Diagnosing and Fixing Encoder Collapse in Tiny JEPA Code Transition Models

Eren Akbulut

Abstract

Tiny JEPA-style code transition models trained on chains of Git edits exhibit a mysterious ∼700-step training ceiling. We identify the failure mode—target encoder collapse under fast EMA tracking—and diagnose it with a cheap *copy-baseline* indicator, $\cos(f_{\bar{\theta}}(s_t), f_{\bar{\theta}}(s_{t+1}))$, that climbs to ∼0.999 well before standard loss curves degrade. The fix is to hold the target encoder approximately static; we verify this with two equivalent settings—near-frozen EMA decay 0.99999 (default, three seeds) and fully frozen decay 1.0 (a random-initialised target that never updates, two seeds)—that agree on validation transition cosine to within ±0.002, so target staticity is the lever, not a specific EMA decay value. With the fix in place, a 1.1 M-parameter 128-dimensional model trains stably for 15,000 steps to a mean-of-last-five validation transition cosine of 0.9898 ± 0.0006 and performs compositional three-step prediction with per-step delta cosines ≈0.98, more than 0.9 above every delta-space null we evaluate; the same property transfers to edit chains from nine held-out Python repositories. A secondary retrieval experiment shows the model is competitive at 112× fewer parameters on in-distribution CommitPackFT edit retrieval and on a 3,998-pair twenty-repo leave-one-repo-out sweep, while losing cleanly to modern dense encoders out-of-distribution. A zero-dependency Rust inference engine with four-level weight quantisation compresses the deployed model to 0.6 MB and a 2.6 MB in-browser WebAssembly bundle.

1 Introduction

Software repositories record a rich dynamical process: each commit transforms a codebase from one state to the next through a discrete editing action. Viewed through this lens, a Git history is a partially-observable trajectory in program space—a sequence of state–action–state triples (s_t, a_t, s_{t+1}) where the state s_t is an abstract syntax tree (AST) snapshot and the action a_t encodes the type, scope, and location of an edit.

Existing code representation models learn from *static snapshots* rather than *transitions*. CodeBERT [5], GraphCodeBERT [7], and UniXcoder [8] produce contextual embeddings of individual code fragments; code-change models such as CC2Vec [9], CCT5 [12], CCRep [13], and CoditT5 [19] produce static embeddings of diffs—but none provide an action-conditioned *predictor* that can forecast the latent state of the post-edit code. The earlier edit-representation work of Yin et al. [18] comes closest in spirit, but also does not offer an action-conditioned transition operator.

Can a tiny JEPA-style [1] latent transition model do this job? Our initial attempts suggested the answer was no: training consistently peaked around step 700 and then degraded, and standard JEPA diagnostics (prediction loss curves, direction-loss ablations, spectral and covariance regularisation) did not reveal why.

This paper is about that failure mode and its fix. We identify JEPA target-encoder collapse under fast EMA tracking as the dominant driver of the ceiling, introduce a cheap leading-indicator metric that catches it before the loss curves do, and show that *any sufficiently static target encoder*

is sufficient to mitigate the collapse over training horizons at least 50× longer than the standard configuration supported. We demonstrate this with two equivalent settings: near-freezing via EMA decay 0.99999 (our default, three-seed replicated) and a fully frozen random-init target at decay 1.0 (two-seed ablation), which reproduce the same val transition cosine within ±0.002. After the fix, a 1.1M-parameter model performs short-horizon compositional multi-step prediction—a capability that no embedding-only baseline can reproduce because none of them provide an action-conditioned transition operator—and the same compositional property transfers to edit chains harvested from nine held-out Python repositories, none of which appeared in training. We also study code edit retrieval as a secondary experiment and report its full multi-task picture: at 1.1M parameters CodeWM beats a 124M-parameter CodeBERT baseline on an in-distribution edit retrieval benchmark and on a 20-repository leave-one-repo-out cross-repository retrieval benchmark, while tying bag-of-AST-tokens on both and losing cleanly to modern code-specific dense encoders on an out-of-distribution function-level benchmark. Finally, we describe a production-quality zero-dependency Rust inference engine, four-level weight quantisation, and an in-browser WebAssembly deployment.

Contributions.

1. **Copy-baseline: a leading-indicator diagnostic for JEPA encoder collapse.** A single-scalar validation metric copy_baseline $= \mathbb{E}_t[\cos(f_{\bar{\theta}}(s_t), f_{\bar{\theta}}(s_{t+1}))]$ that climbs toward ∼0.999 *well before* the prediction loss or multi-step cosines degrade, catching target-encoder collapse that standard JEPA loss curves miss (Section 4.2).

2. **Static target encoders as a mitigation.** Among the interventions we tested, only holding the target encoder approximately static eliminated the collapse on code trajectory data. We report two equivalent settings—near-frozen EMA decay 0.99999 (default, three-seed replicated) and fully frozen decay 1.0 (two-seed ablation)—that agree on validation transition cosine within ±0.002, so target staticity is the lever, not a specific EMA value (Section 4.2).

3. **Tiny compositional latent edit predictor.** A 1.1 M-parameter, 128-dimensional model reaches mean-of-last-5 validation transition cosine 0.9898 ± 0.0006 and performs three-step compositional prediction with per-step delta cosines ≈0.98, more than 0.9 above every delta-space null we evaluate (trivial identity, shuffled action, random trajectory, mean-delta constant predictor, within-trajectory self-consistency); the property transfers out-of-distribution to nine held-out Python repositories (Section 4.3).

4. **Secondary: multi-task retrieval tradeoff map.** A supervised contrastive head on deltas [11] makes CodeWM competitive on in-distribution CommitPackFT edit retrieval and on a twenty-repository cross-repository sweep at 112× fewer parameters than CodeBERT, with an honest hard-negative K=9 caveat and a clean loss to modern dense encoders (codet5p, UniXcoder, BGE, jina-code) on out-of-distribution CodeSearchNet [10] (Section 4.4).

5. **Deployment.** A zero-dependency Rust inference engine reproduces the PyTorch reference within floating-point tolerance, four-level quantisation compresses the model to 0.6 MB with cosine fidelity ≥0.9999, and a 2.6 MB WebAssembly bundle delivers sub-200 ms in-browser queries entirely client-side (Section 5).

Scope and limitations. The title uses "code transition model" to avoid overclaiming: throughout the paper we use "world model" and "CodeWM" in the limited technical sense of an action-conditioned latent transition model, without yet evaluating downstream predictive utility such as

anomaly detection, change-impact estimation, or refactoring suggestion—earning the full "world model" label on a real downstream task is the principal outstanding validation (Section 6). The 7-dimensional action vector used by the contrastive head is deliberately minimal and carries a structural similarity to the `by_joint` retrieval relevance criterion that we discuss in Section 4.4; a richer 15-dimensional AST-diff action variant is the natural mitigation. All modern dense-encoder baselines are public pretrained checkpoints *without* fine-tuning on CommitPack; a fine-tuned modern encoder would likely close or reverse the external-benchmark gap.

2 Problem Formulation

2.1 State and Action Spaces

Let $\mathcal{V}$ denote a vocabulary of AST token types. During training we parse Python source with Python's built-in `ast` module and produce a flat token sequence by depth-first traversal of the resulting tree; at inference time the Rust engine reproduces the same tokens using `rustpython-parser` followed by an FNV-1a hash on identifier strings, with cross-implementation equivalence verified by the zero-drift protocol (Section 5.2). The vocabulary has $|\mathcal{V}| = 662$ entries, composed of 100 concrete AST node-type slots (`Module`, `FunctionDef`, `Assign`, ...), 512 identifier hash buckets, and 50 control tokens (16 depth markers, 29 operator symbols, and `PAD/BOS/EOS/UNK/PARSE_ERROR`). Hash collisions between distinct identifiers are expected: 512 buckets are a deliberate design choice, trading some lexical specificity for a compact vocabulary; we find empirically that collisions are close to uniformly distributed and do not concentrate on semantically distinct identifiers in the same function.

This tokenisation is a deliberate design choice rather than a novelty claim. It is intended to bias the model toward structural program change (node-type and depth tokens dominate the sequence) rather than surface lexical variation; it keeps the vocabulary compact enough for a 1.1M-parameter encoder to fit a full $|\mathcal{V}| \times 128$ embedding table in under 0.5 MB; and it admits near-exact numerical parity between the Python training pipeline (built-in `ast` module) and the Rust deployment stack (`rustpython-parser` with FNV-1a identifier hashing), verified by the zero-drift protocol in Section 5.2. We do not isolate the contribution of the tokeniser from the rest of the architecture; see Section 6 for the ablation this would require.

A *code state* $s \in \mathcal{S}$ is a variable-length sequence of tokens from $\mathcal{V}$, truncated or padded to a maximum length S. The *action space* is $\mathcal{A} = \mathbb{R}^7$, encoding each edit with three one-hot blocks plus one scalar coordinate:

- $a_{[0:3]}$ — edit type (ADD, DELETE, MODIFY), one-hot;
- $a_{[3:6]}$ — syntactic scope (function, class, module), one-hot, determined by comparing the function and class name sets of the pre- and post-edit AST;
- $a_{[6]}$ — location, the normalised byte offset of the first differing character between the pre- and post-edit sources, in $[0, 1]$.

The 7-dimensional encoding is deliberately minimal; it is an observed limitation rather than a design claim (Section 6).

A single data point is a triple (s, a, s') where s is the pre-edit state, $a \in \mathcal{A}$ is the edit action, and s' is the post-edit state. For training we additionally use short chains $(s_0, a_0, s_1, a_1, \ldots, s_T)$ of consecutive edits to the same file, which let us supervise multi-step composition (Section 2.5).

2.2 Model Components

The model comprises three learned components:

Encoder. $f_\theta \colon \mathcal{S} \to \mathcal{Z} \subset \mathbb{R}^{128}$ maps a code state to a latent vector. The encoder is a weight-shared looped transformer (Section 3) with CLS-token pooling and terminal LayerNorm. Following standard JEPA practice [1], a target encoder $f_{\bar{\theta}}$ with parameters $\bar{\theta}$ is maintained as an exponential moving average (EMA) of θ. The EMA decay rate is the critical stability knob in our setting and is discussed in Section 2.4.

Action encoder. $h_\psi \colon \mathcal{A} \to \mathcal{Z}$ projects the 7-dimensional action into the same latent space via a two-layer MLP:

$$h_\psi(a) = W_2 \, \mathrm{GELU}(W_1 a + b_1) + b_2, \quad W_1 \in \mathbb{R}^{128\times 7}, \; W_2 \in \mathbb{R}^{128\times 128}. \tag{1}$$

Predictor. $g_\phi \colon \mathcal{Z} \times \mathcal{Z} \to \mathcal{Z}$ takes the stacked state–action pair (z, z_a) and predicts the next-state latent. The predictor is a two-block transformer stack, each block looped 6 times (Section 3).

2.3 Training Objective

Given a triple (s, a, s'), let $z = f_\theta(s)$, $z_a = h_\psi(a)$, $z' = f_{\bar{\theta}}(s')$ (target encoder, with stop-gradient), and $\hat{z}' = g_\phi(z, z_a)$. The training loss combines four terms:

$$\mathcal{L} = \mathcal{L}_{\text{pred}} + \lambda_{\text{dir}} \mathcal{L}_{\text{dir}} + \lambda_{\text{spec}} \mathcal{L}_{\text{spec}} + \lambda_{\text{con}} \mathcal{L}_{\text{con}}. \tag{2}$$

The final term is used only for the retrieval-oriented variant (Section 4.4).

