SCE-BENCH: FROM SYMPTOMS TO CAUSAL EXECUTIONS OF CONCURRENCY BUGS

Anonymous authors
Paper under double-blind review

ABSTRACT

Concurrency bugs are hard to diagnose and understand: symptoms may not reveal their causes, and failures may depend on nondeterministic schedules. Yet existing LLM benchmarks either omit concurrency bugs or lack deterministic evaluation, limiting assessment of models' reasoning. We present SCE-Bench, an execution-grounded benchmark that tests whether models can turn symptom descriptions into machine-checkable causal executions. Each task tests this reasoning through four stages: diagnose whether a symptom is concurrency-induced, identify the mechanism by which concurrent interactions cause the failure, construct a workload, and submit an execution specification. The grader then runs each execution specification on buggy and repaired source revisions to check for the claimed failure and its absence after repair. SCE-Bench contains 66 positive tasks from disclosed kernel and userspace bugs across 12 mechanism families. Its 86 negative tasks comprise 66 repaired-source counterparts and 20 independent non-concurrency cases to test whether models distinguish actual concurrency bugs from false alarms. At pass@3, GPT-5.5 solved 9 of 66 positive tasks (13.6%), while only 10 were solved across all models. Particularly, they struggled most with the final stage of specifying an execution to trigger the bug. For the negative tasks, every model was less accurate on repaired-source controls than on independent non-concurrency cases.

1 INTRODUCTION

Concurrency bugs are difficult to diagnose and reproduce because program inputs alone do not determine whether a failure occurs. The outcome may depend on rare thread interleavings, object lifetimes, or memory reordering (Jeong et al., 2024). Kernel bugs may additionally require interrupts or hardware unavailable in common test environments (Gong et al., 2026). Repeated execution alone cannot reliably expose all concurrency bugs. Diagnosis therefore requires reasoning across execution contexts.

Security benchmarks emphasize reproduction and exploitation. CyberGym and SEC-bench evaluate vulnerability reproduction using sanitizer oracles (Wang et al., 2026b; Lee et al., 2025). ExploitGym includes concurrency vulnerabilities but acknowledges that some may be unexploitable and uses an agent judge for target-bug attribution (Wang et al., 2026a). SEC-bench Pro reruns PoCs to detect crashes and uses an LLM judge for attribution (Lee et al., 2026). For concurrency cases, these two benchmarks rely on probabilistic execution outcomes and LLM-based attribution.

These benchmarks' reproduction and exploitation tasks start from actual vulnerabilities, so a model can always assume a bug exists and receive credit for triggering it. Success therefore does not establish whether the model can distinguish a supported diagnosis from a false alarm. This gap is especially important for concurrency symptoms that lack direct runtime evidence comparable to an AddressSanitizer report for a memory-safety violation. Diagnosing such symptoms requires reasoning about interacting actors and execution conditions to determine whether the supplied source supports a concurrency failure. Evaluating concurrency reasoning must therefore test both whether a diagnosis is warranted and whether the model can precisely specify the execution that produces the failure.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Existing work evaluates LLMs on concurrent code generation with CONCUR (Huang et al., 2026) and on fault localization and explanation with ConFL (Shao et al., 2026). Neither tests the full reasoning sequence targeted by SCE-Bench: from symptom diagnosis to a workload and execution that demonstrate the bug's mechanism.

Advances in concurrency testing have made nondeterministic execution increasingly controllable. CONCURRIT expresses scheduling constraints to reproduce concurrency bugs (Elmas et al., 2013), while Ozz controls both thread interleavings and memory reordering (Jeong et al., 2024). These mechanisms provide a foundation for validating a model's concurrency explanation through controlled execution.

We present SCE-Bench (**S**ymptom–**C**oncurrency–**E**xecution), a benchmark that assesses whether an LLM can identify a concurrency bug from a symptom and precisely describe how it occurs, with the evaluator validating that explanation through execution. SCE-Bench organizes concurrency reasoning into four stages progressing from high-level understanding to low-level execution details: can the model identify whether the symptom indicates a concurrency bug (T0), what specific type of concurrency bug is it (T1), under what workload can it occur (T2), and how must that workload execute to trigger it (T3)? As the model progresses through the subtasks, it receives an answer interface that defines a bounded answer space. These interfaces enable automatic grading without an LLM judge, even for workload definitions and execution conditions.

SCE-Bench contains 66 concurrency bugs across 12 mechanism families: 35 kernel and 31 userspace cases. We also collected 20 independent negative cases covering sequential defects and external/-configuration failures. Each of the 66 bugs provides an additional negative case, pairing its original symptom with the repaired source. These repaired-source controls test whether the model recognizes that the supplied revision does not support the reported behavior, rather than inferring a concurrency bug from the symptom alone.

We evaluate seven models in three repetitions on 152 tasks. GPT-5.5 achieves the highest observed end-to-end success rate among scored positive episodes (13.1%); across all models, only 10 of 66 positive tasks are solved end to end. Every model has its largest proportional stage loss at T3 and classifies independent negatives more accurately than repaired-source controls. These results expose gaps in both judging whether the source supports a diagnosis and specifying a workload and execution that demonstrate the bug's mechanism.

In summary, this paper makes the following contributions:

- **A benchmark of concurrency reasoning.** We develop SCE-Bench to assess whether LLMs can identify when a symptom warrants a concurrency diagnosis and precisely describe the workload and execution that produce the reported failure, using 66 kernel and userspace bugs across 12 mechanism families. By pairing each symptom with affected and repaired source and including 20 independent negatives, the benchmark requires models to distinguish supported diagnoses from indiscriminate bug predictions.
- **Grading concurrency bugs without an LLM as a judge.** We develop an execution-based grader that binds model-selected source events to workload actors and shared objects, checks execution conformance before evaluating harm, and uses the repaired revision as a causal control.
- **An empirical study of LLM concurrency reasoning.** We evaluate seven models over 3,192 episodes and identify weaknesses in their concurrency reasoning, including distinguishing affected from repaired source and precisely specifying the workloads and execution conditions under which a concurrency bug occurs.

2 Related Work

LLM security benchmarks. Security benchmarks assess vulnerability detection, reproduction, and exploitation (Ding et al., 2025; Zhang et al., 2025b; Zhu et al., 2025; Zhang et al., 2025a). CyberGym and SEC-bench evaluate reproduction from vulnerability reports using sanitizer oracles (Wang et al., 2026b; Lee et al., 2025), while ExploitBench provides deterministic exploitation oracles (Lee & Brumley, 2026). ExploitGym and SEC-bench Pro include concurrency vulnerabilities and use model-based judges for target-bug attribution; the former acknowledges potentially unexploitable

tasks, while the latter retries PoCs to detect crashes (Wang et al., 2026a; Lee et al., 2026). To grade a model-generated reproducer or test suite, the evaluators in CyberGym and SWT-Bench run it on both revisions, checking that it exposes a failure before the fix but not afterward (Wang et al., 2026b; Mündler et al., 2024). SCE-Bench additionally uses repaired source in separate diagnostic tasks: given the same symptom, the model must identify a concurrency bug in affected source and reject that explanation in repaired source. These controls test whether diagnoses are supported by the source and penalize indiscriminate bug predictions.

LLM concurrency benchmarks and analysis. Prior work evaluates concurrent code generation (Huang et al., 2026), fault localization and explanation (Shao et al., 2026), repair (Behrang et al., 2025; Vikram et al., 2026), and test generation (Cao et al., 2026). SCE-Bench assesses the reasoning sequence from symptom diagnosis to mechanism, workload, and execution conditions. Progressively precise answer contracts make these decisions gradable through categorical checks, structural rules, and execution, without an LLM judge.

