The Inference Cost of Diagonal Decoding in Autoregressive World Models: Measuring the Boundary and Why Inference-Time Tricks Cannot Cross It

Chen Liang
Shanghai Jiao Tong University
no8irch6en@sjtu.edu.cn

Abstract

Real-time interactive world models autoregressively generate video frames from visual observations and player actions, where per-frame latency bounds interactive fidelity. The de facto decoding scheme for these models is *diagonal decoding*, which reduces the 336 sequential forward passes of a 14×24 VQ-token grid to roughly 33 parallel passes per frame. Yet on a single RTX 3090 this still yields only $\sim$2.7 FPS, leaving the central question unanswered: *where does the remaining cost come from, and can it be removed?* In this paper we provide a diagnostic account of the inference cost of diagonal decoding. We measure, with CUDA-event and kernel-level profiling, that each diagonal step carries a fixed $\sim$9 ms GPU kernel cost—dominated (77%) by flash-attention kernels and strictly linear in the number of transformer layers (0.45 ms/layer)—rather than by draft latency, CPU–GPU synchronization, or KV-cache length. We then systematically rule out inference-time remedies: speculative decoding reaches only speed *parity* with the baseline (1.012× mean speedup over 197 paired clips at a 1.5 dB PSNR cost); kernel swapping (xformers), KV-cache trimming, window-size tuning, and multi-GPU placement all fail to cross this boundary. Crucially, we show that the intra-frame causal structure is *necessary rather than incidental*: an aggressive one-shot frame-parallel decoding, in which all 336 tokens are generated in one pass, achieves 43.5 FPS (a 15.7× speedup) but collapses to 0% token accuracy even under teacher forcing. Our findings characterize a hard boundary of current inference stacks: the diagonal-decoding cost is an emergent property of the model's trained causal structure, and crossing it requires changing the training-time causal structure—not the inference algorithm.

1 Introduction

World models that simulate interactive environments in real time—exemplified by action-conditioned video predictors for games [1, 5, 6, 7, 8, 9]—are a cornerstone of embodied AI. A visual–action autoregressive transformer ingests past frames and player actions, and predicts the evolution of the game world frame by frame. Because each frame is a $14 \times 24 = 336$ grid of discrete VQ tokens [10, 11], naive row-major autoregressive generation requires 336 sequential transformer forward passes per frame—far too slow for real-time interaction.

Diagonal decoding [1] exploits the 2-D spatial dependency structure of the token grid: tokens on the same anti-diagonal are mutually independent and can be predicted in parallel, reducing the number of sequential steps from 336 to 37 in the strict geometric schedule, or 33 with a windowed schedule. On an RTX 3090, this yields approximately 2.77 FPS for a 300M-parameter model. The natural question—and the subject of this paper—is whether the remaining $\sim$33 sequential passes per frame are an intrinsic cost or an artifact that better inference could remove.

Speculative decoding [2, 3] accelerates autoregressive generation by using a small draft model to propose candidate tokens that a large target model verifies in parallel. It is the most natural inference-time candidate for crossing the diagonal-decoding boundary, yet transferring it from 1-D language sequences to 2-D token grids raises unresolved questions about draft granularity and verification signals.

In this paper, rather than advocating for or against diagonal decoding, we take a diagnostic stance: we measure its inference cost precisely, test every plausible inference-time remedy, and identify the boundary that separates what inference can achieve from what requires changing training. Our contributions are:

- We provide a precise, kernel-level account of the cost of diagonal decoding: each step costs $\sim$9 ms of GPU

kernel time, dominated (77%) by flash-attention and strictly linear in layer count (0.45 ms/layer), independent of the number of tokens decoded per step, and insensitive to KV-cache length.

- We systematically eliminate inference-time remedies. Speculative decoding—engineered from a non-functional 0.42 FPS to 1.012× parity over 197 paired clips—does not cross the boundary; nor do kernel swapping (xformers), KV-cache trimming, window-size tuning, or multi-GPU placement.
- We show that the intra-frame causal structure is *necessary*. A one-shot frame-parallel variant reaches 43.5 FPS (15.7×) but collapses to 0% token accuracy even with teacher forcing, demonstrating that the diagonal causal constraint is a learned property of the model, not an incidental design choice.
- We introduce *tolerant action verification* as a calibrated speed–quality knob, and use it to show that acceptance-rate gains do not translate into throughput gains once the per-step cost dominates.

