# Structured Label Smoothing over Finite Scalar Quantization for Discrete World Models

**Florent Tariolle***
INSA Rouen Normandy

Website Code

## Abstract

Discrete World Models tokenize observations with a learned quantizer and predict next-frame tokens with a transformer, but standard cross-entropy treats every incorrect prediction as equally wrong. Finite Scalar Quantization (FSQ) makes a richer signal available by construction: each code sits on an integer coordinate lattice, so a token one step away in one dimension is a near-miss while a token at the opposite corner is a gross error. We introduce *Structured Label Smoothing* (SLS), which replaces the one-hot training target with a kernel over codebook coordinates, so a near-miss prediction is treated as a near-miss rather than a gross error. An isotropic kernel with bandwidth fixed by a first-neighbour rule gives a zero-calibration hyperparameter that has a single principled choice independent of codebook size. We integrate SLS into a complete real-time Vision-Model-Controller (V-M-C) pipeline for Geometry Dash: an FSQ-VAE tokenizer, a causal transformer dynamics model with action tokens interleaved in the sequence, and a CNN actor-critic in the predicted-token-grid space, trained entirely in imagination on dreamed rollouts and deployed at 30 FPS on the real game via screen capture.

## 1 Introduction

Learning WMs that predict future observations from actions is central to model-based reinforcement learning [1]. Recent approaches tokenize visual frames into discrete codes and train autoregressive transformers to predict token sequences [2, 3]. Finite Scalar Quantization [4] offers a simple alternative to VQ-VAE codebooks: each latent vector is independently rounded to one of $L$ levels per dimension, yielding $\prod_d L_d$ implicit codes with no codebook collapse and no auxiliary losses.

A key property of FSQ, largely unexploited in downstream tasks, is that its codebook has a natural coordinate structure. Each code maps to a point on a $D$-dimensional integer lattice, and the distance between two codes in this space is semantically meaningful: codes that differ by one step in one dimension represent visually similar patches. Yet discrete WMs supervise next-token prediction with one-hot cross-entropy on tokenizer indices [2], which exploits no coordinate metric on the latent space and treats a near-miss (one lattice step from the target) and a gross error (opposite corner of the codebook) as equally informative training signals.

We propose *Structured Label Smoothing* (SLS), a soft-target objective whose target distribution is a kernel over FSQ coordinate distances, so the supervision signal reflects the geometry the quantizer has already learned. Classical uniform label smoothing [5] is the structureless limit of this construction and a natural ablation, not the baseline SLS improves over.

This can be viewed as part of a broader shift from data-agnostic heuristics to structure derived from learned representations, analogous to how learned strided convolutions replaced fixed max-pooling

*Correspondence to florent.tariolle@insa-rouen.fr

for spatial downsampling [6]. Unlike knowledge distillation, which requires a separate teacher model to produce soft targets, SLS exploits geometry already present in the FSQ codebook at no additional cost.

We evaluate SLS within a complete Vision-Model-Controller (V-M-C) pipeline for Geometry Dash, a fast-paced platformer that demands frame-accurate predictions under a 33 ms latency budget. Our system consists of three components: (1) an FSQ tokenizer that encodes $64 \times 64$ Sobel edge maps into $8 \times 8$ grids of discrete tokens, (2) a causal transformer that interleaves action tokens with frame tokens in a single sequence and predicts the next frame's tokens via parallel decoding, and (3) a lightweight CNN actor-critic on the predicted token grid, warm-started with Behavioural Cloning and fine-tuned with PPO [7] entirely in dreamed rollouts. The three components are trained sequentially – each on the frozen output of the previous one – and the full pipeline runs in real time at 30 FPS on consumer hardware.

Our contributions are:

1. **Structured Label Smoothing (SLS)**: a training objective that exploits FSQ coordinate geometry to distribute smoothing mass according to codebook distance. We study three kernel families and adopt an isotropic bandwidth set by a first-neighbour rule, giving a zero-calibration hyperparameter with a single principled choice independent of codebook size. The formulation generalizes to any discrete latent space with a coordinate metric.
2. **A real-time discrete WM system**: a complete V-M-C pipeline for Geometry Dash running at 30 FPS, combining FSQ tokenization, a transformer with action tokens interleaved in the sequence and an action-conditional contrastive auxiliary loss [8], and a CNN actor-critic trained via BC + PPO-in-dreams.