Concurrency reproduction and execution control. Failure-reproduction systems reconstruct executions from crash evidence or recorded paths (Zamfir & Candea, 2010; Bianchi et al., 2017; Huang et al., 2013). Scheduling languages and controlled testing make executions reproducible (Jagannath et al., 2011; Elmas et al., 2013; Li et al., 2025); Ozz additionally controls memory visibility (Jeong et al., 2024). Concurrent witnesses support independent validation (Beyer & Friedberger, 2020; Ponce-de-León et al., 2022). SCE-Bench uses these execution controls to test the model's submitted execution specification. The evaluator executes the certificate and checks that the resulting execution satisfies its source-event, actor, shared-object, and ordering constraints before checking for the claimed harm.

Concurrency bug corpora. RADBench, JaConTeBe, and GoBench collect real concurrency bugs (Jalbert et al., 2011; Lin et al., 2015; Yuan et al., 2021). RaceBench injects observable bugs into real applications (Liang et al., 2023), while kernel studies document interrupt and hardware dependencies (Gong et al., 2026). SCE-Bench turns kernel and userspace bugs into gradable model tasks by adding symptoms, repaired-source controls, bounded workload interfaces, and executable grading of the model's causal explanation.

3 SCE-BENCH

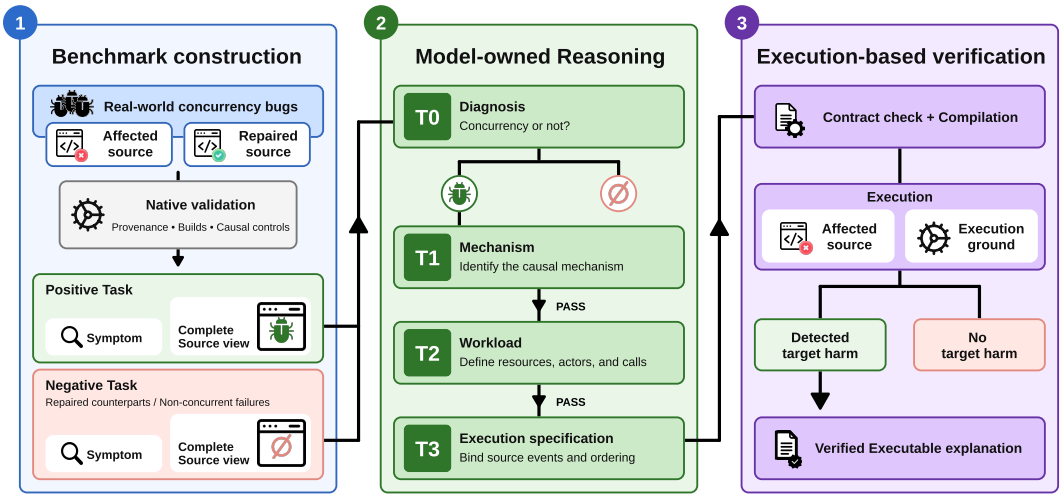


Figure 1: Overview of the SCE-Bench framework.

SCE-Bench assesses concurrency reasoning from high-level symptom diagnosis to a precise executable explanation (Figure 1). Its four stages ask whether a concurrency bug explains the symptom (T0), which type of concurrency bug is responsible (T1), under what workload that mechanism can occur (T2), and how that workload must execute to trigger the bug (T3).

Initially, the page is backed by slot s, holding value v_0 in allocation lifetime g_0.

(a) Reader retaining old data

```
s = backing_id(page)
old = read_backing(s)  # v0 from g0

# The concurrent path can run here.

with mapping_lock(page):
    if backing_id(page) == s:  # s == s
        install(page, old)  # stale v0
```

(b) Concurrent update and reuse

```
make_resident_and_free_slot(page)
write(page, v1)  # complete update
new_id = save_to_backing(page)
# Slot reuse can give new_id == s.
# backing_id(page) = new_id = s
# Now: lifetime g1, stored value v1.
```

Observation: the page yields v_0, although the completed update requires $v_1 \neq v_0$.

Figure 2: Pseudocode abstraction of the K28 swap-slot ABA failure. Each operation summarizes source behavior; the right-hand path groups concurrent activity in the native workload. Slot s can be reused between the reader's load and validation: the identity check passes although the allocation lifetime has changed from g_0 to g_1. The repair protects that lifetime across the read and validation.

Positive cases and negative source controls test whether the initial concurrency diagnosis is warranted. Each stage has an explicit answer contract, graded by categorical checks, workload rules, or execution without an LLM judge. The final stage compiles and executes the model's specification, checking that the observed execution conforms to its constraints and produces the target harm.

3.1 Task Formulation

The four stages progress from diagnosing a symptom to specifying an execution that demonstrates the bug. Each stage isolates one reasoning decision and provides a bounded answer interface with explicit acceptance conditions. Accepted answers remain fixed as later stages supply the information needed for more precise decisions. This structure allows diagnosis, mechanism, workload, and execution to be graded separately, without an LLM judge. Each stage's contract is designed to admit all correct answers within its declared scope.

The stages define the following questions, supplied information, and required answers:

- **T0: diagnosis.** *Does the supplied source support a concurrency diagnosis for the symptom?*
  Given: symptom description, environment context, and source snapshot.
  Answer: one of four diagnoses—concurrency/synchronization defect, sequential defect, external/configuration failure, or behavior unsupported by the source.
- **T1: mechanism.** *What specific type of concurrency bug is responsible?*
  Given: the corpus-wide mechanism taxonomy.
  Answer: the mechanism and its broad class.
- **T2: workload construction.** *Under what workload can this mechanism occur?*
  Given: a catalog of workload operations and their argument domains.
  Answer: shared resources, setup calls, and actors with assigned calls.
- **T3: executable causal certificate.** *How must that workload execute to trigger the bug?*
  Given: the certificate grammar, structural bounds, and harm-oracle interface.
  Answer: a certificate preserving the accepted workload and specifying source anchors, actor/event/object bindings, execution constraints, and the claimed harm.

We illustrate all four stages with kernel case K28, the Linux swap-slot data-loss bug CVE-2024-26759 (NIST, 2024). A swap slot holds an evicted memory page. A reader can retain old page contents while concurrent activity updates the page and reuses the slot for the new contents. The unchanged slot number passes validation, allowing the reader's old copy to replace the completed update (Figure 2).

T0: diagnosis. The model receives a symptom description, environment context, and read-only access to a complete frozen source snapshot. It selects one of four diagnoses: a concurrency/synchronization defect, a sequential defect, an external/configuration failure, or behavior unsupported by the supplied source. A positive task advances to T1 only when the model selects concurrency/synchronization

defect; otherwise it ends as a T0 failure. A negative task always ends at T0, passing only if the expected diagnosis is selected: source-unsupported for repaired-source controls, or the corresponding sequential or external/configuration category for independent controls.

T1: mechanism. After T0 passes, the model receives definitions of the 12 mechanism families listed in Table 1 in §3.3 below. These definitions are identical across tasks, with specialized mechanisms taking precedence over generic conflicting access. To advance to T2, the model must select the correct mechanism family and report its associated broad class. For K28, the required family is ABA/stale reference: the same slot identifier s can refer to different allocation lifetimes (g_0 and g_1 in Figure 2). Equality of identifiers therefore does not establish that a retained value still belongs to the current backing allocation. T0 and T1 use categorical adjudication; their accompanying rationales are recorded but not independently scored.

