Evaluating LLM Reasoning on Operating System Algorithms via Step-Level Verification

Anonymous Authors[1]

Abstract

Large language models (LLMs) are increasingly applied to technical domains requiring structured, multi-step algorithmic reasoning. Yet systematic benchmarks that isolate intermediate reasoning fidelity — rather than merely checking final answers — remain scarce. We introduce SVAC (Step-Level Verification of Algorithm Correctness), a benchmark that evaluates LLMs on their ability to manually trace three canonical Operating System resource-management algorithms: page replacement (FIFO, LRU, OPT, Clock), process scheduling (SJF, Round Robin, MLFQ), and disk scheduling (FCFS, SSTF, SCAN, C-SCAN, LOOK). For each problem instance, models must produce a full execution trace under strict constraints: no code, no external tools, and no step abbreviation. Ground-truth traces generated by deterministic solvers enable fine-grained, step-level scoring across five complexity levels and three prompting strategies (zero-shot, few-shot, chain-of-thought). Our evaluation reveals characteristic failure modes — including early error cascades, hit/miss misclassification, and algorithm-specific state corruption — that vary systematically across model families, algorithm types, and input complexity. SVAC provides the first unified framework for assessing OS-algorithm reasoning in LLMs, with implications for understanding the depth and limits of procedural reasoning in current foundation models.

1. Introduction

Modern large language models achieve remarkable performance on a wide range of tasks, from natural language understanding to mathematical reasoning. Yet a critical and underexplored question remains: can LLMs faithfully trace through deterministic algorithms step by step, maintaining explicit internal state and propagating it correctly across dozens of sequential operations?

Operating System (OS) resource-management algorithms offer an ideal testbed for this question. They are well-specified, produce ground-truth-verifiable traces, and span a meaningful difficulty spectrum from simple queue-based policies to globally-demanding algorithms requiring lookahead or multi-level state. Crucially, each algorithm family imposes a distinct sequential dependency structure: page replacement requires the model to update a frame set and classify every access as a hit or fault; CPU scheduling demands concurrent state tracking across multiple processes with time-quantum boundaries; disk scheduling requires maintaining a one-dimensional head position and enforcing directional sweep constraints at every step. A single error early in any trace propagates to all subsequent steps, making step-level correctness a strict and meaningful metric.

We present SVAC (Step-Level Verification of Algorithm Correctness), a benchmark spanning three OS algorithm families:

- **Page Replacement:** FIFO, LRU, OPT (Bélády's), and Clock. Models must track queue ordering, recency lists, lookahead distances, and circular buffer state with reference bits.
- **Process Scheduling:** SJF (preemptive and non-preemptive), Round Robin, and MLFQ scheduling. Models trace Gantt-chart-level execution, computing waiting times, turnaround times, and CPU utilisation.
- **Disk Scheduling:** FCFS, SSTF, SCAN, C-SCAN, and LOOK. Models simulate disk head movement across a cylinder space, computing per-step seek distances, cumulative total seek distance, and the exact service order of I/O requests under directional sweep constraints.

The three families probe qualitatively distinct reasoning capabilities: working set management and eviction logic; concurrent state tracking across processes and time quanta; and geometric reasoning over a one-dimensional address

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country.**AUTHORERR: Missing \icmlcorrespondingauthor.**

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

space with directional constraints. A model excelling on all three must possess general procedural reasoning ability, not merely pattern-matching to one algorithm class. Each model produces a full JSON execution trace without code or tool use; ground-truth traces from deterministic solvers enable step-level scoring across five complexity tiers, three prompting strategies (zero-shot, few-shot, chain-of-thought), and multiple locality modes per domain. Our primary contributions are:

1. A unified, reproducible benchmark (SVAC) for evaluating procedural algorithm reasoning in LLMs across three OS algorithm families.
2. A step-level scoring framework with domain-specific error taxonomies — seven failure types for page replacement, twenty-two named failure modes for CPU scheduling, and five for disk scheduling — enabling fine-grained diagnosis of model reasoning breakdowns across all three algorithm families.
3. Empirical findings on how model size, prompting strategy, and algorithm complexity jointly affect step-accuracy, error cascade rate, and recovery behavior across multiple LLM families.

2. Related Work

Evaluating LLMs on formal reasoning has attracted significant attention. Benchmarks such as GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021) assess arithmetic and algebraic problem solving, while BIG-Bench (Srivastava et al., 2022) covers a broad range of algorithmic tasks. These benchmarks primarily evaluate final-answer correctness and do not expose intermediate reasoning fidelity.

Closer to our setting, several works study LLM performance on code execution simulation. Liu et al. (2023) demonstrate that models often predict the output of short programs but fail on longer ones, with errors compounding over execution steps — a finding directly parallel to our step-accuracy degradation results across all three OS domains. Similarly, code reasoning benchmarks such as CRUXEval (Gu et al., 2024) measure input-output prediction but do not enforce step-level traces. The role of prompting strategy in procedural reasoning is also underexplored: prior work generally reports chain-of-thought as beneficial, whereas our results show that CoT is actively detrimental for stateful sequential tasks such as CPU and disk scheduling, where extended deliberation introduces spurious state transitions rather than correcting them.