## 2 Related work

**Discrete WMs.** The Vision-Model-Controller paradigm [1] introduced encoding frames with a VAE, modeling dynamics with an RNN, and training a controller in dreamed rollouts. IRIS [2] replaced the VAE with a discrete tokenizer and an autoregressive transformer, alternating tokenizer and dynamics updates; we follow IRIS's architectural template most closely, with a frozen FSQ tokenizer and a transformer trained on top of it. DreamerV3 [3] demonstrated that categorical RSSM latents generalize across diverse domains; this approach was scaled further in [9]. TWISTER [8] added action-conditioned contrastive predictive coding to improve temporal coherence; we adopt their AC-CPC auxiliary loss. FSQ [4] is typically positioned as a frozen tokenizer for downstream tasks, and its coordinate-lattice structure has not been exploited at training time. SLS exploits exactly this structure on the loss side, without modifying the tokenizer or its training.

**Finite Scalar Quantization.** FSQ [4] replaces learned codebooks with per-dimension rounding to fixed levels, eliminating codebook collapse and commitment losses. Each code is a point on a $D$-dimensional integer lattice, giving the codebook a natural coordinate structure. iFSQ [10] introduces a simple modification of the FSQ bounding function that improves bin utilization; we adopt it in our encoder.

**Joint-Embedding Predictive Architectures (JEPA).** LeWorldModel [11] recently demonstrated a stable end-to-end JEPA WM from pixels using continuous latents and a SIGReg anti-collapse regularizer. We explored a JEPA-style joint-embedding variant of our system, in which context and target frames are encoded by a single online FSQ encoder into a shared discrete latent space and prediction gradients flow back into the encoder through a straight-through path, with the pixel decoder retained only as an anti-collapse regularizer. This variant did not improve on the sequentially-trained baseline reported here; we summarize the negative result and the SIGReg adaptation we attempted in Section 5.1.

**Label smoothing.** Label smoothing [5] proposed replacing one-hot targets with a uniform mixture to reduce overconfidence. While effective as a regularizer, uniform smoothing ignores class relationships. Knowledge distillation and soft-target methods distribute mass based on teacher predictions, but require a separate teacher model. We instead derive the target distribution from the codebook geometry itself, requiring no additional model.

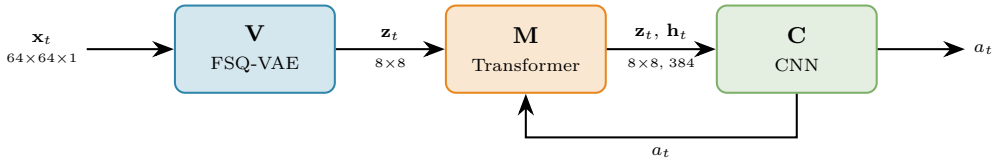


Figure 1: Vision-Model-Controller pipeline. The FSQ-VAE (V) tokenizes Sobel edge maps into a grid of discrete codes, the causal transformer (M) predicts next-frame tokens with action tokens interleaved in the sequence, and the CNN controller (C) reads the predicted token grid alongside the transformer hidden state to select actions. The action feedback loop enables autoregressive dreaming.

**Adaptive conditioning.** FiLM [12] introduced adaptive affine conditioning via scale and shift. DiT [13] applied this as AdaLN-Zero with zero-initialized gates for stable training of diffusion transformers, and QK-norm [14] (RMSNorm on queries and keys per head) was introduced to stabilize attention logits in deep AdaLN models. LeWorldModel [11] applied AdaLN to world-model transformers. Our baseline keeps actions as tokens in the sequence; we treat AdaLN-Zero conditioning as an ablation (Section 5.1).

## 3 Method

Our system follows the Vision-Model-Controller (V-M-C) paradigm [1] (Figure 1). We describe each component below, with Structured Label Smoothing (Section 3.2) as the primary contribution, integrated into a complete real-time pipeline. The three components are trained sequentially – each on the frozen output of the previous one – a deliberate choice motivated by the ablation results in Section 5.1.

