

WAMJET: A Harness for World Action Model Acceleration

Le Chen¹, Lixin Liu^{2,†}, Jan Schneider¹, Zeru Qiu¹, Simon Guist¹, Bernhard Schölkopf¹, Dieter Buehler^{1,3}

Abstract—World Action Models (WAMs) leverage pretrained video foundation models for robot manipulation, but their large backbones and video-action co-prediction are expensive. Although existing acceleration techniques offer many ways to reduce this cost, selecting and composing them requires substantial engineering for each model and hardware platform. To tackle this bottleneck, we present WAMJET, an agentic harness that accelerates WAM inference by equipping coding agents with reusable optimization guidance and measurement and validation tools. WAMJET follows a bottleneck-driven workflow where the agent profiles inference, modifies targeted code, validates effects, and iteratively refines the acceleration stack as bottlenecks shift, while preserving action quality. Experiments span six WAMs, three coding agents, and two GPU architectures. WAMJET achieves up to 9.95× lossless speedup over upstream implementations. Approximation and hardware-aware optimization yield additional latency reductions, with comparable success rates. The results show that WAMJET can produce effective acceleration stacks for WAM deployment.

I. INTRODUCTION

Building generalist policies requires not only mapping visual observations to actions but also understanding and predicting the consequences of actions in diverse environments. Video generation models learn representations of such dynamics from large-scale video datasets. Many of these videos contain intentional behavior like object manipulation and scene changes because someone or something acted. That makes video models attractive backbones for generalist manipulation. Before seeing any robot actions, the backbone already encodes useful links between language, visual change, and plausible object interactions. World Action Models (WAMs) build on these models and extend them with action-generation capabilities. They inherit priors from video pretraining and couple action generation with video-based world modeling for more robust robot manipulation across diverse environments. Compared with Vision-Language-Action Models (VLAs), WAMs are appealing because predicting the future states of the environment may help capture better physical dynamics and temporal information.

Modern video generation models are often billion-scale parameter models and continue to scale toward higher generation quality and higher fidelity. This scaling provides stronger priors about how visual scenes evolve but also increases inference cost. The same backbones that make

WAMs attractive also make them expensive to run. Inference may involve sampling a large video model, several denoising steps, and repeated vision-language processing. Beyond those computations, data movement, synchronization, and cache updates can also add substantial latency. This computational burden poses a significant challenge for real-time control in robotics, where timely decision-making is crucial.

Many techniques can accelerate inference, including compilation, kernel fusion, caching, and quantization. The difficulty is deciding which techniques to apply and how to combine them. A model dominated by repeated conditioning needs different optimizations from one governed by attention or memory transfers. Hardware also affects which kernels or data types are efficient. Moreover, optimizations can interact, which means removing one bottleneck can expose another, and a change that initially gives little benefit may become useful later. Developing an effective acceleration strategy, therefore, requires repeated profiling, implementation, and measurement for each deployment. Preserving the policy’s behavior adds another challenge. Reusing a prompt embedding while the instruction is unchanged can avoid redundant computation. Reusing visual features after the observation changes, however, approximates the policy’s input. Quantization can introduce further error, whose effect on manipulation may be difficult to predict from tensor differences alone. An optimization process must account for both inference speed and the effect of its changes on robot actions.

Accelerating WAM inference is not a one-time optimization but requires a recipe that composes techniques for a specific model, hardware stack, and task. The best recipe for a certain WAM may not directly transfer to other models. As illustrated in Fig. 1, hand-crafting an acceleration strategy that works well on a specific model and hardware setup requires substantial engineering effort and may not generalize to other scenarios. LLM coding agents offer a way to automate this process, but reliable optimization over many iterations requires clear constraints, appropriate tools, and feedback on both latency and correctness. Harness engineering provides this structure through agent-readable workflows, measurement tools, and explicit validation criteria [1]. This shifts human effort toward designing reusable guidance, while the agent conducts experiments, assesses their outcomes, and uses the evidence to guide subsequent optimizations.

In this work, we present WAMJET, an agentic harness for accelerating WAM inference. WAMJET equips coding agents with optimization guidance and tools for profiling, latency measurement, numerical comparison, and paired task analysis. Starting from a runnable policy, the agent identifies bottlenecks, implements lossless or approximate accelerations,

¹Le Chen, Jan Schneider, Zeru Qiu, Simon Guist, Bernhard Schölkopf, and Dieter Buehler are with Max Planck Institute for Intelligent Systems, Tübingen, Germany.

²Lixin Liu is with The Chinese University of Hong Kong, Hong Kong SAR, China.

³Dieter Buehler is also with Johannes Kepler University Linz, Austria.

[†]Corresponding Author: Lixin Liu (lxliu@cse.cuhk.edu.hk).

The code is available at <https://github.com/liulixinkerry/WAMJET>.