In the OS domain specifically, no prior benchmark systematically evaluates LLMs on algorithm tracing across multiple algorithm families at the step level. While educational platforms present OS algorithm problems, they do not provide structured evaluation frameworks or ground-truth step traces. SVAC fills this gap by combining deterministic ground-truth generation, domain-specific error taxonomies, and a multi-metric scoring system spanning page replacement, CPU scheduling, and disk scheduling into a single reproducible benchmark.

Related procedural reasoning benchmarks include ProcBench (Fujisawa et al., 2024), L0-Reasoning Bench (Sun et al., 2025), AQA-Bench (Yang et al., 2024), AlgBench (Sun et al., 2026), OSVBench (Li et al., 2025), and integer sequence generation benchmarks (O'Malley et al., 2024). Hard2Verify (Pandit et al., 2025) proposes step-level verification for mathematics, motivating our analogous framework for OS algorithms.

3. Benchmark Design and Methodology

SVAC is designed around three core principles: (i) deterministic ground truth, enabling exact comparison against model outputs; (ii) step-level granularity, exposing intermediate reasoning rather than only final answers; and (iii) no-code constraints, ensuring the model's trace reflects symbolic reasoning rather than code execution. These principles are applied uniformly across all three algorithm families, with domain-specific instance generators, ground-truth solvers, and error taxonomies for each.

3.1. Problem Formulation

Each benchmark instance consists of a fully specified algorithmic problem (algorithm type, parameter values, input sequence), a ground-truth step trace produced by a deterministic solver, and a prompt that instructs the model to manually produce a complete execution trace in JSON format. The model's output is parsed and aligned with the ground truth at the step level, with each step scored independently. For page replacement, each step records the accessed page, hit-or-fault classification, frame state, and running fault count. For CPU scheduling, each step records the simulation time, event type, active process, ready-queue contents, and remaining burst time. For disk scheduling, each step records the current head position, the next request serviced, per-step seek distance, and cumulative seek distance.

All prompts include an explicit system-level instruction that prohibits code in any form — including Python, pseudocode, and code-like constructs such as loops or variable assignments. The constraint is stated as a hard rule: any code invalidates the entire response. This forces the model to perform genuinely symbolic, step-by-step reasoning rather than silently simulating execution.

3.2. Prompting Strategies

Three prompting strategies are evaluated for each algorithm family:

Zero-Shot: The prompt specifies the algorithm rules and JSON output schema. No worked examples are provided. This measures raw algorithmic comprehension.

Few-Shot: The prompt includes one fully worked example instance with a complete ground-truth trace. This measures the model's ability to generalise from a single demonstration to a new instance.

Chain-of-Thought (CoT): A structured "thought" field is added to each step object in the requested JSON schema, appearing before the state variables. Because LLMs generate text autoregressively, placing the thought field first forces the model to reason through each step before committing to state values — enabling genuine scratchpad-style deliberation rather than post-hoc rationalisation.

3.3. Evaluation Metrics

We measure model performance using six metrics, each capturing a distinct dimension of reasoning quality:

Step Accuracy (SA): The fraction of steps in which all state fields exactly match ground truth. SA is the primary metric; it measures how faithfully the model executes the algorithm across the entire trace. A single wrong eviction or misclassified hit-miss propagates to all dependent state fields in subsequent steps, so SA is a strict and meaningful measure of full-sequence reasoning quality. We use SA rather than final-answer accuracy because a model can reach the correct total fault count via partially incorrect intermediate reasoning, masking real errors.

First Error Position (FEP): The step index at which the first error occurs. FEP isolates where a model's trace diverges from ground truth, distinguishing models that fail immediately from those that sustain correct reasoning for many steps. Normalized FEP (dividing by total steps) enables comparison across instances of different lengths.

Error Cascade Rate (ECR): The fraction of steps after the FEP that also contain errors. High ECR indicates that early mistakes propagate and compound, revealing poor error-recovery ability. This metric is critical because OS algorithms are inherently stateful: an incorrect frame set at step k propagates to all future eviction decisions, making cascades the dominant failure mode.

Recovery Rate: The fraction of steps after the FEP that are nevertheless correct. While counterintuitive alongside ECR, some models recover partial correctness after an error — e.g., re-aligning the fault count even after an incorrect eviction. Recovery Rate distinguishes catastrophic failure from partial recovery.

Fault Count Accuracy: Whether the model's final total fault count matches the ground truth. This is a weaker metric than SA (a model can arrive at the correct count via incorrect reasoning), but it captures whether higher-level summary statistics are correct.

JSON Validity Rate: The fraction of responses that produce a structurally valid JSON object matching the requested schema. This metric captures output reliability independent of reasoning quality, and is especially relevant for smaller models that struggle to maintain JSON structure across long traces. A response with invalid JSON cannot be scored and is treated as a complete failure.