### 3.1 FSQ tokenization (Vision)

We encode $64 \times 64$ grayscale Sobel edge maps using an FSQ-VAE with levels $[8, 5, 5, 5]$, producing 1000 implicit codes. The encoder applies three stride-2 convolutional stages with residual blocks, projecting to a 4-channel latent map of size $8 \times 8$. Each spatial position is independently quantized using the iFSQ bounding function [10], $2\sigma(1.6z) - 1$ in place of $\tanh(z)$, scaled to half-levels and rounded, yielding 64 tokens per frame. The encoder is trained on consecutive frame pairs with a pixel-MSE reconstruction loss and two regularizers in the spirit of geometric regularization: a temporal slowness term that penalizes large $\|\mathbf{z}_e^{t+1} - \mathbf{z}_e^t\|^2$ between adjacent frames, and a uniformity term that spreads pre-quantization features across the FSQ hypercube. The decoder mirrors the encoder with transposed convolutions and is discarded after training.

### 3.2 Structured Label Smoothing

**Motivation.** Standard label smoothing [5] replaces a one-hot target $\mathbf{e}_i$ with

$$q_{\text{uniform}}(j \mid i) = (1 - \varepsilon)\,\delta_{ij} + \frac{\varepsilon}{V - 1}(1 - \delta_{ij}), \tag{1}$$

where $V = 1000$ is the vocabulary size and $\varepsilon$ controls the smoothing strength. This distributes $\varepsilon$ uniformly across all alternatives, ignoring codebook geometry.

**FSQ coordinate distance.** An FSQ codebook with levels $\mathbf{L} = (L_1, \ldots, L_D)$ assigns each code $i$ a coordinate vector $\mathbf{c}(i) = (c_1(i), \ldots, c_D(i))$ on a $D$-dimensional integer lattice. We define the squared lattice distance

$$d^2(i, j) = \sum_{d=1}^{D} \left(c_d(i) - c_d(j)\right)^2. \tag{2}$$

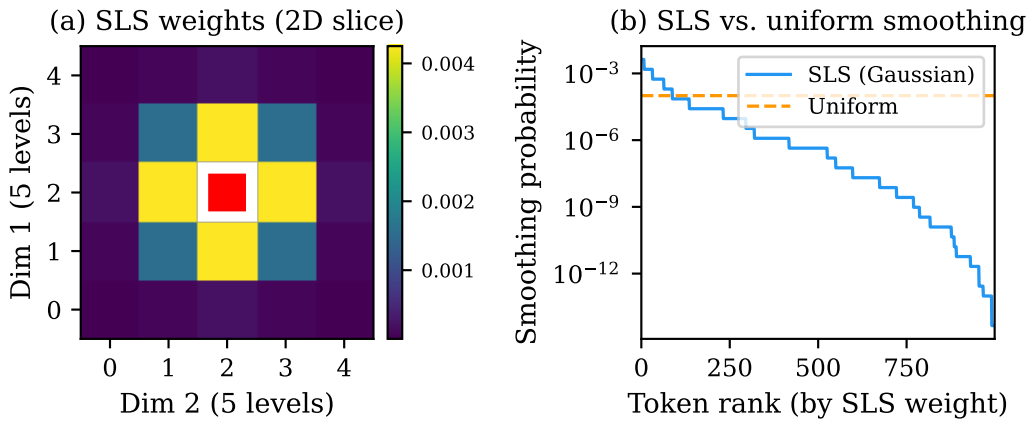


Figure 2: SLS target distribution for a token near the centre of the codebook. **(a)** 2D slice over two lattice dimensions with the remaining two held at the target's values: weights decay as a Gaussian kernel over lattice distance, red square marks the target. **(b)** All $V - 1$ alternative tokens sorted by SLS weight (log scale) vs. uniform label smoothing. SLS concentrates mass on near-miss tokens while suppressing distant ones.