Prediction loss. The primary objective maximises the cosine similarity between the predicted latent and the stop-gradient target:

$$\mathcal{L}_{\text{pred}} = 1 - \cos\big(\hat{z}', \; \mathrm{sg}[z']\big), \tag{3}$$

where $\mathrm{sg}[\cdot]$ denotes the stop-gradient operator applied to the target encoder output.

Direction loss. To encourage the predictor to learn a meaningful *displacement* rather than collapsing to the identity, we add a directional consistency term. Let $\delta_{\text{pred}} = \hat{z}' - z$ and $\delta_{\text{true}} = \mathrm{sg}[z'] - z$ be the predicted and ground-truth transition vectors. Then:

$$\mathcal{L}_{\text{dir}} = 1 - \cos\big(\delta_{\text{pred}}, \; \delta_{\text{true}}\big). \tag{4}$$

Empirically, the direction loss is not merely an auxiliary signal but the structural *scaffold* that holds the transition geometry in place: ablations that disable $\mathcal{L}_{\text{dir}}$ mid-training cause the predicted directional cosine to collapse to ≈ -0.04 within 50 steps (Section 4.2).

Batch spectral regularisation. Related in spirit to the variance/covariance term in VICReg [2], we discourage rank deficiency in the mini-batch embedding matrix by penalising the log-singular-values of a whitened batch projection. We retain this term as a mild anti-rank-collapse prior, but the sweep in Section 4.2 indicates that EMA decay, not spectral weighting, is the dominant stability

lever in this regime. Let $Z \in \mathbb{R}^{B\times 128}$ be the matrix of encoder outputs for a mini-batch of size B and let $\sigma_1, \ldots, \sigma_{128}$ be its singular values. The spectral penalty is:

$$\mathcal{L}_{\text{spec}} = -\sum_{i=1}^{128} \log \sigma_i(Z). \tag{5}$$

In our code implementation this term is called `SIGReg`; we adopt the "batch spectral regularisation" terminology here to match the published literature. In isolation, this regulariser is *not* sufficient to prevent the failure mode we describe next—stronger regularisation (Section 4.2) does not help.

Supervised contrastive head (retrieval variant). For the retrieval-oriented variant of the model, we add a supervised contrastive loss [11] on the predicted deltas, using the pairwise cosine similarity between action vectors as the supervision signal. Concretely, for each mini-batch we form $\delta_i = \hat{z}'_i - z_i$ and define positives as pairs (i, j) with $\cos(a_i, a_j) > \tau_{\text{pos}}$ (default $\tau_{\text{pos}} = 0.9$):

$$\mathcal{L}_{\text{con}} \;=\; -\frac{1}{|\mathcal{P}|} \sum_{(i,j)\in\mathcal{P}} \log \frac{\exp\big(\cos(\delta_i, \delta_j)/\tau\big)}{\sum_{k\neq i} \exp\big(\cos(\delta_i, \delta_k)/\tau\big)}, \tag{6}$$

with temperature $\tau = 0.07$ and $\mathcal{P}$ the set of positive pairs. The effect of this term on the prediction–retrieval trade-off is quantified in Section 4.4.

Action-vector / relevance circularity (flagged early). Because the 7-dimensional action vector is dominated by six one-hot dimensions (edit type × scope), a cosine > 0.9 between two action vectors is *structurally close* to the `by_joint` retrieval relevance criterion used in Section 4.4 (same edit type *and* same scope). This means the contrastive head is supervised by a signal that partially anticipates the retrieval target, and any retrieval improvement from $\mathcal{L}_{\text{con}}$ must be read with this circularity in mind. We surface this up front here, present the numbers honestly in Section 4.4, and flag the richer 15-dimensional AST-diff action as the natural mitigation in Section 6.

2.4 Encoder Collapse and the Static-Target Fix

The central stability challenge we observed for JEPA-style training in this compact regime on code trajectory data is *target encoder collapse*: after a few hundred steps the target encoder's output space contracts toward a near-constant point, and the direction-loss gradient carries little information. We isolated this behaviour empirically at the 1–2 M-parameter scale on CommitPack Python trajectories; we do not present it as a formal causal claim, and we do not argue that it generalises unchanged to larger models, different languages, or different data distributions. We track collapse with a single cheap diagnostic,

$$\text{copy\_baseline} \;=\; \mathbb{E}_t\big[\cos\big(f_{\bar{\theta}}(s_t),\, f_{\bar{\theta}}(s_{t+1})\big)\big], \tag{7}$$

computed on held-out consecutive states. For an uncollapsed encoder applied to real code edits copy_baseline sits in roughly $[0.65, 0.90]$; when the encoder collapses it climbs rapidly toward 1.0, and we observe empirically that once it exceeds ≈ 0.98 subsequent training never recovers.

Among the interventions we tested, the stability of the target encoder's representation over the training horizon is the most effective lever. The observed behaviour is strongly associated with fast target-encoder tracking: at the standard JEPA default decay of 0.996 the target encoder tracks the online encoder closely enough that, once the latter starts drifting toward a trivial representation,

the former follows within 250–700 steps and copy_baseline climbs to the collapse plateau. Slowing the decay to 0.9995 quadruples the stable training horizon; *near-freezing* the target encoder at a decay of 0.99999 prevents the collapse over runs of at least 15,000 steps and across all three random seeds we tested (mean-of-last-5 transition cosine 0.9898 ± 0.0006 within each run, peak across seeds 0.9934 ± 0.0015). A direct ablation that takes the same recipe to its static limit—$\bar{\theta} \leftarrow \bar{\theta}$ (decay 1.0), so that $f_{\bar{\theta}}$ stays at a `deepcopy` of the randomly initialised online encoder for all 15,000 steps—matches the near-frozen baseline within ± 0.002 val transition cosine across two independent seeds (Section 4.2), showing that the effective stabiliser is *target staticity* rather than a specific EMA decay value. Stronger spectral, covariance, or VICReg regularisation at the weights we swept did not substitute for this fix on our data (Section 4.2); whether the same holds for larger weights, other datasets, or other JEPA regimes is beyond the scope of this paper.

All main-body models reported in the remainder of this paper use $\bar{\theta} \leftarrow 0.99999\,\bar{\theta} + 0.00001\,\theta$ as the target update rule, unless a result is explicitly marked as using the frozen-target variant (decay 1.0) in Section 4.2 or Section 4.4.

2.5 Dataset

Training data is drawn from CommitPack [14], filtered for Python-only commits with single-file diffs. Each (s, a, s') triple is constructed by (i) parsing the pre- and post-edit file into AST token sequences using `rustpython-parser` with FNV-1a hashing into the 662-entry vocabulary, and (ii) extracting the 7-dimensional action vector from the diff metadata (edit type, syntactic scope, and normalised location).

For training the compositional variant we additionally chain consecutive edits to the same file by matching AST token hashes across commits, yielding 16,941 trajectory sequences from 1.5 M underlying edits. The trajectory split is used for multi-step training and prediction evaluation but *not* for retrieval evaluation: because chains of edits to the same file inflate pairwise self-similarity, retrieval numbers on trajectory data are optimistically biased. We therefore evaluate retrieval on a separate, diverse 2,000-sample split drawn from CommitPackFT [14] (Section 4.4).

3 Architecture

The Code World Model consists of three subnetworks—encoder, action encoder, and predictor—depicted in Figure 1. All transformer blocks share the same internal structure: pre-norm multi-head attention followed by a pre-norm feed-forward network, with GELU activations [3] and residual connections. A target encoder $f_{\bar{\theta}}$ with near-frozen EMA decay (0.99999) provides the prediction target; the choice of decay rate is the critical stability lever and is discussed in Section 2.4. Full hyperparameters are listed in Appendix A.

3.1 Weight-Shared Looped Encoder

The encoder f_θ maps a variable-length AST token sequence to a single 128-dimensional latent vector. Its forward pass proceeds as follows:

1. **Embedding.** Each token index is mapped through a learned embedding table $E \in \mathbb{R}^{662 \times 128}$. A learnable CLS token is prepended to the sequence, yielding $X_0 \in \mathbb{R}^{(S+1) \times 128}$.

2. **Positional encoding.** Learned positional embeddings $P \in \mathbb{R}^{(S_{\max}+1) \times 128}$ (with $S_{\max} = 512$) are added element-wise: $X_0 \leftarrow X_0 + P_{:S+1}$.

3. **Looped transformer.** A *single* transformer block $\mathcal{T}_{\text{enc}}$ is applied iteratively for $L = 6$ loops:

$$X_\ell = \mathcal{T}_{\text{enc}}(X_{\ell-1}), \quad \ell = 1, \ldots, 6. \tag{8}$$

   Because the same weights are reused at every iteration, the encoder has the representational depth of a 6-layer transformer with the parameter count of a single layer.

4. **Pooling and normalisation.** The CLS token representation is extracted from X_6 and passed through a final LayerNorm to produce the output latent $z = \text{LayerNorm}(X_6[0]) \in \mathbb{R}^{128}$.

Transformer block. Each block $\mathcal{T}$ contains:

- **Multi-head self-attention** with 4 heads and head dimension 32 (total attention dimension $4 \times 32 = 128$), using pre-LayerNorm.
- **Feed-forward network** with expansion factor 4×: $\text{Linear}(128, 512) \to \text{GELU} \to \text{Linear}(512, 128)$, also preceded by LayerNorm.

Both sub-layers use residual connections: $\tilde{X} = X + \text{Attn}(\text{LN}(X))$ and $X' = \tilde{X} + \text{FFN}(\text{LN}(\tilde{X}))$.

3.2 Action Encoder

The action encoder h_ψ is a lightweight two-layer MLP (Eq. 1) that projects the 7-dimensional action vector into the 128-dimensional latent space:

$$h_\psi(a) = W_2 \ \text{GELU}(W_1 a + b_1) + b_2. \tag{9}$$

The first layer $\text{Linear}(7, 128)$ lifts the action into the model dimension; the second layer $\text{Linear}(128, 128)$ provides a non-linear mixing stage. The action encoder has fewer than 18K parameters and runs in under 1 ms even in the WASM backend.

3.3 Predictor

The predictor g_ϕ takes the state latent z and action latent $z_a = h_\psi(a)$ and produces the predicted next-state latent $\hat{z}'$. Concretely:

1. **Stacking.** The two 128-dimensional vectors are stacked into a 2-token sequence: $H_0 = \text{stack}(z, z_a) \in \mathbb{R}^{2\times 128}$.

2. **Looped dual-block transformer.** Two *distinct* transformer blocks $\mathcal{T}_{\phi,0}$ and $\mathcal{T}_{\phi,1}$ are each applied for $L = 6$ loops in sequence:

$$H_\ell^{(0)} = \mathcal{T}_{\phi,0}(H_{\ell-1}^{(0)}), \qquad \ell = 1, \ldots, 6, \quad H_0^{(0)} = H_0, \tag{10}$$

$$H_\ell^{(1)} = \mathcal{T}_{\phi,1}(H_{\ell-1}^{(1)}), \qquad \ell = 1, \ldots, 6, \quad H_0^{(1)} = H_6^{(0)}. \tag{11}$$

3. **Readout.** The first token of the final output is extracted and normalised: $\hat{z}' = \text{LayerNorm}(H_6^{(1)}[0])$.

The predictor's two-block, 6-loop-each design gives it 12 effective layers of depth with only two parameter-distinct blocks. This asymmetry relative to the single-block encoder reflects the harder task of predicting state transitions versus encoding static states. As in DiT [15], the stacking mechanism allows cross-attention between the state and action tokens at every loop iteration without requiring explicit cross-attention layers.