4. Page Replacement: Algorithms, Instances, and Results

We begin with page replacement, the most extensively implemented component of SVAC. Four algorithms are evaluated: FIFO (First-In First-Out), LRU (Least Recently Used), OPT (Bélády's optimal algorithm), and Clock. These algorithms span a spectrum from trivially implementable (FIFO requires only a queue) to globally-demanding (OPT requires foreknowledge of the entire future access sequence), and each requires tracking qualitatively different internal state.

4.1. Instance Generation

Problem instances are generated by a synthetic reference-string generator with five complexity levels, four locality modes, and four target algorithms. Complexity is controlled by two coupled parameters: the number of memory frames (3 to 8) and the reference string length (15 to 60 accesses). Complexity Level 1 uses 3 frames and 15 accesses; Level 5 uses 8 frames and 60 accesses, producing traces of up to 60 JSON step objects.

Locality modes control the statistical structure of the reference string. Uniform random strings are drawn without replacement from a fixed page pool, providing a baseline with no exploitable locality. Temporal strings model a working set with occasional cold misses, reflecting realistic program behavior. Zipfian strings follow a power-law access distribution, with a small number of "hot" pages accessed frequently. Adversarial strings are specifically constructed to maximise fault counts for a given target algorithm — for example, cycling through num_frames + 1 pages to force every access to be a fault for LRU — directly exposing algorithm-specific reasoning vulnerabilities.

4.2. Ground Truth and Scoring

Ground-truth traces are produced by reference solvers for each algorithm. Each solver produces a complete step se-

quence containing the accessed page, hit-or-fault classification, the updated frame state (represented as a queue for FIFO, recency list for LRU, frame set with lookahead for OPT, and circular buffer with reference bits for Clock), the evicted page (if any), and the running fault count.

Scoring proceeds by aligning the model's output steps with the ground truth using step indices, then comparing each field independently. Seven error types are tracked: hit/miss misclassification, wrong eviction choice, correct eviction with incorrect justification (OPT-specific), frame state corruption, fault count drift, premature termination (fewer steps than expected), and algorithm-specific errors (e.g., incorrect Clock hand position or FIFO queue order violations).

4.3. Results

Figures 1–4 show the degradation cliff (SA vs. complexity level), failure mutation (error type distribution as complexity increases), cross-algorithm vulnerability map, FEP distribution, and token efficiency curve.

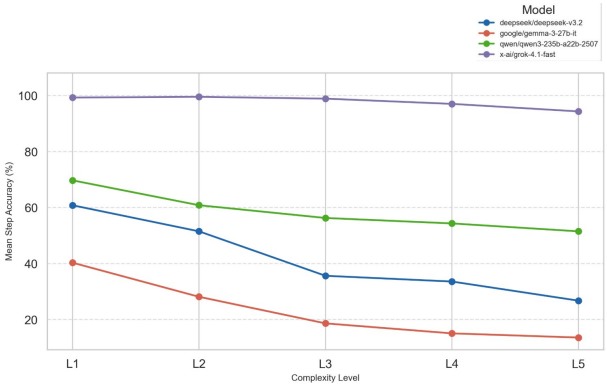

Figure 1:Scaling Law Plot

Figure 1. Scaling Law Plot: grok-4.1-fast maintains above 94% SA even at L5, while gemma-3-27b collapses from 40% to 14% and deepseek drops from 61% to 27% as complexity increases. Procedural reasoning ability does not degrade uniformly — there exists a capability threshold above which models sustain trace fidelity under complexity, and below which performance collapses.

Our results reveal several consistent patterns. First, all models exhibit a sharp degradation cliff between Levels 2 and 3: step accuracy drops significantly as reference string length exceeds 25 accesses and the number of frames increases beyond 4. This finding suggests that LLMs do not fail gradually but rather have a relatively sharp working memory threshold beyond which trace fidelity collapses. Second, error cascade rates are high across all models once the first error occurs, confirming that OS algorithm traces are brittle to early mistakes. Third, adversarial instances expose strong algorithm-specific vulnerabilities: models that perform well on temporal or Zipfian strings often fail substantially on

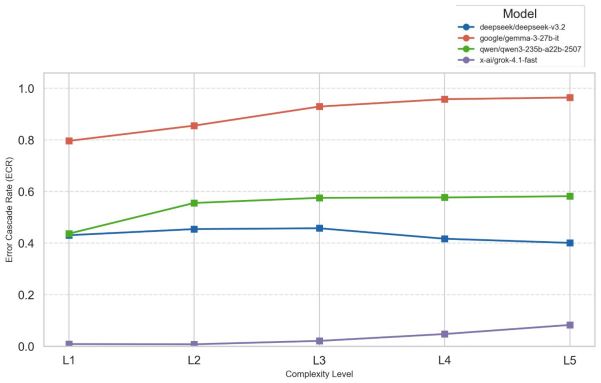

Figure 2: Error Cascade Rate Plot

Figure 2. Error Cascade Rate Plot: gemma's ECR reaches 0.97 at L5 — meaning after its first error, 97% of all subsequent steps are also wrong — while grok's ECR stays below 0.4 across all complexity levels. Weaker models lose internal state coherence entirely after the first mistake.