T2: workload construction. After T1 passes, the model receives a case-specific catalog of three kinds of operations and their argument domains: resource constructors, setup calls, and actor calls. Resource constructors create the objects used by the workload. Setup calls configure those objects before the actors run. Actor calls specify what each actor does; an actor is an independently executing participant, such as a thread. The model chooses which resources to create, which setup calls to make, how many actors are needed, and which calls and arguments to assign to each actor. This defines the workload without specifying how the actors interleave; T3 constrains the ordering of selected events across its actors. The catalog and argument bounds define a workload space in which every submission can be checked against explicit acceptance rules (§3.2).

For K28, T2 provides a bounded vocabulary: the constructors `bounded_swap_area()` and `initialized_page(swap_area)`, the actor operations `retain_read_until_slot_reuse(page)`, `fault_update_and_pageout(page, swap_area)`, and `fault_reused_slot(page, swap_area)`, and no setup operations. These operations may span multiple Linux functions. An accepted workload creates one swap area and one anonymous page on it, and uses each actor operation exactly once. Additional or repeated calls are unnecessary to demonstrate the bug and therefore fall outside the case's workload contract. The operations serve three roles:

- The *stalled fault* reads the page's old value v_0 from swap and retains it while paused before installation.
- The *completing fault* faults in the same page, writes v_1, and pages it out, allowing the swap slot to be reused for the updated contents.
- The *reuse fault* faults on the updated page and confirms reuse of the same numeric slot.

The reuse fault may run on a third actor or, in the two-actor profile, after pageout on the completing actor. It cannot follow the retained-read call on the stalled actor: that call waits for slot reuse and therefore cannot finish before the later call starts. The stalled actor corresponds to Figure 2(a); the completing and reuse operations realize the activity grouped in Figure 2(b). Resource and actor names and declaration order are immaterial, while calls within an actor execute in order. The grader accepts workloads according to the case's contract; §3.2 describes these acceptance rules.

T3: executable causal certificate. After T2 passes, the model receives the `ExecSpec` grammar, its accepted workload, the harm-oracle signature, and any case-specific limits on the certificate's structure. It submits an `ExecSpec` program specifying how that workload must execute to produce the claimed harm. The program preserves the workload unchanged and binds model-selected source events to its actors and shared objects, with execution constraints and a final harm observation. The interface defines the specifications that can be compiled and checked through execution (§3.2). The same specification is also checked against the repaired revision as a causal control.

For K28, the prompt requires two source anchors, four events, one swap-slot object, one same-object group, and five order terms including the observation. The certificate preserves the accepted two- or three-actor workload; in the two-actor profile, the completing and reuse roles share an actor. A passing answer selects the swap-entry lookup and direct page read inside `do_swap_page()`, binds their runtime hits to the accepted T2 actors, and relates the events through the same numeric swap slot. It orders the stalled fault's entry lookup and old-data read before the completing fault's entry lookup, then the reuse fault's entry lookup, and finally the observation of v_0 where the completed

update requires v_1. The recorded certificate conforms on both revisions: the affected kernel restores the stale value v_0, whereas the repaired kernel preserves the completed update v_1.

3.2 Workload Contracts and Execution Grading

Workload contracts. The workload interface deliberately bounds the answer space for which the grader must define acceptance rules. Within this space, the rules are designed to accept every workload satisfying the case's requirements—including permitted alternatives in actor assignment and resource configuration—and reject those that violate them. Every submission is thus checked by explicit rules, without an LLM judge. Correctness is scoped to the disclosed workload contract, rather than arbitrary programs that might reproduce the bug.

A generic validator checks submission format and interface bounds. Case-specific rules then check resource relationships, actor assignments, call sequences, and argument values against the workload contract. These rules admit permitted alternatives, such as K28's two- and three-actor profiles, while excluding unnecessary operations. Consistent renaming and reordering of resource and actor declarations do not affect acceptance; argument bindings and call order remain significant.

ExecSpec and compilation. The `ExecSpec` interface defines the source events, object relations, and execution controls that a certificate can express. The compiler is designed to translate every specification valid within this interface into an executable check of the model's hypothesis. It resolves the submitted source anchors, checks actor/object bindings, and uses a case-specific adapter to generate instrumentation and backend controls. The compiler preserves the model's submitted constraints without adding missing parts of the execution hypothesis. Every successfully compiled certificate is run by the case's execution backend to determine whether the specified execution can be realized and whether it produces the claimed harm. Compilation and grading are applied separately to affected and repaired source.

Execution backends. We use three backend families. *Linux VMs* boot an instrumented kernel under QEMU/KVM; a generated driver runs the workload, while a kernel controller enforces the submitted event constraints. *Ozz VMs* translate source anchors and supported visibility relations into instruction-level scheduling and memory-visibility controls (Jeong et al., 2024). *Native runtimes* execute instrumented C/C++ code through POSIX, or Go, .NET, and Python code through their respective runtimes, using source hooks and runtime-specific controllers. This family also includes the FreeRTOS POSIX port and bounded extractions of Linux production code. Across backends, execution records must establish the specified source events, actor and object identities, dynamic occurrences, and required relations before the case-specific harm oracle is evaluated. Oracles check incorrect values, wait cycles, or matching sanitizer/runtime diagnostics; a crash alone does not establish the specified harm.

3.3 Corpus

We collected 71 concurrency bugs; five could not be included in the positive evaluation for the reasons discussed in §5. The remaining 66 bugs span 12 mechanism families (Table 1): 31 userspace and 35 kernel bugs (34 Linux and one FreeRTOS). For each bug, we retain the available bug reports, affected source commits, fix commits, patches, and reproducers, together with provenance and date metadata. Appendix A provides per-bug details.

The recorded fixes span October 2013–September 2026; 36 of 66 (54.5%) were committed in 2026. These are fixing-commit dates (Figure 3); Appendix A lists per-case provenance, including verified CVE identifiers or dated public records where available.

We also collected 20 independent negative cases: 10 sequential defects and 10 external/configuration failures, with five kernel and five userspace cases in each category. These cases test whether the model distinguishes concurrency bugs from failures with independently established alternative causes.

4 Evaluation

We evaluate SCE-Bench using a flagship and a lightweight model from each of OpenAI (GPT-5.5 and GPT-5.4 Mini), Anthropic (Opus 4.6 and Sonnet 4.6), and Google (Gemini 3.1 Pro and Gemini 3.1

Table 1: Positive cases by mechanism.

Mechanism family	Userspace	Kernel
ABA/stale reference	2	2
Duplicate release	2	2
Lifetime overlap	2	12
Lock-cycle deadlock	2	3
Lost update	2	4
Missed notification	2	2
Recursive-lock deadlock	2	2
Scheduler livelock	2	2
Shared intermediate-state interference	5	2
Stale snapshot/check-then-use	2	2
Unsynchronized conflicting access	5	0
Visibility/publication failure	3	2
Total	31	35

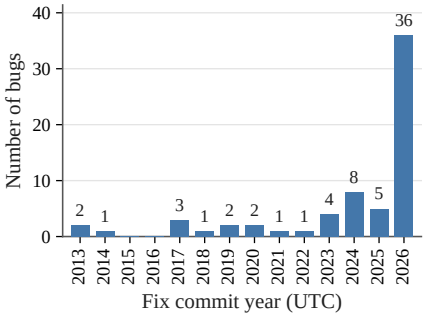


Figure 3: Fix years for the 66 positive bugs, using UTC committer dates.