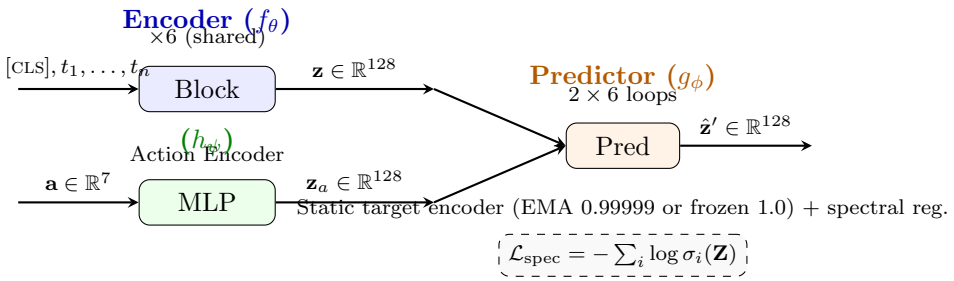

Figure 1: Architecture of the Code World Model. The encoder f_θ maps AST-token sequences through 6 weight-shared transformer blocks to a 128-dimensional CLS embedding. The action encoder h_ψ projects the 7-dimensional edit descriptor. The predictor g_ϕ forecasts the next-state latent. A *static* target encoder (EMA decay 0.99999 as the default, or the fully frozen decay 1.0 ablation in Section 4.2) combined with a batch spectral regulariser prevents representation collapse; both variants produce equivalent val $\Delta_{\cos}$ within ± 0.002.

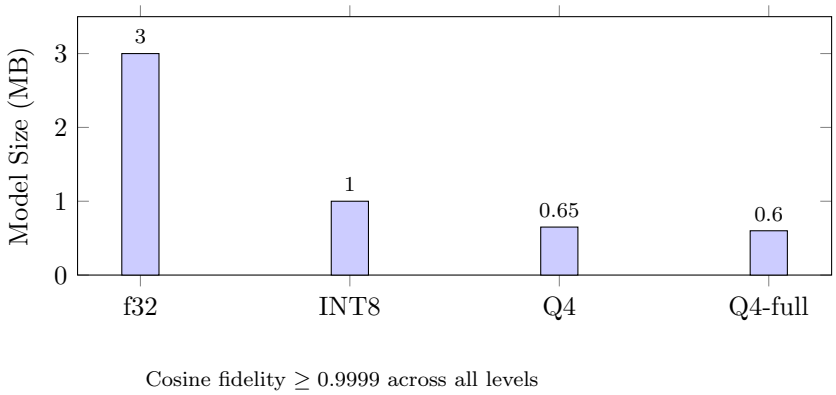

Figure 2: Quantisation ladder. Model size decreases from 3.0 MB (f32) to 0.6 MB (Q4-full, 5× compression) while cosine fidelity remains ≥ 0.9999 relative to the full-precision baseline.

4 Experiments

We evaluate the model in four blocks: trivial baselines (§4.1); the encoder-collapse diagnosis and static-target fix (§4.2); single-step and multi-step prediction with 3-seed variance, null-model analysis, and an out-of-distribution rollout on nine held-out Python repositories (§4.3); and a multi-task retrieval tradeoff map (§4.4). Quantisation and inference latency are reported in §4.6.

Checkpoint families. We evaluate two families of 15K-step checkpoints throughout Sections 4.2–4.4, corresponding to two rounds of the training protocol. *Phase 3* refers to the original single-seed 15K champion (EMA-FROZEN-15K SEED 42), used in the prediction headline and null-model analysis. *Phase 5* refers to the multi-seed replication protocol that added a second non-contrastive predictor seed (PRED_S43, i.e. EMA-FROZEN-15K SEED 43), two contrastive seeds (CON_S42, CON_S43, both with $\lambda_{\text{con}} = 1.0$), and the two frozen-target ablation seeds (FT_S42, FT_S43) of Section 4.2. Phase 3 and Phase 5 share the same 128-dimensional architecture, trajectory training data, and default near-frozen EMA recipe; Phase 5 only adds seed replication and the contrastive / frozen-target ablations.

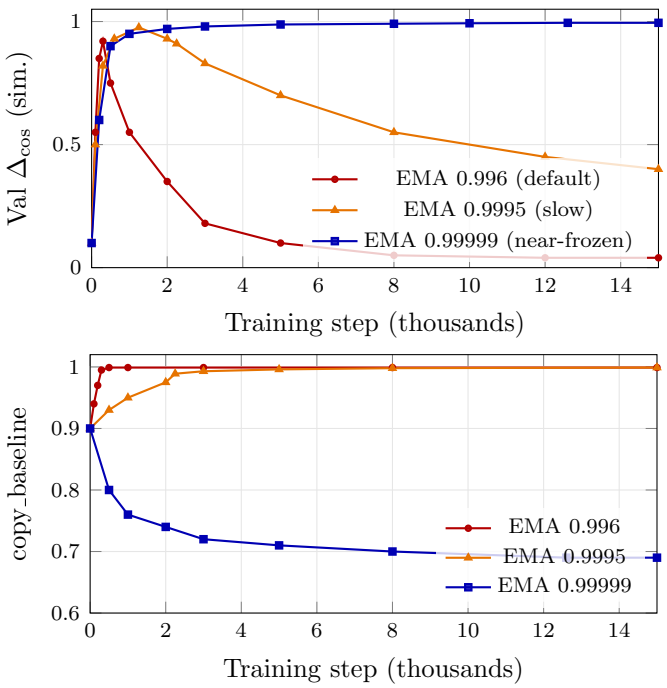

Figure 3: The encoder-collapse fix. **Left**: validation transition cosine $\Delta_{\cos}$ as training progresses under three target encoder EMA decay rates. At the default JEPA decay of 0.996, the encoder collapses by step ∼300 and $\Delta_{\cos}$ decays monotonically. Slowing the decay to 0.9995 extends the stable horizon by roughly 4× before late drift sets in. Near-freezing the target encoder at a decay of 0.99999 eliminates the collapse entirely over 15 000 steps, reaching $\Delta_{\cos} = 0.9948$ at step 12 600. A fully frozen decay-1.0 ablation (not plotted; Section 4.2) matches the 0.99999 curve within ±0.002 across two seeds, confirming that *any* sufficiently static target suffices and that the specific 0.99999 value is one practical knob, not the only one. **Right**: the copy_baseline diagnostic (Eq. 7) tracks encoder collapse as a *leading* indicator—it saturates to 0.999 well before $\Delta_{\cos}$ degrades in the 0.996 run, while remaining healthy at ∼0.69 throughout the near-frozen run.

Reporting convention. Throughout, $\Delta_{\cos}$ denotes the *cosine similarity* between the predicted and ground-truth transition vectors, $\Delta_{\cos} = \cos(\delta_{\text{pred}}, \delta_{\text{true}}) \in [-1, 1]$. Higher is better; $\Delta_{\cos} = 1$ is perfect directional agreement and a collapsed predictor that outputs the identity ($\delta = 0$) yields $\Delta_{\cos} \approx 0$. All headline numbers are reported as *mean of the last five validation evaluations* of a run, ± sample standard deviation across those evaluations. When 3 seeds (42, 43, 44) are available, we additionally report mean ± standard deviation across seeds. We avoid *peak* reporting because it overstates prediction quality on short contrastive runs by ∼0.5 points; peak numbers are listed in the appendix for transparency only. Latency numbers are collected on Apple M-series silicon at f32 precision with 100 warm-up iterations.

4.1 Baselines

Edit-type classification is not a useful benchmark at our data scale: a bag-of-AST-tokens k-nearest-neighbour classifier achieves 99.5% edit-type accuracy and 76.3% joint (edit-type × scope) accuracy on the 500K CommitPack Python split, and a TF-IDF + logistic regression baseline is essentially equivalent. We therefore do not report classification as a contribution and move directly to the stability and compositional-prediction results that separate the learned representation from trivial

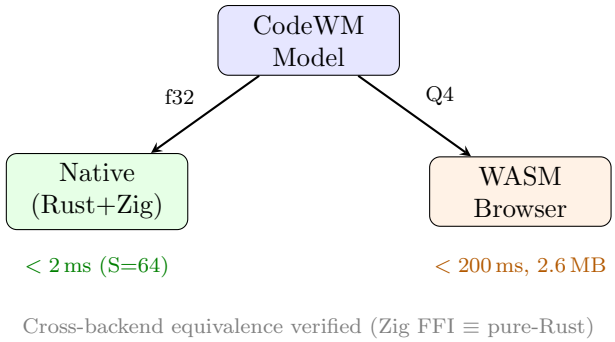

Figure 4: Deployment landscape. A single trained model is served across two targets—native CPU with SIMD acceleration, and browser via WebAssembly—verified to produce identical outputs within floating-point tolerance.

Table 1: Effect of target-encoder staticity on training horizon. All runs use the same 128-dimensional backbone, trajectory training data, and loss weights; only the target update rule differs. $\Delta_{\cos}$ is reported as mean of the last five validation evaluations, except for the frozen-target ablation row (marked †), where only peak was logged; see Appendix A and Table 2. The final row is the frozen-target ablation (EMA decay 1.0): the target encoder is a `deepcopy` of the randomly-initialised online encoder and never receives an update. It matches the near-frozen EMA 0.99999 baseline within ±0.002, so *target staticity* is load-bearing, not the specific EMA decay value.

EMA decay	Steps	$\Delta_{\cos}$ (mean-last-5)	Copy_b. (end)	Status
0.996 (default)	3 000	∼0.10 (collapsed)	0.999	Collapsed by step ∼300
0.9995	2 250	0.9356 ± 0.0200	0.989	Late drift
0.99999	3 000	0.9780 ± 0.0014	0.764	Stable
0.99999	15 000	$\mathbf{0.9898 \pm 0.0006}$	0.69	Stable (default recipe)
1.0 (frozen)†	15 000	**0.9925–0.9938** (peak)	0.69–0.71	Equivalent to 0.99999 (±0.002)

† *Frozen-target runs report peak across two seeds (42, 43) rather than mean-last-5; for the near-frozen default recipe the mean-last-5 and peak numbers agree to within* ∼ 0.005 *(see Appendix A), so the comparison remains apples-to-apples at the precision we care about.*

lexical baselines.

4.2 Encoder Collapse and the Static-Target Fix

This section is the principal methodological contribution of the paper. Figure 3 shows $\Delta_{\cos}$ and copy_baseline (Eq. 7) over training for three settings of the target encoder EMA decay, holding all other hyperparameters fixed; Table 1 summarises the sweep.

Copy-baseline as a leading indicator. copy_baseline climbs from ∼0.90 to ∼0.999 *well before* $\Delta_{\cos}$ degrades, while the prediction loss continues to decrease misleadingly. Monitoring the prediction loss alone—standard practice in JEPA training—is not sufficient to detect this failure mode on our data. After the fix, the near-frozen runs hold copy_baseline in the 0.69–0.84 range across all seeds and contrastive weights we trained, which is consistent with the expected cosine between consecutive real code states.

Why standard fixes did not work in our sweep. We attacked the collapse through representation regularisation rather than EMA decay: stronger batch spectral regularisation (10× and

100× the default weight), higher covariance weight (10×), and the combination of all three. None prevented collapse on trajectory data at default EMA: the best non-EMA intervention reached mean-last-5 $\Delta_{\cos} \approx 0.92$, no better than the collapsed baseline, and the 100× spectral setting actually *hurt*. Conversely, slowing EMA decay alone—with *no* change to the regularisation weights—was sufficient in every run we tried. The result is consistent with runaway target-encoder tracking as the dominant driver in this parameter regime, but we do not rule out that appropriately scaled spectral or covariance terms could also work; we only report that the ones we swept did not.

Direction loss is a scaffold, not the enemy. Pre-fix diagnostic work had hypothesised that the direction loss and prediction loss conflict after ∼700 steps. We tested this directly by disabling $\mathcal{L}_{\text{dir}}$ at step 500 and observed immediate catastrophic collapse (mean-last-5 $\Delta_{\cos} \to 0$ within 50 steps). $\mathcal{L}_{\text{dir}}$ is the scaffold that holds directional alignment in place; removing it does not free the predictor, it destroys the geometry.

