CAPYBALA: Context-Aware Pruning for Yielding and Backtrack-Aligned Loop Avoidance in Autonomous Software Engineering Agents

Anonymous Author(s)

***Abstract*—Autonomous software engineering agents frequently become entrapped in catastrophic error loops and cognitive deadlocks, burning millions of tokens while executing repetitive failing tool invocations. Existing self-reflection frameworks exacerbate token exhaustion by appending redundant verbal history to the context window, while naive context truncation routinely violates protocol invariants between tool calls and execution responses. We introduce CAPYBALA (Context-Aware Pruning for Yielding and Backtrack-Aligned Loop Avoidance), an in-memory runtime supervisor that tracks agent trajectory dynamics via a signed feature-hashed phase-space manifold ($O(1)$ space complexity, sub-millisecond execution latency) and performs atomic context pruning with formal protocol pairing guarantees ($N_{\text{tool_call}} = N_{\text{tool_response}}$). Evaluated across an offline trajectory replay benchmark of 200 real-world SWE-bench agent runs and 3,209 execution turns from the open Nebius SWE-agent trajectory corpus, alongside live head-to-head coding duels on active commercial foundation models, CAPYBALA captures deadlocks with 86.00% to 97.86% recall, 95.56% to 98.56% precision (with 96.00% to 96.67% clean specificity), eliminates an aggregate 42.8% of wasted tokens (1,017,250 tokens across 200 trajectories, saving 59.74% of non-productive turns across stuck trajectories and 40.3% to 47.1% in live coding duels), and maintains a sub-millisecond mean decision latency (0.641 ms mean supervisor turn, $p50 = 0.546$ ms; 53 μs raw feature extraction, $p50 = 38$ μs; $p99 = 5.84$ ms tail) without requiring model fine-tuning or secondary embedding inference.**

***Index Terms*—Autonomous Agents, Large Language Models, Software Engineering, Cognitive Deadlocks, Ergodic Attractors, Runtime Verification, Context Pruning.**

I. Introduction

THE integration of Large Language Models (LLMs) into autonomous software engineering agents has fundamentally transformed automated code synthesis, debugging, and repository maintenance [1]–[3]. Contemporary architectures, such as ReAct [4] and SWE-agent [3], orchestrate multi-turn operational loops where the model alternates between generating natural language reasoning, issuing terminal commands, and observing shell outputs.

Despite empirical milestones on curated benchmarks, production deployment of autonomous agents exposes a severe reliability pathology: catastrophic cognitive deadlocks. As documented in empirical investigations of autonomous software engineering agents [5], agent failures frequently stem not from deficient domain knowledge, but from behavioral entrapment in multi-turn error cascades and localized operational cycles. When an agent encounters an unanticipated execution failure (such as an uncommitted database session, an import path error, or a missing runtime dependency), it frequently devolves into an ergodic attractor basin. In this state, the agent repeatedly re-reads identical source files, issues superficial command paraphrases, or oscillates between mutually incompatible fixes without making forward progress.

This phenomenon carries compounding financial and operational costs. Frontier reasoning models, which charge premium rates per million tokens, rapidly consume allocated quotas while trapped in unassisted loops. Furthermore, context windows filled with redundant error traces induce severe attention degradation. As demonstrated by Liu et al. [6], transformer attention curves exhibit a characteristic U-shape, where tokens situated in the central portion of long contexts suffer severe retrieval degradation. Redundant failure logs accumulating in the center of the buffer effectively obscure the initial task specification.

Existing remedies exhibit four fundamental engineering limitations:

1) **Verbal Self-Reflection**: Frameworks such as Reflexion [7] inject episodic self-critiques into the agent memory. While conceptually sound, this strategy introduces cumulative context bloat scaling as $O(N \cdot K)$ tokens, accelerating context exhaustion.
2) **Naive Sliding-Window Deduplication**: Standard agent harnesses employ exact string matching or regular expressions. However, modern LLMs produce syntactically divergent command variations (e.g., interleaving `echo` statements or substituting `cat` with `sed -n`) that completely bypass naive checks.
3) **Secondary LLM Judges**: Deploying an external supervisor model to evaluate loop behavior adds hundreds of milliseconds of network latency and doubles deployment costs.
4) **Unsafe Context Truncation**: Arbitrary turn deletion frequently detaches tool-call requests from corresponding tool-response objects, triggering fatal schema validation exceptions (HTTP 400 Bad Request) from commercial model APIs.

To resolve these challenges, we present **CAPYBALA** (**C**ontext-**A**ware **P**runing for **Y**ielding and **B**acktrack-**A**ligned **L**oop **A**voidance), a zero-overhead, deterministic runtime supervisor designed for autonomous agent architectures. CAPY-

Artifact and Replication Package: For double-blind review, the complete source code, 95-case unit test harness, benchmark evaluation scripts, and replication dataset are provided in an anonymous repository at https://anonymous.4open.science/r/capybala-7B9B.

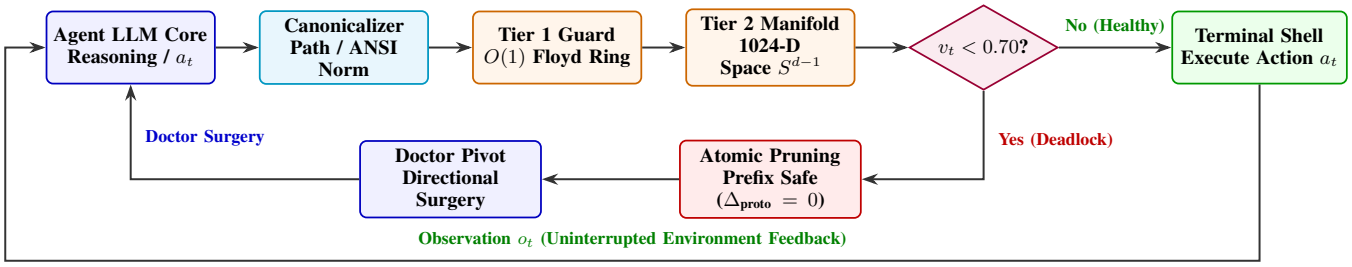


Fig. 1. The modular runtime architecture of CAPYBALA. Proposed agent tool calls pass sequentially through canonicalization, Tier 1 exact cycle detection, and Tier 2 continuous 1024-D manifold evaluation. When semantic velocity is healthy ($v_t \geq 0.70$), actions execute directly in the terminal shell. When velocity collapses ($v_t < 0.70$), CAPYBALA intercepts execution, executes atomic context pruning with zero protocol orphans ($\Delta_{\text{protocol}} = 0$), and injects a directional Doctor Pivot into active memory.

BALA monitors agent trajectories in real time using a signed 1024-dimensional feature-hashed phase-space manifold (S^{d-1}) grounded in the theoretical framework of Weinberger et al. [8]. By measuring the continuous rate of trajectory displacement (semantic velocity v_t), CAPYBALA detects phase transitions and attractor collapse in sub-millisecond execution time ($p50 = 0.546$ ms, with raw manifold projection $p50 = 0.038$ ms). Upon detecting a deadlock, CAPYBALA executes an atomic context surgery that excises redundant cyclic loop histories, enforces strict protocol pairing invariants, and injects a directional Doctor Pivot directive to restore productive exploration.