**Kernel-family study.** We compare three kernels over $d = \sqrt{d^2(i,j)}$:

$$k_{\text{Gauss}}(d) = e^{-d^2/2\sigma^2}, \qquad k_{\text{Laplace}}(d) = e^{-d/\sigma}, \qquad k_{\text{Cauchy}}(d) = \frac{1}{1 + d^2/\sigma^2}. \tag{3}$$

The three families differ in tail behaviour. Cauchy has polynomial tails and leaks meaningful mass to distant neighbours; Laplace has a cusp at $d = 0$ and an exponential tail; Gaussian has the thinnest tails, decaying faster than either alternative once $d$ exceeds the bandwidth. We take Gaussian with $\sigma = 0.7$ as the current default: thin tails ensure distant neighbours receive negligible mass ($k(2) \approx 0.02$), while the bandwidth still places meaningful mass on the immediate lattice neighbour ($k(1) \approx 0.37$, matching Laplace at $\sigma = 1$).

**Isotropic first-neighbour bandwidth.** A natural alternative is per-dimension weights $\alpha_d$ calibrated from an empirical sensitivity probe: perturbing each FSQ dimension by $\pm 1$, decoding, and measuring reconstruction MSE. The probe gives data-dependent weights, but ties the smoothing target to a particular trained codebook and varies across training runs. We instead use an isotropic kernel and fix $\sigma$ by a geometric rule: choose the bandwidth so that the immediate integer lattice neighbour sits near the kernel's half-maximum. This gives a zero-calibration hyperparameter with a single principled choice, independent of the codebook size or the realized dimension scales. The sensitivity probe is retained as an analysis tool to characterize a learned codebook's anisotropy and cross-dimension coupling, but not as a hyperparameter source.

**Structured smooth target.** We replace the uniform distribution in Equation 1 with a kernel over FSQ distances:

$$q_{\text{SLS}}(j \mid i) = \begin{cases} 1 - \varepsilon & \text{if } j = i, \\ \varepsilon \cdot \dfrac{k(d(i,j))}{\sum_{l \neq i} k(d(i,l))} & \text{otherwise.} \end{cases} \tag{4}$$

The full target matrix $\mathbf{Q} \in \mathbb{R}^{V \times V}$ is precomputed once per kernel choice and stored on GPU (Figure 2). We use $\varepsilon = 0.1$.

**Integration with focal loss.** To handle class imbalance (background tokens dominate over obstacle and spike tokens), we combine SLS with focal loss [15]:

$$\mathcal{L}_{\text{SLS-focal}} = -\sum_{j=1}^{V} (1 - p_j)^{\gamma} \, q_{\text{SLS}}(j \mid i) \, \log p_j, \tag{5}$$

where $p_j = \text{softmax}(\mathbf{z})_j$ and $\gamma = 2.0$.

### 3.3 Transformer WM (Model)

The WM is a causal transformer (384 embedding dim, 8 heads, 8 layers) that predicts the next frame's tokens in parallel given a context of $K = 4$ past frames and their associated actions, on top of a frozen FSQ tokenizer (Section 3.1).

**Sequence construction.** Each frame contributes 64 visual tokens (from the $8 \times 8$ FSQ grid) plus one status token (ALIVE or DEATH), for 65 positions per frame block. Actions are embedded into the same 384-dimensional space and inserted as single tokens between consecutive frame blocks, giving the input sequence

$$[f_0\, a_0\, f_1\, a_1\, \ldots\, f_{K-1}\, a_{K-1}\, f^{\text{tgt}}]$$

of length $K \cdot (65 + 1) + 65 = 329$. All target-frame positions are replaced with a learnable `[MASK]` embedding so the model never sees the ground-truth target tokens during training.

**3D Rotary Position Embedding.** We apply factored RoPE [16] encoding three positional axes: spatial row, spatial column, and temporal frame index. The head dimension is partitioned into three bands (rows, columns, time) with separate base frequencies ($\theta_{\text{spatial}} = 10$, $\theta_{\text{time}} = 10{,}000$). Status and action tokens receive the grid centre coordinates to represent whole-frame summaries.