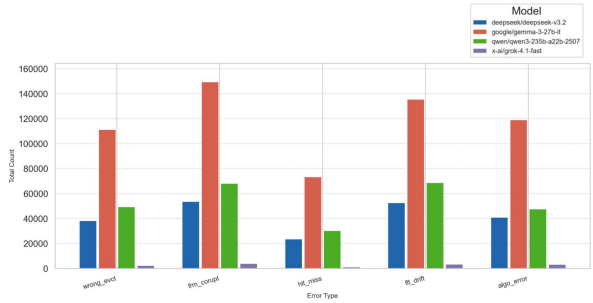

Figure 3: Error Type Comparison Plot

Figure 3. Error Type Comparison Plot: frame state corruption and fault count drift are the dominant error types for gemma and qwen, occurring at roughly 2–3 times the rate of wrong eviction choices, while grok shows near-zero counts across all five error categories.

adversarial strings for LRU and FIFO, while OPT — which requires global lookahead — is the most uniformly difficult algorithm regardless of locality mode.

5. CPU Process Scheduling: Algorithms, Instances, and Results

We evaluate CPU process scheduling, assessing three canonical algorithms: Shortest Job First (SJF), Round Robin (RR), and Multi-Level Feedback Queue (MLFQ). These algorithms span a spectrum of state-tracking complexity, from simple non-preemptive priority evaluation requiring basic selection logic (SJF) to globally demanding preemptive systems requiring tracking of per-process queue levels, quantum timers, and periodic priority boosts (MLFQ).

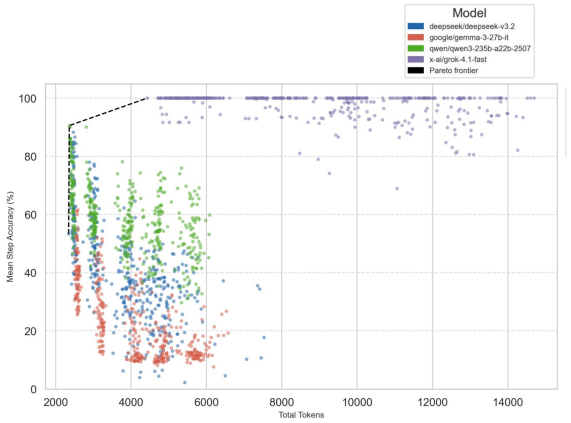

Figure 4: Token Efficiency Plot

Figure 4. Token Efficiency Plot: grok achieves near-perfect SA at 2,000–4,000 total tokens and maintains it up to 14,000 tokens, while deepseek and gemma scatter widely between 10–62% SA across the same token range with no clear accuracy-token correlation.

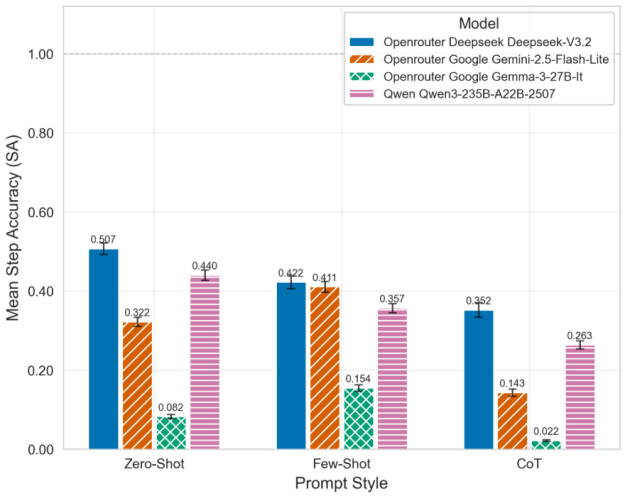

Figure 5: Prompt Sensitivity Plot

Figure 5. Mean Step Accuracy by Prompting Strategy. Performance varies across prompting methods. DeepSeek-V3.2 and Qwen3-235B perform best under zero-shot prompting (peak 0.507), while Gemini-2.5-Flash-Lite benefits from few-shot prompting (0.411). Chain-of-thought prompting yields the lowest accuracy across all models.

5.1. Instance Generation

Problem instances are generated by a synthetic process generator with five complexity levels per algorithm. Complexity is controlled by multiple independent axes: process count (3 to 14), arrival variance (0 to 18), burst range ([1,8] to [4,36]), tie-injection probability (0% to 65%), and algorithm-specific configurations.

Complexity Level 1 represents fundamental correctness checks, featuring 3 processes with zero arrival variance and simple queue management. Complexity Level 5 introduces compound reasoning challenges under high process counts and complex event interleaving (e.g., 14 processes with frequent tie-breaking for SJF, or 4 queues with periodic priority boosts and small time quantums for MLFQ).