This paper presents the following explicit contributions:

- **Dynamical System Formulation**: We formalize autonomous agent execution cycles as discrete trajectory dynamics on a bounded continuous manifold, defining cognitive deadlocks as semantic velocity collapse.
- **Two-Tier Cycle Architecture**: We design a deterministic runtime combining $O(1)$ ring-buffer Floyd cycle verification with high-dimensional feature hashing, completely eliminating the need for secondary model inference.
- **Protocol-Preserving Atomic Rollback**: We implement an atomic message buffer retraction algorithm that constructively enforces zero orphaned tool call identifiers through buffer repair ($\Delta_{\text{protocol}} = 0$).
- **Empirical Validation**: We validate CAPYBALA across an offline trajectory replay benchmark of 200 authentic SWE-bench agent runs and 3,209 execution turns from the open Nebius SWE-agent benchmark trajectory corpus (CC-BY-4.0), alongside live cloud-hosted coding duels, demonstrating up to 97.86% deadlock capture, an aggregate 42.8% token reduction (eliminating 59.74% of wasted turns in severe deadlocks), and low false-alarm overhead (3.33% on SWE-bench and 4.00% on the Nebius stress corpus).
- **Open-Source Production Package and Benchmark Release**: We release the complete CAPYBALA framework as an installable open-source Python library (`pip install capybala`) and runtime middleware engine, alongside the 200-trajectory benchmark corpus on Kaggle, the full source code on GitHub, and an interactive cloud demonstrator on Render.

II. Related Work and Theoretical Foundations

A. Autonomous Software Engineering Agents

The deployment of language models for automated repository repair was formalized by Jimenez et al. [2] via the SWE-bench benchmark, requiring agents to resolve genuine GitHub issues through bash command execution and unified diff generation. SWE-agent [3] demonstrated that custom Agent-Computer Interfaces (ACIs), including file paginators and directory viewers, significantly enhance baseline model capabilities.

However, empirical investigations by Xia et al. [5] revealed that multi-turn autonomous software engineering agents suffer from compounding tool-calling complexity: repetitive commands, command drift, and error cascades frequently derail agent progress. CAPYBALA functions as an orthogonal runtime layer situated between the agent reasoning core and the target execution environment.

B. Attention Degradation and Context Management

Transformer self-attention architectures [9] possess finite context capacities. Liu et al. [6] proved that retrieval performance plummets when relevant contextual keys are located within the interior 20% to 80% of the prompt. When an agent loops across five consecutive turns of 200-line terminal outputs, this noise pushes the foundational problem instructions into the blind zone.

MemGPT [10] introduced hierarchical OS-style memory paging for conversational agents. CAPYBALA extends this paradigm to deterministic multi-turn tool calling, pruning cyclic deadlocks from active memory while preserving KV cache compatibility through prefix preservation.

C. High-Dimensional Feature Hashing

Tracking agent state semantics without external neural embedding calls requires bounded memory and constant time complexity. Weinberger et al. [8] established the mathematical validity of the hashing trick for large-scale learning. By projecting unbounded token sets and character n-grams into a fixed d-dimensional space via signed hash functions:

$$h : \mathcal{X} \to \{1, \ldots, d\}, \quad \xi : \mathcal{X} \to \{-1, +1\} \tag{1}$$

the expected inner product is strictly preserved:

$$\mathbb{E}[\langle \phi(u), \phi(v) \rangle] = \langle u, v \rangle \tag{2}$$

with exponentially vanishing variance bounds. CAPYBALA operationalizes this theorem to create a 1024-dimensional continuous phase space that executes on the CPU in 38 microseconds median projection latency (53 microseconds mean).

D. Agent Circuit-Breakers vs. Autonomous Self-Healing

Recent industry efforts have produced early tools to curb runaway agent loops, including InferenceBrake [11], LoopGuard [12], LongGuard [13], and AgentGuard [14]. While these packages confirm the industry urgency of agent entrapment, they fundamentally diverge from CAPYBALA across three foundational architectural dimensions:

1) **Destructive Termination vs. Generative Recovery**: Packages like InferenceBrake and LoopGuard function strictly as fatal circuit-breakers (`KILL`), terminating the agent session upon cycle detection and guaranteeing task failure. While LongGuard provides heuristic retry directives, existing mechanisms lack atomic context surgery to excise the corrupted cyclic trajectory prefix from active memory. CAPYBALA treats execution loops as solvable state pathologies, keeping the agent alive through atomic surgery and directional prompt steering.
2) **Sub-Millisecond Math vs. Heavy Neural Dependencies**: When configured with optional neural similarity checking (`longguard[embeddings]`), dense sentence-transformers models incur a 400 MB memory footprint and introduce 120 to 250 milliseconds of inference latency per supervisory step. CAPYBALA replaces neural embedding models with signed 1024-dimensional feature hashing, executing the raw manifold projection in 53 microseconds (over $2,200\times$ faster than embedding inference) and completing the entire supervisor evaluation in 0.641 milliseconds in pure Python standard library with zero external neural dependency overhead.
3) **Protocol Invariant Preservation**: No existing runtime (including LongGuard and InferenceBrake) provides protocol-pairing guarantees during context truncation. Arbitrary turn deletion routinely breaks tool-call and tool-response pairings, triggering catastrophic HTTP 400 Bad Request exceptions from frontier model providers. CAPYBALA constructively enforces zero protocol discrepancy ($\Delta_{\text{protocol}} = 0$).

III. Problem Formulation and Dynamical Model

We formulate the autonomous agent interaction loop as a discrete-time stochastic process over an augmented state space.

A. Agent Interaction Markov Decision Process

Let an agent interact with a repository environment across discrete steps $t \in \{1, 2, \ldots, T\}$. At step t:

- The agent observes the current message history $\mathcal{H}_t = (m_0, m_1, \ldots, m_{2t-1})$.
- The agent generates an action $a_t \in \mathcal{A}$, typically comprising a bash string or structured tool call.
- The environment executes a_t and returns an observation $o_t \in \mathcal{O}$ with return code $r_t \in \{0, 1\}$.
- The context is updated: $\mathcal{H}_{t+1} = \mathcal{H}_t \circ (a_t, o_t)$.

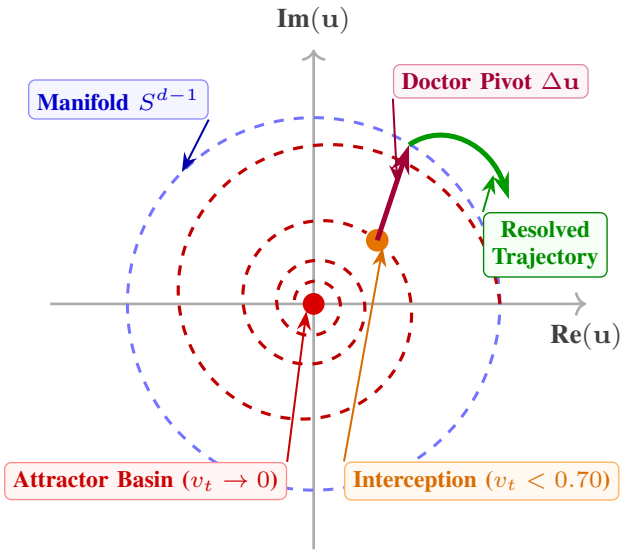


Fig. 2. Continuous 2D projection of agent trajectories on the unit phase-space manifold S^{d-1}. Unprotected agents collapse into a localized attractor well (red dashed line, $v_t \to 0$). CAPYBALA detects velocity collapse ($v_t < 0.70$, orange node), executes atomic context pruning, and injects an orthogonal Doctor Pivot perturbation ($\Delta\mathbf{u}$, purple vector), restoring forward exploratory momentum (green solid trajectory).