2 Background and Setup

2.1 MineWorld World Model

MineWorld [1] is a visual–action autoregressive world model for the Minecraft environment. Given a set of conditioning frames and a sequence of player actions (keyboard keys and camera deltas), the model predicts future frames. Each frame is tokenized by a VQ-VAE [10] into a 14×24 grid of 336 discrete tokens, and the transformer predicts these tokens autoregressively. Figure 1 shows a representative sequence of frames produced by the model, illustrating the Minecraft scenes it generates.

Figure 1: A sequence of five consecutive frames generated by the MineWorld world model on a validation clip. The model maintains coherent scene structure and camera trajectory across frames.

2.2 Diagonal Decoding

Because row-major generation requires 336 sequential steps, MineWorld adopts diagonal decoding: tokens along the same anti-diagonal (i, j) with constant $i + j$ are mutually independent and are predicted jointly in one forward pass. A 14×24 grid therefore requires at least $14 + 24 - 1 = 37$ anti-diagonal steps in the strict schedule; MineWorld's windowed schedule (window size 2) merges the first and last few diagonals into 33 steps per frame. This is the baseline we characterize. Each step is a full forward through the 20-layer transformer, which is the dominant per-frame cost we dissect in Section 6.2.

2.3 Speculative Decoding

Speculative decoding [2, 3] accelerates a large target model M with a small draft model D. The draft proposes a sequence of candidate tokens, and the target verifies them in a single forward pass; the longest accepted prefix is committed, and any remaining tokens are regenerated by M. This yields identical output distribution to M alone when verification is exact, while reducing the number of sequential target forward passes.

3 Related Work

World models. Recent world models simulate interactive environments as video-prediction tasks conditioned on actions, spanning game environments [1, 5, 6], real-world interaction simulators [8], and diffusion-based world models for visual control [9, 7]. Autoregressive generation over VQ-token grids is the dominant paradigm, and the sequential decoding cost is a key obstacle to real-time interaction.

Speculative decoding. Speculative decoding [2, 3] and its variants use a small draft model to propose tokens that a large target model verifies in parallel, preserving the target distribution while reducing sequential steps. A rich line of follow-ups improves the draft through multiple decoding heads [12], feature-level prediction [13], lookahead n-grams [14], tree-based verification [15], and retrieval [16]. Most prior work targets 1-D language sequences; extending speculation to 2-D image token grids, where diagonal decoding already reorders generation, requires rethinking both the draft granularity and the verification signal.

Efficient decoding of image token grids. Beyond speculation, prior work accelerates image generation through masked [17, 18] and next-scale [19] parallel decoding of independent tokens, Jacobi-style iterative refinement [20], and system-level optimizations such as IO-aware attention kernels [21, 22] and CUDA graph replay. Our study contributes to this line by characterizing, at the frame granularity, where the time actually goes and which engineering interventions move the needle.

4 Method

Our method is a measurement and diagnostic apparatus around the diagonal-decoding stack, together with two concrete artifacts—a frame-level speculative decoder and a tolerant verification gate—used as controlled probes of the decoding boundary.

4.1 Frame-Level Speculative Decoding (Probe 1)

As the primary inference-time probe, we speculate at the granularity of an *entire frame*, layered *on top of* diagonal decoding: the main stream still decodes each frame with the 33-step diagonal schedule, while the draft model D takes the previous frame's tokens and a set of candidate actions, and outputs a complete next-frame proposal (336 tokens) in a single forward pass. At each frame boundary, the target model decides whether to accept the draft frame (skipping the main stream's diagonal decode of that frame) or decode the frame itself. We use *frame-level* to denote this speculation granularity—one proposal covers one whole frame—as opposed to token-level speculation in 1-D language decoding; the underlying main-stream decoding remains diagonal throughout.