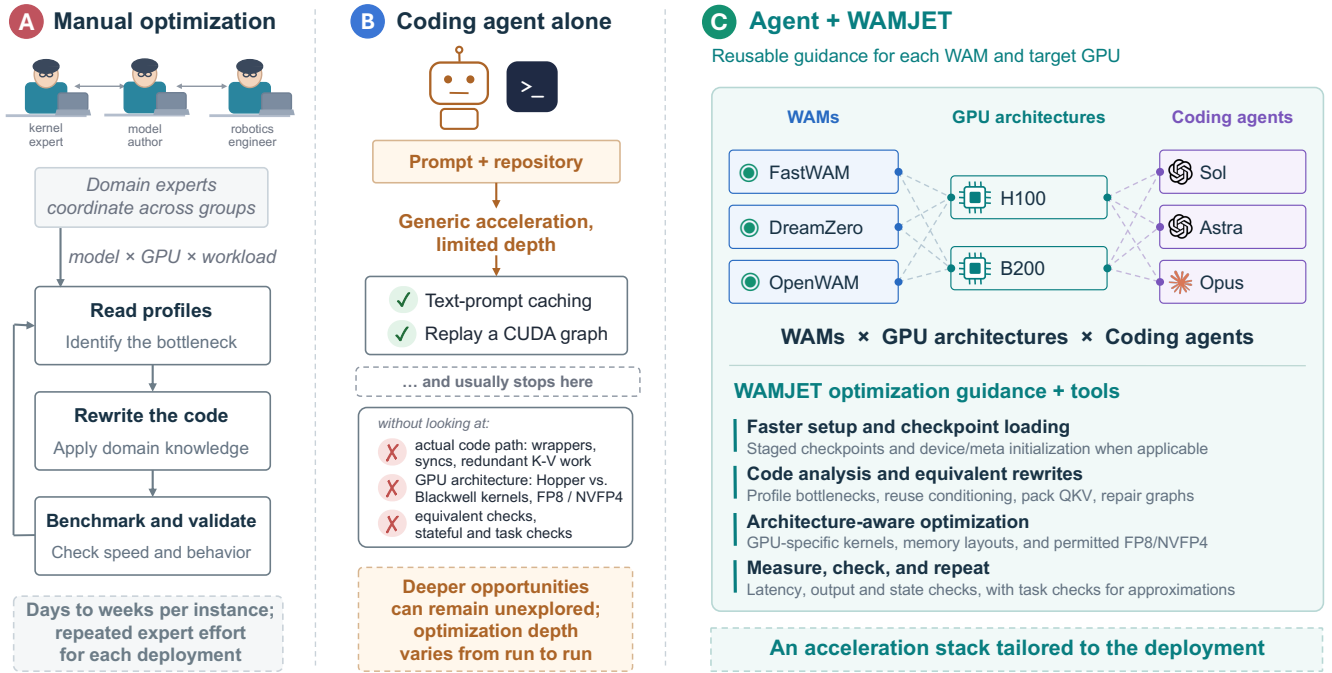


Fig. 1: Overview of WAMJET. (A) Manual optimization requires diverse domain expertise, with substantial engineering for each deployment. (B) A coding agent without guidance may only apply generic optimizations while leaving deeper optimization opportunities unexplored. (C) WAMJET equips the agent with reusable guidance and tools for faster startup, bottleneck analysis, hardware-aware optimization, and iterative validation, supporting acceleration strategies tailored to different WAMs and GPU architectures.

and benchmarks and validates each candidate. It retains useful changes and repeats the process within a specified budget. Throughout the search, the agent adapts its optimization choices to the target hardware, records experiment outcomes, and stores reusable findings in external memory. This iterative process produces acceleration stacks tailored to the target policy and hardware.

Our contributions can be summarized in several aspects:

- We introduce WAMJET, a harness that combines reusable optimization guidance with measurement and validation tools for accelerating WAM inference.
- We develop a bottleneck-driven workflow that iteratively composes acceleration techniques and checks their effects on outputs, recurrent state, and task performance.
- We evaluate WAMJET across different WAMs and GPU architectures, examining the optimizations found by different agent campaigns, the benefit of further search, and the gains and limitations of approximate inference. Extensive experiments demonstrate that our training-free framework can achieve significant acceleration.

II. RELATED WORK

A. Foundation Models for Robot Manipulation

Foundation models support robot manipulation through both direct action prediction and higher-level decision making. RT-2 transfers vision-language pretraining into control by representing robot actions as tokens, while OpenVLA provides an open-source vision-language-action (VLA) model trained on diverse robot demonstrations [2], [3]. π_0 combines

a pretrained vision-language backbone with flow-matching action generation, and Octo learns a generalist diffusion policy from heterogeneous robot data [4], [5]. Other approaches leverage language models: SayCan selects skills based on learned affordances [6], while Code as Policies generates executable programs for robot control [7].

A related line connects policy learning with future visual prediction. UniPi formulates planning as text-conditioned video generation and extracts executable actions from the resulting visual plans [8]. Unified models couple video and action learning: UVA learns a shared latent representation with decoupled diffusion heads, while UWM integrates video and action diffusion in a single transformer with independently controlled diffusion timesteps [9], [10]. DreamZero builds on a pretrained video diffusion backbone to jointly predict future observations and robot actions [11]. WAMJET accelerates the inference code of a trained policy, with numerical and task evaluations used to assess the effect of each optimization.

B. Accelerating Video Generation and World Action Models

Diffusion acceleration operates at several levels of the inference computation. DPM-Solver reduces the number of model evaluations through higher-order numerical integration, whereas progressive distillation trains a student to reproduce a longer sampling process with fewer steps [12], [13]. Training-free caching methods instead exploit redundancy along the denoising trajectory: TeaCache uses timestep-conditioned input changes to guide reuse, and TaylorSeer forecasts intermediate features rather than simply reusing them [14], [15]. Sparse VideoGen reduces attention computation through