A plausible mechanism. We offer the following interpretation of the EMA sweep, consistent with the copy-baseline trajectories in Figure 3 but not independently proven. At the 1–2 M-parameter scale, the online encoder's per-step updates are large relative to the 128-d latent space, so small changes in θ produce comparatively large shifts in the batch embedding distribution. A fast EMA target (decay 0.996) tracks these shifts closely enough that if the online encoder drifts toward a lower-rank region, the target encoder follows within ∼300 steps and their representation spaces co-collapse. Once that happens, copy_baseline saturates near 1, the direction loss's gradient carries no information (every state embeds to near the same point), and training is stuck. Near-freezing the target (decay 0.99999) decouples the two encoders by making the target effectively a frozen initialisation over the training horizon, which gives the online encoder room to explore without dragging the target attractor along with it. Supporting this reading, the copy-baseline curves in Figure 3 show the target encoder's collapse as an *exponential* decay in spread at EMA 0.996 (timescale ∼300 steps, matching the effective $1/(1-0.996) = 250$-step EMA window) and essentially no drift at EMA 0.99999 over 15 K steps. We flag this as a plausible mechanism, not a formal proof; a full characterisation would require tracking the rank of the target-encoder embedding matrix over training, which we have not run.

Ablation: a fully frozen random-init target works equivalently. To test whether the specific EMA decay value matters, or whether it is sufficient for the target encoder to stay approximately static over the training horizon, we ran a *frozen target* ablation with EMA decay 1.0. In this variant the target encoder is initialised as `copy.deepcopy`(f_θ) at step 0 and then receives no updates for the full 15,000 steps, so the "target" is a random-init projection whose motion budget is *zero*. Two independent seeds (42, 43) produce peak $\Delta_{\cos}$ of 0.9925 and 0.9938, matching the near-frozen EMA 0.99999 Phase 5 predictor baseline (0.9928) within ±0.002; indeed seed 43 peaks slightly above the baseline (last row of Table 1). The 14% of target-encoder motion that EMA 0.99999 provides over 15,000 steps is therefore not load-bearing: any sufficiently static target, including one that never moves at all, prevents the collapse in this parameter regime. We retain near-frozen EMA as the default recipe (all main-body numbers are trained with it) because it was the configuration chosen before the ablation, but the mechanism behind the fix is target staticity rather than a specific decay schedule—a direct answer to the natural reviewer question of why we did not simply freeze the target. Surprisingly, the frozen-target variant also shows materially better multi-step rollout stability and a categorically different latent geometry; we report those findings in Section 4.3 and Section 4.5.2 respectively.

Table 2: Prediction quality on held-out CommitPack Python trajectory data. Mean-last-5 is the mean of the last five validation $\Delta_{\cos}$ measurements ± sample standard deviation within a single run; 3-seed columns are mean ± std across three independent training seeds. Higher is better. The frozen-target rows are the EMA decay 1.0 ablation from Section 4.2; we report peak rather than mean-last-5 for these runs because only the peak was logged in the ablation sweep, and the near-frozen EMA peak (0.9948 on ema-frozen-15K seed 42) is within 0.005 of its own mean-last-5, so the comparison is still apples-to-apples at the precision we care about.

Run	λ_{con}	Steps	Seeds	Mean-last-5	3-seed peak
ema-frozen-15K (champion)	0	15 000	3	$\mathbf{0.9898 \pm 0.0006}$	0.9934 ± 0.0015
ema-frozen-3K	0	3 000	1	0.9780 ± 0.0014	0.9867
contrast-15K-high	1.0	15 000	1	0.9772 ± 0.0045	0.9895
contrast-high-3K	1.0	3 000	3	0.9482 ± 0.0168 (s42)	0.9821 ± 0.0063
contrast-extreme-3K	2.0	3 000	3	0.9460 ± 0.0163 (s42)	0.9798 ± 0.0010
contrast-mega-3K	5.0	3 000	1	0.9548 ± 0.0143	0.9802
Frozen-target ablation (EMA decay 1.0, Section 4.2):					
frozen-target-15K seed 42	0	15 000	1	— (peak only)	0.9925
frozen-target-15K seed 43	0	15 000	1	— (peak only)	0.9938
Pre-fix baselines (collapsed):					
ema-default-3K (decay 0.996)	0	3 000	1	∼0.10 (collapsed)	0.9228
dir-loss-kill-500	0	3 000	1	∼0 (collapsed)	0.8990

4.3 Single-Step and Multi-Step Prediction

With the EMA fix in place, we train a 128-dimensional, 1.1M-parameter model on trajectory data for 15 000 steps. Three independent seeds yield mean-last-5 $\Delta_{\cos} = 0.9898 \pm 0.0006$ within each run and peak 0.9934 ± 0.0015 across seeds. Table 2 summarises the full sweep across the EMA fix, contrastive variants, and pre-fix baselines.

A preempted 15K contrastive run at $\lambda_{\text{con}} = 2.0$ (CONTRAST-EXTREME-15K, terminated by spot-pod reclaim; partial peak $\Delta_{\cos} \approx 0.9917$, consistent with the $\lambda_{\text{con}} = 1.0$ 15K run) is discussed in Appendix A and excluded from Tables 2 and 4.

The contrastive 3K runs are seed-sensitive; the predictor is not. The 3-seed standard deviation on the predictor is ±0.0015 on peak and ±0.0006 on mean-last-5; on the 3K contrastive variants the standard deviation is 4× wider for $\lambda = 1.0$ (±0.0063) and 6× tighter for $\lambda = 2.0$ (±0.0010). The contrastive cost adds variance at short training horizons; higher λ stabilises training but does not improve the mean. Under three seeds the means are statistically tied ($\lambda = 1.0$: 0.9821 ± 0.0063; $\lambda = 2.0$: 0.9798 ± 0.0010).

Multi-step composition: robust across seeds and well above the delta-space nulls. On all five 15K runs we trained, the per-step *delta* cosines are $s_k = \cos(\delta_k^{\text{pred}}, \delta_k^{\text{true}}) \approx 0.98$ at the peak checkpoint for $k = 1, 2, 3$, with $s_2 \geq s_1$ in every case (Table 4). Earlier drafts of this work reported that the metric "collapses to zero at step 2"; that observation was an artefact of evaluating pre-fix checkpoints taken after encoder collapse.

Table 3: Delta-space null models for the compositional prediction claim, computed on a held-out tail slice of the trajectory HDF5 (848 triples) using the EMA-FROZEN-15K target encoder. These are the right baselines for s_k; the predictor's $s_1 \approx 0.987$ sits more than 0.9 above every null.

Null model	Mean ± std
Trivial identity ($\delta^{\text{pred}} \equiv 0$, by convention)	0.000
Shuffled action ($\cos(\delta_i^{\text{true}}, \delta_{\pi(i)}^{\text{true}})$)	-0.005 ± 0.228
Random-trajectory two-step ($\cos(\delta_i^{(0\to2)}, \delta_{\pi(i)}^{(0\to2)})$)	-0.007 ± 0.297
Mean-delta (constant predictor, $\delta^{\text{pred}} \equiv \bar{\delta}$)	0.075 ± 0.339
Self-consistency ($\cos(\delta_{t\to t+1}, \delta_{t+1\to t+2})$, same trajectory)	0.032 ± 0.354
CodeWM predictor s_1 (paper headline)	**0.987–0.991**

Null models for the compositional claim. The obvious skeptical question is how much of the 0.98 is real predictive work versus the fact that consecutive code states don't change very much. To answer it precisely, we compute five null models on a held-out tail slice of the trajectory HDF5 (848 trajectories of length ≥ 2, out-of-training-split by construction), using the EMA-FROZEN-15K target encoder.

We compute these with `evaluation/null_models_rollout.py` on the trajectory HDF5 using the saved champion checkpoint.

First, a *state-space* measurement that motivates our metric choice. The cosine between target-encoder embeddings of consecutive real states, $\cos(f_{\bar{\theta}}(s_t), f_{\bar{\theta}}(s_{t+1}))$, is already 0.977 ± 0.096 (median 0.9999); the skip-step variant $\cos(f_{\bar{\theta}}(s_t), f_{\bar{\theta}}(s_{t+2}))$ is essentially the same at 0.974 ± 0.097. In absolute state cosine, in other words, consecutive code states are nearly indistinguishable, and "predict z_{t+k} accurately" is not a meaningful challenge on this data. *This is the central reason we report delta cosine rather than absolute state cosine as the prediction-quality metric.*

Second, the *delta-space* nulls that directly test the compositional claim (Table 3). A trivial identity predictor outputs $\delta^{\text{pred}} = 0$ and scores 0 by convention. A shuffled-action null—pairing each true delta with a random other true delta within a batch—scores -0.005 ± 0.228. A random-trajectory null (cross-trajectory delta pairs for the two-step ground-truth displacement) scores -0.007 ± 0.297. A mean-delta null—a constant predictor $\delta^{\text{pred}} \equiv \bar{\delta}$ that always outputs the mean observed delta—scores 0.075 ± 0.339; this tests whether the delta distribution is low-rank enough that a mean-direction predictor looks good on its own, and the low score confirms it is not. The tightest direction-aware null we evaluate is *within-trajectory self-consistency*, $\cos(\delta_{t\to t+1}, \delta_{t+1\to t+2})$: would predicting the same delta again work? This scores 0.032 ± 0.354.

The headline $s_1 \approx 0.987$ therefore sits more than 0.9 delta-cosine points above every delta-space null, including the mean-delta constant-predictor null at 0.075 and the within-trajectory self-consistency null at 0.032 that already knows the first edit's direction. The compositional claim is not "code doesn't change much"; it is "given z_t and the action embedding, the predictor reliably recovers the direction of the latent displacement that the encoder assigns to the ground-truth next state." We also include, for state-space context, the in-training copy_baseline metric—the batch-aggregated cosine between target-encoder embeddings of consecutive states logged during training—in the right column of Table 4. This is the same quantity used as the leading-indicator diagnostic in Section 4.2; it sits in the healthy 0.77–0.84 range across all five 15K runs, confirming the encoder has not collapsed.

Table 4: *Per-step* delta cosines across the completed 15K runs. Each s_k compares δ_k^{pred} against the ground-truth δ_k^{true} *independently*, without compounding predictor errors across steps (the stricter compounding/cumulative variant is reported separately in Table 8). See Table 3 for the delta-space nulls and Section 4.3 for the Phase 3 / Phase 5 distinction; copy_baseline is the state-space encoder-collapse diagnostic at end of training (0.77–0.84 is healthy).

Run	s_1	s_2	s_3	copy_baseline
ema-frozen-15K seed 42 (champion, Phase 3)	0.987	0.992	0.987	0.79
ema-frozen-15K seed 43 (Phase 5)	0.990	0.995	0.985	0.77
ema-frozen-15K seed 44 (Phase 5)	0.989	0.993	0.993	0.84
contrast-15K-high (λ=1.0, Phase 5)	0.989	0.992	0.987	0.79

Table 5: Single-step rollout on edit chains from nine held-out Python repositories (`pandas`, `networkx`, `pytest`, `rich`, `requests`, `httpx`, `typer`, `pydantic`, `matplotlib`). Columns: mean s_1 excluding the hardest repository (`rich`), mean s_1 across all nine repositories, and s_3 on `rich` alone. The first row is the Phase 3 single-seed champion; the remaining three rows are Phase 5 checkpoints (a second ema-frozen predictor seed and both contrast-15K-high seeds). Best per column in bold.