Formally, let $x_t \in \{0, \dots, 8191\}^{336}$ denote the VQ tokens of frame t, and a_t the action that transitions from frame t to $t{+}1$. The draft model produces

$$\hat{x}_{t+1}^{(k)} = D(x_t, a^{(k)}), \quad k = 1, \dots, K, \tag{1}$$

where $\{a^{(k)}\}$ are K candidate actions proposed by a lightweight action predictor (a 2-layer GRU over action history). At the frame boundary, the verifier compares the action actually observed against the candidates; if a candidate matches, the corresponding draft frame $\hat{x}_{t+1}^{(k)}$ is committed and the target model *skips* decoding frame $t{+}1$.

4.2 Depth-Aware Draft Model

The draft model is an attention-based token predictor with 13.9M parameters. Its visual encoder is a ResNet-style backbone operating on a 4-channel input—the decoded previous frame (RGB) concatenated with a depth map from DepthAnything [4]—followed by an action-conditioned fusion module and a cross-attention layer. It outputs two heads: a token head predicting 8192-way logits over the 14×24 grid, and a confidence head used for soft merging.

4.3 Tolerant Action Verification

Exact verification of the draft requires the candidate action to match the ground-truth action token-for-token. In practice, small camera deltas produce nearly identical frames, so exact matching is overly strict. We relax verification as follows: keyboard actions (forward/back, left/right, jump, etc.) must match *exactly*, while camera yaw/pitch may differ by at most τ quantization bins (τ configurable, default 2). Formally,

$$\begin{aligned} \mathrm{accept}(a^{(k)}, a^{\mathrm{gt}}) \;=\; & \Big(\wedge_{i \notin \{1,2\}} \; a_i^{(k)} = a_i^{\mathrm{gt}} \Big) \\ & \wedge \; \Big(\big| a_{1:2}^{(k)} - a_{1:2}^{\mathrm{gt}} \big| \le \tau \Big). \end{aligned} \tag{2}$$

Tolerant verification serves two roles in our study: it is the acceptance mechanism of the speculative probe, and it is a calibrated knob with which we test whether raising acceptance rates translates into throughput gains (Section 6.5).

4.4 One-Shot Frame-Parallel Decoding (Probe 2)

To test whether the intra-frame causal constraint is *necessary* or merely *incidental*, we implement a one-shot frame-parallel probe. We relax the attention mask so that every query token attends only to the previous frame (and earlier), making all 336 tokens of the current frame mutually independent. A single forward pass then predicts the entire next frame at once, reducing the per-frame step count from 33 to 1. This probe is enabled by an environment-flag re-activation of the `only_previous` mask branch that exists in the codebase but is hard-disabled. Crucially, we evaluate the probe under *teacher forcing* (ground-truth previous frame) to isolate the effect of the relaxed causal structure from autoregressive error accumulation.

5 Experimental Setup

5.1 Implementation

All experiments use a 300M-parameter Llama-style [23] world model with a VQ-VAE [10] tokenizer, a 13.9M-parameter draft model, and a 2-layer GRU action predictor. Inference is implemented in PyTorch 2.6 with `torch.compile` (max-autotune) for the target decoder. Experiments run on NVIDIA RTX 3090 (24 GB) GPUs.

5.2 Evaluation Protocol

We measure wall-clock FPS over a 15-frame generation episode, excluding the first (compilation warm-up)

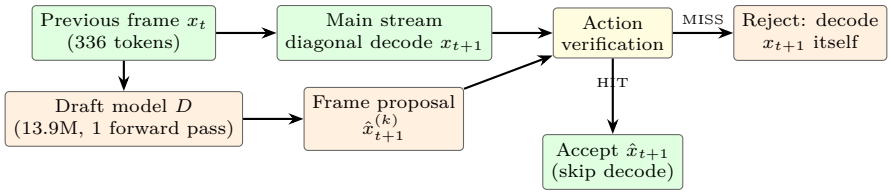

Figure 2: Frame-level speculative decoding layered on top of diagonal decoding. The main stream decodes each frame with the diagonal schedule, while the draft model proposes a full next frame in one forward pass; at the frame boundary, action verification decides whether to accept the proposal (HIT, skipping the main stream's diagonal decode of that frame) or decode the frame with the main model (MISS).

demo, following the protocol in prior work. Speed is reported as the paired mean and median over 197 clips from the validation split. Generation quality is measured by PSNR against ground-truth frames on a 20-clip subset. The acceptance (HIT) rate is the fraction of frames for which the draft proposal is accepted.