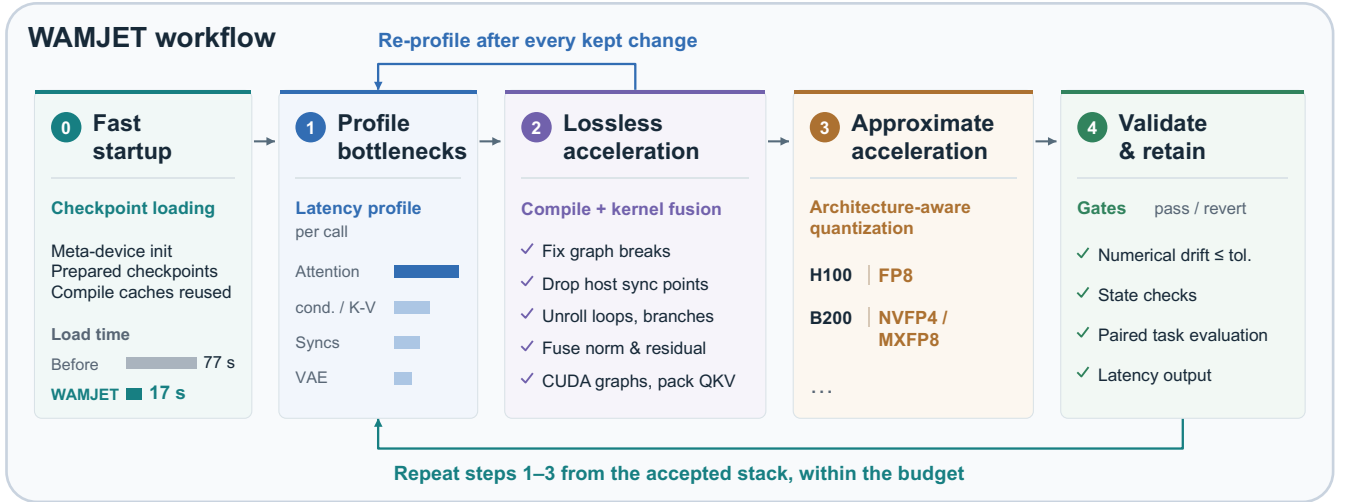


Fig. 2: WAMJET’s bottleneck-driven optimization workflow. The agent reduces startup costs, profiles inference to identify bottlenecks, explores lossless acceleration followed by approximate techniques when permitted, and validates candidates. The agent retains accepted changes, reassesses the remaining bottlenecks, and continues searching within the specified budget.

spatial-temporal sparsity, token merging reduces redundant token processing, and SVDQuant enables low-bit diffusion inference with a high-precision low-rank branch [16], [17], [18]. These approximations differ from implementations such as FlashAttention, which accelerates exact dense attention through IO-aware tiling and reduced memory traffic [19].

WAM-specific works explore both architectural efficiency and deployment-time optimization. DreamZero combines caching, quantization, compilation, kernel optimization, and asynchronous execution in a model-specific control stack [11]. Fast-WAM retains video co-training but removes test-time future-video generation [20]. Light-WAM further adopts a compact video backbone and state-fusion action decoding, predicting action chunks without a generative action expert [21]. These methods require retraining and thus are not drop-in optimizations for arbitrary checkpoints. In contrast, C³ache is a training-free WAM accelerator that reuses action-expert residuals across consecutive inference chunks, exploiting cross-chunk redundancy [22].

While these methods offer complementary acceleration mechanisms, they differ in the evaluation criteria: numerical equivalence, perceptual similarity, and task success are not interchangeable, e.g., video fidelity alone does not ensure manipulation performance. WAMJET builds on these techniques, selecting and combining them according to the bottlenecks of each model and hardware platform.

C. Agentic Workflows and Harness Engineering

Agentic systems make iterative tool use and external feedback central to problem-solving. ReAct introduced the reasoning–action loop [23], SWE-agent built an agent–computer interface for solving software engineering tasks [24], and agent harnesses go further by designing the whole runtime system around the agent [25]. In performance engineering, CUDA-LLM and CudaForge use executable correctness checks and hardware-level performance measurements to iteratively improve GPU kernels [26], [27].

Robotics harnesses apply these ideas to different stages of the robot-learning lifecycle. HARBOR automates robot RL training by coordinating specialized agents from environment setup to reward tuning and to policy training [28]. Nautilus standardizes policy-benchmark-robot integration through typed contracts and validated workflows [29]. Show-Harness instead operates within the embodied control loop: it exposes compact semantic actions to a VLM and deterministically maps them into robot motion, with execution feedback from observations [30]. WAMJET instead applies agentic optimization to the inference code of a trained WAM.

The most related framework is the Sol Video Inference Engine [31], which optimizes video generation by assigning caching, sparse attention, token pruning, quantization, and kernel fusion to parallel agents, then integrating results using human quality feedback. WAMJET addresses the additional requirements of action generation: optimizations must preserve the policy’s recurrent state, respond correctly to changing observations, and be evaluated for their effect on manipulation. Our workflow combines bottleneck analysis with numerical and stateful checks and uses paired task evaluation to assess approximations.

III. PRELIMINARIES

A visuomotor policy maps current observations plus a goal or instruction to robot actions. A world model predicts future visual or latent states from the current state plus an action or goal abstraction. A WAM sits at the overlap: it leverages a pretrained video generation world model backbone as a prior and predicts both future states and robot actions.

We consider robot manipulation from visual observations and language instructions. At time step t , let $x_t = (o_t, q_t, l)$ denote the visual observation, proprioceptive state, and language instruction. A standard policy models the conditional distribution over an action chunk $\mathbf{a}_t = (a_t, \dots, a_{t+H-1})$ of horizon H given x_t :

$$\mathbf{a}_t \sim p_\theta(\mathbf{a}_t | x_t) \quad (1)$$

where θ denotes the policy parameters.

WAMs learn to predict robot actions alongside future states. Let $\mathbf{s}_t = (s_{t+1}, \dots, s_{t+T})$ denote future video frames or their latent representations over a horizon T . WAMs jointly generate both future states and actions:

$$(\mathbf{s}_t, \mathbf{a}_t) \sim p_\theta(\mathbf{s}_t, \mathbf{a}_t \mid x_t). \quad (2)$$

The joint distribution can be factorized as

$$p_\theta(\mathbf{s}_t, \mathbf{a}_t \mid x_t) = p_\theta(\mathbf{s}_t \mid x_t) \cdot p_\theta(\mathbf{a}_t \mid x_t, \mathbf{s}_t) \quad (3)$$

where future states are first imagined by the video backbone, and actions are then inferred conditioned on the imagined future, so that the action distribution effectively marginalizes over imagined futures rather than mapping the current context to actions directly. In practice, the inference procedure depends on the architecture; this factorization is instantiated either by jointly denoising future frames and actions within a shared model [11], or by first generating future video and then feeding it to a separate inverse dynamics module [32], [20]. The expensive stages, therefore, differ across WAMs, even when they share the same video model backbone. The dominant cost may lie in visual encoding, repeated denoising, action generation, or the transfers and synchronization between these stages. It depends on the architecture and deployment platform, so a technique that helps one WAM might not help another.

