EQUINOX: BENCHMARKING NATURAL-LANGUAGE-TO-TLA+ FORMALISATION BY BOUNDED TRACE EQUIVALENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

A TLA+ specification describes a protocol's behaviour. The distributed computing community uses TLA+ to find protocol defects before deployment. Writing these specifications requires substantial time and expertise, which motivates the use of AI agents. Current natural-language-to-TLA+ benchmarks accept a specification that parses, runs, and satisfies its declared properties. However, a specification can pass all three checks yet describe a different protocol by omitting required behaviour, allowing extra behaviour, or both. We present EQUINOX, a benchmark for protocol formalisation in TLA+. It contains 115 tasks and evaluates each output using *bounded trace equivalence*. Each task uses a published specification as a hidden reference. EQUINOX compares the behaviours of the agent output and the hidden reference in both directions. We evaluate five agents: Codex + GPT-5.6 Sol, Codex + GPT-5.6 Terra, Claude Code + Opus 5, Claude Code + Sonnet 5, and OpenCode + GLM-5.3. We report three results. First, execution is necessary but not sufficient for successful formalisation. Nontrivial execution accepts 73.0% of the 1723 eligible attempts, but only 24.0% satisfy bounded trace equivalence. Second, the agents solve similar sets of tasks: no agent solves 71 of 115 tasks, while all five agents solve 24 tasks. Third, more attempts rarely resolve modelling errors. Codex + GPT-5.6 Sol solves 1 of 11 selected failed tasks across 110 further attempts. Together, these results show a capability gap for frontier AI agents in protocol formalisation.

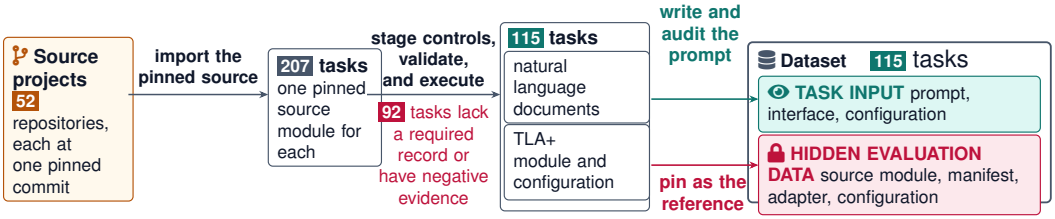
1 INTRODUCTION

Background. TLA+ is a language for specifying a protocol's behaviour. Its model checker, TLC, explores the specification's finite reachable state space (Lamport, 1994; 2002; Yu et al., 1999). PlusCal provides an algorithm-like syntax for TLA+ (Lamport, 2009). Engineering teams use TLA+ and TLC to find concurrency defects overlooked by testing and design review (Newcombe et al., 2015). However, writing these specifications requires substantial time and expertise.

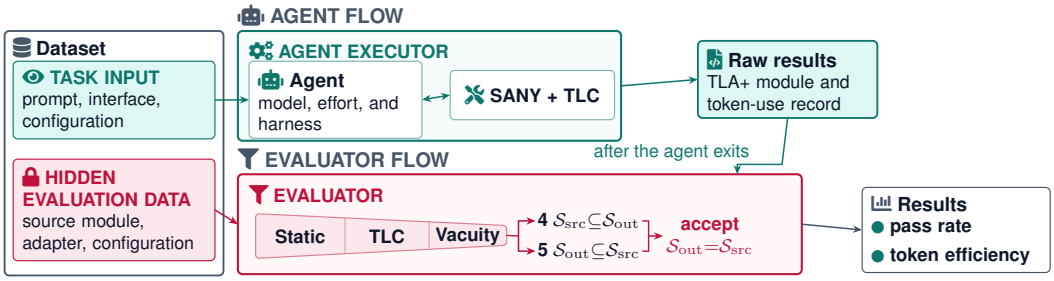
An AI agent can reduce this effort and make formal specification practical for more protocol designs. The domain also provides automated checks that can guide an agent's work. Recent work in formal mathematics shows that AI agents can produce machine-checked proofs. In these systems, a proof checker verifies every step. This feedback enables training against checker results instead of human labels. Such systems now reach olympiad-level performance (Trinh et al., 2024; Hubert et al., 2025). Spencer et al. (2026) apply this feedback loop to TLA+, with TLC as the reward signal and no learned reward model. The usefulness of an AI agent in this domain therefore depends on whether the checker accepts semantically correct specifications.