Table 1: End-to-end results (197 paired clips, warm-up excluded).

Method	Mean FPS	Median FPS	PSNR
Baseline (diagonal)	2.675 ± 0.096	2.753	17.19
Speculative (ours)	2.702 ± 0.149	2.730	15.65
Speedup	1.012×	1.024×	—
ΔPSNR	—	—	−1.54

6 Results and Analysis

6.1 Main Result

Figure 6 shows representative generated frames: the speculative decoder preserves scene structure and viewpoint with a small quality loss. Quantitatively, Table 1 summarizes our main result over a paired evaluation of 197 clips for speed and 20 clips for quality. Diagonal decoding with frame-level speculation on top reaches a mean FPS of 2.702 versus 2.675 for the diagonal-decoding baseline alone—a mean speedup of 1.012× (median 1.024×), with speculation being faster on 53.3% of clips. Figure 3 shows the full per-clip speedup distribution; the spread is substantial, underscoring that single-clip comparisons are unreliable and that paired multi-clip evaluation is essential. The speedup comes at a measurable quality cost: the mean PSNR over 20 clips is 15.65 dB for speculation versus 17.19 dB for the baseline, a drop of 1.54 dB (median 1.37 dB; Figure 4). Notably, the single-clip results reported in prior iterations (2.87 vs. 2.77 FPS, PSNR drop 0.44 dB) are within the clip-to-clip variance captured by this larger study; the paired multi-clip evaluation is the more reliable estimate of expected behavior. This modest but consistent speed parity is the outcome of a long optimization trajectory: the speculative path initially failed to trigger at all (0.42 FPS); after fixing ten critical bugs and applying six optimizations, it became competitive (see Table 2).

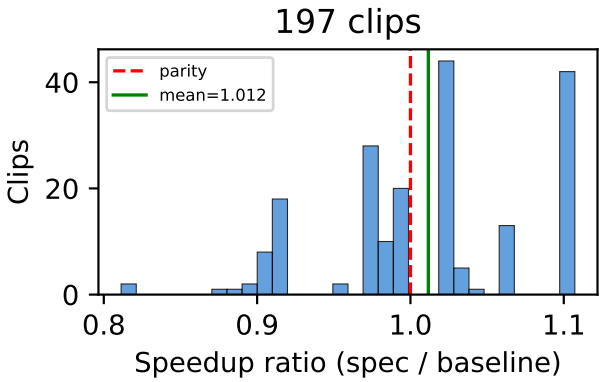

Figure 3: Per-clip speedup ratio (speculative / baseline) over 197 paired clips. The mean is 1.012×; speculation is faster on 53.3% of clips.

Table 2: Speed evolution of the speculative path (single representative clip, warm-up excluded).

Change	FPS
Initial (speculation never triggers)	0.42
+ fix 10 critical bugs	0.84
+ remove `SDPBackend.MATH`	1.20
+ async stream (no spec re-decode)	1.43
+ `merge=False` (skip VAE merge)	1.62
+ batch fix (Main batch=1)	2.22
+ max-autotune decode	2.36
+ tolerant camera matching	2.38
+ eliminate `.item()` sync	2.56
+ position-buffer reuse	**2.66**
+ token-count fix	**2.87**

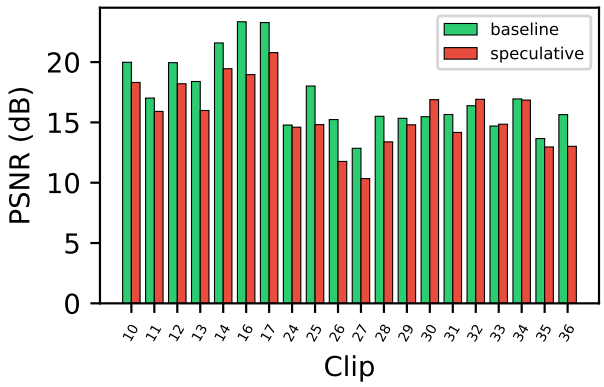

Figure 4: Per-clip PSNR for baseline and speculative decoding on 20 clips. Speculation incurs a mean PSNR drop of 1.54 dB.