IV. WAMJET

A. Overview

WAMJET is a harness that guides a coding agent in accelerating a trained WAM policy. Its optimization skill describes how to establish a baseline, identify bottlenecks, test changes, and validate results. Python tools support this workflow through inference timing, trace analysis, numerical comparisons, and paired task analysis. Table I summarizes these components by functional role. The user specifies the policy code and checkpoint, inference configuration, quality requirements, and search budget. Within these constraints, the agent searches for a faster implementation, recording each experiment and its accept/revert decision. It returns the selected implementation with the measurements and validation evidence needed to reproduce and assess the result.

As demonstrated in Fig. 2, WAMJET guides the agent through an iterative bottleneck-driven optimization workflow. The agent first establishes a runnable baseline and reduces startup costs to allow more experiments within the budget. It then profiles inference to identify bottlenecks, explores corresponding lossless acceleration techniques, and benchmarks each candidate under comparable conditions. Experiment records guide subsequent trials, while reusable findings are retained in external memory. When approximation is permitted, the agent also explores approximate techniques, accepting candidates only after the required checks establish that they satisfy the user’s quality requirements. After each change, the agent reassesses the remaining bottlenecks and continues searching until it meets the user requirements or reaches the budget limit.

TABLE I: Functional components of the WAMJET harness.

Component	Function
<i>Agent guidance</i>	
Run configuration	Defines policy, checkpoint, workload, quality requirements, and search budget limits.
Optimization skill	Provides procedures for bottleneck analysis, code changes, accept/revert decisions, and validation.
External memory	Records validated notes for reuse across optimization runs.
<i>Execution and Analysis</i>	
Policy runner	Runs the policy with reference checks, timing measurement, and run records.
Environment preflight	Checks package dependencies, software/GPU compatibility, and inference execution.
Endpoint timing	Measures synchronized warmed inference latency.
Trace analysis	Locates GPU hotspots, synchronization, memory traffic, compilation, and fusion opportunities.
Startup analysis	Measures loading phases and checkpoint page-cache residency.
Deadline analysis	Compares inference latency with the available replanning interval.
<i>Validation</i>	
Output checks	Checks output structure, finiteness, input identity, and numerical drift.
State checks	Replays changed inputs against saved references to detect stale outputs.
Paired task analysis	Compares matched episode outcomes and reports success differences and paired statistics.

B. Agent-Native Acceleration Workflow

Establish a baseline. The agent uses environment preflight checks to establish a runnable inference call on the target GPU and resolves any compatibility issues. The same fixes are applied to both the baseline and subsequent candidates. It then saves the original outputs and measures latency over repeated calls after warmup, keeping the checkpoint, inputs, random seeds, and inference settings fixed. Repeated baseline outputs can also reveal numerical variation that should be considered when checking later changes. Reusing the environment, local checkpoints, and compiler caches reduces the time spent on setup. When loading limits iteration, startup analysis measures loading phases and checkpoint page-cache residency so comparisons use comparable cache states.

Find the bottleneck. The agent inspects the inference code and profiles the stages that appear expensive. Trace analysis helps locate costly modules, repeated operations, memory transfers, and synchronization. It then chooses an optimization that addresses the measured bottleneck using the techniques in Section IV-D. As tracing can substantially slow execution, latency is measured without the profiler. The inference timing tool synchronizes the GPU, retains the latency series, and checks whether the measurements have reached a steady level.

Test the change. The agent measures the modified implementation on the same GPU and compares its outputs with the saved baseline. Cache and CUDA-graph changes are also tested with changing inputs and resets. Changes that make inference slower, produce invalid outputs, or violate the required state or quality constraints are reverted. Unexpected

numerical differences are investigated using the criteria in Section IV-C. A correct change with unchanged measured latency may be retained if it enables a specific follow-up optimization. Such changes are recorded as having no measured speedup. Approximate candidates can be explored before task evaluation is available; their acceptance remains pending until that evaluation is complete.

Repeat and validate. After an improvement, the agent examines the remaining computation and decides the next experiment. This step matters because the benefit of an optimization depends on the changes already applied. Search continues within the time budget, with time reserved to remeasure the original policy and the best candidate under comparable conditions. The final evaluation checks numerical outputs, stateful behavior, and task performance as required by the type of change. Loading and compilation costs are recorded separately from warmed inference latency. Deadline analysis relates inference latency to the available replanning interval. For K actions executed between replans at control frequency f , this interval is K/f . This reports deployment headroom while keeping the workload fixed.

External memory records validated notes from earlier runs, such as environment fixes and model-specific findings. The agent reuses these notes and can incorporate broader lessons into the optimization skill after a run. In Sec. V, when evaluating the harness itself, the guidance is held fixed, and both settings receive the same resources, search budget, and access to prior information.

C. Correctness and Candidate Acceptance

Different optimizations require different checks. WAMJET distinguishes three tiers:

- **Tier A: lossless acceleration.** The change preserves the mathematical computation, precision, workload, and state behavior. The agent assesses eligibility through mathematical and code review and focused output and state checks. Equivalent compilation, fusion, and changes to GEMM or reduction order may introduce floating-point differences. Unexpected drift is investigated rather than rejected solely for exceeding an earlier numerical tolerance.
- **Tier B: validated approximation.** The change approximates internal computation while keeping the denoising steps, schedule, action horizon, and required outputs fixed. It must pass paired task evaluation within a specified bound on quality loss.
- **Tier C: changed workload or unvalidated approximation.** This includes fewer denoising steps, changed horizons or schedules, and approximations that lack passing task evaluation.