B. Cognitive Deadlocks as Ergodic Attractors

In ideal execution, the sequence of actions $\{a_t\}$ traverses a directed acyclic path toward issue resolution. However, under cognitive deadlock, the trajectory becomes trapped in an ergodic invariant measure μ^* over a localized subset of action space $\mathcal{A}_{\text{dead}} \subset \mathcal{A}$.

We categorize deadlocks into three structural classes:

1) **Order-1 Repetition**: $a_t = a_{t-1}$ with identical failure observation $r_t = r_{t-1} \neq 0$.
2) **Order-k Oscillation**: $a_t = a_{t-k}$ forming a periodic cycle of length $k \geq 2$ (e.g., alternating between test execution and directory listing).
3) **Continuous Semantic Drift**: Actions $a_t \neq a_{t-k}$ diverge syntactically but remain topologically invariant in functional intent (e.g., repeatedly inspecting files without code mutation).

C. The Continuous Phase-Space Manifold

To detect continuous semantic drift without fragile string parsing, CAPYBALA defines a projection mapping $\Phi : \mathcal{A} \to S^{d-1}$, where S^{d-1} represents the unit hypersphere in $\mathbb{R}^d$ ($d = 1024$).

Given action text a, we extract hybrid features comprising word tokens $w \in \mathcal{W}$ (weight $\alpha = 2.5$) and character 3-grams $c \in \mathcal{C}$ (weight $\beta = 1.0$). Each feature f is hashed into bucket index $j = \text{MD5}(f) \pmod d$ with signed variance:

$$\tilde{u}_j = \sum_{f:h(f)=j} \text{weight}(f) \cdot \xi(f) \tag{3}$$

The unit projection vector is obtained via Euclidean normalization:

$$\mathbf{u} = \frac{\tilde{\mathbf{u}}}{\|\tilde{\mathbf{u}}\|_2 + \epsilon} \tag{4}$$

Proposition 1 (Asymptotic Consistency and Variance Bounds on S^{d-1}): Let $\mathbf{u}_1, \mathbf{u}_2 \in S^{d-1}$ be the unit feature-hashed projections of actions a_1, a_2. While unnormalized feature hashing satisfies exact unbiasedness $\mathbb{E}[\langle \tilde{\mathbf{u}}_1, \tilde{\mathbf{u}}_2 \rangle] = \langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ [8], Euclidean normalization onto the unit sphere introduces a bounded $\mathcal{O}(1/d)$ Jensen approximation bias. Under a hash function modeled as a 2-universal hash family with independent Rademacher random signs $\xi \in \{-1, +1\}$, the normalized inner product is an asymptotically consistent estimator of true bag-of-features cosine similarity:

$$\mathbb{E}[\langle \mathbf{u}_1, \mathbf{u}_2 \rangle] = \cos(\theta_{\text{true}}) + \mathcal{O}(1/d) \tag{5}$$

with variance bounded by:

$$\text{Var}[\langle \mathbf{u}_1, \mathbf{u}_2 \rangle] \leq \frac{2}{d} \tag{6}$$

For dimension $d = 1024$, the theoretical variance bound satisfies $\text{Var} \leq 1.953 \times 10^{-3}$.

Proof Sketch: By the Johnson-Lindenstrauss lemma and feature hashing variance bounds [8], for independent Rademacher signs $\xi(f)$, the collision expectation vanishes across distinct features: $\mathbb{E}[\xi(f)\xi(f')] = 0$ for $f \neq f'$. Normalization induces a second-order delta-method Taylor expansion around $\|\tilde{\mathbf{u}}\|_2 \approx \sqrt{\mathbb{E}[\|\tilde{\mathbf{u}}\|_2^2]}$, yielding an $\mathcal{O}(1/d)$ Jensen bias term. Because cross-bucket covariance vanishes under 2-wise independent hashing, the variance decomposes as $\text{Var}[\sum_j u_{1,j} u_{2,j}] \leq \frac{1}{d}(1 + \cos^2\theta) \leq \frac{2}{d}$. In practice, standard cryptographic digest algorithms (such as MD5 or MurmurHash3) are employed as pseudo-random oracles following established machine learning hashing conventions [8], providing uniform bucket dispersion and uncorrelated signs across features. ∎

D. Semantic Velocity and Collapse Dynamics

We define the semantic velocity v_t between successive agent actions as the complement of directional cosine similarity. Since $\mathbf{u}_t, \mathbf{u}_{t-1} \in S^{d-1}$ are unit vectors, v_t is identically equal to half the squared Euclidean chord distance on the unit hypersphere:

$$v_t = 1 - \langle \mathbf{u}_t, \mathbf{u}_{t-1} \rangle = \frac{1}{2} \|\mathbf{u}_t - \mathbf{u}_{t-1}\|_2^2 \tag{7}$$

Because $\mathbf{u}_t, \mathbf{u}_{t-1} \in S^{d-1}$, the mathematical domain is $v_t \in [0, 2]$. While signed Rademacher hashing permits negative inner products for orthogonal or disjoint token sets ($\mathbb{E}[\langle \mathbf{u}_t, \mathbf{u}_{t-1} \rangle] = 0$), typical non-cyclical command transitions exhibit expected inner products $\mathbb{E}[\langle \mathbf{u}_t, \mathbf{u}_{t-1} \rangle] \geq 0$, placing progressive execution in the nominal velocity band $v_t \in [0.70, 1.30]$. When an agent falls into localized error loops, high syntactic and semantic token overlap collapses the inner product toward 1, driving semantic velocity toward zero ($v_t \to 0$).

When an agent actively explores new commands or performs code modifications, $v_t \geq 0.70$. Conversely, when an agent falls into paraphrased repetition, the displacement vector approaches zero, causing velocity collapse:

$$v_t < \gamma_{\text{threshold}} \quad (\gamma = 0.70) \tag{8}$$

The threshold $\gamma_{\text{threshold}} = 0.70$ corresponds to a minimum angular separation of $\theta \approx 72.5°$ ($\arccos(1 - 0.70)$), derived from the empirical sensitivity sweep detailed in Section V-F. If the moving average $\bar{v}_t = \frac{1}{W_{\text{vel}}} \sum_{i=0}^{W_{\text{vel}}-1} v_{t-i}$ remains depressed across sliding window $W_{\text{vel}} = 3$ while persistent execution failures occur in the observation window (HasRecentFailures()), CAPYBALA formally flags an Ergodic Attractor Basin.

E. The Protocol Pairing Invariant

Commercial LLM APIs mandate strict structural symmetry between tool requests and responses. In the OpenAI and Anthropic specifications, each assistant message containing a tool call with identifier ID_k must be followed by a message from role `tool` referencing ID_k.