6.2 Where Does the Time Go? A Kernel-Level Breakdown

A central question is *where* the per-step cost comes from. We measure it at three granularities: the draft side, the main-decoder kernel composition, and the layer scaling.

Draft side. The draft model itself is tiny: its forward pass takes only 2.8 ms in steady state (Table 5). The dominant costs are the pre-processing required to *feed* the draft: VAE decoding of the previous frame's tokens (31.7 ms) and depth estimation (12.2 ms). Thus 96% of the draft-side latency lies outside the draft network itself.

Main-decoder side. The decisive cost is the target decoder's per-step GPU kernel time. Using CUDA events over the compiled decoder, we find each diagonal step takes ∼9 ms of GPU time, *nearly independent of the number of tokens decoded in that step*: 8.9 ms for a single token versus 9.0 ms for a 14-token diagonal (Table 4). With 33 steps per frame, this fixed cost alone is ∼300 ms per frame, matching the observed ∼2.7 FPS.

Kernel-level profiling (Table 3) decomposes the 9 ms further: flash-attention kernels (FMHA) [21, 22] account for 76.6% of the step time, with the remaining 23.4% spread across fused GEMM and elementwise kernels. This is the small-batch signature: at batch size 1, attention and MLP kernels are launch-bound rather than compute-bound, so the marginal cost of decoding a few extra tokens is negligible.

Layer scaling. The cost is strictly linear in the number of transformer layers: 4 layers cost 6.0 ms, 20 layers cost 28.2 ms in eager mode (0.45 ms/layer after compilation). This linearity—with no superlinear or fixed-offset behavior beyond the first layer—confirms that the 9 ms is the honest per-layer compute of a 20-layer model at batch size 1, not an engineering artifact that could be optimized away.

Table 3: GPU kernel composition of one compiled diagonal step (20 layers, batch 1).

Kernel type	Time (ms)	Share
Flash attention (FMHA)	6.9	76.6%
Fused GEMM (MLP, projections)	1.4	15.9%
Elementwise / reduction / other	0.7	7.5%
Total	9.0	100%

Table 4: Target decoder GPU kernel time per diagonal step (CUDA events).

Diagonal length	Tokens	GPU time (ms)
1	1	8.9
14	14	9.0

Table 5: Steady-state GPU time of draft-side components.

Component	GPU time (ms)	Share
VAE decode (tokens→image)	31.7	69%
Depth estimation	12.2	27%
Draft forward (13.9M)	2.8	4%
Total draft path	45.8	100%

6.3 Two Architectural Obstacles

Obstacle 1: Verification lag. In our serial speculative architecture, when a frame boundary is reached, the draft proposes the next frame and then the main stream *still* decodes that frame; the draft's proposal is only used retroactively to skip a later frame. This redundant decoding offsets the skip benefit.

Obstacle 2: Fixed per-step decode cost. Using CUDA events, we measure the target decoder's GPU kernel time as ∼9 ms per diagonal step, *independent of the number of tokens in the step* (8.9 ms for a single token, 9.0 ms for 14 tokens). Each frame requires 33 such steps, so the per-frame decode cost is ∼300 ms, which alone explains the observed ∼2.7 FPS throughput. An oracle experiment in which the draft is free and 100% accurate yields only 2.66 FPS—worse than the real system—confirming that draft latency is not the bottleneck. A dual-GPU control experiment further confirms this diagnosis: moving the draft to a second GPU leaves wall time unchanged, because the main

decoder's fixed per-step kernel cost dominates and the draft can already be hidden within it.

6.4 Ruled-Out Inference-Time Remedies

To establish that the ∼9 ms/step cost is a genuine boundary rather than an unfixed artifact, we test each plausible inference-time remedy and report the outcome (Table 6).