Flash-Lite), plus the open-weight GLM-5.3. We run 3 independent repetitions on 152 tasks (66 positive and 86 negative), yielding 3,192 model–task episodes at a total LLM cost of $9,047.66.

4.1 Experimental Setup

We selected the most recent versions available through our endpoints that ran reliably during compatibility screening without refusals triggered by provider security safeguards. We completed each provider's security-access process for this evaluation. Appendix B gives model release dates, documented cutoffs, and inference settings.

All models use the same agent loop and prompts. The agent has read-only tools to browse and read the frozen source snapshot. Each episode starts with a fresh conversation and retains its message history across stages T0–T3. The agent submits each stage answer as JSON.

Within each repetition, an episode interrupted by a safeguard-related refusal or an infrastructure failure receives up to three attempts in total. Episodes still unresolved after the third attempt are reported as refusal or infrastructure outcomes.

Each episode shares a budget of 100 actions across stages T0–T3. Every source-tool call and submitted answer costs one action, including corrections and individual calls within a batch. Completions are limited to 16,384 tokens per request. Syntax, schema, or disclosed interface violations permit up to three corrections per stage using public validator feedback. Corrections address formatting and interface violations only; incorrect diagnoses, mechanisms, workloads, or execution hypotheses receive no retry.

4.2 Performance

End-to-end performance. Among scored positive episodes, GPT-5.5 achieves the highest observed end-to-end success rate, 13.1% (26/198), at an average cost of $2.12 and 26.8 model turns per episode. Gemini 3.1 Flash has the lowest rate, 0.0% (0/192), averaging $0.06 and 17.9 turns (Table 2). Observed pass@3 remains close to the average success rate: GPT-5.5 reaches 13.6% (9/66), compared with 13.1%, while Gemini 3.1 Flash remains at 0.0%. Across all models, only 10 of 66 positive tasks are fully solved; no single model covers this entire set at pass@3. No end-to-end pairwise difference survives Holm correction.

T0 performance. Excluding unresolved episodes, positive T0 accuracy ranges from 36.0% (71/197) for Gemini 3.1 Pro to 84.8% (168/198) for GPT-5.5. Across all negatives, accuracy ranges from 11.9% (30/252) for Gemini 3.1 Pro to 61.9% (135/218) for Opus 4.6. All seven models achieve higher T0 accuracy on independent negatives than on repaired-source controls. Gemini 3.1 Flash has the largest gap, 88.0 percentage points (93.3% versus 5.4%); GPT-5.4 Mini has the smallest, 29.3 percentage points (83.3% versus 54.0%).

Table 2: Counts: mean ± sample SD over three repetitions (66 positive, 86 negative tasks each); parentheses give distinct successes (@3). Success is E2E for positives and T0 for negatives; failures mark the first unpassed stage. Unresolved = infrastructure + refusal. Mean cost and turns (model responses) cover scored episodes only, excluding operational attempts.

Model	Cases	Success (@3)	Failures				Unresolved	Avg. cost (USD)	Avg. turns
			T0	T1	T2	T3			
Opus 4.6	Pos.	2.7±0.6 (4)	15.7±2.3	9.3±3.1	6.0±2.6	19.3±2.5	13.0±1.0	10.20	31.6
	Neg.	45.0±3.6 (55)	27.7±0.6	—	—	—	13.3±3.5	11.91	34.1
Sonnet 4.6	Pos.	3.7±0.6 (4)	19.7±1.5	17.0±1.0	8.7±2.9	17.0±2.0	0.0±0.0	4.94	27.9
	Neg.	38.0±1.0 (47)	48.0±1.0	—	—	—	0.0±0.0	4.26	25.3
GPT-5.5	Pos.	8.7±0.6 (9)	10.0±1.0	12.7±1.5	12.3±2.3	22.3±3.8	0.0±0.0	2.12	26.8
	Neg.	47.3±1.5 (57)	38.7±1.5	—	—	—	0.0±0.0	1.41	21.5
GPT-5.4 Mini	Pos.	1.0±1.0 (2)	35.3±1.5	8.3±1.5	7.0±0.0	13.0±2.0	1.3±1.5	0.23	17.9
	Neg.	52.3±1.5 (67)	33.7±1.5	—	—	—	0.0±0.0	0.14	13.8
Gemini 3.1 Pro	Pos.	3.3±0.6 (4)	42.0±1.0	7.3±0.6	2.7±2.1	10.3±1.2	0.3±0.6	1.49	66.9
	Neg.	10.0±0.0 (11)	74.0±0.0	—	—	—	2.0±0.0	2.34	80.0
Gemini 3.1 Flash	Pos.	0.0±0.0 (0)	15.3±2.1	18.0±2.0	16.3±1.2	14.3±3.1	2.0±2.6	0.06	17.9
	Neg.	22.0±2.6 (25)	60.0±3.6	—	—	—	4.0±2.0	0.03	17.6
GLM-5.3	Pos.	4.3±1.2 (5)	40.3±1.5	4.7±1.5	2.7±0.6	14.0±1.7	0.0±0.0	0.74	28.9
	Neg.	31.7±1.5 (39)	54.3±1.5	—	—	—	0.0±0.0	0.72	30.8

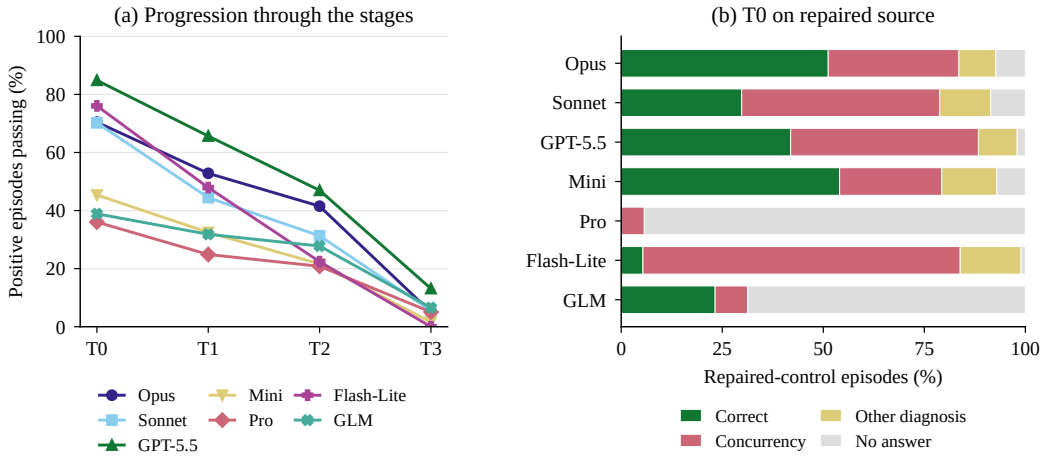


Figure 4: (a) Cumulative positive success. (b) Repaired-control diagnoses: only *Correct* (source-unsupported) passes. Both panels exclude unresolved episodes. Model abbreviations follow Table 2.

T1–T3 performance. Among episodes passing the preceding stage, 71.0% pass T1 (569/801) and 70.7% pass T2 (402/569), but only 17.7% pass T3 (71/402). T3 has the largest proportional loss for every model (Figure 4(a)). Its conditional success rate is highest for GPT-5.5 (28.0%), followed by Gemini Pro (24.4%), GLM-5.3 (23.6%), Sonnet (17.7%), Opus (12.1%), Mini (7.1%), and Gemini Flash (0.0%). GLM-5.3 leads conditional T1 and T2 performance (81.8% and 87.3%), showing that the strongest intermediate-stage performance need not translate into the strongest execution performance. These rates describe the subsets reaching each stage.