**Block-causal attention.** Within each frame block, attention is bidirectional (tokens attend to all positions in the same frame). Across frame and action blocks, attention is strictly causal: each block attends only to current and past blocks. At inference, the next frame's 64 visual tokens are decoded in a single parallel forward pass from the masked target block.

**Action-conditional contrastive auxiliary loss.** Following TWISTER [8], we add an action-conditional contrastive predictive coding (AC-CPC) auxiliary loss: for each $k \in \{1, \ldots, K\}$, an MLP head predicts the future hidden state $\mathbf{h}_{t+k}$ from the current $\mathbf{h}_t$ and the intervening actions $a_{t:t+k-1}$, scored against in-batch negatives with InfoNCE. The auxiliary loss enters the total objective with weight $\lambda_{\text{cpc}} = 0.1$.

**FSQ-neighbor token noise.** On context tokens, with probability 5% per token we replace the token with a lattice neighbour obtained by perturbing a single FSQ dimension by $\pm 1$. An earlier dual-noise scheme also applied uniform random-token replacement, but uniform replacement contradicts the SLS geometry (it presents distant tokens as substitutable for the target, the opposite of the SLS signal on the loss side). We kept only the FSQ-neighbor branch so that context and loss agree on what counts as a near-miss.

**Output projection and total loss.** The output projection shares weights with the input embedding (weight tying). The total training loss is

$$\mathcal{L} = \mathcal{L}_{\text{SLS-focal}} + \lambda_{\text{cpc}}\, \mathcal{L}_{\text{AC-CPC}}, \tag{6}$$

applied to the transformer parameters with the FSQ tokenizer frozen.

### 3.4 PPO controller (Controller)

The controller is a lightweight CNN actor-critic, in the spirit of IRIS [2] and DIAMOND, that reads the predicted token grid alongside the transformer's hidden state and outputs a jump probability and a state value estimate.

**Architecture.** Token IDs are embedded into a small learnable space and reshaped into the $8 \times 8$ spatial grid; two stride-1 convolutions with max-pooling produce a 256-dimensional feature vector, which is concatenated with the transformer's hidden state $\mathbf{h}_t \in \mathbb{R}^{384}$ and passed to zero-initialized linear actor and critic heads. The full controller has $\sim$45K parameters.

**Training in dreams.** The controller is first pretrained via Behavioural Cloning (BC) on expert demonstrations, then fine-tuned with PPO [7] in dreamed rollouts. At each iteration, several hundred rollouts of up to 45 steps are generated by autoregressively sampling from the frozen WM. Rollouts terminate when the predicted death probability exceeds 0.5. Rewards are $+1$ per survival step with

a $-0.2$ penalty per jump to discourage gratuitous jumping (an idle agent dies immediately upon reaching an obstacle, while a jumping agent survives longer by spending time airborne, creating a deceptive local optimum). We use GAE [7] with $\gamma = 0.995$, $\lambda = 0.95$, and clip ratio $\varepsilon_{\text{PPO}} = 0.2$. A Dreamer-style $\lambda$-return actor-critic is a natural replacement: on-policy imagined rollouts make PPO's off-policy clipped surrogate unnecessary. We leave this to future work.

## 4 Experimental setup

**Environment.** Geometry Dash is a rhythm-based platformer where the player controls a cube moving at constant horizontal speed. The only action is a binary jump. Colliding with any obstacle causes instant death. Levels are deterministic: the same action sequence always produces the same outcome, but frame-accurate timing is required.

**Data collection.** We collect roughly 4,200 gameplay episodes ($\sim$250K frames) including both intentional-death episodes (covering diverse failure modes) and 36 expert demonstrations (full level clears). Vertical shift augmentation ($\pm 2$, $\pm 4$ pixels) expands the dataset $5\times$. Frames are preprocessed with Sobel edge detection and resized to $64 \times 64$.