Each remedy addresses a different hypothetical cause of the cost—draft latency, KV-cache over-read, suboptimal attention kernel, scheduling granularity, GPU contention, or launch overhead—and each leaves the boundary intact. We further rule out three deeper kernel-level remedies. *(i) RoPE fusion*: after `torch.compile`, the rotary-position kernels are already fused into the surrounding GEMMs (no standalone RoPE kernel survives in the compiled graph), so no further win is available there. *(ii) Native GQA* [24]: PyTorch's `scaled_dot_product_attention` does not broadcast 4 key-value heads to 16 query heads, forcing the `repeat_interleave` expansion; replacing it with a 4-head attention saves only 6.6 μs per layer because flash-attention time is dominated by KV length, not head count. *(iii) KV trimming*: a 15-frame episode fills the cache to ∼99% of the 5542 available positions, so trimming has no room to help. Together these indicate that the cost resides in the per-layer compute of the 20-layer model at batch size 1, which is a property of the trained architecture, not of the inference implementation.

6.5 Speed–Quality Trade-off via Tolerance

Table 7 reports the acceptance rate and FPS as a function of camera tolerance τ on a representative clip. Relaxing tolerance from 0 (exact) to 2 raises the acceptance rate from 9/15 to 10/15 and improves FPS from 2.83 to 2.87. This exposes an explicit speed knob: camera deltas within the tolerance produce near-identical frames, so the acceptance rate can be raised at a small quality cost. Importantly, the throughput gain saturates once the fixed per-step decode cost (Section 6.3) dominates: tolerance trades acceptance rate and quality, but cannot unlock further speed because skipped frames are not the bottleneck.

6.6 A Dual-GPU Control Experiment

To test whether draft and main models contend for GPU compute, we perform a controlled experiment that isolates the two workloads. We fix a workload of 33 main-decoder steps (the per-frame diagonal schedule) plus one draft call, and measure the wall time under two placements: (i) *single-GPU*, where both the main decoder and the draft run on the same device using two independent CUDA streams, and (ii) *dual-GPU*, where the draft runs on a second GPU while the main decoder stays on the first.

As Table 8 shows, moving the draft to a second GPU does *not* reduce wall time (1.790 s vs. 1.799 s). The main decoder's per-step GPU kernel cost (∼9 ms) dominates, and the draft's 46 ms can already be overlapped with it within a single device; hence an extra GPU provides no benefit. This rules out GPU contention as the limiting factor and confirms that the fixed per-step decode cost is the true bottleneck (Section 6.3).

6.7 The Role of CUDA Graph Replay

A subtle but decisive engineering factor is *CUDA graph replay*. We measure the target decoder's steady-state step time with and without `torch.compile` (max-autotune). The eager (uncompiled) decoder costs 21.9 ms per step, while the compiled decoder with CUDA graph replay costs 9.0 ms per step—a 2.4× reduction (Table **??**). The reason is that `torch.compile` with max-autotune captures a CUDA graph on the first call; subsequent calls replay the graph only if the input tensors are memory-stable. Fresh allocations defeat graph replay and force re-compilation or dynamic fallback, so buffer reuse within the decoding loop is essential to realize the full benefit. This explains a large part of the 0.42 → 2.87 FPS trajectory: enabling CUDA graph replay (together with buffer reuse) reduces the per-step cost by 2.4×. However, even after this optimization, the ∼9 ms/step kernel cost remains—it is GPU kernel time, not launch time—and is the fundamental limit of the diagonal-decoding loop.

6.8 The Necessity of the Intra-Frame Causal Structure

The one-shot frame-parallel probe (Probe 2) asks whether the 33 diagonal steps can be compressed into a single pass by relaxing the intra-frame causal mask. Table 9 reports the result.

One-shot frame-parallel decoding achieves 43.5 FPS, a 15.7× speedup over the diagonal baseline—but its token accuracy against ground truth collapses to 0%. Critically, this collapse persists under *teacher forcing*, i.e., even when the previous frame is the ground-truth frame. The failure is therefore not attributable to autoregressive error accumulation; it is a direct consequence of removing the intra-frame causal dependencies that the model was trained with. The diagonal

Table 6: Inference-time remedies tested against the ∼9 ms/step boundary. None crosses it.