The user sets the maximum acceptable tier. Invalid outputs, stale state, and known task-quality failures are rejected in all tiers. Lossless acceleration does not require bit-identical outputs. The numerical checker verifies output keys, shapes, dtypes, finite values, and input identity against the saved reference. For each tensor, it reports the maximum absolute error and relative ℓ_2 error. These measurements are assessed against

the saved tolerances and, when available, the baseline’s run-to-run variation. The original reference is retained throughout the search so that successive small changes cannot hide cumulative drift. However, we find that the numerical drift checker could possibly make the acceleration too conservative, so in practice, the numerical checker is usually turned off.

The agent tests stateful optimizations using input sequences such as $A \rightarrow B \rightarrow A$, comparing each output with its baseline reference under the same state history. It also verifies that resets restore the intended state. These checks help detect stale caches and incorrect reuse of buffer contents from earlier calls. Task performance is evaluated by running the baseline and candidate on matched episodes with identical initial states and random seeds. The number of episodes and acceptable performance loss are specified in advance, and final evaluation uses episodes excluded from calibration and optimization. WAMJET reports the difference in success rate and applies an exact McNemar test to the paired outcomes. Acceptance requires evidence that the candidate meets the predefined quality criterion.

D. Acceleration Techniques

The bottleneck determines which techniques the agent tries first. When inference repeatedly computes the same conditioning, prompt embeddings or K/V projections can be reused as long as their inputs remain unchanged. Unnecessary synchronization and distributed wrappers can be removed, while static buffers and CUDA graphs reduce allocation and dispatch overhead. These changes must preserve cache updates and ensure that graph replay receives the current inputs.

When most time is spent in GPU kernels or memory transfers, the agent can try packed projections, faster implementations of the same attention operation, different tensor layouts, or compiler specialization. Fusing normalization, modulation, activation, and residual operations can reduce intermediate memory traffic. Each change is evaluated through its effect on the complete inference call, since a faster kernel may have little effect on overall latency.

When approximation is allowed, the search can also explore quantization and approximate reuse of features or K/V tensors. The agent verifies that the intended low-precision or reuse path executes and measures the resulting error. Conversion and cache-maintenance costs can offset the expected speedup, so the benefit must be measured for the actual workload. Useful candidates are then evaluated on robotic tasks. As earlier optimizations change the bottleneck, the agent can revisit techniques that initially gave little benefit.

V. EXPERIMENTS

A. Experimental Setup

We evaluate WAMJET across six WAMs, three coding agents, and two GPU architectures. To assess the harness’s generality, we study lossless acceleration across different combinations of coding agents and WAMs on NVIDIA H100. We further evaluate approximate acceleration on both H100 and B200. As a robustness check, we apply WAMJET

TABLE II: Latency on H100 across baseline, w/o WAMJET, and w/ WAMJET Lossless across LLMs and WAMs.

LLM	WAM	Baseline	w/o WAMJET		w/ WAMJET Lossless (Ours)				
		Latency (ms)	Latency (ms)	Speedup	R_1 Latency (ms)*	R_2 Latency (ms)*	R_1/R_2 †	Speedup‡	
GPT-5.6-sol	Fast-WAM	84.7	59.2	1.43x	50.4	37.4	1.35x	2.27x	
	Cosmos-Policy	384.2	369.2	1.04x	236.2	163.1	1.45x	2.36x	
	DreamZero	2780.7	2687.7	1.03x	2373.9	2030.6	1.17x	1.37x	
	LingBot-VA	5323.6	2670.2	1.99x	755.8	686.5	1.10x	7.75x	
Claude-opus-5	Fast-WAM	84.7	54.1	1.57x	47.5	43.1	1.10x	1.96x	
	Cosmos-Policy	384.2	263.7	1.46x	143.0	134.9	1.06x	2.85x	
	DreamZero	2780.7	2454.1	1.13x	1948.7	1904.5	1.02x	1.46x	
	LingBot-VA	5323.6	894.4	5.95x	902.2	537.5	1.68x	9.91x	
GPT-6-astra	Fast-WAM	84.7	59.2	1.43x	47.5	34.9	1.36x	2.43x	
	Cosmos-Policy	384.2	264.6	1.45x	164.8	131.9	1.25x	2.91x	
	DreamZero	2780.7	2502.9	1.11x	1938.2	1785.2	1.09x	1.56x	
	LingBot-VA	5323.6	966.7	5.51x	634.9	534.9	1.19x	9.95x	

All accelerations reported in this table are lossless, with no statistically significant difference in policy performance.

* R_x denotes executing the WAMJET optimization for x rounds. Our final reported results use the R_2 latency.

† R_1/R_2 denotes the speedup of the second round R_2 result over the first round R_1 result.

‡ Total speedup denotes the speedup of our final R_2 result over the baseline.

TABLE III: Latency of two WAMs under baseline and different WAMJET acceleration levels on two GPU architectures.

WAM	GPU	Baseline	w/ WAMJET Lossless		w/ WAMJET Approx.		
		Latency (ms)	Latency (ms)	Speedup	Latency (ms)	Speedup	Approx. Method
DreamZero	H100	2780.7	1785.2	1.56x	1297.0	2.14x	FP8
DreamZero	B200	1598.5	823.9	1.94x	722.6	2.21x	NVFP4 + FP8
Cosmos3-Nano-Policy-DROID	H100	814.8	768.8	1.06x	651.2	1.25x	FP8
Cosmos3-Nano-Policy-DROID	B200	446.3	368.3	1.21x	304.7	1.47x	MXFP8

to a recently released state-of-the-art WAM on B200 to examine whether it can further accelerate an already optimized inference pipeline. Inference latency is measured after warm-up, excluding model loading, compilation, and CUDA-graph capture. Each experiment follows its recorded timing and state-update protocol.