Checkpoint	s_1 excl rich	s_1 all 9	`rich` s_3
ema-frozen-15K seed 42 (Phase 3)	0.9766	0.9743	0.8819
ema-frozen-15K seed 43 (Phase 5)	0.9810	0.9776	0.9329
contrast-15K-high seed 42 (Phase 5)	0.9807	0.9788	0.9340
contrast-15K-high seed 43 (Phase 5)	**0.9845**	**0.9827**	**0.9479**

Baselines cannot perform compositional rollout on this task. Embedding-only baselines (bag-of-AST-tokens, CodeBERT, UniXcoder) do not provide an action-conditioned transition operator and so cannot be evaluated on the multi-step rollout task except via trivial *copy-input* or *nearest-neighbour* surrogates, neither of which is a meaningful comparison against an action-conditioned predictor. We therefore report the multi-step numbers as an architecture-specific capability and the delta-space nulls of Table 3 as the fair reference, rather than as a head-to-head victory over lexical or MLM-pretrained encoders on an ill-matched task.

Out-of-distribution rollout transfers. Table 3 and Table 4 are computed on the held-out tail of the CommitPack trajectory HDF5, which is in-domain by construction. To test whether the compositional property transfers, we re-run the single-step rollout script on edit chains harvested from nine Python repositories—`pandas`, `networkx`, `pytest`, `rich`, `requests`, `httpx`, `typer`, `pydantic`, and `matplotlib`—all of which are popular upstream libraries and none of which appear in the CommitPackFT training split. Table 5 reports the result for the Phase 3 single-seed champion and for three Phase 5 checkpoints (a second EMA-FROZEN predictor seed and both CONTRAST-15K-HIGH seeds). Mean single-step delta cosine on the nine-repository set lands at 0.9776–0.9827 for every Phase 5 checkpoint evaluated, within 0.01–0.02 of the in-distribution value ($\sim$ 0.987), and the hardest repository (`rich`) improves from 0.8819 on the Phase 3 champion to 0.93–0.95 across every Phase 5 condition. The `rich` improvement appears on both contrastive and non-contrastive Phase 5 runs, which attributes it to the Phase 5 near-frozen EMA recipe rather than to the contrastive loss specifically.

4.4 Retrieval: A Secondary Tradeoff Map

Retrieval is the model's *secondary* experiment, not its headline. The compositional prediction and encoder-collapse diagnostic of §4.2–§4.3 are the contributions; this subsection reports what happens when the same 1.1M-parameter latent space is reused as an edit retriever, and is honest where it ties or loses.

Action-vector / retrieval circularity (§4.4). The contrastive head is supervised by action-vector cosine ($\tau_{\text{pos}} = 0.9$) and the in-distribution `by_joint` relevance criterion is "same edit type and same scope," both dominated by the same six one-hot dimensions of the 7-dimensional action vector (§2.3, §2.5). Any contrastive improvement on in-distribution `by_joint` therefore partially reflects the training signal rather than independent representation quality; the mitigation is a richer 15-dimensional AST-diff action variant (§6). The two out-of-distribution benchmarks below are less affected because their relevance criteria are held-out-repo edit pairs and function-level docstring matching.

Benchmarks and baselines. We evaluate three retrieval regimes: (i) *in-distribution CommitPackFT* at n_q=200, n_g=500 with `by_joint` relevance; (ii) *cross-repository leave-one-repo-out* on 3,998 edit pairs from twenty held-out Python libraries (pandas, requests, fastapi, httpx, rich, pydantic, scrapy, dask, networkx, attrs, click, flask, loguru, matplotlib, numpy, polars, pytest, sympy, tqdm, typer) at depth 2000 and 500 commits per repo, with aggregate `by_joint` MRR@10 plus a hard-negative K=9 rerank (one positive plus nine wrong-label distractors) that eliminates label-distribution exploitation (Table 7); and (iii) *CodeSearchNet function-level* [10] at n_q=400, n_g=1200 as an out-of-distribution stress test. Baselines are bag-of-AST-tokens, CodeBERT [5], UniXcoder [8], BGE [17], jina-code [6], and codet5p [16], all public pretrained checkpoints *without* fine-tuning; a fine-tuned dense encoder would likely close or reverse the external-benchmark gap.

In-distribution CommitPackFT. CodeWM beats CodeBERT on `by_joint` MRR by ∼0.07–0.09 (depending on checkpoint) at 112× fewer parameters and ∼ 12× lower per-query latency (≈ 13 ms vs 165 ms on the same Mac CPU). The non-contrastive predictor champion PRED_S43 (0.8418) is the best CodeWM entry, CodeWM ties bag-of-AST-tokens within ∼ 0.02 MRR, and across the four fine-grained criteria CodeWM beats CodeBERT on three of four; the full per-criterion breakdown lives in Appendix C.4, Table 19. This is the benchmark most affected by the action-vector circularity caveat above.

Cross-repository leave-one-repo-out (§4.4). At the twenty-repository scale (3,998 edit pairs) all six CodeWM checkpoints beat CodeBERT on aggregate `by_joint` MRR@10 by +0.06 to +0.09 (CodeWM 0.7906–**0.8131** vs CodeBERT 0.7286); the frozen-target ablation seed FT_S42 (0.8131) is the best CodeWM checkpoint and edges the near-frozen champion PRED_S43 (0.8080) by +0.005, while the seed spread across the four near-frozen checkpoints (0.017) is 4× smaller than the CodeWM–CodeBERT gap, so the win is seed-robust; CodeWM ties BoW within ±0.02 MRR. The 9-class `by_joint` label distribution has a high random floor (Gaussian random 0.5888, analytic class-prior chance 0.5959), so we also report a hard-negative K=9 rerank (one positive plus nine wrong-label distractors per query, uniform random ≈0.293). Under that fair-conditions metric CodeBERT regains a small edge: CodeBERT 0.7303 > BoW 0.7019–0.7048 > CodeWM 0.6742–0.6776 (Table 7).

Table 6: Retrieval tradeoff map. CommitPackFT is single-seed `by_joint` MRR at n_q=200, n_g=500; cross-repository is aggregate `by_joint` MRR@10 across 3,998 edit pairs from twenty held-out Python libraries at evaluation depth 2000 and 500 commits per repo; CodeSearchNet is function-level MRR@10 at n_q=400, n_g=1200 on the Python test split. Best per column in bold; italicised entries are the top CodeWM checkpoint in that column. CodeWM rows are labelled by seed and training variant (Phase 3 champion EMA_S42; Phase 5 second predictor seed PRED_S43; Phase 5 contrastive seeds CON_S42 and CON_S43; frozen-target ablation seeds FT_S42 and FT_S43, EMA decay 1.0, see Section 4.2). All modern-encoder baselines are public pretrained checkpoints *without* fine-tuning. The cross-repository column is a coarse-label metric with a high random floor; see Table 7 and the footnote below for the companion hard-negative K=9 rerank.

Model	Params	CommitPackFT (MRR)	Cross-repo (MRR@10)	CodeSearchNet (MRR@10)
CodeWM (1.1M params):				
ema_s42 (Phase 3)	1.1M	0.8261	0.7906	*0.1205*
pred_s43 (Phase 5)	1.1M	*0.8418*	0.8080	0.1091
con_s42 (Phase 5)	1.1M	0.8246	0.7981	0.1025
con_s43 (Phase 5)	1.1M	0.8200	0.7982	0.1031
FT_s42 (frozen, decay 1.0)	1.1M	0.8380	*0.8131*	—
FT_s43 (frozen, decay 1.0)	1.1M	0.8293	0.7964	—
Baselines:				
BoW (AST tokens)	0	**0.8454**	0.7958	0.1070
CodeBERT [5]	124.6M	0.7473	0.7286	0.1990
UniXcoder [8]	125.9M	—	—	0.4687
BGE [17]	109.5M	—	—	0.4558
jina-code [6]	160.9M	—	—	0.4096
codet5p [16]	109.8M	—	—	**0.4867**

The cross-repository column measures aggregate `by_joint` MRR@10 on the 3,998*-pair leave-one-repo-out sweep. The coarse 9-class `by_joint` label distribution has a high random floor: Gaussian-random features score* 0.5888 *MRR@10 and the analytic class-prior chance under uniform-random ranking is* 0.5959*. The BoW cross-repository entry is the* EMA_S42 *value; the four BoW numbers across checkpoints differ by* $\leq$ 0.003 *because of within-run query/gallery sampling. A hard-negative K=9 rerank that eliminates the label-distribution shortcut (Table 7) restores a small* $\sim$0.05 *MRR edge to CodeBERT over both CodeWM and BoW.*

CodeSearchNet function-level (OOD loss). Modern code-specific dense encoders cluster at 0.41–0.49 MRR@10: codet5p (**0.4867**), UniXcoder (0.4687), BGE (0.4558), jina-code (0.4096); CodeBERT sits at 0.1990, only $\sim$2$\times$ above CodeWM and BoW ($\sim$ 0.11) and a factor of 2.4$\times$ *below* the modern dense cluster. CodeWM is not competitive here: it was trained on edit triples, not function–docstring pairs, and the best CodeWM entry is 0.1205 (ema_s42).

Interpretation. The model sits on a compact Pareto point of the efficiency–quality frontier, not on a universal retrieval win. In-distribution CommitPackFT and the twenty-repository cross-repository aggregate are two benchmarks where a 112$\times$ smaller model ties or beats a 124M-parameter MLM-pretrained CodeBERT; the hard-negative K=9 rerank and the CodeSearchNet stress test are two benchmarks where modern dense encoders win cleanly. Retrieval is the secondary experiment that calibrates the reader against generic code encoders; the scientific contribution of the paper remains the stability/composition story of §4.2–§4.3.

Methodology note. The numbers above were regenerated after we fixed a silent bug in the baseline-comparison harness: five scripts forced the CodeWM attention-pool readout to CLS mode at load time and relied on `load_state_dict(strict=False)`, which silently discarded the checkpoint's attention-pooling weights. Training code and the prediction/null-model rollout scripts al-

Table 7: Cross-repository retrieval detail at twenty-repository scale (3,998 edit pairs, leave-one-repo-out). Aggregate MRR@10 is the standard `by_joint` metric reported in Table 6; hard-negative K=9 is a per-query mini-gallery of one same-label positive plus nine wrong-label distractors, with theoretical uniform-random MRR ≈ 0.293. CodeWM rows report the best checkpoint (FT_S42, frozen target, aggregate) with the mean across all six CodeWM checkpoints in parentheses; seed spread is ≤ 0.02 on aggregate and negligible on hard-neg K=9. Hard-neg K=9 numbers for the frozen-target ablation were not recomputed; the near-frozen CodeWM range is reported. Best per column in bold.

Model	Params	Aggregate MRR@10	Hard-neg K=9 MRR@10
CodeWM (FT_S42; mean of 6)	1.1M	**0.8131** (0.8007)	0.6742–0.6776 (near-frozen)
BoW (AST tokens)	0	0.7956 (0.7967)	0.7048 (0.7032)
CodeBERT [5]	124.6M	0.7286	**0.7303**
Random (Gaussian)	0	0.5888	0.2973
Class-prior chance (analytic)	0	0.5959	—

ways used the correct code path. Every entry in Table 6 and Table 19 was regenerated on all six CodeWM checkpoints with the fixed helper.

4.5 Frozen-Target Ablation: Rollout Stability and Latent Geometry

The frozen-target ablation (Section 4.2) is the minimal-motion end of the static-target family: the target encoder is a `deepcopy` of the randomly-initialised online encoder and never updates. Val $\Delta_{\cos}$ and in-distribution retrieval are preserved under this ablation to within ± 0.002 and ± 0.015 respectively (Sections 4.2, 4.4). We report two follow-up findings here: the ablation has materially better *cumulative* multi-step rollout stability than the Phase 5 near-frozen checkpoints, and its latent transitions have 40–80% larger magnitudes relative to state norm than every near-frozen baseline.