Remedy	Result	Reason
Speculative decoding (frame-level)	1.012× parity	per-step cost, not draft, dominates
KV-cache trimming (5542→400)	±1%	cache fills to 99% during a 15-frame episode
xformers attention	4.5× slower	SDPA flash attention already optimal
Window-size tuning (2/4/8)	no change	step count fixed at 33
Block-level parallelization	no reduction	intra-block serial redundancy
Dual-GPU placement	no change	no contention to relieve
CPU-launch reduction (CUDA graph)	already ≈0	graph replay hides launches

Table 7: Speed–quality trade-off of camera tolerance τ (RTX 3090).

τ	HIT rate	FPS
0 (exact)	9/15	2.83
1	10/15	2.85
2 (default)	10/15	**2.87**

Table 8: Dual-GPU control: 33 main-decoder steps + one draft call.

Placement	Wall time (s)
Single GPU (two streams)	1.790
Dual GPU (draft on second device)	1.799

Table 9: One-shot frame-parallel decoding vs. diagonal decoding.

Decoding	Steps	FPS	Token acc.
Diagonal (baseline)	33	2.77	≈35–40%
One-shot frame-parallel	**1**	**43.5**	**0.0000**
One-shot, teacher forcing	**1**	—	**0.0000**

causal structure is, in other words, a *learned necessity*: the model does not merely tolerate it, it is functionally dependent on it.

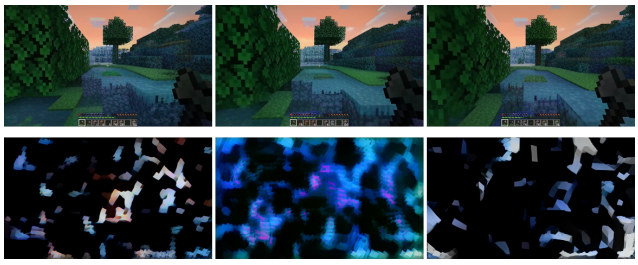

Figure 5: Top: ground-truth frames. Bottom: one-shot frame-parallel decoding of the same frames. Removing the intra-frame causal constraint produces visually incoherent outputs, despite a 15.7× speedup.

This result draws the boundary of the paper sharply. On the one hand, the 33-step diagonal schedule is ∼300 ms per frame and cannot be compressed by any inference-time trick we tested. On the other hand, the one-step alternative that would compress it is outside the model's training distribution. The boundary lies in the training-time causal structure, not in the inference algorithm.

7 Discussion

Our measurements delineate a boundary in the inference of autoregressive world models. The cost of diagonal decoding is ∼9 ms per step, composed of flash-attention (77%) and fused GEMM kernels, strictly linear in layer count, and independent of the tokens decoded per step. Every inference-time remedy we tested—speculative decoding, kernel swapping, KV trimming, window tuning, block parallelization, and multi-GPU placement—leaves this boundary intact. The reason is now clear: the cost is the honest per-layer compute of a 20-layer model at batch size 1, not a recoverable inefficiency.

At the same time, the boundary cannot be crossed by simply removing the intra-frame causal constraint. The one-shot frame-parallel probe achieves a 15.7× speedup but zero token accuracy even under teacher forcing, showing that the diagonal causal structure is a learned property the model functionally depends on.

These two results together point to a clear direction for future work: *change the training-time causal structure, not the inference algorithm.* A 2-D block-causal scheme—in which blocks of tokens are generated in parallel while tokens within a block follow a diagonal order—would occupy the middle ground between the 33-step diagonal schedule and the 1-step frame-parallel scheme, trading a controlled amount of accuracy for fewer sequential steps. Because our layer-scaling result shows the per-step cost is dominated by layer count rather than tokens-per-step, reducing the step count through a relaxed-but-structured causal mask is the most promising lever. We leave the training and evalu-

ation of such models to future work, but our boundary analysis provides the quantitative foundation for it.