5.2. Ground Truth and Scoring

Ground-truth traces are produced by deterministic Python reference solvers for each algorithm. Each solver emits a complete step-level execution trace, generating an object for every event trigger (arrival, dispatch, preemption, completion, or idle transition). Each state records the simulation time, event type, running CPU process and its remaining burst, the ordered ready queue across all levels, completed PIDs, and whether a tie-breaking rule was applied.

Scoring proceeds by aligning the model's parsed output steps with the ground truth using step indices, then comparing each field independently. Step-level correctness requires an exact match across all six correctness-criterion fields. Errors are classified into a robust taxonomy of 22 named failure modes, encompassing generic state corruption (e.g., wrong clock time, queue contents) and algorithm-specific vulnerabilities (e.g., `SJF_FCFS_FALLBACK`, `RR_ARRIVAL_ORDER_ERROR`, `MLFQ_WRONG_DEMOTION`). Performance is quantified using Step Accuracy (SA), Conditional Step Accuracy (CSA), First Error Position (FEP), and Error Cascade Rate (ECR).

5.3. Results

Figures 5–8 show mean step accuracy by prompting strategy, step accuracy vs. complexity level, error cascade rate, and error type distribution across frontier models.

6. Disk Scheduling: Algorithms, Instances, and Results

We now turn to disk scheduling, the third major component of SVAC. Five algorithms are evaluated: FCFS (First Come First Serve), SSTF (Shortest Seek Time First), SCAN, C-SCAN (Circular SCAN), and LOOK. These algorithms span a spectrum from trivially stateless (FCFS services requests in arrival order, requiring no auxiliary state beyond the queue) to directionally-aware and globally-sensitive (LOOK requires the model to track current head position, pending requests, and sweep direction simultaneously), and each demands qualitatively different spatial reasoning over a one-dimensional cylinder space.

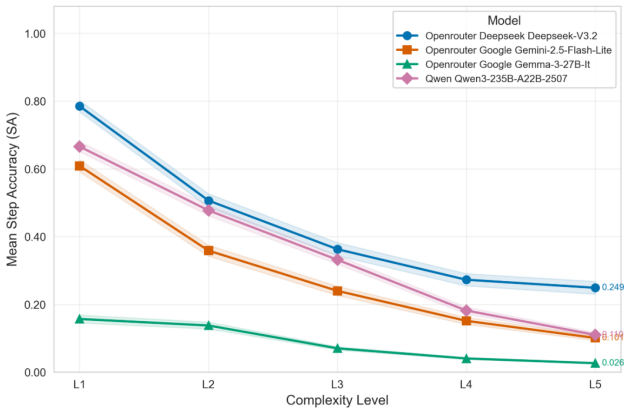

Figure 6: Scaling Law Plot

Figure 6. Step Accuracy vs. Complexity Level. Step Accuracy declines near-linearly with increasing complexity. DeepSeek-V3.2 performs best across all levels (0.785 → 0.249), while Qwen3-235B drops more sharply (0.666 → 0.110). Gemma-3-27B performs worst, never exceeding 0.157.

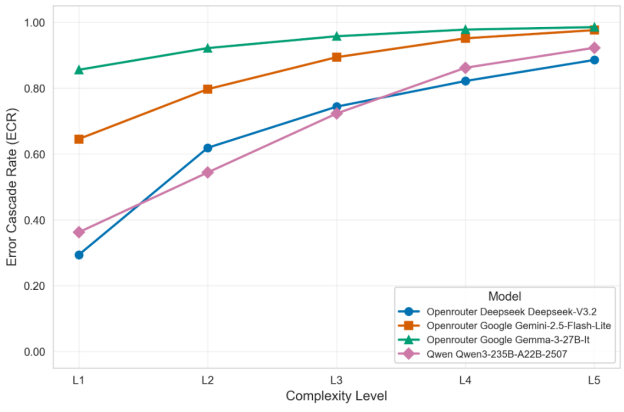

Figure 7: ECR Plot

Figure 7. Error Cascade Rate (ECR) vs. Complexity Level. ECR increases with complexity, highlighting the brittleness of OS algorithm traces to early reasoning errors. All models converge to high ECR values (0.88–0.99) by Level 5. Gemma-3-27B consistently records the highest ECR.

6.1. Instance Generation

Problem instances are generated by a synthetic disk-request generator with five complexity levels and six locality modes. Complexity is controlled by two coupled parameters: disk size (100 to 300 cylinders) and request queue length (8 to 30 requests). Complexity Level 1 uses a 100-cylinder disk with 8 pending requests; Level 5 uses a 300-cylinder disk with 30 requests, producing traces of up to 30 JSON step objects. Each instance also carries an initial head position and an initial sweep direction, both drawn uniformly at random, which are required inputs for the directional algorithms.