Flagship versus lightweight models. Flagship models do not consistently outperform their lightweight counterparts. GPT-5.5 has higher end-to-end success than Mini (13.1% versus 1.5%), as does Gemini Pro relative to Flash (5.1% versus 0.0%), whereas Sonnet slightly exceeds Opus (5.6% versus 5.0%). Negative T0 accuracy reverses the ordering within each pair: Mini exceeds GPT-5.5 (60.9% versus 55.0%), Flash exceeds Pro (26.8% versus 11.9%), and Opus exceeds Sonnet (61.9% versus 44.2%). Lightweight models also have lower average API cost per scored episode on both positives and negatives (Table 2).

Does positive-stage progress predict better negative diagnosis? We match affected and repaired episodes by model, task, and repetition, retaining pairs with both episodes scored. Pooled repaired-control accuracy is 30.4%, 36.1%, 42.6%, and 54.9% when the positive counterpart passes through T0, T1, T2, and T3, respectively. Later-stage success is associated with better repaired-source diagnosis, although these subsets differ in model and case composition. Even among end-to-end successes, only 39/71 repaired counterparts are classified correctly; 13/71 still receive a concurrency diagnosis.

4.3 Failure Analysis

T0: diagnostic tendencies. Incorrect diagnoses on positive cases concentrate on source-unsupported behavior (135 episodes) and sequential defects (121), with only 13 attributing the symptom to external or configuration failures. The tendencies differ across models: Mini accounts for 73 of the unsupported-source predictions, whereas Flash predicts concurrency on 146/186 repaired controls (Figure 4(b)). Sonnet and GPT-5.5 also predict concurrency on 97/198 and 92/198 repaired controls, respectively. These results expose both missed concurrency bugs and false concurrency diagnoses; missing answers, especially for Pro and GLM, represent a separate failure mode.

T1: confusing related mechanisms. The most frequent confusion labels unsynchronized conflicting access as missed notification (23 episodes). Other common confusions include recursive-lock deadlock as lock-cycle deadlock (21) and duplicate release as lifetime overlap (17). For example, on a duplicate-release case involving a channel, one answer selects lifetime overlap by interpreting the bug as one actor sending on the channel after another actor closes it. This substitutes a use-after-close explanation for the repeated-close mechanism.

T2: nontrivial workload formation. T2 failures involve incorrect resources, setup, actor assignment, and call order. Our running example, K28, illustrates an actor-assignment error. GPT-5.5 selects the expected resources and operations but places the confirming reuse fault after the stalled read on the same actor. The read cannot return until the reuse fault confirms slot reuse, while that actor cannot begin the reuse fault until the read returns. This creates a circular dependency. GPT-5.5 is the only model to reach T2 for K28 and makes this assignment in all three repetitions. The grader rejects these workloads before execution.

T3: grounding the execution hypothesis. T3 failures involve incorrect source anchors, inconsistent object bindings, incomplete event ordering, and missing execution controls. For example, one certificate requests the ninth occurrence of a source statement that appears only once, preventing the evaluator from resolving the selected anchor. Other submissions duplicate a source event as the final observation or omit events from shared-object and ordering constraints. 28 terminal errors concern missing or mismatched mediation declarations needed to control execution, and 6 change the accepted T2 workload. These failures show the difficulty of translating an execution explanation into precise, mutually consistent constraints that the backend can enforce.

5 Limitations

Five of 71 collected cases are excluded: four userspace races (c-ares, gofeed, libevent, and SQLite) lack independently established behavioral symptoms; the Linux quota/filesystem-freeze livelock lacks reliable single-run grading because conforming executions may not trigger harm. SCE-Bench has limited coverage, and task construction requires significant engineering effort.

6 Conclusion

SCE-Bench evaluates concurrency reasoning from symptom to mechanism, workload, and execution. Bounded interfaces enable grading without an LLM judge; negative controls test diagnosis, and execution certificates connect explanations to harm. Across seven models and three repetitions, the highest end-to-end success rate is 13.1%, and only 10 of 66 positive tasks are solved by any model. All models perform worse on repaired controls than independent negatives and suffer their largest proportional stage loss at T3. By identifying these gaps, SCE-Bench offers concrete targets for improving agents' ability to diagnose concurrency bugs and ground their explanations in execution.

AI USE STATEMENT

Generative AI tools assisted our research, code generation, and manuscript preparation, including drafting, editing, and reviewing text. At least one author reviewed and verified all AI-generated content. The authors are responsible for the paper's claims and the accompanying code and artifacts.

ETHICS STATEMENT

The corpus contains security-relevant technical details about concurrency defects. All included concurrency defects are publicly disclosed and paired with recorded fixing commits.

REPRODUCIBILITY STATEMENT

§3 describes the task interfaces, grading rules, and execution backends; §4.1 specifies the evaluation protocol, agent tools, and budgets. The appendix provides additional data and implementation details. After discussion forums open, we will share an anonymous repository link through a comment visible only to reviewers and area chairs. The repository will contain task manifests, prompts, grading implementations, execution drivers, model configurations, and evaluation records.

REFERENCES

Farnaz Behrang, Zhizhou Zhang, Georgian-Vlad Saioc, Peng Liu, and Milind Chabbi. DR.FIX: Automatically fixing data races at industry scale. *Proceedings of the ACM on Programming Languages*, 9(PLDI), 2025. doi: 10.1145/3729265. URL https://doi.org/10.1145/3729265.

Dirk Beyer and Karlheinz Friedberger. Violation Witnesses and Result Validation for Multi-Threaded Programs: Implementation and Evaluation with CPAchecker. In *Proceedings of the 9th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), Part I*, volume 12476 of *Lecture Notes in Computer Science*, pp. 449–470. Springer, 2020. doi: 10.1007/978-3-030-61362-4_26. URL https://www.sosy-lab.org/research/witnesses-concurrency/.

Francesco A. Bianchi, Mauro Pezzè, and Valerio Terragni. Reproducing concurrency failures from crash stacks. In *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, pp. 705–716, 2017. doi: 10.1145/3106237.3106292. URL https://valerio-terragni.github.io/assets/pdf/bianchi-fse-2017.pdf.

Hongchen Cao, Jingzhu He, Ting Dai, and Guoliang Jin. Themis: Detecting distributed concurrency bugs through RPC-driven race-directed test generation and fuzzing. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 1685–1706. USENIX Association, 2026. URL https://www.usenix.org/conference/nsdi26/presentation/cao.

Yangruibo Ding, Yanjun Fu, Omniyyah Ibrahim, Chawin Sitawarin, Xinyun Chen, Basel Alomair, David Wagner, Baishakhi Ray, and Yizheng Chen. Vulnerability detection with code language models: How far are we? In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering*, ICSE '25, pp. 1729–1741. IEEE Press, 2025. ISBN 9798331505691. doi: 10.1109/ICSE55347.2025.00038. URL https://doi.org/10.1109/ICSE55347.2025.00038.

Tayfun Elmas, Jacob Burnim, George Necula, and Koushik Sen. CONCURRIT: A domain specific language for reproducing concurrency bugs. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pp. 153–164, 2013. URL https://people.eecs.berkeley.edu/~necula/Papers/concurrit-pldi13.pdf.