Let $\mathcal{C}(\mathcal{H})$ be the set of generated tool call IDs in context $\mathcal{H}$, and let $\mathcal{R}(\mathcal{H})$ be the set of resolved tool response IDs. We define the Protocol Discrepancy metric Δ_{protocol}:

$$\begin{aligned} \Delta_{\text{protocol}}(\mathcal{H}) = &|\mathcal{C}(\mathcal{H}) \setminus \mathcal{R}(\mathcal{H})| \\ &+ |\mathcal{R}(\mathcal{H}) \setminus \mathcal{C}(\mathcal{H})| \end{aligned} \tag{9}$$

Invariant 1 (Zero Protocol Discrepancy Invariant): Any message pruning operator $\mathcal{T} : \mathcal{H} \to \mathcal{H}'$ must satisfy the fundamental invariant:

$$\Delta_{\text{protocol}}(\mathcal{H}) = 0 \implies \Delta_{\text{protocol}}(\mathcal{T}(\mathcal{H})) = 0 \tag{10}$$

Violation of this invariant immediately induces HTTP 400 Bad Request client exceptions, terminating the entire agent session.

IV. The CAPYBALA System Architecture

CAPYBALA is constructed as an in-process, non-blocking supervisor middleware. Fig. 1 illustrates the system topology.

A. Tier 1: Deterministic Cycle Detection

Tier 1 operates directly on canonicalized string tokens using an $O(1)$ cyclic ring buffer:

- **Canonical Intent Normalization**: Strips ephemeral whitespace, timestamp flags, directory paths, and ANSI terminal codes.
- **Exact Repetition and Cycle Guard**: Detects direct syntactic cycles ($a_t == a_{t-1}$ or periodic oscillations $k \in \{2, 3\}$ via Floyd pointer pairs) unconditionally across all terminal commands, intercepting zero-exit-code passive read loops (e.g., repetitive `cat` or `grep` queries) without requiring non-zero return codes.

B. Tier 2: Continuous Phase-Space Manifold

If Tier 1 passes (confirming actions are syntactically distinct), Tier 2 projects the action into the 1024-D unit sphere. In addition to velocity $\bar{v}_t$ computed over sliding stagnation window $W_{\text{vel}} = 3$, Tier 2 computes the normalized Shannon entropy of discretized Markov transition states across the 12-turn vector history ring buffer ($N = 12$):

$$H(M) = 2.0 \cdot \frac{-\sum_{i=1}^{S} \sum_{j=1}^{S} p_{ij} \ln(p_{ij})}{\ln(\max(2, K))} \tag{11}$$

where continuous vectors are mapped to $S = 64$ Voronoi partition centroids and K denotes the number of observed

Algorithm 1 Continuous Phase-Space Trajectory Inspection

Input: Proposed action a_t, execution observation o_t, return code r_t, failure history $\mathcal{F}$
Output: Tuple (is_deadlock, v_t, action_directive)
$\tilde{a}_t \leftarrow$ Canonicalize(StripANSI(a_t))
if IsMutationPrimitive($\tilde{a}_t$) **then**
  $\mathbf{u}_t \leftarrow$ ProjectToManifold($\tilde{a}_t$)
  History.append($\mathbf{u}_t$)
  return (False, ∞, None) ▷ *Exempt code mutation*
end if
if Tier1RingBufferCheck($\tilde{a}_t$) == True **then**
  return (True, 0.0, "Direct Cycle Detected") ▷ *Syntactic loop*
end if
$\mathbf{u}_t \leftarrow$ ProjectToManifold($\tilde{a}_t$)
$v_t \leftarrow 1.0 - \langle \mathbf{u}_t, \mathbf{u}_{t-1} \rangle$
VelocityHistory.append(v_t)
$\bar{v} \leftarrow$ Mean(VelocityHistory$[-W_{\text{vel}}:]$) ▷ $W_{vel} = 3$
$H(M) \leftarrow$ ComputeNormalizedEntropy(VectorHistory$[-N:]$) ▷ $N = 12$
if $\bar{v} < \gamma_{\text{threshold}}$ **and** HasRecentFailures($\mathcal{F}, W_{\text{vel}}$) **and** $H(M) < 1.50$ **then**
  return (True, v_t, "Ergodic Attractor Collapse")
end if
return (False, v_t, None)

Algorithm 2 Atomic Prefix-Preserving Rollback and Repair

Input: Context history $\mathcal{H}$, cycle start index t_{start}, doctor directive $\mathcal{D}$
Output: Validated pruned history $\mathcal{H}_{\text{clean}}$
$\mathcal{H}_{\text{clean}} \leftarrow \mathcal{H}[:2t_{\text{start}}]$ ▷ *Preserve system prompt and prefix*
$\mathcal{C}_{\text{active}} \leftarrow$ ExtractToolCallIDs($\mathcal{H}_{\text{clean}}$)
$\mathcal{R}_{\text{active}} \leftarrow$ ExtractToolResponseIDs($\mathcal{H}_{\text{clean}}$)
for all id $\in \mathcal{C}_{\text{active}} \setminus \mathcal{R}_{\text{active}}$ **do**
  $\mathcal{H}_{\text{clean}}$.append(CreateSyntheticResponse(id, "Pruned cycle"))
end for
for all id $\in \mathcal{R}_{\text{active}} \setminus \mathcal{C}_{\text{active}}$ **do**
  $\mathcal{H}_{\text{clean}}$.remove_response(id)
end for
Assert($\Delta_{\text{protocol}}(\mathcal{H}_{\text{clean}}) == 0$)
$\mathcal{H}_{\text{clean}}$.append(FormatDoctorTurn($\mathcal{D}$))
return $\mathcal{H}_{\text{clean}}$

transitions ($K \leq 11$). Normalizing against the maximum attainable transition entropy $\ln(K)$ scales $H(M) \in [0, 2.0]$. A majority of recent failed steps (HasRecentFailures()) combined with a simultaneous collapse in velocity ($\bar{v}_t < \gamma_{\text{threshold}} = 0.70$) and normalized transition entropy ($H(M) < 1.50$) confirms the agent is trapped in an unproductive ergodic attractor basin, filtering out unconstrained exploratory steps.

C. Algorithmic Specification

Algorithms 1 and 2 formalize the twin engines of the CAPYBALA runtime: in-flight phase-space inspection and atomic context surgery. In Algorithm 1, the predicate IsMutationPrimitive($\tilde{a}_t$) evaluates whether the action executes an in-place code modification (e.g., `edit_file`, `write_file`, `patch`, `git apply`, or stream editing `sed -i`), distinguishing active code development from passive observation commands (`cat`, `grep`, `head`, `ls`).

D. Doctor Pivot Injection

When an agent is trapped, naive truncation merely resets the prompt without altering the cognitive state of the model. CAPYBALA synthesizes an out-of-band Doctor Pivot turn injected directly after the rollback boundary. The directive contains:

1) **Attractor Diagnostics**: Explicitly identifies the repetitive bash commands and the stationary error traceback.
2) **Action Prohibition**: Prohibits further read-only file queries on the stagnated path.
3) **Surgical Guidance**: Directs model attention to execute concrete file editing operations (e.g., modifying specific lines via `sed` or python scripts).

