. We use existing assets under their stated licenses or service terms. CommitPack is distributed under the MIT License [29]. C3PO’s public code and data release is under the MIT License, with examples drawn from the C# GitHub projects listed by the benchmark authors [7]. The Dream-Coder initialization checkpoint and companion code are released under the Apache-2.0 License [45]. The AR baseline is trained from Seed-Coder, whose public release is under the MIT License [8]. Mercury-Coder is used through Inception Labs’ hosted API under its service terms [18].

C Training and Inference Hyperparameters

In our CT and SFT experiments, we use weighted cross-entropy over edit labels: ordinary token and same predictions use weight 1, while the rarer structural actions `insert`, `delete`, and `replace` use weight 5. Prompt tokens are excluded from the prediction loss. All training stages use the AdamW optimizer, gradient clipping at 1.0, and a warmup-decay learning-rate schedule. For advanced post-training, we first run a reference-only Diff-DPO pass to cache the chosen and rejected transition log-probabilities, then optimize the actor with the Diff-DPO loss using $\beta = 0.1$. AGRPO samples $G = 8$ complete edit trajectories for each prompt. When a trajectory contains more than 25 denoising steps, 25 transitions are sampled for the policy-gradient estimate, while shorter trajectories optimize all transitions. In our inference settings, the context window is set to 5.5K tokens, the maximum generation length is set to 356 tokens, and the maximum number of iterations is set to 178.

D Online A/B Experiment Setup

We conduct an online A/B experiment within an existing IDE edit-prediction feature. Eligible users are randomly assigned to persistent control and treatment groups, which receive the AR baseline and Diff Diffusion, respectively, under the same user-facing configuration. We apply consistent eligibility

and validity criteria to both groups and compute aggregate metrics over valid suggestion impressions and successful requests during the evaluation window.