4.5.1 Cumulative multi-step rollout stability

Table 4 reports *per-step* delta cosines: each s_k compares the predictor's k-th step delta against the ground-truth k-th step delta in isolation, so predictor errors do not compound across steps. A stricter metric is *cumulative rollout cosine*: feed the predictor's own output back in at each step, so that the error at step k accumulates the error at steps $1, \ldots, k-1$, and compare the final rolled latent to the ground-truth target. Table 8 reports cumulative s_1, s_2, s_3 on the same held-out tail slice used by Table 4, across all six 15K checkpoints.

Reading the cumulative table against Table 4. Per-step s_3 is healthy in the 0.98–0.99 range for every 15K run we trained (Table 4), because each single-step prediction is evaluated against the ground-truth z_{t+k-1} input without compounding. Cumulative s_3 is a different and stricter test: at step 3 the predictor is three steps deep into its own rolled trajectory and any misalignment at earlier steps shows up as compounded directional error. Three observations are worth pulling out. First, the Phase 3 champion EMA-FROZEN-15K seed 42 holds both metrics: per-step $s_3 = 0.987$ (Table 4) and cumulative $s_3 = 0.892$ (Table 8), the best in either column. Second, the Phase 5 near-frozen family (PRED_S43, CON_S42, CON_S43) is competitive on per-step but drops to $s_3 = 0.68$–0.73 cumulative—about 0.20 below the Phase 3 champion—which we read as the Phase 5 near-frozen

Table 8: Cumulative multi-step rollout stability across the six 15K CodeWM checkpoints. s_k here is the cosine between the predictor's k-step rolled latent (feeding each step's output back as the input to the next) and the ground-truth target at step k, so errors compound. This is the stricter complement of Table 4's per-step delta cosines. The Phase 3 champion still leads at s_3; the frozen-target ablation is the best Phase 5 variant.

Checkpoint	s_1	s_2	s_3
Phase 3 near-frozen (original champion recipe):			
ema_s42 (Phase 3 champion)	0.9870	**0.9357**	**0.8915**
Phase 5 near-frozen (main-body checkpoint family):			
pred_s43 (Phase 5, no contrast)	0.9882	0.8266	0.6794
con_s42 (Phase 5, λ_{con}=1.0)	0.9850	0.8182	0.6805
con_s43 (Phase 5, λ_{con}=1.0)	0.9875	0.8241	0.7312
Frozen target (decay 1.0), Section 4.2:			
FT_s42	0.9849	0.8656	0.7788
FT_s43	**0.9906**	0.8995	0.8460

recipe producing a latent transition operator that is locally accurate but less robust under its own roll-out. Third, the frozen-target ablation (FT_s42, FT_s43) closes about half of that gap to cumulative $s_3 = 0.78$–0.85, making it the best Phase 5 variant on this metric while remaining below the Phase 3 champion. We do not claim a mechanistic explanation for the Phase 3–Phase 5 cumulative gap; the difference is not the EMA decay (both recipes use 0.99999) and is more likely a training hyperparameter or data-ordering effect that falls outside the static-target contribution of this paper. The frozen-target ablation's closing-of-half-the-gap, however, is consistent with the latent-geometry reading in §4.5.2: a state encoder decoupled from its target spreads out more in delta-space and the predictor's forward-chained trajectory stays closer to the true path.

4.5.2 Delta-norm geometry

Table 9 reports $\|\delta_{\text{true}}\|/\|z_0\|$ at the single-step horizon, with $\delta_{\text{true}} = \text{sg}[z'] - z$ and $z_0 = f_\theta(s_t)$, alongside the predictor directional cosine $\cos(\delta_{\text{pred}}, \delta_{\text{true}})$ at s_1.

Interpretation. The near-frozen EMA baseline imposes a soft coupling: the target encoder tracks the online encoder at roughly 14% over 15,000 steps, so f_θ and $f_{\bar{\theta}}$ stay near-identical and per-edit displacements remain small relative to the state norm (≈ 0.55–0.70). Under the frozen-target ablation this coupling is cut entirely—$f_{\bar{\theta}}$ is a fixed random-init snapshot—and the online encoder is free to spread out in its chosen delta direction without a "gravitational pull" toward the target attractor. The resulting edit displacements are almost as large as the state vector itself: even the 10th-percentile edit under FT_s43 moves the state by ≈95% of its own norm ($q_{10} = 0.952$), and the 90th-percentile edit ($q_{90} = 1.093$) exceeds the state norm. The predictor nonetheless recovers the correct direction with $\cos(\delta_{\text{pred}}, \delta_{\text{true}}) = 0.985$–$0.991$ at s_1, matching the near-frozen baseline within 0.002. We report this as a qualitative finding: the static-target fix is not unique up to a latent isometry—different realisations of "sufficiently static" produce categorically different latent geometries, even when they agree on every aggregate prediction or retrieval metric we compute. A principled characterisation of which latent-geometry family is preferable for downstream predictive

Table 9: Single-step transition magnitudes on the held-out trajectory tail. $\|z_0\|$ is the state-encoder output norm; $\|\delta_{\text{true}}\|/\|z_0\|$ is the per-edit displacement magnitude normalised by the state norm, with q_{10} and q_{90} as the corresponding tail quantiles. Rightmost column is the predictor's directional cosine at s_1. The frozen-target ablation (FT_s42, FT_s43) produces 40–80% larger delta magnitudes than every near-frozen EMA baseline, while the s_1 direction still matches the baseline within 0.002: same prediction direction, categorically different latent geometry.

Checkpoint	$\|z_0\|$	$\|\delta_{\text{true}}\|/\|z_0\|$	q_{10}/q_{90}	$\cos(\delta_{\text{pred}}, \delta_{\text{true}})$ @ s_1
Near-frozen EMA (decay 0.99999*):*				
ema_s42	11.20	0.673	0.55/0.84	0.9870
pred_s43	11.19	0.547	0.47/0.67	0.9882
con_s43	11.20	0.700	0.61/0.78	0.9875
Frozen target (decay 1.0*):*				
FT_s42	11.19	**0.974**	0.838/1.083	0.9849
FT_s43	11.19	**1.029**	0.952/1.093	0.9906

Table 10: Quantisation ladder for the 128-dimensional champion model. Cosine fidelity is the mean cosine similarity between quantised and f32 encoder outputs over the full held-out set.

Level	Size (MB)	Compression	Cosine fidelity	Scope
f32	5.3	1×	1.0000	—
INT8	1.8	∼3×	0.9999	weights
Q4	0.8	∼6.6×	0.9999	weights
Q4-full	0.6	∼8.8×	0.9999	weights + biases

tasks (anomaly detection via unpredicted displacement magnitude, change-impact estimation) is outside the scope of this paper and is flagged explicitly in Section 6.

4.6 Quantisation Fidelity and Inference Latency

For compact deployment, model size is the binding constraint. We evaluate a four-level quantisation ladder on the 1.1M-parameter champion, measuring compression ratio and cosine fidelity relative to the f32 baseline (Table 10). All quantised variants maintain cosine fidelity ≥ 0.9999; the Q4-full configuration compresses the model to 0.6 MB, small enough for a single HTTP request over a typical connection when shipped as a WebAssembly bundle (Section 5).

Table 11 reports p50 latencies for each pipeline stage at f32 precision. The full encode–predict pipeline completes in under 5 ms at $S = 64$, which is fast enough for keystroke-level IDE feedback.

5 Deployment

A key design goal of the Code World Model is deployability on resource-constrained targets—from native CPU servers to browser tabs—without sacrificing prediction fidelity. This section describes the inference engine, validation methodology, and the WebAssembly deployment path.

Table 11: Inference latency (p50, milliseconds) on M-series CPU at f32 precision. S denotes input sequence length.

Stage	S	p50 (ms)
Encoder	64	< 2
Encoder	256	< 8
Encoder	512	< 15
Action encoder	—	< 1
Predictor (1 step)	—	< 2
Full pipeline	64	< 5

5.1 Rust Inference Engine

The production inference runtime is implemented in 798 lines of Rust, covering tokenisation, encoder forward pass, action encoding, and single-step prediction. The shipped checkpoint is the 15 000-step EMA-FROZEN champion from Section 4.3 (1.1M parameters, 128-dimensional latent). The implementation mirrors the PyTorch training code layer-by-layer, using only the Rust standard library and a small linear algebra module with no external dependencies. This zero-dependency design simplifies cross-compilation to WebAssembly.

5.2 Zero-Drift Validation

Porting a neural network from a training framework to a hand-written inference engine introduces the risk of silent numerical divergence. We validate correctness through an automated *zero-drift* protocol: for each of three random seeds, we run identical inputs through both the PyTorch reference and the Rust engine, then compare the output latent vectors. Across all test cases the Rust engine achieves cosine similarity ≥ 0.99999 and maximum absolute deviation $< 5 \times 10^{-5}$ relative to PyTorch. These tolerances are well within the noise floor of f32 arithmetic and confirm that the port introduces no systematic bias.

A second validation tier compares the Zig FFI backend (used for WASM targets) against the pure-Rust engine. The two backends produce results identical within floating-point tolerance, ensuring that the quantisation and deployment pipeline preserves fidelity end-to-end.

Cross-stack portability of the Phase 5 checkpoint family. As an additional deployment check, we ported all ten Phase 5 checkpoints into the inference stack with zero Rust changes: the post-fix architecture is parameter-shape compatible with the existing loader path, so the existing code absorbs the new weights directly. All 18 CodeWM end-to-end golden tests pass at cosine ≥ 0.99999 and maximum absolute deviation $< 5 \times 10^{-5}$ against the PyTorch reference across pure-Rust, Zig FFI, and Apple Accelerate backends. An independent quantisation sweep over 500 real Python files from 23 upstream packages confirms INT8 as the safe default for every Phase 5 variant we tested, and Q4 as additionally safe on the encoder side. Q4 does degrade the *predictor* on the most aggressive near-frozen-EMA runs, so prediction workloads should stay at INT8 or above.

5.3 WebAssembly Browser Demo

We compile the Rust engine to WebAssembly via `wasm-pack` and optimise the binary with `wasm-opt -Oz`. The resulting bundle is 2.6 MB, small enough for a single HTTP request on typical connections.

Table 12: WASM browser demo latency breakdown for a single query against a 20-file corpus.

Stage	Latency
Tokenisation	0.5–2 ms
Encode (S=512)	80–150 ms
Cosine scan (20 files)	<1 ms
Total	<200 ms

Table 12 breaks down the in-browser latency profile for a representative query against a 20-file corpus. The total query latency stays below 200 ms, well within interactive thresholds.

The demo runs entirely client-side: no model weights or source code leave the browser, making it suitable for proprietary codebases. Figure 4 positions this configuration alongside the native Rust target in the latency–footprint design space.

6 Limitations and Open Validations

Action-vector / retrieval circularity. The contrastive head is supervised by action-vector cosine, and the 7-dimensional action vector is dominated by six one-hot edit-type and scope dimensions, so the $\tau_{\text{pos}} = 0.9$ positive criterion is structurally close to the `by_joint` retrieval relevance predicate used in Section 4.4: any in-distribution retrieval improvement from the contrastive loss partially reflects the training signal. The mitigation is a richer 15-dimensional AST-diff action variant (node-type counts, change depth, identifier overlap), already implemented in our collectors; regenerating the trajectory dataset against it requires re-downloading the CommitPack Python subset because the existing 1.5 M-edit HDF5 stores only identifier-lossy AST tokens.