**Training details.** The FSQ-VAE and the transformer are trained sequentially with AdamW using cosine annealing and warmup. The FSQ-VAE is trained for 200 epochs (LR $4 \cdot 10^{-3}$, batch 32) with the slowness and uniformity regularizers weighted at $\alpha_{\text{slow}} = 0.1$ and $\alpha_{\text{uniform}} = 0.01$, after which the encoder is frozen and its tokens are cached. The transformer is then trained for 200 epochs (LR $10^{-3}$, weight decay $0.05$, dropout $0.1$, batch 256) with 5% FSQ-neighbor token noise on context tokens, and death frames oversampled $15\times$ via weighted sampling. BF16 mixed-precision training is used throughout; BF16 requires no loss scaling and is numerically more stable than FP16. Training runs on a single NVIDIA A100 GPU.

**Training throughput.** We benchmarked `torch.compile` modes under BF16 on our joint-training workload using subprocess isolation to prevent CUDA allocator fragmentation across runs. All `torch.compile` modes are within a few percent of each other; we adopt `reduce-overhead` with `fullgraph=True` as the default: it is faster than `default` mode while avoiding the lengthy autotune phase that `max-autotune` adds at startup in exchange for marginal additional steady-state throughput. This configuration is applied to the world-model and controller training scripts. A secondary NVIDIA RTX 2060 SUPER (Turing, SM 7.5) was used in parallel for auxiliary experiments. Turing lacks native BF16 (software-emulated and much slower than FP32 on this card) and has no TF32 support; Inductor's channels-last layout heuristic additionally triggers a stride assertion in the convolution backward, which we disable by setting `torch._inductor.config.layout_optimization = False`. Under these constraints the best local configuration is `torch.compile` in `default` mode with FP16 and a `GradScaler`.

**Real-time inference.** The deployed agent processes each frame at a fixed 30 FPS cadence, with a per-iteration sleep padding any spare time so the capture rate stays constant rather than racing the GPU. Within each iteration we minimize stalls: the captured image is staged into a pinned-memory tensor and uploaded with `non_blocking=True`, so the CPU returns from the H2D copy immediately and queues the subsequent CUDA dispatches (FSQ encode, transformer, controller) without waiting for the transfer to physically complete. The transformer's context window is held as a GPU-resident ring buffer of FSQ token indices, eliminating a GPU$\rightarrow$CPU$\rightarrow$GPU round-trip per frame. The encoder forward pass is wrapped in a CUDA Graph after a short warm-up, and the entire model graph is compiled with `torch.compile` in `reduce-overhead` mode. Under these optimizations the full pipeline (screen capture, Sobel preprocessing, FSQ encode, transformer step, controller action) fits comfortably within the 33 ms budget on a single RTX 2060 SUPER.

**BC pretraining ablation.** To verify that BC pretraining is a convergence accelerator rather than a stability requirement, we train one PPO run from random initialization. Cold-start PPO converges to the same plateau as BC-initialized PPO on all metrics (eval survival, entropy, jump ratio) but requires roughly 2,000 additional iterations ($\approx$6 hours on A100); BC itself takes $\sim$5 minutes, giving a large compute ROI. BC is retained in the pipeline on this basis.

**Evaluation metrics.** We evaluate on: (1) next-token prediction accuracy and cross-entropy on held-out episodes, (2) death prediction F1 score, and (3) real-environment gameplay performance measured as percentage of level completed (mean over $N = 100$ automated runs).

# 5 Results

*Results pending model freeze and ablation experiments.*

## 5.1 Ablations and negative results

We report a set of architectural changes which, on paper, were expected to improve over the baseline of Section 3, but did not yield real-game gains in our experiments. Together they motivate the deliberately conservative system design of the current paper. Detailed per-ablation tables are pending model freeze.

**Joint training of FSQ and transformer.** We trained a JEPA-style variant in which the FSQ encoder and the transformer are optimized together in a single AdamW group, with prediction gradients routed back into the encoder through a learnable straight-through projection (`fsq_grad_proj`) inserted between the pre-rounding continuous latents and the transformer input. A light pixel-MSE loss is retained as an anti-collapse regularizer on the encoder, and a single global gradient-norm clip is applied to the combined parameter set following [11]. This setup trains stably and produces lower validation cross-entropy than the sequentially-trained baseline, but the resulting FSQ codebook drifts toward a representation that is harder to act on: real-game level progress consistently dropped relative to the sequentially-trained baseline of identical capacity. Removing the pixel-MSE anchor (pure-JEPA) causes the encoder to collapse; our adaptation of SIGReg [11] to the discrete factorized FSQ case, using a Cramér-von-Mises test against a factorized uniform target, was insufficient to prevent joint-distribution degeneracy when the prediction target comes from the online encoder. Finding a decoder-free discrete anti-collapse signal remains open work.