Locality modes control the statistical structure of the request queue. **Uniform random** queues are drawn independently from the full cylinder range, providing a baseline with no spatial structure. **Clustered** queues place requests around two to five randomly chosen centres with bounded spread, modelling workloads with spatially local I/O bursts. Hotspot queues concentrate the majority of requests near a single high-traffic region while scattering the remainder uniformly, reflecting a single hot database region. Temporal queues maintain a sliding working set with configurable reuse probability, modelling sequential-scan workloads with occasional cold seeks. Zipfian queues follow a power-law distribution over cylinder addresses, with a small number of frequently-accessed cylinders dominating the queue. Adversarial queues are constructed per-algorithm to maximise seek distance: FCFS instances alternate between near-zero and near-maximum cylinders to force large oscillations; SSTF instances cluster most requests at one end and hide two far-outlier cylinders, exposing the greedy algorithm's susceptibility to starvation; SCAN instances interleave low and high requests to force repeated full-sweep traversals.

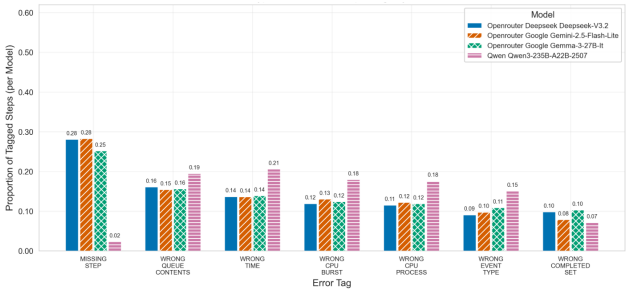

Figure 8: Error Type Distribution Plot

Figure 8. Error Type Distribution. `MISSING_STEP` dominates for DeepSeek, Gemini, and Gemma (25–28%), indicating incomplete traces. Qwen3-235B is an outlier with minimal truncation (2%) but elevated state errors, including `WRONG_TIME` (21%) and `WRONG_QUEUE_CONTENTS` (19%).

6.2. Ground Truth and Scoring

Ground-truth traces are produced by reference simulators for each algorithm. Each simulator produces a complete step sequence containing the current head position, the next request serviced, the seek distance for that step, the cumulative seek distance, and the remaining unserviced queue. FCFS services requests in arrival order. SSTF greedily selects the nearest pending request at each step. SCAN sweeps to the disk boundary before reversing, whereas LOOK reverses at the highest (or lowest) pending request rather than the physical boundary — a distinction that is the primary source of model confusion. C-SCAN sweeps only in one direction and wraps to cylinder 0 upon reaching the maximum, requiring models to recognise that no requests are serviced during the return sweep.

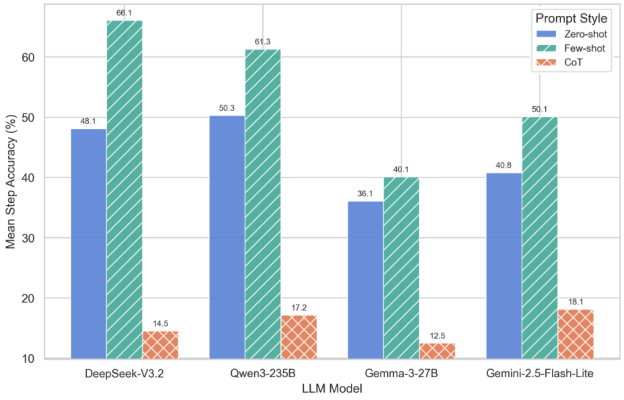

Figure 9: Model Comparison Plot

Figure 9. Model Comparison Plot: few-shot prompting dominates across all models, while CoT collapses below zero-shot for every model — the prompt-style gap exceeds any inter-model gap. Gemma-3-27B trails in every prompting regime.

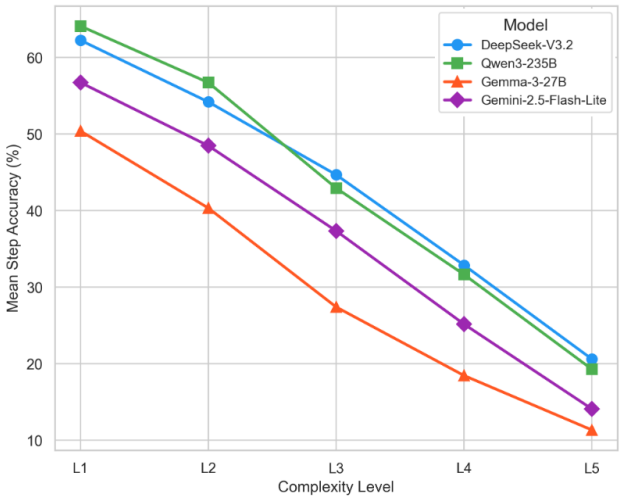

Figure 10: Scaling Law Plot

Figure 10. Scaling Law Plot: all models degrade steeply and continuously from L1 to L5 with no plateau. DeepSeek-V3.2 and Qwen3-235B track closely throughout, while Gemma-3-27B collapses earliest and Gemini-2.5-Flash-Lite converges toward Gemma by L5.