Sishuai Gong, Chih-En Lin, Kevin Wu, Edwin Lu, and Pedro Fonseca. A Comprehensive Study of Concurrency Bugs in the Linux Kernel. In *Proceedings of the 48th IEEE/ACM International Conference on Software Engineering (ICSE)*, 2026. doi: 10.1145/3744916.3787790. URL https://www.cs.purdue.edu/homes/pfonseca/papers/icse26-concurrencybugstudy.pdf.

Jeff Huang, Charles Zhang, and Julian Dolby. CLAP: Recording local executions to reproduce concurrency failures. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2013. URL https://www.cse.ust.hk/~charlesz/academic/clap.pdf.

Jue Huang, Tarek Mahmud, Corina Pasareanu, and Guowei Yang. CONCUR: Benchmarking LLMs for concurrent code generation. *arXiv preprint arXiv:2603.03683*, 2026. URL https://arxiv.org/abs/2603.03683.

Vilas Jagannath, Milos Gligoric, Dongyun Jin, Qingzhou Luo, Grigore Rosu, and Darko Marinov. Improved multithreaded unit testing. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pp. 223–233, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450304436. doi: 10.1145/2025113.2025145. URL https://doi.org/10.1145/2025113.2025145.

Nicholas Jalbert, Cristiano Pereira, Gilles Pokam, and Koushik Sen. RADBench: A concurrency bug benchmark suite. In *3rd USENIX Workshop on Hot Topics in Parallelism (HotPar)*. USENIX Association, 2011. URL https://www.usenix.org/events/hotpar11/tech/final_files/Jalbert.pdf.

Dae R. Jeong, Yewon Choi, Byoungyoung Lee, Insik Shin, and Youngjin Kwon. Ozz: Identifying Kernel Out-of-Order Concurrency Bugs with In-Vivo Memory Access Reordering. In *Proceedings of the 30th ACM Symposium on Operating Systems Principles (SOSP)*, Austin, TX, November 2024. doi: 10.1145/3694715.3695944.

Hwiwon Lee, Ziqi Zhang, Hanxiao Lu, and Lingming Zhang. SEC-bench: Automated benchmarking of LLM agents on real-world software security tasks. In *The thirty-ninth annual conference on neural information processing systems*, 2025. URL https://openreview.net/forum?id=QQhQIqons0.

Hwiwon Lee, Jiawei Liu, Dongjun Kim, Wubing Xia, Ziqi Zhang, Chunqiu Steven Xia, and Lingming Zhang. SEC-bench Pro: Can language models solve long-horizon software security tasks? *arXiv preprint arXiv:2605.26548*, 2026. URL https://arxiv.org/abs/2605.26548v2.

Seunghyun Lee and David Brumley. ExploitBench: A capability ladder benchmark for LLM cybersecurity agents. *arXiv preprint arXiv:2605.14153*, May 2026. URL https://arxiv.org/abs/2605.14153.

Ao Li, Byeongjee Kang, Vasudev Vikram, Isabella Laybourn, Samvid Dharanikota, Shrey Tiwari, and Rohan Padhye. Fray: An efficient general-purpose concurrency testing platform for the JVM. *Proceedings of the ACM on Programming Languages*, 9(OOPSLA2):4035–4063, 2025. doi: 10.1145/3764119. URL https://rohan.padhye.org/files/fray-oopsla25.pdf.

Jiashuo Liang, Ming Yuan, Zhanzhao Ding, Siqi Ma, Xinhui Han, and Chao Zhang. RaceBench: A triggerable and observable concurrency bug benchmark. In *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security*, pp. 415–428. ACM, 2023. doi: 10.1145/3579856.3595787.

Ziyi Lin, Darko Marinov, Hao Zhong, Yuting Chen, and Jianjun Zhao. JaConTeBe: A benchmark suite of real-world Java concurrency bugs. In *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 178–189, 2015. URL https://mir.cs.illinois.edu/marinov/publications/LinETAL15JaConTeBe.pdf.

Niels Mündler, Mark Niklas Müller, Jingxuan He, and Martin Vechev. SWT-Bench: Testing and validating real-world bug-fixes with code agents. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 81857–81887. Curran Associates, Inc., 2024. doi: 10.52202/079017-2601. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/94f093b41fc2666376fb1f667fe282f3-Paper-Conference.pdf.

NIST. CVE-2024-26759 Detail. National Vulnerability Database, April 2024. URL https://nvd.nist.gov/vuln/detail/cve-2024-26759. Accessed: 2026-09-24.

Hernán Ponce-de-León, Thomas Haas, and Roland Meyer. Dartagnan: SMT-based Violation Witness Validation (Competition Contribution). In *Proceedings of the 28th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Part II*, volume 13244 of *Lecture Notes in Computer Science*, pp. 418–423. Springer, 2022. doi: 10.1007/978-3-030-99527-0_24. URL https://link.springer.com/chapter/10.1007/978-3-030-99527-0_24.

Shuai Shao, Dingbang Wang, Yiming Zeng, and Tingting Yu. ConFL: Explainable concurrent fault localization via hierarchy-guided LLM reasoning. *arXiv preprint arXiv:2608.02974*, August 2026. URL https://arxiv.org/abs/2608.02974.

Vasu Vikram, Ao Li, and Rohan Padhye. Spaghetti Bench: Evaluating AI agents on concurrency bug fixes. PASTA Lab project report, February 2026. URL https://pastalab.org/spaghetti-bench/blog.html.

Zhun Wang, Nico Schiller, Hongwei Li, Srijiith Sesha Narayana, Milad Nasr, Nicholas Carlini, Xiangyu Qi, Eric Wallace, Elie Bursztein, Luca Invernizzi, Kurt Thomas, Yan Shoshitaishvili, Wenbo Guo, Jingxuan He, Thorsten Holz, and Dawn Song. ExploitGym: Can AI agents turn security vulnerabilities into real attacks? *arXiv preprint arXiv:2605.11086*, May 2026a. URL https://arxiv.org/abs/2605.11086.

Zhun Wang, Tianneng Shi, Jingxuan He, Matthew Cai, Jialin Zhang, and Dawn Song. CyberGym: Evaluating AI agents' real-world cybersecurity capabilities at scale. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2026b. URL https://openreview.net/forum?id=2YvbLQEdYt.

Ting Yuan, Guangwei Li, Jie Lu, Chen Liu, Lian Li, and Jingling Xue. GoBench: A benchmark suite of real-world Go concurrency bugs. In *IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021. doi: 10.1109/CGO51591.2021.9370317. URL https://lujie.ac.cn/files/papers/GoBench.pdf.

Cristian Zamfir and George Candea. Execution synthesis: A technique for automated software debugging. In *Proceedings of the 5th European Conference on Computer Systems (EuroSys)*, pp. 321–334, 2010. URL https://llvm.org/pubs/2010-04-EUROSYS-ExecutionSynthesis.pdf.

Andy Zhang, Joey Ji, Celeste Menders, Riya Dulepet, Thomas Qin, Ron Wang, Junrong Wu, Kyleen Liao, Jiliang Li, Jinghan Hu, Sara Hong, Nardos Demilew, Shivatmica Murgai, Jason Tran, Nishka Kacheria, Ethan Ho, Denis Liu, Lauren McLane, Olivia Bruvik, Dai-Rong Han, Seungwoo Kim, Akhil Vyas, Cuiyuanxiu Chen, Ryan Li, Weiran Xu, Jonathan Ye, Prerit Choudhary, Siddharth M. Bhatia, Vikram Sivashankar, Yuxuan Bao, Dawn Song, Dan Boneh, Daniel Ho, and Percy Liang. Bountybench: Dollar impact of ai agent attackers and defenders on real-world cybersecurity systems. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025a. doi: 10.52202/085713-5725. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/faed4276b52ef762879db4142655c699-Paper-Datasets_and_Benchmarks_Track.pdf.