B. Lossless Acceleration

In this section, we evaluate WAMJET’s effectiveness in lossless acceleration and its generality across different combinations of coding agents and WAMs. Since the acceleration here is lossless, the policy performance remains unchanged. On H100, we evaluate three coding agents: GPT-5.6-sol xhigh (Codex), GPT-6-astra xhigh (Codex), and Claude-opus-5 xhigh (Claude Code). We pair them with four WAMs: Fast-WAM [20], DreamZero [11], Cosmos-Policy-Predict2-2B [33], and LingBot-VA [32]. To assess the benefit of WAMJET beyond the coding agents’ own optimization capabilities, we compare three settings:

- **Baseline:** the official upstream implementation, with any compatibility fixes shared across all settings.
- **Without WAMJET:** the coding agent starts from the baseline and optimizes inference independently.
- **With WAMJET:** the coding agent starts from the baseline and optimizes inference using WAMJET’s guidance and tools.

Table II reports the first-round (R_1) and second-round (R_2) WAMJET configurations. From the results, we can see that: (1) **WAMJET can effectively accelerate inference speed across different coding agents and WAMs combinations.** All WAMJET configurations achieve speedups over the

shared model baselines and have lower reported latency than the corresponding runs without WAMJET in all 12 entries. For LingBot-VA, our harness achieves a significant $9.95\times$ speedup with GPT-6-astra, showing that our harness could benefit more from more advanced coding agents. (2) **More budget leads to better optimization.** Even though the first round has already accelerated the latency quite a lot, the ratio R_1/R_2 demonstrates that executing WAMJET for a second round can still further speed up the inference.

C. Approximate Acceleration across GPU Architectures

We evaluate WAMJET with GPT-6-astra on Cosmos3-Nano-Policy-DROID [34] and DreamZero [11], across H100 and B200 GPUs. We further enable approximate acceleration on top off lossless acceleration to quantify acceleration across different WAMs and GPU architectures. Since we apply approximate acceleration here, we also evaluate task success on RoboLab [35] using 20 tasks with 10 episodes per task to assess whether these speedups preserve task performance.

As shown in Table III, WAMJET accelerates both policies across H100 and B200. Lossless optimization achieves $1.56\times$ and $1.94\times$ speedups for DreamZero, and $1.06\times$ and $1.21\times$ for Cosmos3, on H100 and B200, respectively. Adding approximate optimization further increases the overall speedups to $2.14\times$ and $2.21\times$ for DreamZero, and $1.25\times$ and $1.47\times$ for Cosmos3, respectively. These results illustrate the complementary benefits of lossless and approximate optimization. In addition, from Table IV we can see that WAMJET can preserve action quality and task performance even for approximate acceleration, indicating the effectiveness of our iterative validation-involved workflow.

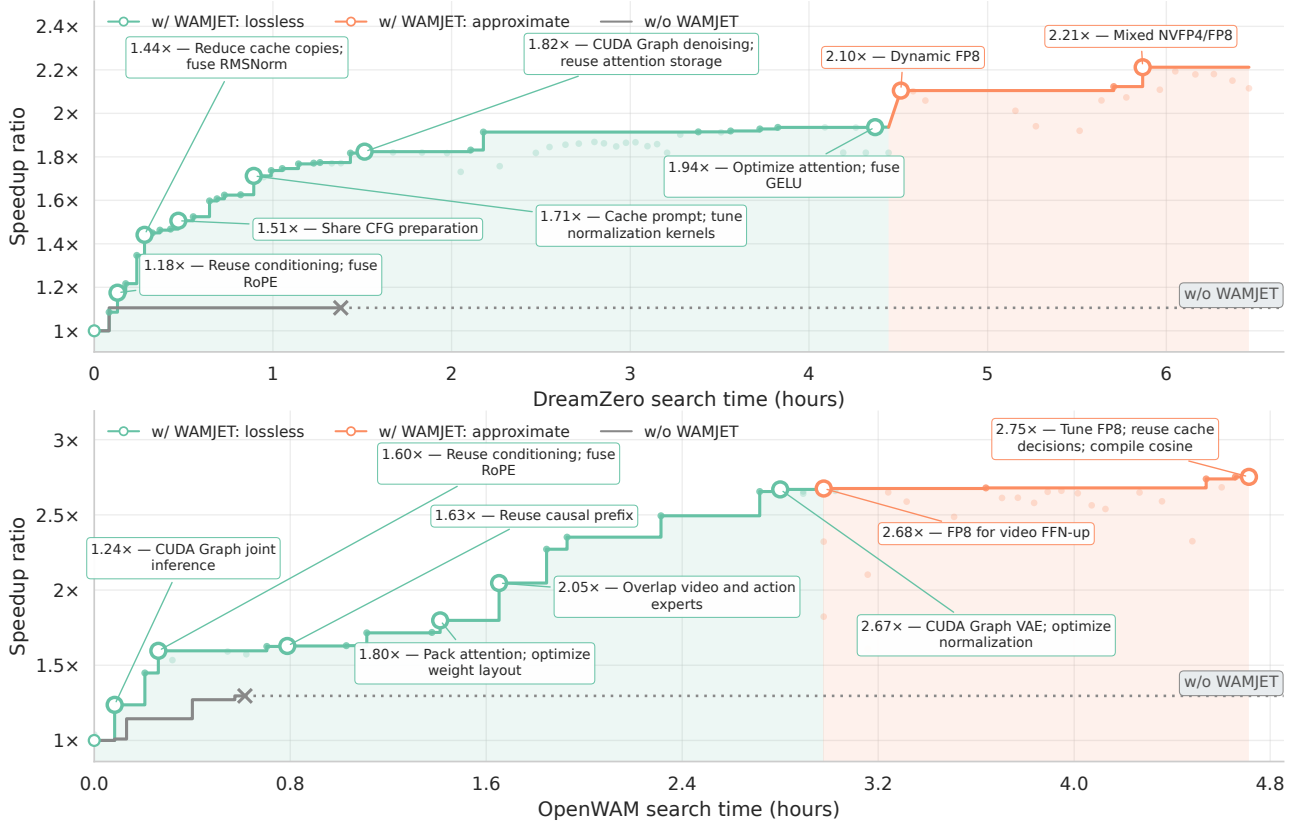


Fig. 3: Optimization progress over search time for DreamZero (top) and OpenWAM (bottom) on B200.