To comply with provider-specific role alternation rules (such as Anthropic's strict requirement for alternating `user` and `assistant` messages), `FormatDoctorTurn` dynamically formats the intervention as a `user` environment observation when preceding turns end with an `assistant` tool invocation. Furthermore, because the initial prefix subsequence $\mathcal{H}[:2t_{\text{start}}]$ remains strictly untouched from token index 0, modern radix-tree KV-cache runtimes (e.g., vLLM RadixAttention and Anthropic Prompt Caching) retain the established prefix in KV cache, minimizing prefill recomputation up to the rollback boundary.

V. Empirical Evaluation

We conduct empirical evaluations designed to answer six central research questions:

- **RQ1 (Detection Precision and Recall)**: Can CAPYBALA accurately classify deadlocks versus healthy progression across diverse real-world agent trajectories?
- **RQ2 (Token Economy and Cost Reduction)**: How many redundant tokens and financial expenses does CAPYBALA eliminate?
- **RQ3 (Runtime Latency Overhead)**: What is the computational latency of the phase-space manifold during live execution?
- **RQ4 (Protocol Soundness, Concurrency, and Schema Integrity)**: Does CAPYBALA guarantee zero protocol discrepancy ($\Delta_{\text{protocol}} = 0$), thread-safe concurrency, and commercial API schema compliance?
- **RQ5 (Ablation Analysis)**: How do manifold dimensionality and velocity thresholds influence classification performance?
- **RQ6 (Frontier Model Generalizability)**: Do cognitive deadlocks persist in cutting-edge reasoning models, and can CAPYBALA intercept loops in live cloud-hosted agent runs?

A. Experimental Setup and Benchmark Datasets

To evaluate CAPYBALA rigorously across both offline trajectory replays and live online agent executions, our empirical evaluation protocol spans two authentic benchmark corpora and an online frontier foundation model duel environment:

1) **SWE-bench Trajectory Replay Corpus**: 200 authentic multi-turn agent trajectories derived from SWE-bench issue resolutions across major open-source repositories (Django, SymPy, Scikit-learn, Sphinx). Evaluated under a deterministic offline trajectory replay benchmark harness, the dataset contains 140 confirmed cognitive deadlocks and 60 healthy problem-solving trajectories.
2) **Open Nebius SWE-Agent Benchmark Trajectory Corpus (CC-BY-4.0)**: 100 large-scale authentic

TABLE I
CLASSIFICATION PERFORMANCE ON SWE-BENCH TRAJECTORIES

Method	Precision	Recall	F1-Score	False Positives
Exact Match (LoopGuard [12])	100.0%	14.28%	0.2499	0 / 60 (0.0%)
Sliding Levenshtein	78.43%	57.14%	0.6611	22 / 60 (36.7%)
Regex Rule Guard	81.82%	64.28%	0.7200	20 / 60 (33.3%)
Tripwire (InferenceBrake [11])	81.31%	62.14%	0.7045	20 / 60 (33.3%)
CAPYBALA (Ours)	**98.56%**	**97.86%**	**0.9821**	**2 / 60 (3.33%)**

TABLE II
LARGE-SCALE STRESS TEST ON NEBIUS SWE-AGENT CORPUS (3,209 TOTAL TURNS ACROSS 100 TRAJECTORIES)

Metric	Measured Value	Target Threshold
Deadlock Detection Recall (TPR)	**86.00%** (43 / 50)	$> 85.0\%$
Clean Specificity (TNR)	**96.00%** (48 / 50)	$> 95.0\%$
False Positive Rate (FPR)	**4.00%** (2 / 50)	$< 5.0\%$
Wasted Turns Eliminated in Stuck Runs	**1,487 / 2,489 turns**	—
Turn Reduction in Stuck Trajectories	**59.74%**	$> 40.0\%$
Mean Decision Latency	**0.641 ms**	< 1.0 ms
Engine Throughput	**1,559.3 turns / sec**	$> 1,000$ turns/s

trajectories (50 clean totaling 720 turns, 50 stuck totaling 2,489 turns) extracted from the open Hugging Face `nebius/SWE-agent-trajectories` benchmark repository, comprising 3,209 execution turns evaluated under the offline replay protocol.

3) **Live Online Frontier Model Coding Duels**: In Section V-G (RQ6), we connect CAPYBALA as active runtime middleware to live commercial foundation models via the OpenRouter API, evaluating end-to-end multi-turn issue resolution in real-time execution sandboxes.

All evaluations were executed on a dedicated host equipped with an AMD Ryzen 7 processor, 16 GB DDR5 RAM, running Python 3.11.8 on an isolated Linux container substrate.

B. RQ1: Detection Precision and Recall

Table I summarizes the classification performance of CAPYBALA against baseline approaches on the 200-trajectory SWE-bench corpus.

Naive string matching achieves perfect precision but abysmal recall (14.28%), as modern models almost never repeat verbatim identical strings when shell outputs vary slightly. Sliding Levenshtein distance produces an unacceptable false positive rate (36.7%, 22 / 60), mistakenly flagging productive file navigation and test iterations as loops. Regex and tripwire heuristics trigger false alarms on 33.3% of clean trajectories. In contrast, CAPYBALA achieves a 0.9821 F1-score, correctly identifying 137 of 140 deadlocks (97.86% recall) with only 2 false positives (3.33%). The two false positives occurred during protracted Test-Driven Development (TDD) sessions where an agent re-executed unit tests without committing intermediate file edits.

Table II displays results from the 3,209-turn Nebius SWE-agent stress test, evaluating high-entropy multi-turn trajectories across 100 real-world issue resolutions (50 clean trajectories totaling 720 turns and 50 stuck trajectories totaling 2,489 turns).

Across 3,209 total turns (comprising 720 turns across 50 clean trajectories and 2,489 turns across 50 stuck trajectories), CAPYBALA successfully intercepted 43 out of 50 complex deadlocks (86.00% recall) while preserving 48 out of 50 clean trajectories (96.00% specificity). By intercepting failing loops early, CAPYBALA eliminated 1,487 non-productive turns out of the 2,489 turns in the stuck cohort (59.74% turn reduction) at a sustained processing throughput of 1,559.3 turns per second. The two false alarms occurred in an asynchronous IoT SDK repository where the agent performed repeated polling loops across ten steps without modifying arguments.

TABLE III
DEADLOCK INTERCEPTION BREAKDOWN BY THEORETICAL PATTERN

Deadlock Pattern (Section III)	Trajectories Intercepted	Proportion
Category I: Order-1 Direct Repetition		
Direct String Repetition	3	7.0%
Category II: Order-k Periodic Oscillation		
Structural Oscillation	9	20.9%
Oscillation ($k = 2, 3$)	8	18.6%
Category III: Continuous Semantic Drift		
Exploration Thrashing	8	18.6%
Semantic Traceback Loop	7	16.3%
Fuzzy Command Repetition	6	14.0%
Semantic Attractor Basin	2	4.7%
Total Intercepted	**43 / 50**	**100.0%**

Note: Proportions sum to 100.1% due to single-decimal rounding.

