Inspectable Control for Structure-Preserving Software Regeneration

Alexey Gavrilov
ITMO University
Saint-Petersburg, Russia
alexgavrilov28@gmail.com

Alan-Barsag Gazzaev
ITMO University
Saint-Petersburg, Russia
alanrbtx@gmail.com

Mikhail Mozikov
AXXX
Moscow, Russia
mozikov@axxx.tech

Ilya Makarov
AXXX
Moscow, Russia
iamakarov@itmo.ru

Sergey Muravyov
ITMO University
Saint-Petersburg, Russia
mursmail@gmail.com

Abstract

Software-engineering workflows such as constrained repair, staged refinement, and structure-preserving modification require control over what changes and what remains fixed. Token-level generation is a weak control surface for these operations because it constrains local surface text rather than the coarse structural invariants that software engineering often aims to preserve. We study hierarchical discrete latents as an inspectable intermediate representation for software artifacts: a hierarchical VQ-VAE compresses a 64-token Python function into coarse and fine discrete codes, and masked discrete generation regenerates only selected positions under partial constraints. On 2,000 preprocessed Python functions, locking four top-level codes improves parse rate from 0.453 to 0.591 while preserving substantial change in unlocked positions (edit freedom, 0.936) and near-maximal sample uniqueness (diversity, 0.998). Under fixed coarse context, lower-level refinement is weaker but remains monotonic, supporting a coarse-to-fine reading of the hierarchy. Overall, these results provide early evidence for a practical control layer that supports bounded, structure-preserving software-artifact regeneration above the token level.

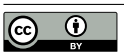
CCS Concepts

• **Software and its engineering → Automatic programming**; **Software maintenance tools**.

Keywords

controllable code generation, structure-preserving code regeneration, hierarchical discrete latent representations, latent-space control, masked discrete generation, discrete diffusion, code editing, software engineering

ACM Reference Format:
Alexey Gavrilov, Alan-Barsag Gazzaev, Mikhail Mozikov, Ilya Makarov, and Sergey Muravyov. 2026. Inspectable Control for Structure-Preserving Software Regeneration. In *34th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE Companion '26), July 05–09, 2026, Montreal, QC, Canada.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3803437.3807386



FSE Companion '26, Montreal, QC, Canada

ACM ISBN 979-8-4007-2636-1/2026/07
https://doi.org/10.1145/3803437.3807386

1 Problem Setting and Method

Many software-engineering tasks are better described as *controlled modification* than as free-form generation: a developer wants to preserve a high-level plan, regenerate only a bounded region, and inspect where change propagates [4, 5]. Surface-token interfaces make that awkward because the directly manipulable object is text.

We therefore study hierarchical discrete latents as an *inspectable control layer* for software artifacts. Our system combines a hierarchical VQ-VAE with masked discrete regeneration [1–3]. Each 64-token Python function is encoded into 16 top-level codes and 32 lower-level codes. Control is exposed through *lock-and-regenerate*: selected latent positions are frozen, the rest are regenerated, and the decoder reconstructs the final function. Developers therefore do not edit embeddings directly; they choose which latent positions remain fixed. We intentionally study a narrow setting—short Python functions, argmax decoding, and structural proxies—to isolate controllability before broader evaluations.

Contributions. (1) We introduce an inspectable hierarchical latent control layer for code. (2) We show that freezing coarse latent structure improves parse rate while preserving substantial freedom in editable regions. (3) We demonstrate a controllability–freedom trade-off that enables bounded, structure-preserving regeneration above the token level.

2 Experimental Setting and Main Results

We evaluate on 2,000 Python functions from a preprocessed CodeParrot Clean subset. All diagnostics use argmax decoding. *Parse* is the fraction of outputs accepted by the Python parser; *Skeleton* measures coarse program-form preservation; *Signature* tracks the function-interface region; *Unlocked change* is the fraction of editable positions that change; and *Diversity* is sample uniqueness under repeated regeneration. Pipeline checks confirm exact recovery under full locking and perfect preservation of locked positions.