Andy K. Zhang, Neil Perry, Riya Dulepet, Joey Ji, Celeste Menders, Justin W. Lin, Eliot Jones, Gashon Hussein, Samantha Liu, Donovan Jasper, Pura Peetathawatchai, Ari Glenn, Vikram Sivashankar, Daniel Zamoshchin, Leo Glikbarg, Derek Askaryar, Mike Yang, Teddy Zhang, Rishi Alluri, Nathan Tran, Rinnara Sangpisit, Polycarpos Yiorkadjis, Kenny Osele, Gautham Raghupathi, Dan Boneh, Daniel E. Ho, and Percy Liang. Cybench: A framework for evaluating cybersecurity capabilities and risks of language models. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 25094–25243, 2025b. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/3e9412a9c1d93810ef3ef7825115016b-Paper-Conference.pdf.

Yuxuan Zhu, Antony Kellermann, Dylan Bowman, Philip Li, Akul Gupta, Adarsh Danda, Richard Fang, Conner Jensen, Eric Ihli, Jason Benn, Jet Geronimo, Avi Dhir, Sudhit Rao, Kaicheng Yu, Twm Stone, and Daniel Kang. CVE-Bench: A Benchmark for AI Agents' Ability to Exploit Real-World Web Application Vulnerabilities. In *Proceedings of the 42nd International Conference on*

Machine Learning, volume 267 of *Proceedings of Machine Learning Research*, pp. 79850–79867, 2025. URL https://proceedings.mlr.press/v267/zhu25i.html.

A Positive Case Inventory

Table 3 and Table 4 list all 66 scored positive cases. Userspace cases are labeled U1–U31 and kernel cases K1–K35. Mechanism labels abbreviate the families in Table 1. Each source pair lists the affected revision above its repair; both identifiers link to upstream source records. The kernel cases are from Linux except for the named FreeRTOS case. For U6, the listed revision is the initial crash repair for CVE-2018-1000030; upstream later superseded it with a compatibility-preserving repair.

Fix dates are UTC committer dates. The public-record column links to an assigned CVE where verified as of September 25, 2026; otherwise, a date links to the retained GitHub issue (I), pull request (P), or syzbot report (S). An em dash means that the temporal inventory has no separately dated public record for that case; it does not mean that no report exists. Absence of a CVE identifier means that no assignment was verified, not that no CVE exists. The retained temporal inventory uses 19 public-record dates and the fix dates for the remaining 47 cases. Five of these public-record dates are represented by CVE identifiers in the tables, leaving 14 dates displayed. These dates do not establish the first disclosure or the release date of a software package.

A.1 Userspace Cases

Table 3: Userspace positive cases (31). Source pairs list the affected revision above its repair.

ID	Project / defect	Mechanism	Public record	Fix date	Source pair
U1	async_simple ConditionVariable ABA/UAF	ABA	2026-03-16[I]	2026-03-17	e50c0a866257 8df816f11876
U2	async_simple FutureState continuation publication failure	Publication	—	2026-01-26	82f2b09f1280 12c2a726f511
U3	Boost.Lockfree queue next-tag ABA	ABA	2020-06-03[I]	2020-06-23	771e8c1016a5 df78b9da42d7
U4	Caddy UsagePool constructor/Range lock-cycle deadlock	Lock cycle	—	2026-08-29	502691f51821 eaaacccf50ed
U5	CockroachDB virtual-intent-resolution livelock	Livelock	2026-02-19[I]	2026-03-17	6abd5bbf44de 6c4e614cd01c
U6	CPython 2.7 shared-file readahead crash	Shared state	CVE-2018-1000030	2017-11-10	1bce4efdb462 6401e5671781
U7	curl blank-header thread-safety	Shared state	—	2019-02-19	5908e900be1e 942eb09e8a97
U8	curl OpenSSL shared-session data-race	Lifetime overlap	—	2024-09-02	2c2292ecaf41 a2bcec0ee089
U9	Dapper concurrent type-handler registration lost update	Lost update	2022-08-09[I]	2024-10-31	00a3808f7db1 9f4f783a4184
U10	Fontconfig cache-version lazy-initialization race	Stale snapshot	—	2026-08-25	2c648f834e27 d9941dd1031a
U11	FreeRDP WinPR timer-queue cancellation race	Conflicting access	—	2024-04-15	09060a9a49a6 e4498df60750
U12	golang-lru expirable callback-reentry deadlock	Recursive lock	2026-08-18[I]	2026-09-03	d76085cdab99 9c13c57de0be
U13	Gotenberg `downloadFrom` concurrent-map-writes crash	Conflicting access	CVE-2026-45742	2026-05-12	f9a01c9fb308 6671b5e5d302
U14	gRPC-Go `WriteStatus` duplicate close	Duplicate release	2017-03-08[I]	2017-10-04	cf79c849790f 22c3f92f5fae

Continued on next page

Table 3 (continued)

ID	Project / defect	Mechanism	Public record	Fix date	Source pair
U15	HiGHS binary-semaphore publication race	Publication	2026-02-06[I]	2026-02-06	980d45467541 9b2e3cd1ed2c
U16	libarchive built-in CRC32 lazy-initialization race	Publication	—	2026-06-02	235165e3e5fb 95154465d902
U17	libgit2 built-in SHA-256 wrong digest	Shared state	—	2026-05-16	d23f18fd881b 455e513120cf
U18	libuv slow-I/O threadpool counter race	Conflicting access	—	2018-10-07	6781db5c789b daf04e83cb45
U19	LocalAI watchdog stop double close	Duplicate release	2026-07-15[I]	2026-07-16	6a985d13ea4e 808312b4b96e
U20	Playwright-Go load-state lost wakeup	Missed notification	—	2026-09-09	b4b4642de835 a75a7a3b79f6
U21	PostgreSQL RI fast-path race with `REINDEX CONCURRENTLY`	Stale snapshot	—	2026-08-18	92e20d4a4382 d02a9084d910
U22	Prometheus `promhttp` dynamic-label shared-map race	Conflicting access	2023-05-15[I]	2023-08-01	7f2db5f1ed43 59c00e3e9cdf
U23	SGLang cached-prefix resume livelock	Livelock	2026-07-25[P]	2026-07-25	e943e609dc50 69a3c54c7023
U24	SignalR group-map concurrent-access race	Conflicting access	—	2026-08-03	9b75d16d216e 7286f347d118
U25	AWS Smithy Go event-stream close/write race	Lifetime overlap	—	2026-08-24	95f86a699b66 7e0c19975a09
U26	SQLite memdb open/last-close deadlock	Lock cycle	—	2021-05-12	85ffcae177ba 52d1407891ad
U27	SQLite WAL shared-cache callback-field data race	Shared state	—	2014-12-12	2ea0bafae2c4 ef15c6e9e65c
U28	Viam RDK GraphNode recursive-close deadlock	Recursive lock	2024-05-17[P]	2024-06-18	df1281698fb8 37d9d51b8e92
U29	xsync concurrent-resize lost update	Lost update	2023-10-27[I]	2023-11-01	abcfab602348 42e3390a5c0b
U30	XZ threaded-decoder partial-output deadlock	Missed notification	—	2022-03-25	e0394e94230f bd93b776c1bd
U31	zstd adaptive compression-level data race	Shared state	—	2017-08-02	69ef22c0ac97 8be7bba08c1c

A.2 Kernel Cases

Table 4: Kernel positive cases (35). Source pairs list the affected revision above its repair.