Scoring aligns the model's output steps with ground truth by step index and independently compares each field. A step is considered correct only when both the current head position and the chosen next request match ground truth exactly, enforcing joint head-tracking and selection accuracy. Five error types are tracked: servicing a cylinder not in the remaining queue (**not-in-remaining**), re-servicing an already-completed request (**already-served**), making a suboptimal greedy choice (**suboptimal-choice**), breaking ties incorrectly among equidistant requests (**tiebreak-error**), and violating the current sweep direction for SCAN, C-SCAN, or LOOK (**direction-violation**). Three summary metrics are computed per instance: Step Accuracy (SA), the proportion of steps where both head and choice are correct; First Error Position (FEP), the normalised index of the first incorrect step; and Error Cascade Rate (ECR), the proportion of post-error steps in which the head position is also corrupted.

6.3. Results

Figures 9–12 show the model comparison across prompt styles, the degradation cliff, prompt sensitivity, and the error cascade rate across complexity levels.

7. Discussion

Across all three algorithm families, SVAC reveals that LLM reasoning quality degrades with sequence length and state complexity, with error cascades dominating performance at higher complexity levels. This is not merely a token-length issue — even models with large context windows fail to maintain correct intermediate state when the number of steps exceeds moderate thresholds. However, the shape of degradation differs across domains: page replacement exhibits a sharp capability cliff between Levels 2 and 3, whereas CPU and disk scheduling degrade more continuously, with no model sustaining meaningful trace fidelity at high complexity in either domain. This distinction suggests that working-set management admits a threshold capability, whereas stateful scheduling and seek-sequence tracing place demands that scale unfavorably with instance complexity for all evaluated models.

The choice of OS algorithm family matters substantially. Page replacement places demands on ordered data structure maintenance and eviction logic, with OPT being the most uniformly difficult algorithm due to its global lookahead requirement. CPU scheduling requires simultaneous tracking of multiple process states, arrival events, and time-quantum boundaries, with MLFQ introducing additional complexity through multi-level demotion and periodic priority boosts. Disk scheduling demands geometric reasoning over a one-dimensional address space with directional constraints, where LOOK is the hardest algorithm due to models consistently conflating its reversal condition with SCAN's boundary-seeking behaviour. Models that perform well on one family do not automatically generalise to others, confirming that procedural reasoning is at least partially domain-specific within the OS context.

The effect of prompting strategy is domain-dependent and, in two out of three families, counterintuitive. For page replacement, chain-of-thought prompting provides modest

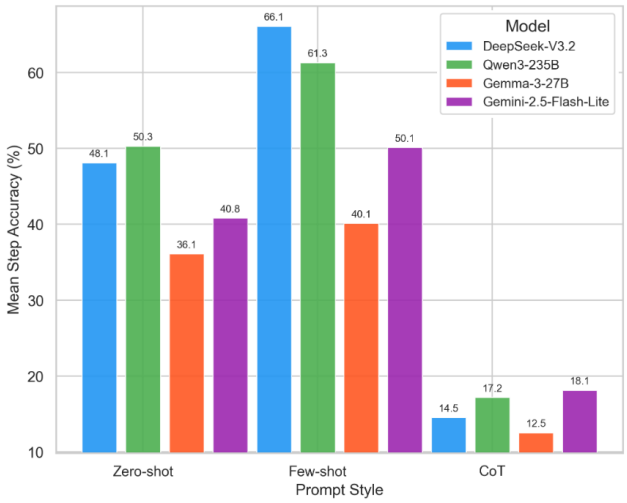

Figure 11: Prompt Sensitivity Plot

Figure 11. Prompt Sensitivity Plot: model rankings are stable across zero-shot and few-shot but collapse into a near-indistinguishable low-accuracy band under CoT, where between-model variance nearly disappears.

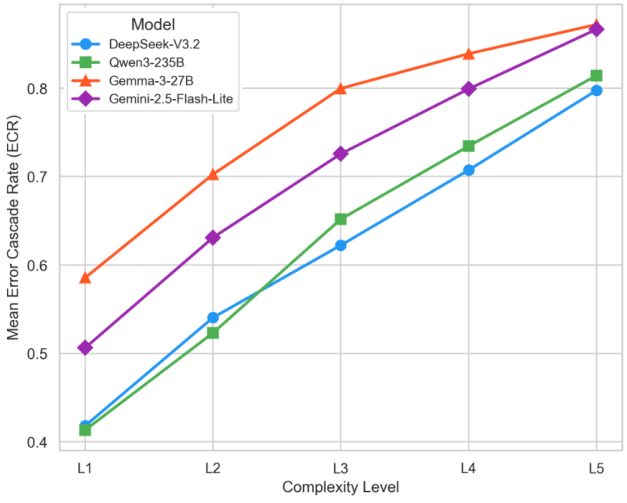

Figure 12: ECR Plot

Figure 12. Error Cascade Rate Plot: ECR is high for all models even at L1 — no model begins in a low-cascade regime — and all four rise steeply through L5. Gemma-3-27B and Gemini-2.5-Flash-Lite converge near the ceiling at L5.