**AdaLN-Zero action conditioning.** Rather than inserting actions as tokens, we conditioned each transformer block on the action history via Adaptive Layer Normalization [13, 12]. The $K$-step action sequence was projected through an MLP to per-layer scale, shift, and gate parameters modulating each sublayer. To stabilize training we needed both QK-norm [14] (RMSNorm on queries and keys per head) and gate-bias initialization to 1 rather than DiT's standard zero: zero-initialized gates suppressed both conditioning and learning, and without QK-norm, attention logits diverged. Even with both fixes, the AdaLN variant matched but did not surpass the in-sequence-action baseline on real-game level progress, while adding architectural complexity. The token-in-sequence design is what we ship.

**Larger transformer ($384 \to 512$).** Scaling the embedding dimension from $384 = 384$ to $512$ at otherwise matched capacity improves dream-quality metrics (validation cross-entropy and AC-CPC score) but does not translate into real-game gains, with extra training time per epoch. We retain the smaller model.

**MLP controller on $\mathbf{h}_t$ alone.** We tested a controller that consumes only the transformer hidden state $\mathbf{h}_t$, with no CNN over the predicted token grid, motivated by the fact that $\mathbf{h}_t$ already encodes spatial and temporal structure through $8$ transformer layers with 3D-RoPE. This MLP variant converges 2–5$\times$ faster in PPO and has the smallest train-eval gap of any controller we trained, but the CNN-on-tokens variant from Section 3.4 dominates it on real-game level progress, suggesting the explicit token-grid signal carries information that is washed out in the global hidden-state summary.

# 6 Limitations

Our evaluation is limited to a single game (Geometry Dash) with a binary action space and deterministic level layouts. The generalization of SLS to environments with higher-dimensional actions, stochastic dynamics, or RGB observations remains to be validated. The system is trained sequentially – FSQ, then transformer, then controller – which is operationally simple but forfeits the possibility of

prediction gradients shaping the codebook. Our preliminary attempts at a JEPA-style joint-training variant did not produce real-game gains over this sequential baseline (Section 5.1); whether a more careful version of joint training, or a stronger decoder-free anti-collapse signal, would close this gap is open work.

## 7 Conclusion

We introduced Structured Label Smoothing, a training objective that exploits the coordinate geometry of FSQ codebooks. A kernel-family study over Gaussian, Laplace, and Cauchy families, combined with an isotropic first-neighbour bandwidth, gives a zero-calibration hyperparameter with a single principled choice independent of codebook size. We integrate SLS into a real-time discrete-WM system on Geometry Dash: an FSQ-VAE tokenizer, a causal transformer with action tokens interleaved in the sequence and a TWISTER-style action-conditional contrastive auxiliary loss, and a CNN actor-critic on the predicted token grid trained via BC and PPO entirely in dreamed rollouts. The three components are trained sequentially, and the full pipeline runs at 30 FPS on consumer hardware. A set of architectural variants (joint FSQ+M training, AdaLN-Zero action conditioning, scaling to a larger transformer, an MLP controller on the hidden state) is reported as ablations: each is more elegant on paper but did not improve real-game level progress in our experiments, motivating the deliberately conservative baseline of this paper. SLS is in principle applicable to any model that predicts over a discrete latent space with a coordinate metric. A natural extension is to make the kernel bandwidth $\sigma$ learnable and jointly optimized with the WM, moving SLS from a fixed prior to a fully end-to-end objective; care must be taken to prevent degenerate solutions in which the learned targets collapse to uniform smoothing.

## Acknowledgments and Disclosure of Funding

We thank Maël Planchot for contributing gameplay data. The NVIDIA A100 GPU used for all primary training runs was provided by MesoNet (CRIANN, Rouen) through the Representation Learning course at INSA Rouen Normandy. Geometry Dash is developed by RobTop Games.