Table 1 shows the main top-level result. Partial latent locking improves parse rate over unconditional generation while preserving non-trivial freedom in editable regions. Locking four prefix codes raises parse from 0.453 to 0.591; locking the latent span aligned to the function signature reaches 0.600. Unlocked change

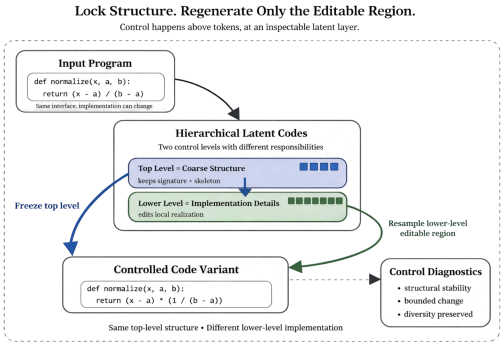


Figure 1: Encode to discrete codes, lock selected coarse positions, regenerate only editable positions, then inspect structural stability and scope of change.

Table 1: Top-level controlled regeneration. "Sig.-span" locks the code prefix covering the function-signature token span.

Setting	Parse	Skeleton	Signature
Codec reconstruction	0.857	0.848	0.493
Unconditional generation	0.453	0.080	0.000
Conditional, prefix k=4	0.591	0.295	0.061
Conditional, sig.-span	0.600	0.302	0.063

Table 2: Why operate above tokens?

Capability	Token-level	Latent control
Freeze coarse structure	limited	native
Partial regeneration	fragile	native
Inspectable control points	no	yes
Structured drift diagnosis	weak	direct

remains 0.936 and conditional diversity remains 0.998. Exact signature preservation stays low, which suggests that the earliest top-level codes capture coarse structural regularities more strongly than exact lexical interfaces.

A prefix sweep shows a clean controllability–freedom trade-off. At k=0, the conditional model matches the unconditional baseline (parse 0.460; unlocked change 0.995). At k=8, parse rate rises to 0.675 while 86.7% of unlocked positions still change. At k=16, the process reduces to codec reconstruction (parse 0.857; unlocked change 0.0). Lower-level diffusion under fixed top-level context is weaker but remains monotonic under locking, supporting a coarse-to-fine reading: the top level is the stronger structural control point, while the lower level supports bounded implementation refinement.

3 Implications and Limits

The main implication is practical: hierarchical discrete latents can act as an inspectable control layer above tokens. They expose explicit control points that improve structural stability during regeneration while preserving freedom in editable regions, which is closer to bounded modification than to unconstrained code generation. This is the main benefit beyond parse rate alone: exact preservation of locked positions, bounded scope of change, and diagnostics that make the control surface inspectable.

The current evidence is intentionally narrow. We do not yet evaluate downstream correctness, refactoring utility, or optimized wall-clock latency; nor do we claim semantic equivalence. The present poster therefore establishes *structural controllability* rather than full functional preservation. Future work should evaluate larger-scale settings and broader software-engineering tasks such as bounded repair and structure-preserving refactoring, while also assessing practical properties such as downstream correctness and latency.

Acknowledgments

This research is financially supported by the Foundation for National Technology Initiative's Projects Support as a part of the roadmap implementation for the development of the high-tech field of Artificial Intelligence for the period up to 2030 (agreement 70-2021-00187).

References

[1] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. 2019. Generating Diverse High-Fidelity Images with VQ-VAE-2. In *Advances in Neural Information Processing Systems*.

[2] Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. 2021. Structured Denoising Diffusion Models in Discrete State-Spaces. In *Advances in Neural Information Processing Systems*.

[3] Subham Sekhar Sahoo, Marianne Arriola, Aaron Gokaslan, Edgar Mariano Marroquin, Alexander M. Rush, Yair Schiff, Justin T. Chiu, and Volodymyr Kuleshov. 2024. Simple and Effective Masked Diffusion Language Models. In *Advances in Neural Information Processing Systems*.

[4] Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. Grounded Copilot: How Programmers Interact with Code-Generating Models. *Proceedings of the ACM on Human-Computer Interaction* 7, CSCW1 (2023), Article 78.

[5] Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. 2023. RepoCoder: Repository-Level Code Completion Through Iterative Retrieval and Generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*.