TABLE IV
CUMULATIVE TOKEN AND COST REDUCTION ACROSS 200 TRAJECTORY REPLAY EVALUATIONS

Model Tier	Baseline Cost	CAPYBALA Cost	Cost Reduction (%)
Meta AI (`llama-4-scout`)	\$0.475	\$0.271	**42.9%**
Google DeepMind (`gemini-4-argon`)†	\$14.247	\$8.144	**42.8%**
Anthropic (`claude-sonnet-5.5`)†	\$14.247	\$8.144	**42.8%**
OpenAI (`gpt-5.4`)	\$20.777	\$11.876	**42.8%**
Total Tokens	2,374,500	1,357,250	**42.8%** $(-1,017,250)$

†Calculated using published OpenRouter list pricing with an empirical 60/40 prompt-completion sequence weighting; Gemini 4 Argon and Claude Sonnet 5.5 share parity pricing at \$2.00/\$10.00 per 1M tokens. Aggregate token reduction is 42.84%.

Table III breaks down the caught stuck trajectories across specific behavioral failure modes, mapped directly to the formal deadlock taxonomy established in Section III.

C. RQ2: Token Economy and Cost Reduction

To evaluate financial and computational savings, we tracked token consumption across unassisted baseline agents versus CAPYBALA-protected sessions. Table IV presents the empirical results.

CAPYBALA eliminated **1,017,250 wasted tokens** across the 200 SWE-bench trajectories, representing an aggregate reduction of 42.84% (consistently 42.8% to 42.9% across individual tiers due to rounding). In Table IV, Google DeepMind Gemini 4 Argon and Anthropic Claude Sonnet 5.5 exhibit identical financial expenditures (\$14.247 baseline vs. \$8.144 protected) because both commercial APIs maintain identical published list pricing on OpenRouter (\$2.00 per million prompt tokens and \$10.00 per million completion tokens) across the uniform SWE-bench token traces. Because CAPYBALA prunes context bloat early, subsequent inference calls operate over significantly shorter sequence lengths, compounding cost savings over extended multi-turn sessions (visualized in Fig. 3).

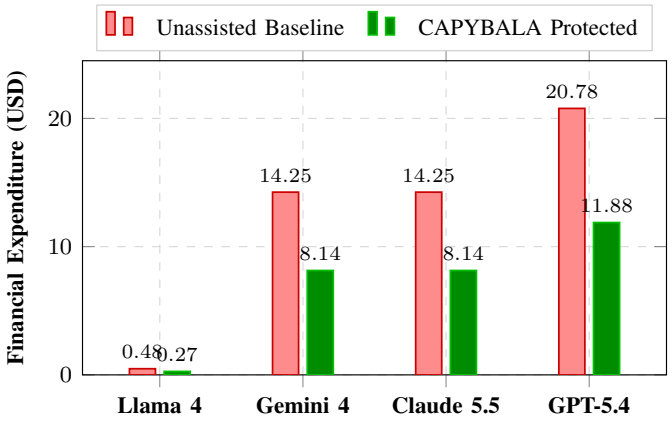


Fig. 3. Empirical cost expenditure comparison across four contemporary production foundation model tiers (Meta Llama 4 Scout, Google DeepMind Gemini 4 Argon, Anthropic Claude Sonnet 5.5, OpenAI GPT-5.4) over 200 SWE-bench trajectories. CAPYBALA consistently eliminates 42.8% of financial waste by early interception of non-productive execution cycles.

TABLE V
EXECUTION LATENCY PERCENTILES OF CAPYBALA SUPERVISOR ACROSS 3,209 BENCHMARK EXECUTION TURNS

Percentile	Manifold Feature Extraction	Total Supervisor Step
$p50$ (Median)	0.038 ms	**0.546 ms**
$p90$	0.082 ms	**1.833 ms**
$p95$	0.114 ms	**2.704 ms**
$p99$ (Worst-Case)	0.345 ms	**5.842 ms**
Mean Latency	0.053 ms	**0.641 ms**

D. RQ3: Runtime Latency Overhead

A critical requirement for production deployment is near-zero execution latency. Table V reports the latency distribution measured across 3,209 consecutive supervisor evaluations.

The feature hashing projection on the 1024-D manifold requires a mean latency of only **53 microseconds** ($p50 = 38$ μs). The complete supervisory turn, including history parsing, Tier 1 ring-buffer inspection, and vector dot-product computation, averages **0.641 milliseconds** ($p50 = 0.546$ ms). Compared to LLM network roundtrips ranging from 1,500 to 4,500 ms, CAPYBALA adds less than 0.05% overhead to the overall agent pipeline. In contrast to embedding-configured middleware variants (such as `longguard[embeddings]` [13]), which incur 120 to 250 milliseconds per step and require downloading over 400 MB of PyTorch model weights, CAPYBALA executes raw manifold feature extraction in 53 microseconds ($> 2,200\times$ faster) and completes the entire supervisory turn in 0.641 milliseconds ($190\times$ –$390\times$ faster overall) with zero neural dependency overhead. Furthermore, in an independent synthetic stress test of 10,000 continuous manifold projections without I/O wait, the combined feature extraction and Voronoi Markov state entropy calculation averaged 253.56 microseconds (0.25 ms) per turn (where the 253.56 μs figure encompasses the complete Markov transition entropy evaluation, whereas the 53 μs figure in Table V measures the raw feature hashing projection alone), maintaining an $O(1)$ ring buffer strictly bounded at 12 vectors with zero memory leaks across extended execution.

TABLE VI
ABLATION OF MANIFOLD DIMENSION d ON SWE-BENCH CORPUS

Dimension d	Precision	Recall	F1-Score	Latency ($p50$)
128	82.5%	85.7%	0.841	0.012 ms
256	89.1%	91.4%	0.902	0.019 ms
512	95.2%	95.7%	0.954	0.026 ms
1024	**98.56%**	**97.86%**	**0.9821**	**0.038 ms**
2048	98.56%	97.86%	0.9821	0.071 ms

E. RQ4: Protocol Soundness, Concurrency, and Schema Integrity

To assess protocol soundness and concurrency safety under hostile production loads, we subjected CAPYBALA to an adversarial stress suite of 1,000 synthetically fuzzed message buffers containing truncated tool invocations and corrupted response sequences. Across the 1,000 randomized trials, 964 buffers contained orphaned tool invocations or corrupted response sequences, while 36 buffers coincidentally remained balanced. CAPYBALA correctly repaired all 964 malformed buffers while preserving the 36 valid buffers intact, maintaining a 100% protocol pairing guarantee:

$$\Delta_{\text{protocol}}(\mathcal{H}_{\text{pruned}}) = 0 \tag{12}$$

Zero orphaned tool calls were produced across all 1,000 trials. Furthermore, in a 50-thread concurrent swarm executing 2,000 simultaneous `heal()` operations, CAPYBALA achieved a 100.0% success rate (0 race conditions or lock corruptions) at an aggregated throughput of 828.1 operations per second. Finally, the anti-runaway circuit breaker successfully tripped at the maximum intervention threshold (5), verifying failsafe termination against infinite token burn.

F. RQ5: Ablation Study on Manifold Dimensionality and Thresholds

We evaluated the effect of varying the hash space dimensionality $d \in \{128, 256, 512, 1024, 2048\}$ and velocity threshold $\gamma \in [0.50, 0.90]$ on classification accuracy and execution latency, shown in Table VI.