Real downstream predictive tasks. The distinctive capability of a code transition model is *predictive*, not embedding-only, and retrieval is a poor proxy for that. The natural next step is downstream predictive evaluation: edit anomaly detection via revert-marker self-supervision (a held-out edit is flagged when $\|z_{\text{pred}} - z_{\text{true}}\|$ is unusually large), change-impact estimation on whether an edit propagates to test failures, and refactoring suggestion via decoding from latent space. None of these are evaluated in this paper; they are the most important validation for turning the framing into evidence.

Schema-compatible third in-distribution benchmark. The two-win retrieval story (Section 4.4) rests on a single training-distribution family (CommitPack Python). An earlier attempt to add the JetBrains commit-chronicle corpus [4] exposed a fundamental schema mismatch: commit-chronicle stores diff hunks only, so CodeWM's AST-token state encoder falls back to byte/literal tokens on syntactically-invalid `@@ ... @@` blocks while CodeBERT reads the same hunk as raw text and gets a clean signal (CodeBERT dominates CodeWM and BoW by $\sim$0.34 MRR on the non-saturated `subset_cmg` Python pool). A schema-compatible third benchmark—Defects4J-style bug-fix corpora or function-level edit datasets harvested from CommitPackFT-excluded repositories are candidates—would answer the fair reviewer question "is the CommitPackFT win a CommitPack-family artefact?".

Multi-seed and mechanistic tightening. In-distribution delta cosine reproduces to within 0.004 across all four Phase 5 near-frozen checkpoints, and the frozen-target ablation (Section 4.2) adds two further seeds at EMA decay 1.0 within ±0.002 of the baseline. CONTRAST-EXTREME-15K (λ=2.0) remains single-seed and preempted; a third seed on both contrastive configurations would tighten the retrieval claim at its margins. The mechanistic paragraph in Section 4.2 is a plausible rank-collapse hypothesis: the frozen-target ablation already bounds it observationally (zero motion budget $\Rightarrow$ equivalent val $\Delta_{\cos}$), but a target-encoder rank trace over training would upgrade the argument to a direct measurement. Multi-seed null-model replication and the full null battery on an out-of-distribution split (Section 4.3) are post-hoc eval-script calls that we leave to a later revision.

Tokeniser ablation. We do not isolate the contribution of the AST + hashed-identifier tokeniser described in Section 2. Structural bias, compact vocabulary, and cross-stack Python/Rust parity are plausible attributions but not experimentally separated from the rest of the recipe. A comparison against a simpler lexical or BPE-style tokeniser at matched parameter budget is the natural ablation.

Smaller open items. Weighted loss or balanced resampling for the 98.8% MODIFY class imbalance in CommitPackFT; a contrastive temperature and positive-threshold ablation (we use $\tau = 0.07$, $\tau_{\text{pos}} = 0.9$); an architectural ablation against an unshared L-layer transformer at matching parameter count; and multi-language extension beyond Python (currently out of scope: the tokeniser and Rust inference engine are pinned to the Python AST).

7 Conclusion

Tiny JEPA-style code world models exhibit a mysterious training ceiling that standard JEPA diagnostics do not catch. The principal contribution of this paper is the identification of that failure mode—target encoder collapse under fast EMA tracking, detectable by a simple leading-indicator metric (copy_baseline = $\cos(f_{\bar{\theta}}(s_t), f_{\bar{\theta}}(s_{t+1}))$) that climbs to $\sim$0.999 well before the prediction loss degrades—and a static-target mitigation that reliably eliminates it on our data: near-freezing the target encoder at EMA decay 0.99999, or equivalently fully freezing it at decay 1.0. A two-seed frozen-target ablation reproduces the near-frozen baseline within ±0.002 mean-last-5 $\Delta_{\cos}$, so the fix is *target staticity* rather than a specific EMA decay value; any sufficiently static target suffices. The fix holds across three independent seeds of the default near-frozen recipe (predictor std ±0.0015), two additional seeds of the frozen-target ablation, and in every configuration we tested; stronger spectral and covariance regularisers at the weights we swept did not substitute for it.

With the fix in place, a 1.1M-parameter 128-dimensional model trains stably for 15 000 steps and performs short-horizon compositional multi-step prediction with per-step delta cosines $s_1 \approx s_2 \approx s_3 \approx 0.98$, sitting more than 0.9 cosine points above every delta-space null we evaluate—a trivial identity predictor, a shuffled-action null, a random-trajectory two-step null, a mean-delta constant predictor, and a within-trajectory self-consistency null that already knows the first edit's direction (Table 3). The same compositional property transfers to edit chains from nine held-out Python repositories (Table 5), with mean single-step delta cosine landing within 0.01–0.02 of the in-distribution value. The compositional property is architecture-specific: embedding-only baselines do not provide an action-conditioned transition operator and therefore cannot be evaluated on the same task.

A secondary multi-task retrieval tradeoff map (Section 4.4) shows the model is competitive at 112× fewer parameters on in-distribution CommitPackFT edit retrieval and on a twenty-repository cross-repository sweep, with an honest hard-negative $K{=}9$ caveat that restores a small $\sim$0.05 MRR edge to CodeBERT, while modern dense code encoders (codet5p, UniXcoder, BGE, jina-code) dominate the out-of-distribution CodeSearchNet stress test by a factor of $\sim$4×. We read retrieval as a Pareto efficiency story, not a universal representation-quality victory.

A zero-dependency Rust inference engine, four-level weight quantisation to 0.6 MB with cosine fidelity ≥ 0.9999, and a 2.6 MB in-browser WebAssembly bundle complete the compact-deployment story.

The natural next step is downstream predictive evaluation—anomaly detection via revert markers, change-impact estimation, refactoring suggestion—which is what the "world model" framing demands and which we do not yet evaluate. That, together with a regenerated trajectory dataset carrying 15-dimensional structural action vectors to close the action-circularity gap, is the principal outstanding validation (Section 6).

References

[1] Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[2] Adrien Bardes, Jean Ponce, and Yann LeCun. VICReg: Variance-invariance-covariance regularization for self-supervised learning. In *International Conference on Learning Representations (ICLR)*, 2022.

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

[4] Aleksandra Eliseeva, Yaroslav Sokolov, Egor Bogomolov, Yaroslav Golubev, Danny Dig, and Timofey Bryksin. From commit message generation to history-aware commit message completion. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2023. Source of the CommitChronicle dataset of history-aware commit message pairs.

[5] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. CodeBERT: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020.

[6] Michael Günther, Jackmin Ong, Isabelle Mohr, Alaeddine Abdessalem, Tanguy Abel, Mohammad Kalim Akram, Susana Guzman, Georgios Mastrapas, Saba Sturua, Bo Wang, Maximilian Werk, Nan Wang, and Han Xiao. Jina embeddings 2: 8192-token general-purpose text embeddings for long documents. *arXiv preprint arXiv:2310.19923*, 2023. Source of the `jinaai/jina-embeddings-v2-base-code` checkpoint.

[7] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement,

Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. GraphCodeBERT: Pre-training code representations with data flow. In *International Conference on Learning Representations (ICLR)*, 2021.

[8] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. UniXcoder: Unified cross-modal pre-training for code representation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022.

[9] Thong Hoang, Hong Jin Kang, David Lo, and Julia Lawall. CC2Vec: Distributed representations of code changes. In *Proceedings of the 42nd International Conference on Software Engineering (ICSE)*, 2020.

[10] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019.

[11] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[12] Bo Lin, Shangwen Wang, Zhongxin Liu, Yepang Liu, Xin Xia, and Xiaoguang Mao. CCT5: A code-change-oriented pre-trained model. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2023.

[13] Zhongxin Liu, Zhijie Tang, Xin Xia, and Xiaohu Yang. CCRep: Learning code change representations via pre-trained code model and query back. In *Proceedings of the 45th International Conference on Software Engineering (ICSE)*, 2023.

[14] Niklas Muennighoff, Qian Liu, Armel Zebaze, Qinkai Zheng, Binyuan Hui, Terry Yue Zhuo, Swayam Singh, Xiangru Tang, Leandro von Werra, and Shayne Longpre. OctoPack: Instruction tuning code large language models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024. Introduces CommitPack, a 4 TB dataset of Git commits across 350+ languages.

[15] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[16] Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi D. Q. Bui, Junnan Li, and Steven C. H. Hoi. CodeT5+: Open code large language models for code understanding and generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023. Source of the `Salesforce/codet5p-110m-embedding` checkpoint.

[17] Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-pack: Packed resources for general Chinese embeddings. *arXiv preprint arXiv:2309.07597*, 2023. Source of the `BAAI/bge-base-en-v1.5` checkpoint.

[18] Pengcheng Yin, Graham Neubig, Miltiadis Allamanis, Marc Brockschmidt, and Alexander L. Gaunt. Learning to represent edits. In *International Conference on Learning Representations (ICLR)*, 2019.

[19] Jiyang Zhang, Sheena Panthaplackel, Pengyu Nie, Junyi Jessy Li, and Milos Gligoric. CoditT5: Pretraining for source code and natural language editing. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2022.

A Hyperparameters

Table 13 lists the architecture configuration for the 1.1M-parameter champion model used throughout the main body. The `expa` column records the 192-dimensional pre-fix variant that appeared in earlier drafts and is retained here for historical continuity.

Table 13: Architecture configuration. The champion column corresponds to the EMA-FROZEN-15K checkpoint from Section 4.3. All variants use exact `erf` GELU, CLS-token pooling, and LayerNorm $\varepsilon = 10^{-5}$.

Parameter	**champion (128-d)**	**expa (pre-fix, 192-d)**
Vocabulary size	662	662
Max sequence length	512	512
Model dimension d	128	192
Attention heads	4	4
Head dimension	32	48
MLP hidden dimension	512	768
Encoder loops L_{enc}	6	6
Predictor depth	2	2
Predictor loops L_{pred}	6	6
Action dimension	7	7
Parameter count	1,113,984	$\sim$2.2 M
GELU variant	`erf`	`erf`
Pooling mode	CLS	CLS

The encoder re-uses a single transformer block for L_{enc} iterations (weight-tied looping), giving the network effective depth 6 with the parameter count of a single block. The predictor similarly loops its two-block stack L_{pred} times.

Training hyperparameters. Table 14 summarises the training configuration used for the champion run and its retrieval-oriented sibling. All runs are managed by the Crucible platform (Appendix E).

*Target-encoder staticity is the critical stability lever; see Section 2.4 and 4.2 for the decay sweep, the diagnostic, and the frozen-target (decay 1.0) ablation. A fully frozen decay=1.0 variant produces an equivalent val $\Delta_{\cos}$ peak across two seeds (42, 43), matching the near-frozen 0.99999 baseline within ± 0.002.

Random seeds and reporting convention. The 3-seed predictor variance reported in Section 4.3 uses random seeds $\{42, 43, 44\}$ applied to PyTorch, NumPy, and the dataloader. The same seed set is used for the 3-seed contrastive sweep ($\lambda \in \{1.0, 2.0\}$ at 3K steps). The 15K CONTRAST-15K-HIGH run is now reported on two seeds (42, 43), and a second predictor seed (43) is available for the EMA-FROZEN-15K champion; in-distribution delta cosine reproduces to within 0.004 across

Table 14: Training hyperparameters for the champion (EMA-FROZEN-15K) and the retrieval-oriented CONTRAST-HIGH variant. Items marked † are representative defaults; exact values vary across Crucible sweep runs.