TABLE IV: RoboLab subset success rates of Cosmos3-Nano-Policy-DROID and DreamZero under baseline and different WAMJET acceleration levels.

WAM	Configuration	Approx. Method	Success Rate
DreamZero	Baseline	-	25.0%
	w/ WAMJET Lossless	-	24.0%
	w/ WAMJET Approx.	FP8	26.0%
	w/ WAMJET Approx.	NVFP4 + FP8	25.0%
Cosmos3-Nano-Policy	Baseline	-	43.0%
	w/ WAMJET Lossless	-	47.0%
	w/ WAMJET Approx.	FP8	45.5%
	w/ WAMJET Approx.	MXFP8	44.5%

WAMJET supports hardware-aware optimization. The WAMJET-guided agent selects and refines quantization strategies through profiling and empirical search, adapting them to the target model and GPU architecture. On H100, both DreamZero and Cosmos3 use FP8 with per-output-channel weight scaling and per-token activation scaling. On B200, DreamZero uses a mixed-precision configuration: NVFP4 for the feed-forward projections and FP8 for the remaining linear projections within the diffusion-transformer blocks. Cosmos3 instead uses MXFP8 for its generation projections. The agent combines these quantization strategies with lossless optimizations, including compilation and scheduling, and reassesses the remaining bottlenecks after each change.

D. Further Optimization of a Strong WAM Baseline

Wang et al. recently released OpenWAM [36], which achieves state-of-the-art performance across several robotic benchmarks. OpenWAM already incorporates several in-

TABLE V: Latency and success rates for OpenWAM.

Config	Precision	Latency (ms)	Speedup	LIBERO	LIBERO-plus
Baseline	BF16	87.25	1.00x	98.95%	69.27%
w/ WAMJET Lossless	BF16	32.69	2.67x	99.10%	69.24%
w/ WAMJET Approx.	FP8	31.70	2.75x	99.25%	69.30%

ference optimizations, making it a useful test of whether WAMJET can guide an agent to find further optimization opportunities. We apply WAMJET-guided GPT-6-astra to optimize OpenWAM on B200, and evaluate on full LIBERO [37] and full LIBERO-plus [38].

The results are shown in Table V. OpenWAM already enables the PyTorch compiler and velocity cache, achieving a latency of 87.25 ms on B200. WAMJET further reduces latency by 32.69 ms with lossless optimizations and 31.70 ms with approximate optimizations, achieving 2.67 $\times$ and 2.75 $\times$ speedups, respectively, while preserving action quality. The similar speedups of the lossless and approximate methods indicate that quantization provides limited additional benefit in this setting, with most of the gains coming from lossless methods. These results show that the guided agent can identify substantial acceleration opportunities even in an already highly optimized inference pipeline.

E. Optimization Progress over Search Time

We plot the optimization progress of GPT-6-astra on DreamZero and OpenWAM using B200. Fig. 3 shows individual trials and accepted optimization trajectories, illustrating how performance improves through iterative exploration. With WAMJET, the coding agent autonomously profiles

each candidate, identifies new bottlenecks, and iteratively explores optimization opportunities. In contrast, without our harness, coding agents tend to be overly conservative and terminate early, leaving substantial optimization opportunities unexplored. Furthermore, Fig. 3 also annotates representative points with the key optimizations introduced during the search. Although DreamZero and OpenWAM share techniques such as CUDA Graphs and operator fusion, their final optimization recipes differ even on the same GPU. The results indicate that the guided agent adopts adaptive, model-specific optimization strategies according to each model’s bottlenecks.

F. Startup Cost Acceleration

WAMJET also guides the coding agent in optimizing model startup cost. In recorded B200 runs, DreamZero’s startup time decreases from 208.1 to 30.3 s, and OpenWAM’s decreases from 79.6 to 8.2 s.

VI. CONCLUSION

We presented WAMJET, a harness for accelerating WAM inference through iterative code optimization and evaluation. WAMJET guides a coding agent to find bottlenecks, test acceleration techniques, and check their effects on the policy. Experiments across six models show improvements from conditioning reuse, reduced execution overhead, and kernel optimization, with additional gains from continued search. Quantization can further reduce latency, but its effect on task performance requires evaluation. Future work will focus on broader paired task tests and controlled comparisons of the harness under matched search budgets.

ACKNOWLEDGMENT

We used AI for editing, grammar checking, coding assistance under human guidance, and to gain inspiration for style and art design for figure color and layout, but not for the final figures or any other technical content in this manuscript.