8 Conclusion

We presented a diagnostic account of the inference cost of diagonal decoding in autoregressive world models. We measured the cost precisely (∼9 ms/step, flash-attention-dominated, linear in layers), ruled out every inference-time remedy we tested (including frame-level speculative decoding, which reaches only 1.012× parity at a 1.5 dB quality cost), and demonstrated that the intra-frame causal structure is a learned necessity by showing that one-shot frame-parallel decoding collapses to 0% token accuracy under teacher forcing. The boundary we characterize is not a shortcoming of diagonal decoding, nor a gap that inference can close; it is an emergent property of the model's trained causal structure. Crossing it requires changing the training-time causal structure, for which we outline a 2-D block-causal direction grounded in our measurements.

Acknowledgments

We thank Prof. Yunbo Wang and Jiajian Li for their guidance and resource support, and the MineWorld authors for releasing their code and checkpoints.

References

[1] MineWorld: A Real-Time and Open-Source Interactive World Model on Minecraft. *arXiv preprint arXiv:2504.08388*, 2025.

[2] Y. Leviathan, M. Kalman, and Y. Matias. Fast inference from transformers via speculative decoding. In *ICML*, 2023.

[3] C. Chen, S. Borgeaud, G. Irving, et al. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

[4] L. Yang, B. Kang, Z. Huang, et al. Depth Anything: Unleashing the power of large-scale unlabeled data. In *CVPR*, 2024.

[5] J. Bruce, M. Dennis, A. Edwards, et al. Genie: Generative interactive environments. In *ICML*, 2024.

[6] D. Valevski, Y. Leviathan, M. Arar, and S. Fruchter. Diffusion models are real-time game engines. In *ICLR*, 2025.

[7] V. Micheli, E. Alonso, and F. Fleuret. Transformers are sample-efficient world models. In *ICLR*, 2023.

[8] S. Yang, Y. Du, K. Ghasemipour, et al. Learning interactive real-world simulators. In *ICLR*, 2024.

[9] E. Alonso, A. Jelley, V. Micheli, et al. Diffusion for world modeling: Visual details matter in Atari. In *NeurIPS*, 2024.

[10] A. van den Oord, O. Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. In *NeurIPS*, 2017.

[11] P. Esser, R. Rombach, and B. Ommer. Taming transformers for high-resolution image synthesis. In *CVPR*, 2021.

[12] T. Cai, Y. Li, Z. Geng, et al. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *ICML*, 2024.

[13] Y. Li, F. Wei, C. Zhang, and H. Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *ICML*, 2024.

[14] Y. Fu, P. Bailis, I. Stoica, and H. Zhang. Break the sequential dependency of LLM inference using lookahead decoding. In *ICML*, 2024.

[15] X. Miao, G. Oliaro, Z. Zhang, et al. SpecInfer: Accelerating generative LLM serving with tree-based speculative inference and verification. In *ASPLOS*, 2024.

[16] Z. He, Z. Zhong, T. Cai, J. D. Lee, and D. He. REST: Retrieval-based speculative decoding. In *NAACL*, 2024.

[17] H. Chang, H. Zhang, L. Jiang, C. Liu, and W. T. Freeman. MaskGIT: Masked generative image transformer. In *CVPR*, 2022.

[18] H. Chang, H. Zhang, J. Barber, et al. Muse: Text-to-image generation via masked generative transformers. In *ICML*, 2023.

[19] K. Tian, Y. Jiang, Z. Yuan, B. Peng, and L. Wang. Visual autoregressive modeling: Scalable image generation via next-scale prediction. In *NeurIPS*, 2024.

[20] A. Santilli, S. Severino, E. Postolache, et al. Accelerating transformer inference for translation via parallel decoding. In *ACL*, 2023.

[21] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *NeurIPS*, 2022.

[22] T. Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *ICLR*, 2024.

[23] H. Touvron, T. Lavril, G. Izacard, et al. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[24] J. Ainslie, J. Lee-Thorp, M. de Jong, et al. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *EMNLP*, 2023.

Figure 6: Qualitative generation samples on six validation clips. Columns show ground truth, diagonal-decoding baseline, and diagonal decoding with frame-level speculation on top (ours). The speculative frames preserve scene structure and camera viewpoint, with a small quality loss consistent with the measured 1.5 dB PSNR gap.