## References

[1] David Ha and Jürgen Schmidhuber. World models. In *Neural Information Processing Systems*, 2018.

[2] Vincent Micheli, Eloi Alonso, and François Fleuret. Transformers are sample-efficient world models. In *International Conference on Learning Representations*, 2023.

[3] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 2025.

[4] Fabian Mentzer, David Minnen, Eirikur Agustsson, and Michael Tschannen. Finite scalar quantization: VQ-VAE made simple. In *International Conference on Learning Representations*, 2024.

[5] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the Inception architecture. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[6] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. In *International Conference on Learning Representations (Workshop)*, 2015.

[7] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[8] Maxime Burchi and Radu Timofte. TWISTER: Learning transformer-based world models with contrastive predictive coding. In *International Conference on Learning Representations*, 2025.

[9] Danijar Hafner, Wilson Yan, and Timothy Lillicrap. Training agents inside of scalable world models. *arXiv preprint arXiv:2509.24527*, 2025.

[10] Mohammad Hassan Vali, Tom Bäckström, and Arno Solin. iFSQ: Improving FSQ for image generation with 1 line of code. In *International Conference on Learning Representations*, 2026.

[11] Lucas Maes, Quentin Le Lidec, Damien Scieur, Yann LeCun, and Randall Balestriero. LeWorldModel: Stable end-to-end joint-embedding predictive architecture from pixels. *arXiv preprint arXiv:2603.19312*, 2026.

[12] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer. In *AAAI Conference on Artificial Intelligence*, 2018.

[13] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *International Conference on Computer Vision*, 2023.

[14] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorber, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *International Conference on Machine Learning*, 2024.

[15] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *International Conference on Computer Vision*, pages 2980–2988, 2017.

[16] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

## NeurIPS Paper Checklist

1. **Claims**

   Answer: [Yes]

   Justification: The abstract and introduction state our two contributions (SLS and the real-time V-M-C pipeline) and scope them to a single game environment.

2. **Limitations**

   Answer: [Yes]

   Justification: Section 6 discusses single-environment evaluation, binary action space, and calibration transferability.

3. **Theory assumptions and proofs**

   Answer: [N/A]

   Justification: The paper does not include theoretical results. SLS is an empirical method.

4. **Experimental result reproducibility**

   Answer: [Yes]

   Justification: Section 4 provides full training details. Code will be released.

5. **Open access to data and code**

   Answer: [Yes]

   Justification: Code and dataset are publicly available at `https://github.com/Tariolle/sls-wm`. Data collection procedure is described in Section 4.

6. **Experimental setting/details**

   Answer: [Yes]

   Justification: All hyperparameters, optimizers, schedules, and data augmentation are specified in Section 4.

7. **Experiment statistical significance**

   Answer: **[TODO]**

   Justification: **[TODO]**

8. **Experiments compute resources**

   Answer: [Yes]

   Justification: All experiments run on a single NVIDIA A100 GPU. Training times are reported in Section 4.

9. **Code of ethics**

   Answer: [Yes]

   Justification: The research involves no human subjects, sensitive data, or dual-use concerns.

10. **Broader impacts**

    Answer: [N/A]

    Justification: This work applies to a single-player video game with no direct societal impact.

11. **Safeguards**

    Answer: [N/A]

    Justification: The released model is a small world model for a video game and poses no misuse risk.

12. **Licenses for existing assets**

    Answer: [Yes]

    Justification: All referenced works are properly cited. Geometry Dash is a commercial game by RobTop Games; gameplay data was collected from a legally purchased copy.

13. **New assets**

Answer: [Yes]

Justification: Code and model checkpoints will be released under an open-source license with documentation.

14. **Crowdsourcing and research with human subjects**

    Answer: [N/A]

    Justification: No crowdsourcing or human subjects research was conducted.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Answer: [N/A]

    Justification: No human subjects research was conducted.

16. **Declaration of LLM usage**

    Answer: [N/A]

    Justification: LLMs were not used as a component of the core methodology.