REFERENCES

- [1] Y. Lee, R. Nair, Q. Zhang, K. Lee, O. Khattab, and C. Finn, “Meta-Harness: End-to-end optimization of model harnesses,” *arXiv preprint arXiv:2603.28052*, 2026.
- [2] A. Brohan, *et al.*, “RT-2: Vision-language-action models transfer web knowledge to robotic control,” *arXiv preprint arXiv:2307.15818*, 2023.
- [3] M. J. Kim, *et al.*, “OpenVLA: An open-source vision-language-action model,” *arXiv preprint arXiv:2406.09246*, 2024.
- [4] K. Black, *et al.*, “ π_0 : A vision-language-action flow model for general robot control,” *arXiv preprint arXiv:2410.24164*, 2024.
- [5] Octo Model Team, *et al.*, “Octo: An open-source generalist robot policy,” in *Proceedings of Robotics: Science and Systems*, 2024.
- [6] M. Ahn, *et al.*, “Do As I Can, Not As I Say: Grounding language in robotic affordances,” *arXiv preprint arXiv:2204.01691*, 2022.
- [7] J. Liang, *et al.*, “Code as Policies: Language model programs for embodied control,” in *2023 IEEE International conference on robotics and automation (ICRA)*. IEEE, 2023, pp. 9493–9500.
- [8] Y. Du, *et al.*, “Learning universal policies via text-guided video generation,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.
- [9] S. Li, Y. Gao, D. Sadigh, and S. Song, “Unified video action model,” *arXiv preprint arXiv:2503.00200*, 2025.
- [10] C. Zhu, R. Yu, S. Feng, B. Burchfiel, P. Shah, and A. Gupta, “Unified World Models: Coupling video and action diffusion for pretraining on large robotic datasets,” in *Proceedings of Robotics: Science and Systems (RSS)*, 2025.
- [11] S. Ye, *et al.*, “World action models are zero-shot policies,” *arXiv preprint arXiv:2602.15922*, 2026.
- [12] C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu, “DPM-Solver: A fast ODE solver for diffusion probabilistic model sampling in around 10 steps,” *Advances in neural information processing systems*, 2022.
- [13] T. Salimans and J. Ho, “Progressive distillation for fast sampling of diffusion models,” *arXiv preprint arXiv:2202.00512*, 2022.
- [14] F. Liu, *et al.*, “Timestep embedding tells: It’s time to cache for video diffusion model,” in *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2025, pp. 7353–7363.
- [15] J. Liu, C. Zou, Y. Lyu, J. Chen, and L. Zhang, “From reusing to forecasting: Accelerating diffusion models with TaylorSeers,” in *2025 IEEE/CVF International Conference on Computer Vision*, 2025.
- [16] H. Xi, *et al.*, “Sparse VideoGen: Accelerating video diffusion transformers with spatial-temporal sparsity,” *arXiv preprint arXiv:2502.01776*, 2025.
- [17] D. Bolya and J. Hoffman, “Token merging for fast stable diffusion,” *CVPR Workshop on Efficient Deep Learning for Computer Vision*, 2023.
- [18] M. Li, *et al.*, “SVDQuant: Absorbing outliers by low-rank components for 4-bit diffusion models,” in *The Thirteenth International Conference on Learning Representations*, 2025.
- [19] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, “FlashAttention: Fast and memory-efficient exact attention with io-awareness,” *Advances in neural information processing systems*, 2022.
- [20] T. Yuan, Z. Dong, Y. Liu, and H. Zhao, “Fast-WAM: Do world action models need test-time future imagination?” *arXiv preprint arXiv:2603.16666*, 2026.
- [21] Z. Li, *et al.*, “Light-WAM: Efficient world action models with state-fusion action decoding,” *arXiv preprint arXiv:2606.08242*, 2026.
- [22] W. Zhao, L. Nguyen, Z. Lu, and Y. Shang, “C³ache: Accelerating world action models with cross inference chunk cache,” *arXiv preprint arXiv:2606.08962*, 2026.
- [23] S. Yao, *et al.*, “ReAct: Synergizing reasoning and acting in language models,” in *International Conference on Learning Representations (ICLR)*, 2023.
- [24] J. Yang, *et al.*, “SWE-agent: Agent-computer interfaces enable automated software engineering,” *Advances in Neural Information Processing Systems*, vol. 37, pp. 50 528–50 652, 2024.
- [25] H. Zhong and S. Zhu, “AI Harness Engineering: A runtime substrate for foundation-model software agents,” *arXiv preprint arXiv:2605.13357*, 2026.
- [26] W. Chen, J. Zhu, Q. Fan, Y. Ma, and A. Zou, “CUDA-LLM: LLMs can write efficient CUDA kernels,” *arXiv preprint arXiv:2506.09092*, 2025.
- [27] Z. Zhang, R. Wang, S. Li, Y. Luo, M. Hong, and C. Ding, “CudaForge: An agent framework with hardware feedback for cuda kernel optimization,” *arXiv preprint arXiv:2511.01884*, 2025.
- [28] Z. Li, *et al.*, “HARBOR: A harness framework for agentic robot reinforcement learning,” *arXiv preprint arXiv:2606.08610*, 2026.
- [29] Y. Jin, *et al.*, “Nautilus: From one prompt to plug-and-play robot learning,” *arXiv preprint arXiv:2605.11665*, 2026.
- [30] Y. Chen, *et al.*, “Show-Harness: Just a VLM agent can play robots,” *arXiv preprint arXiv:2609.10522*, 2026.
- [31] Y. Li, *et al.*, “Sol Video Inference Engine: Agent-native full-stack acceleration framework for efficient video generation,” *arXiv preprint arXiv:2606.23743*, 2026.
- [32] L. Li, *et al.*, “Causal world modeling for robot control,” *arXiv preprint arXiv:2601.21998*, 2026.
- [33] M. J. Kim, *et al.*, “Cosmos Policy: Fine-tuning video models for visuomotor control and planning,” *arXiv preprint arXiv:2601.16163*, 2026.
- [34] N. Agarwal, *et al.*, “Cosmos 3: Omnimodal world models for physical AI,” *arXiv preprint arXiv:2606.02800*, 2026.
- [35] X. Yang, *et al.*, “RoboLab: A high-fidelity simulation benchmark for analysis of task generalist policies,” in *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2026.
- [36] Y. Wang, *et al.*, “OpenWAM: An open, modular exploration towards systematic world-action model pretraining,” *arXiv preprint arXiv:2609.07398*, 2026.
- [37] B. Liu, *et al.*, “LIBERO: Benchmarking knowledge transfer for lifelong robot learning,” in *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023.
- [38] S. Fei, *et al.*, “LIBERO-Plus: In-depth Robustness analysis of vision-language-action models,” *arXiv preprint arXiv:2510.13626*, 2025.