gains on algorithms with complex eviction decisions (OPT, LRU) by allowing the model to reason through lookahead or recency before committing to state values. However, for both CPU scheduling and disk scheduling, CoT is actively harmful: it yields the lowest step accuracy of all three prompting regimes across every model, falling well below zero-shot baselines. Extended deliberation appears to introduce spurious direction reversals, incorrect queue reorderings, and over-ridden greedy decisions — suggesting that chain-of-thought reasoning interferes with the procedural discipline required for correct state propagation in these domains. Few-shot exemplars, by contrast, provide the most reliable performance signal across all three families, anchoring the model to the correct output schema and decision logic without the over-reasoning pathology induced by CoT.

8. Conclusion

We presented SVAC, a benchmark for evaluating LLM procedural reasoning on OS resource-management algorithms via step-level trace verification. By requiring models to produce complete, no-code execution traces in structured JSON format across three algorithm families — page replacement, CPU scheduling, and disk scheduling — SVAC exposes reasoning failures that final-answer benchmarks miss. Our evaluation reveals degradation cliffs and high error cascade rates in page replacement; continuous accuracy collapse and scheduling-state mismanagement in CPU scheduling; and seek-sequence breakdown with structurally unavoidable cascades in disk scheduling. A cross-domain finding of particular practical significance is that chain-of-thought prompting, widely assumed to benefit reasoning-intensive tasks, is actively detrimental for two of the three domains evaluated. The results underline both the promise and the limits of current LLMs as general-purpose procedural reasoners, and we hope SVAC provides a useful testbed for future work on improving algorithmic trace fidelity in foundation models.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning by providing a structured benchmark for evaluating procedural reasoning capabilities in large language models. The algorithms studied — page replacement, CPU process scheduling, and disk scheduling — are well-established computer science topics with fully deterministic, publicly documented behaviour. SVAC evaluates these algorithms using synthetically generated instances across controlled complexity levels and locality modes, with no human subject data collected or used. The benchmark is intended to support research into the limits of LLM algorithmic reasoning, and no direct negative societal consequences are anticipated from this work.

References

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*,

2021.

Fujisawa, I., Nobe, S., Seto, H., Onda, R., Uchida, Y., Ikoma, H., Chien, P.-C., and Kanai, R. ProcBench: Benchmark for multi-step reasoning and following procedure. 2024. doi: 10.48550/ARXIV.2410.03117. URL https://arxiv.org/abs/2410.03117.

Gu, A., Roziere, B., Leather, H., Solar-Lezama, A., Synnaeve, G., and Wang, S. I. CRUXEval: A benchmark for code reasoning, understanding and execution. *arXiv preprint arXiv:2401.03065*, 2024.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the MATH dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Li, S., Jiang, J., Zhao, T., and Shen, J. OSVBench: Benchmarking LLMs on specification generation tasks for operating system verification. 2025. doi: 10.48550/ARXIV.2504.20964. URL https://arxiv.org/abs/2504.20964.

Liu, H., Ning, R., Teng, Z., Liu, J., Zhou, Q., and Zhang, Y. Evaluating the logical reasoning ability of ChatGPT and GPT-4. *arXiv preprint arXiv:2304.03439*, 2023.

O'Malley, D., Bhattarai, M., Ranasinghe, N. R., Draayer, E., and Santos, J. Benchmarking large language models with integer sequence generation tasks. 2024. doi: 10.48550/ARXIV.2411.04372. URL https://arxiv.org/abs/2411.04372.

Pandit, S., Xu, A., Nguyen, X.-P., Ming, Y., Xiong, C., and Joty, S. Hard2verify: A step-level verification benchmark for open-ended frontier math. *ArXiv*, abs/2510.13744, 2025. doi: 10.48550/arxiv.2510.13744.

Srivastava, A. et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*, 2022.

Sun, H., Yu, K., Wang, Y., Liu, B., Li, X., Li, R.-H., Chen, N., and Li, J. AlgBench: To what extent do large reasoning models understand algorithms? 2026. doi: 10.48550/ARXIV.2601.04996. URL https://arxiv.org/abs/2601.04996.

Sun, S., Hsieh, C.-P., Ladhak, F., Arakelyan, E., Serano, S. A., and Ginsburg, B. L0-reasoning bench: Evaluating procedural correctness in language models via simple program execution. 2025. doi: 10.48550/ARXIV.2503.22832. URL https://arxiv.org/abs/2503.22832.

Yang, S., Zhao, B., and Xie, C. AQA-bench: An interactive benchmark for evaluating LLMs' sequential reasoning ability. 2024. doi: 10.48550/ARXIV.2402.09404. URL https://arxiv.org/abs/2402.09404.