ID	Project / defect	Mechanism	Public record	Fix date	Source pair
K1	FreeRTOS `vTaskListTasks()` stale-snapshot overflow	Stale snapshot	—	2026-06-19	83e56c38ee18 e146d6444c3d
K2	ALSA aloop peer-runtime use-after-free	Lifetime overlap	CVE-2026-46090	2026-04-27	901ac0ff15ed e5c33cdc6f40
K3	Apple SMC cache publication failure	Publication	—	2026-08-10	d33baed3fea2 5a151274c1fd

Continued on next page

Table 4 (continued)

ID	Project / defect	Mechanism	Public record	Fix date	Source pair
K4	audit executable-rule recursive directory-lock deadlock	Recursive lock	CVE-2026-68096	2026-05-27	888a0396e154 81905b5acbe7
K5	Bluetooth L2CAP cleanup race	Lifetime overlap	CVE-2026-90091	2026-08-24	59eecbe2f2f3 66d6ef18548a
K6	bpffs inode RCU use-after-free	Lifetime overlap	CVE-2026-74363	2026-06-02	51321158a365 b93c55b4932d
K7	CAN transmitted-frame statistics race	Lost update	—	2025-03-14	1d22a122ffb1 80b5f90158d1
K8	cfg80211 `cmp_bss()` use-after-free	Lifetime overlap	CVE-2025-39864	2025-08-26	f64768bec0d5 26e84445f02c
K9	cgroup PSI pressure-write use-after-free	Lifetime overlap	CVE-2026-52991	2026-04-17	d730905bc3c0 a5b98009f16d
K10	e1000 reset-work/RTNL deadlock	Lock cycle	CVE-2025-38114	2025-06-10	a5a441ae283d b4a8085ceefb
K11	epoll exclusive-wait-entry lost-wakeup	Missed notification	—	2020-05-08	474328c06e3e 412895f03cbf
K12	eventpoll `tfile_check_list` use-after-free	Lifetime overlap	—	2026-06-04	b466bfe0ec0f a1e9718b406b
K13	FUSE synchronous-init abort use-after-free	Lifetime overlap	CVE-2026-80857	2026-08-17	6e64df0f73f1 64b0b5cacbd2
K14	PI-futex exit livelock	Livelock	—	2019-11-20	ac31c7ff8624 3ef240eaff36
K15	edge-IRQ affinity-migration livelock	Livelock	—	2025-07-22	4e879dedd571 8d39d6ec4db5
K16	I2C debugfs adapter-removal use-after-free	Lifetime overlap	CVE-2026-89983	2026-08-28	27c9445be86b b15b548d52b4
K17	ieee802154 hwsim PIB duplicate release	Duplicate release	CVE-2026-97599	2026-09-02	bf79662bc85e 979d5b8de8ed
K18	IPComp per-CPU scratch-buffer race	Shared state	—	2013-10-18	e9e4ea74f066 12e3594698f6
K19	lockdep stale-class-cache ABA	ABA	2026-08-24[S]	2026-09-02	2af470916a20 02c6be7d675b
K20	loop size-update/queue-freeze lock cycle	Lock cycle	—	2025-06-11	6f65947a1e68 4bb08cf974c5
K21	NET/ROM routing lock-order deadlock	Lock cycle	CVE-2024-38589	2024-05-17	67708158e732 e03e7f20ebf7
K22	network-watchdog ref-tracker race	Duplicate release	CVE-2026-74264	2026-06-13	56a0b00c5a04 8eed5519e496
K23	NFSD reclaim lost-update	Lost update	—	2026-08-10	09ea3eb9a518 0a5a548861ab
K24	rtl8150 transmit/completion use-after-free	Lifetime overlap	CVE-2026-52982	2026-04-28	9e6bf146b559 23f0e34c64ac
K25	SBS cross-battery serial substitution	Shared state	—	2026-07-26	df55823b98fe 6027892925b8
K26	BPF sockmap callback-teardown race	Lifetime overlap	CVE-2024-26731	2024-02-21	5c138a8a4abe 4cd12c6065df
K27	SLIP `sl_sync()` teardown use-after-free	Lifetime overlap	CVE-2026-81001	2026-08-27	9a56a27e6002 2c4e7c42d77e
K28	direct-swap-in slot ABA data loss	ABA	CVE-2024-26759	2024-02-20	16e96ba5e92c 13ddaf26be32

Continued on next page

Table 4 (continued)

ID	Project / defect	Mechanism	Public record	Fix date	Source pair
K29	TCP reuseport-migration lost-wakeup	Missed notification	CVE-2026-46015	2026-04-23	e08a9fac5cf8 3864c6ba1e04
K30	shadow-timekeeper TAI lost update	Lost update	—	2013-12-23	0e6601eee039 f55c07607a38
K31	TLS initialization-publication	Publication	CVE-2024-36489	2024-05-23	3b1c92f8e537 91e61dd7a0af
K32	`trace_pipe`/sub-buffer-order use-after-free	Lifetime overlap	CVE-2026-89747	2026-08-22	649bc7df3e5d 372f8534244d
K33	UDF AED allocator recursive-mutex deadlock	Recursive lock	2026-06-20[S]	2026-07-08	120ec50984b8 31e5e0c8c8e2
K34	VMCI host-poll race	Stale snapshot	CVE-2023-54007	2023-04-05	48a6c7bced2a ae13381da5ff
K35	workqueue started-statistics race	Lost update	CVE-2023-53329	2023-08-29	b6a46f7263bd fe48ba7daefe

B Model Details

Table 5 lists the seven evaluated models, their release dates, documented cutoffs, and reasoning-effort settings. Release dates and cutoffs link to provider documentation, checked on September 25, 2026. The Anthropic, OpenAI, and GLM models use the configured LiteLLM route; the Gemini models use OpenRouter. All models share the action and completion limits in §4.

Table 5: Evaluated models and request settings. K denotes a documented knowledge cutoff; T denotes a documented training-data cutoff. Month-only dates retain the precision reported by the provider.

Model	Release date	Cutoff	Effort
GPT-5.5	2026-04-24	2025-12-01 K	high
GPT-5.4 Mini	2026-03-17	2025-08-31 K	high
Claude Opus 4.6	2026-02-05	2025-08 T	high
Claude Sonnet 4.6	2026-02-17	2026-01 T	high
Gemini 3.1 Pro	2026-02-19	2025-01 K	high
Gemini 3.1 Flash-Lite	2026-05-07	2025-01 K	high
GLM-5.3	2026-08-18*	Undocumented	—

Date conventions. Release dates refer to API availability for OpenAI, Anthropic, and Google; the Flash-Lite date is its stable release. *For GLM-5.3, the date is the provider's release-note date. The cutoff column gives the boundary used for temporal analysis: Anthropic's training-data cutoffs differ from its reliable-knowledge cutoffs (May 2025 for Opus 4.6 and August 2025 for Sonnet 4.6). No cutoff is documented in GLM-5.3's model documentation or model card. The frozen temporal artifact instead uses August 25, 2026, the model repository's creation date, as a separately labeled proxy; this is neither a training cutoff nor a verified public-release date.

Inference settings. Effort denotes the requested reasoning setting; identical labels do not imply equal reasoning compute across providers. The dash indicates that GLM-5.3's reasoning effort is not explicitly set.