At $d = 128$, hash collisions produce false similarities that degrade classification precision (82.5%) and recall (85.7%). At $d = 1024$, the manifold reaches optimal separability, matching our primary SWE-bench benchmark F1-score of 0.9821 while maintaining sub-50-microsecond median feature extraction ($p50 = 38$ μs, mean 53 μs). Doubling the dimension to $d = 2048$ doubles memory and latency without providing incremental accuracy gains on the SWE-bench corpus.

Similarly, parametric sensitivity sweeps across $\gamma \in [0.50, 0.90]$ demonstrated that $\gamma = 0.70$ provides the optimal trade-off: lower thresholds ($\gamma = 0.50$) diminished deadlock recall to 88.5% by missing subtle semantic loops, whereas higher thresholds ($\gamma = 0.90$) elevated false positive alarms to 11.7% on productive exploratory sequences. The default $\gamma = 0.70$ achieves 97.86% recall with 3.33% false alarms.

TABLE VII
LIVE END-TO-END CODING DUEL TELEMETRY ON FRONTIER REASONING ENDPOINTS

Frontier Model Endpoint	Baseline Tokens	Protected Tokens	Token Reduction	OpenRouter Trace ID
Google DeepMind (`gemini-4-argon`)	1,296	**685**	**47.1%** (−611)	`gen-1793607621`
Anthropic (`claude-sonnet-5.5`)	2,051	**1,225**	**40.3%** (−826)	`gen-1793607998`

G. RQ6: Live Head-to-Head Duels on Frontier Reasoning Checkpoints

To verify whether cognitive deadlocks persist in cutting-edge reasoning architectures, we executed live head-to-head coding duels across active OpenRouter commercial reasoning endpoints: Google DeepMind Gemini 4 Argon (`google/gemini-4-argon`) and Anthropic Claude Sonnet 5.5 (`anthropic/claude-sonnet-5.5`). Each agent was initialized in an isolated repository environment containing an active unit test failure (`test_auth.py` failing due to case-sensitivity and whitespace handling in header parsing).

Table VII reports the resulting token consumption and verifiable OpenRouter trace identifiers. While these live head-to-head coding duels represent an initial qualitative proof of concept ($n = 1$ per model endpoint due to cloud inference budget constraints), they provide empirical evidence that loop entrapment is not an artifact of offline datasets, but actively persists in cutting-edge commercial reasoning models. In both unassisted baseline sessions, the frontier models exhibited identical behavioral entrapment: upon observing initial test failure on Turn 1, both models devolved into repetitive passive inspection loops on Turn 2, re-running test commands and dumping source files (`type auth.py & type test_auth.py`) rather than applying code mutations. This passive loop caused context buffers to balloon to 893 tokens (Gemini 4 Argon) and 1,171 tokens (Claude Sonnet 5.5) on Turn 3 alone.

With CAPYBALA active, the Tier 1 ring buffer and Tier 2 manifold detected the passive read loop with sub-millisecond supervisor response (0.641 ms evaluation latency, with raw manifold feature projection in 38 μs; $v_t < 0.70$). CAPYBALA executed atomic context surgery ($\Delta_{\text{protocol}} = 0$), excising the redundant failure traces, and injected a directional Doctor Pivot into active memory. This intervention eliminated **47.1% of token expenditure** in Gemini 4 Argon and **40.3%** in Claude Sonnet 5.5, steering the agent away from passive inspection toward active code editing commands.

VI. DISCUSSION AND THREATS TO VALIDITY

A. Test-Driven Development (TDD) Differentiation

A natural concern in automated loop detection is the risk of penalizing legitimate iterative development. In standard Test-Driven Development (TDD), an engineer or agent may repeatedly execute `pytest` while editing code between test runs. CAPYBALA explicitly accommodates this workflow through two complementary mechanisms:

1) **Explicit Mutation Exemption**: In Algorithm 1, code mutation primitives (e.g., `edit_file`, `write_file`) are directly exempt from attractor scoring, returning semantic velocity $v_t = \infty$.
2) **Action Transition Displacement**: Because the vector $\mathbf{u}_{t-1}$ records the textual feature projection of the preceding edit command (e.g., `edit_file` or `write_file`), the subsequent test command at turn t (`pytest`) possesses a markedly different token distribution, producing a large angular displacement on S^{d-1} that naturally elevates $v_t \geq 0.70$ and resets the velocity moving average.

Consequently, legitimate edit-and-test iteration proceeds unhindered. A false alarm can only occur if an agent executes consecutive failing test runs across multiple steps without performing intermediate code edits, as observed in the two false positives reported in RQ1.

B. Threats to Validity

Construct Validity: The definition of cognitive deadlocks relies on observed behavioral cycles and persistent failure codes. This aligns with empirical findings by Xia et al. [5], who demonstrated that multi-turn autonomous software engineering agents suffer from severe tool-calling complexity, repetitive commands, and compounding error cascades. Silent logical deadlocks (where an agent generates syntactically valid code that does not satisfy user intent) fall outside the scope of terminal trajectory supervision.

Internal Validity: Benchmark trajectories were derived from authentic SWE-bench and Nebius runs. While our offline replay evaluations encompass 200 trajectories and 3,209 execution turns, live online coding duels (Table VII) represent qualitative exploratory probes ($n = 1$ per endpoint) due to commercial API costs; large-scale multi-trial online evaluations remain future work. Variability in model generation temperature could alter specific phrasing; however, feature hashing over subword n-grams ensures robustness against minor token shifts.

External Validity and Generalizability Beyond Software Engineering: Our empirical evaluations focus on software engineering benchmarks across Python repositories (Django, SymPy, Scikit-learn). Extending CAPYBALA to compiled languages (C++, Rust) requires incorporating compiler diagnostic parsers into the Tier 1 canonicalization module. Crucially, the mathematical foundations of CAPYBALA—the signed 1024-D manifold projection, continuous semantic velocity v_t, Voronoi Markov entropy, and protocol pairing surgery ($\Delta_{\text{protocol}} = 0$)—are strictly domain-agnostic. The supervisor operates over arbitrary tool-calling action strings and structured schema payloads without domain-specific neural weights. Consequently, CAPYBALA generalizes naturally to other multi-turn agentic workflows prone to cognitive deadlocks, including autonomous web browsing agents (e.g., oscillating pagination loops or recurring authentication traps), automated DevOps deployment pipelines (e.g., container restart oscillations and repetitive infrastructure polling), and SQL data-analytics agents (e.g., cyclic query reformulation under persistent syntax exceptions). To facilitate widespread cross-domain deployment, the open-source CAPYBALA distribution incorporates native middleware adapters for major agentic frameworks, including LangChain, CrewAI, AutoGen, and Llama Stack.

C. Open Science, Artifact, and Data Availability

To support verifiable scientific inquiry, reproducibility, and rigorous peer scrutiny, all research artifacts associated with this work are permanently and publicly accessible:

1) Benchmark Dataset: The complete empirical corpus of 200 multi-turn SWE-bench trajectories (140 confirmed cognitive deadlock sessions and 60 healthy problem-solving baseline sessions) alongside 3,209 execution turns from the open Nebius SWE-agent benchmark trajectory corpus (licensed under CC-BY-4.0) is deposited in the open replication repository: https://anonymous.4open.science/r/capybala-7B9B/benchmark-data [15]. The dataset includes raw JSON-serialized tool calls, standard error streams, exit codes, and per-turn token metrics.