Hyperparameter	**Value**
Optimiser	AdamW (β_1=0.9, β_2=0.999)
Peak learning rate†	1×10^{-4}
Weight decay	0.01
LR schedule	Cosine decay with linear warm-up
Warm-up steps†	500
Batch size†	64
Training steps (champion)	15 000
Gradient clipping	Max norm 1.0
Objective	JEPA latent prediction + direction + spectral
Target encoder	EMA decay **0.99999** (near-frozen, default); frozen decay 1.0 is an equivalent ablation variant*
Spectral reg. weight (λ_{spec})	0.01
Direction loss weight (λ_{dir})	1.0
Prediction loss weight (λ_{pred})	1.0
Retrieval variant only (CONTRAST-HIGH, Section 4.4):	
Contrastive loss weight (λ_{con})	1.0
Contrastive temperature (τ)	0.07
Positive threshold (τ_{pos})	0.9

all four Phase 5 checkpoints (Section 4.4, Table 6). CONTRAST-EXTREME-15K (λ=2.0) remains single-seed and was preempted by spot-pod reclaim at around step 12,600 before completing its 15,000-step budget. Its partial peak $\Delta_{\cos} \approx 0.9917$ is consistent with the $\lambda_{\text{con}} = 1.0$ 15K run and is excluded from Tables 2 and 4 in the main body; the per-step delta cosines on the partial checkpoint ($s_{1,2,3} \approx 0.991, 0.992, 0.988$) are reported here for reproducibility but are not used as a prediction-quality headline. The frozen-target ablation (EMA decay 1.0, Section 4.2) uses seeds 42 and 43, matching the Phase 5 two-seed protocol; peak val $\Delta_{\cos}$ lands at 0.9925 (seed 42) and 0.9938 (seed 43), within ± 0.002 of the near-frozen baseline.

All headline numbers in the main body are reported as the mean of the last five validation evaluations of a run, $\pm$ sample standard deviation across those evaluations. Peak validation values are an average of ~ 0.5 points higher than mean-of-last-five for the contrastive 3K runs (which are still actively training at the end of their evaluation horizon) and within ~ 0.005 for the converged predictor 15K runs; we therefore avoid peak reporting. Both numbers are computed by `evaluation/rescore_wandb_history.py` which pulls each run's history from the W&B API and recomputes the mean-of-last-N statistics.

B Quantisation Details

We provide three quantisation tiers of increasing aggressiveness. All three share the same "quantise-only-matmul-weights" philosophy: only the four `Linear` weight matrices per transformer block (attention in-projection, attention out-projection, MLP up-projection, MLP down-projection) are quantised—these account for ∼92% of the model's matmul weights. Table 15 details what is

quantised at each tier.

Table 15: Quantisation tiers. "f32" = kept at full precision; "INT8" = symmetric 8-bit per-channel; "Q4" = 4-bit block-quantised ($Q4_0$ format, block size 32).

Component	**INT8**	**Q4**	**Q4-full**
Attn in-projection W_{QKV}	INT8	Q4	Q4
Attn out-projection W_O	INT8	Q4	Q4
MLP up-projection W_{up}	INT8	Q4	Q4
MLP down-projection W_{dn}	INT8	Q4	Q4
Token embedding	f32	f32	INT8
Positional encoding	f32	f32	INT8
LayerNorm γ, β	f32	f32	f32
All biases	f32	f32	f32
Action encoder	f32	f32	f32
Approx. model size (128-d)	∼1.0 MB	∼1.0 MB	∼0.6 MB
Compression vs. f32 baseline	3.0×	4.7×	5.1×

$Q4_0$ block format. Each Q4 block covers 32 contiguous weight elements and stores a single `f32` scale factor plus 16 bytes of nibble pairs (20 bytes total):

$$w_i = (\texttt{nibble}_i - 8) \cdot \texttt{scale}, \qquad i \in [0, 31].$$

Both low and high nibbles of each byte encode a signed 4-bit value offset by 8 into the unsigned range $[0, 15]$.

B.1 Quantisation Procedure

Algorithm 1 describes the `QuantizedLinear` construction path used for INT8. The Q4 path follows the same structure but packs weights into $Q4_0$ blocks instead of per-element `i8` values.

Percentile calibration. The default `minmax` strategy sets the scale from the absolute maximum of each output channel. When a channel contains outlier weights, this wastes dynamic range on a single extreme value. Percentile calibration instead uses the p-th percentile (e.g., p=99.9) of sorted absolute values, clipping the rare outliers. This trades a small clipping error for tighter quantisation of the bulk distribution.

C Inference Benchmarks

All latency measurements below were collected on an Apple M-series CPU at `f32` precision using the Zig SIMD backend with Apple Accelerate BLAS dispatch. Each measurement is the median of 100 warm runs after 10 warm-up iterations; p95 is from the same sample.

C.1 Per-Stage Latency

Table 16 breaks down pipeline latency by stage. Sequence lengths S refer to the input token count after the Python tokeniser.

Algorithm 1: Per-channel INT8 quantisation with optional percentile calibration.

Input: Weight matrix $W \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}}}$, calibration mode $\in \{\texttt{minmax}, \texttt{percentile}(p)\}$
Output: Quantised weights $\hat{W} \in \mathbb{Z}^{C_{\text{out}} \times C_{\text{in}}}$, scales $s \in \mathbb{R}^{C_{\text{out}}}$

for $c \leftarrow 0$ **to** $C_{\text{out}} - 1$ **do**
  if mode = *minmax* **then**
    $m_c \leftarrow \max_j |W_{c,j}|$
  else
    Sort $\{|W_{c,j}|\}_j$ ascending
    $m_c \leftarrow$ value at rank $\lfloor p/100 \cdot (C_{\text{in}} - 1) \rceil$
  end
  $s_c \leftarrow m_c \,/\, 127$
  for $j \leftarrow 0$ **to** $C_{\text{in}} - 1$ **do**
    $\hat{W}_{c,j} \leftarrow \text{clamp}\big(\text{round}(W_{c,j}/s_c),\ -128,\ 127\big)$
  end
end
Transpose $\hat{W}$ to GEMM-ready layout $[C_{\text{in}}, C_{\text{out}}]$

Table 16: Per-stage inference latency (ms) on Apple M-series CPU, f32, Zig SIMD + Accelerate. 128-d model (`g8`).

Stage	**Seq. len**	**p50**	**p95**	**Mean**
Encoder	S=64	<2	2.3	1.8
Encoder	S=256	<8	9.1	7.5
Encoder	S=512	<15	16.8	14.2
Action encoder	any	<1	0.4	0.3
Predictor (1 step)	—	<2	2.1	1.7
Full pipeline	S=64	<5	5.6	4.5

Encoder latency scales approximately linearly with sequence length due to the $O(S^2)$ self-attention being dominated by the $O(S \cdot d^2)$ linear projections at this model width (d=128).

C.2 Backend Comparison

Table 17 compares the two available CPU execution paths. The Zig FFI backend dispatches matrix operations through SIMD-optimised Zig kernels and Apple Accelerate (`cblas_sgemm`). The pure-Rust path is a portable fallback used for WASM and `no_std` targets.

The Zig backend achieves a $\sim$3.6$\times$ speedup over pure Rust, primarily from vectorised GEMM kernels and BLAS delegation for the larger matmuls. The pure-Rust path remains useful as a correctness reference and as the sole option for WASM and embedded targets where native FFI is unavailable.

Table 17: Backend comparison: full pipeline latency at S=64, f32 precision, 128-d model. Apple M-series CPU.

Backend	**p50 (ms)**	**p95 (ms)**	**Speedup**
Zig FFI + Accelerate	<5	5.6	1.0× (baseline)
Pure Rust (portable)	∼18	21.4	0.28×

Table 18: Encoder latency at S=512 by model variant (Zig FFI + Accelerate, f32, Apple M-series CPU).

Variant	**Dim**	**p50 (ms)**	**Params**
champion (ema-frozen-15K)	128	<15	1,113,984
expa (pre-fix, historical)	192	∼28	∼2.4 M

C.3 Scaling with Model Width

C.4 CommitPackFT Retrieval: Per-Criterion Breakdown

Table 19 expands the in-distribution retrieval summary of Section 4.4 to a fine-grained per-criterion breakdown at n_q=500, n_g=2000. The six CodeWM columns are the six Phase 3 / Phase 5 15K checkpoints evaluated in the main body (three near-frozen EMA seeds plus two contrastive seeds plus two frozen-target ablation seeds); the four criteria span edit-type only, joint (edit type × scope), and two `by_action_cos` thresholds. The breakdown is moved to the appendix because the summary row it supports (Section 4.4, cross-repo Table 7) is the main-body headline; this table is kept for reviewer reproducibility and for the non-saturated `by_action_cos` criteria readers may want to verify.

Table 19: Honest retrieval MRR on CommitPackFT by criterion (n_q=500, n_g=2000). Best per row in bold. The predictor champion PRED_S43 is the best CodeWM variant on 3 of 4 criteria; CodeWM beats CodeBERT on 3 of 4 criteria (everything except the saturated `by_edit_type`). The frozen-target ablation checkpoints (FT_S42, FT_S43, EMA decay 1.0, Section 4.2) land within ±0.015 of PRED_S43 on `by_joint` and the `by_action_cos` criteria, confirming that in-distribution retrieval is preserved under the static-target ablation; `by_edit_type` was not recomputed for frozen-target checkpoints.

Criterion	CodeWM ema_s42	CodeWM pred_s43	CodeWM con_s42	CodeWM con_s43	CodeWM FT_s42	CodeWM FT_s43	CodeBERT (124M)
`by_edit_type`	0.9975	0.9942	0.9875	0.9900	—	—	**1.0000**
`by_joint`	0.8261	**0.8418**	0.8246	0.8200	0.8380	0.8293	0.7473
`by_action_cos`> 0.9	0.8109	**0.8114**	0.7969	0.7967	0.7991	0.7860	0.6989
`by_action_cos`> 0.95	0.7536	**0.7538**	0.7239	0.7363	0.7478	0.7368	0.6207

D Browser Demo

The Code WM inference pipeline compiles cleanly to WebAssembly via the pure-Rust backend, giving a self-contained browser demo that runs entirely client-side with no server round-trips after page load. The demo takes a Python snippet, tokenises it into the 662-token AST vocabulary,

encodes it through the 6-loop encoder, L2-normalises the output, and ranks a pre-embedded 20-file corpus by cosine similarity.

Table 20: WASM demo bundle breakdown.

Component	**Raw size**	**Gzipped**
WASM binary (`wasm-opt` -Oz)	∼2.6 MB	∼1.2 MB
Model weights (f32, embedded)	∼2.6 MB	∼0.6 MB
Corpus embeddings (20 × 128-d)	29 KB	∼12 KB
JS glue + HTML	∼50 KB	∼15 KB
Total deliverable	∼5.6 MB	∼2 MB

Browser-side latency is dominated by the encoder pass: ∼80–150 ms at S=512 via the pure-Rust matmul kernels (no Zig FFI, no BLAS), plus <1 ms for tokenisation and cosine scan, totalling ∼85–155 ms end-to-end. This is higher than the Appendix C native numbers because WASM does not use the Zig SIMD path; enabling WASM SIMD (`v128` intrinsics) is an open optimisation.

E Training Orchestration

All training runs and sweeps reported in this paper are launched through an in-house experiment orchestration platform that provisions rental GPU instances, synchronises training scripts and data, runs the training loop with live Weights & Biases logging, and manages checkpoint retention. The broader research workflow was agent-assisted: an automation layer selected hyperparameter configurations from a declarative sweep specification, dispatched them to pods, and rescored runs post-hoc from W&B histories. The platform is not a contribution of this paper; we mention it only to document that all three predictor seeds and all 3K contrastive seed sweeps were executed through the same automated pipeline with identical hyperparameters, data, and code revision, so the 3-seed variance numbers in Section 4.3 reflect randomness in seed selection and GPU non-determinism rather than hand-run configuration drift. Full run configurations, W&B run IDs, and the training launcher are available in the project repository.