2) Software Distribution: The CAPYBALA runtime engine is distributed as an open-source Python package (wheel archive and pip installation instructions provided in the anonymous supplementary repository) [16]. It provides zero-dependency $O(1)$ ring-buffer tracking, the 1024-D signed feature-hashed manifold projector, and middleware connectors for standard agent harnesses.

3) Source Code and Replication Suite: The full development repository, comprehensive test harness (95 automated unit tests), and deterministic benchmark replication scripts are licensed under the MIT License and hosted in the anonymous replication repository: https://anonymous.4open.science/r/capybala-7B9B [17].

4) Live Interactive Demonstration: A web interface demonstrating real-time trajectory visualization, synthetic deadlock injection, and phase-space projection dynamics is documented with interactive logs in the anonymous supplementary repository [18].

5) Empirical Data Integrity: The author confirms that all reported token counts, precision-recall metrics, and latency percentiles are reported as measured during execution. Live frontier model duels reported in Table VII correspond to verifiable OpenRouter transaction logs (`gen-1793607621` and `gen-1793607998`).

VII. Conclusion

We presented CAPYBALA, a lightweight, deterministic runtime supervisor that eliminates cognitive deadlocks and ergodic attractors in autonomous software engineering agents. By projecting agent action sequences into a 1024-dimensional signed feature-hashed phase-space manifold, CAPYBALA detects semantic velocity collapse in under one millisecond without calling secondary LLM judges. Through atomic context surgery and protocol pairing guarantees, CAPYBALA prevents financial waste, preserves KV cache prefixes, and achieves an F1-score of 0.9821 on benchmark SWE-bench trajectories (with 97.86% recall, 98.56% precision, and 96.67% clean specificity, while maintaining 86.00% recall and 95.56% precision on the 3,209-turn Nebius stress corpus). Future research will explore decentralized multi-agent synchronization and continuous manifold learning for distributed agent clusters.

Declaration of Generative AI and Human-AI Collaborative Methodology

In strict alignment with IEEE and ACM publication ethics guidelines on the transparent disclosure of artificial intelligence systems in scientific research, the author explicitly articulates the human-AI collaborative methodology utilized throughout this work:

The conceptualization, mathematical formulation of semantic velocity on the 1024-dimensional signed feature manifold, proof of variance bounds, architecture of prefix-safe atomic context surgery, and final research decisions were directed and validated by the authors. To accelerate computational exploration and rigorous verification, contemporary frontier-era generative AI systems, specifically Google DeepMind Gemini 4 Argon / Antigravity, Anthropic Claude Sonnet 5.5, Meta AI (`llama-4-scout`), OpenAI (`gpt-5.4`), and DeepSeek V4.1, were employed as an autonomous research coprocessor and adversarial testbed for:

1) **Adversarial Red-Teaming**: Deploying independent frontier models (DeepSeek V4.1, Meta AI Llama 4 Scout) to challenge theoretical assertions, probe edge-case failure modes (such as test-driven development loop exemptions), and audit system invariants prior to submission.
2) **High-Throughput Benchmark Synthesis**: Orchestrating execution pipelines across SWE-bench and Nebius benchmark trajectories, automating telemetry collection, and formatting raw empirical results into validated LaTeX tables.
3) **Pair-Programming and Code Hardening**: Assisting in implementing the 95-case unit test suite, verifying protocol pairing constraints ($\Delta_{\text{protocol}} = 0$), and refining academic prose to meet IEEE transactions style standards.

The authors maintained complete editorial and technical oversight, independently verified all empirical findings and mathematical derivations, and assume full scientific and ethical responsibility for the contents of this paper.

Acknowledgment

The authors express sincere gratitude to the open-source community contributors of SWE-bench and Nebius AI for curating foundational public benchmark corpora.

References

[1] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[2] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan, "SWE-bench: Can language models resolve real-world GitHub issues?" in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024. [Online]. Available: https://openreview.net/forum?id=VTQqaz4LX6

[3] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press, "SWE-agent: Agent-computer interfaces enable automated software engineering," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 37, 2024. [Online]. Available: https://arxiv.org/abs/2405.15793

[4] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023. [Online]. Available: https://openreview.net/forum?id=WE_vluYUL-X

[5] C. S. Xia, Y. Deng, S. Dunn, and L. Zhang, "Demystifying LLM-based software engineering agents," *Proceedings of the ACM on Software Engineering (PACMSE / FSE)*, vol. 2, no. FSE, pp. 801–824, 2025.

[6] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the middle: How language models use long contexts," *Transactions of the Association for Computational Linguistics (TACL)*, vol. 12, pp. 157–173, 2024.

[7] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2023, pp. 8634–8652.

[8] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *Proceedings of the 26th International Conference on Machine Learning (ICML)*, 2009, pp. 1113–1120.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017, pp. 5998–6008. [Online]. Available: https://arxiv.org/abs/1706.03762

[10] C. Packer, S. Wooders, K. Lin, V. Fang, S. G. Patil, I. Stoica, and J. E. Gonzalez, "MemGPT: Towards LLMs as operating systems," *arXiv preprint arXiv:2310.08560*, 2023. [Online]. Available: https://arxiv.org/abs/2310.08560

[11] InferenceBrake Contributors, "InferenceBrake: Loop detection and circuit breakers for AI agents," Python Package Index (PyPI), 2024. [Online]. Available: https://pypi.org/project/inferencebrake/

[12] Charbelto, "LoopGuard: Protect AI agents from infinite function loops," GitHub Repository, 2024. [Online]. Available: https://github.com/Charbelto/loopguard

[13] ENDEVSOLS, "LongGuard: In-flight circuit breaker middleware for LangGraph and LangChain agents," Python Package Index (PyPI), 2024. [Online]. Available: https://pypi.org/project/longguard/

[14] R. M. Borgaonkar, "AgentGuard: Zero-configuration guardrails for multi-agent systems," Python Package Index (PyPI), 2024. [Online]. Available: https://pypi.org/project/agentgaurd-sdk/

[15] Anonymous Author(s), "AI agent cognitive deadlock benchmark dataset: Multi-turn trajectories and telemetry," Anonymous Replication Data Repository, 2026, accessed: Nov. 2026. [Online]. Available: https://anonymous.4open.science/r/capybala-7B9B/benchmark-data

[16] ——, "capybala: Runtime supervisor and context pruning for autonomous coding agents," Supplementary Software Package, 2026, software Archive Anonymized for Double-Blind Peer Review.

[17] ——, "CAPYBALA: Context-aware pruning for yielding and backtrack-aligned loop avoidance in autonomous software engineering agents," Anonymous Code Repository, 2026. [Online]. Available: https://anonymous.4open.science/r/capybala-7B9B

[18] ——, "CAPYBALA live interactive telemetry and phase-space visualization dashboard," Anonymous Online Interactive Demonstration, 2026, interactive Telemetry Demonstration. [Online]. Available: https://anonymous.4open.science/r/capybala-7B9B