The problem. Current natural-language-to-TLA+ benchmarks check syntax, execution, or selected properties (Bisharat et al., 2026a; Spencer et al., 2026; Bisharat et al., 2026b; Cao et al., 2025). These criteria can accept a nontrivial specification that describes a different protocol (Table 1). This mismatch also affects leaderboards and training loops, which can reward an incorrect specification. The limited pool of public TLA+ specifications creates a second concern: a model can have seen a reference during training. An evaluation must report how it handles that risk.

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.



(a) Dataset construction from 207 packages in 52 pinned repositories. Bounded execution admits 118 and records 89 defects. Evaluator filters yield 115 tasks. Generated aliases separate public task input from hidden evaluation data.



(b) Evaluation. The executor records the agent output. The evaluator combines it with hidden data, applies five checks, and reports the equivalence-pass rate and token efficiency.

Figure 1: EQUINOX constructs source-grounded tasks and evaluates agents against hidden behavioural references.

The EQUINOX benchmark (§3.1). A specification describes a set of behaviours, so it must contain the complete transition relation. One omitted action removes a required execution. One extra action allows behaviour that the protocol forbids. Any single run can miss either fault. An acceptance criterion must therefore check the complete transition relation rather than a sample of it.

EQUINOX addresses this gap by applying bounded trace equivalence, an established technique in autoformalisation (Liu et al., 2025; Poiroux et al., 2025; Thakur et al., 2025), to natural-language-to-TLA+ evaluation (Figure 1). Let $\mathcal{S}_{\text{src}}$ and $\mathcal{S}_{\text{out}}$ be the behaviours of a hidden source specification and an agent output over the declared observable variables at one finite scope. The evaluator checks $\mathcal{S}_{\text{src}} \subseteq \mathcal{S}_{\text{out}}$ for omitted behaviour and $\mathcal{S}_{\text{out}} \subseteq \mathcal{S}_{\text{src}}$ for extra behaviour. It accepts only when both inclusions hold. A private evaluator applies this criterion after the agent session ends, so the reference never enters the generation environment. This paper makes three contributions. It defines bounded trace equivalence as an acceptance criterion for TLA+. It builds 115 tasks from published specifications that remain inaccessible to the agent. It evaluates five agents on those tasks.

Dataset construction (§3.2). EQUINOX contains 115 tasks built from published executable specifications. Each task gives the agent a protocol description, an observable interface, and finite configurations. The published specification and all source-name mappings remain hidden. The dataset records source provenance and reports potential contamination instead of claiming that public sources are unseen.

Evaluation (§4). We evaluate five agents, each with three attempts on each of the 115 tasks. Each agent combines a model, an effort setting, and the software that runs the agent, so the study does not measure an isolated model. Pass@3 ranges from 23.5% for OpenCode + GLM-5.3 to 35.7% for Claude Code + Opus 5. Nontrivial execution accepts 73.0% of eligible attempts, while bounded trace equivalence accepts only 24.0%. Success also concentrates on a shared set: at least one agent solves 44 tasks, all five solve 24, and no agent solves the other 71. Repeated attempts by one agent rarely solve the failed tasks. We run 10 new attempts with Codex + GPT-5.6 Sol on each selected task that it failed in the baseline experiment. The agent solves only 1 task across these 110 attempts. These values describe the tested agents on this fixed benchmark and do not rank model families. We release all artefacts so that other researchers can reproduce and extend the study.

Table 1: Natural-language-to-formal-specification benchmarks. Columns state acceptance criteria and contamination control. A dash marks an inapplicable criterion.

Benchmark			Evaluation criteria					Data control
Work	Formal output	Tasks	Static validity	Execution	Selected properties	Nontrivial output	Semantic equivalence	Contamination control
Natural-language description → TLA+ specification								
FormalLM (Bisharat et al., 2026a)	TLA+	205	●	●	●	○	○	○
TLA-Prover (Spencer et al., 2026)	TLA+	30	●	●	●	●	○	○
TLA+-Bench (Bisharat et al., 2026b)	TLA+	403	●	●	◑[g]	●	○	○
EQUINOX (ours)	TLA+	115	●	●	●	●	●	●
Natural-language description → other formal language								
nl2spec (Cosler et al., 2023)	LTL	36	●	○	–	–	○	○
nl2postcond (Endres et al., 2024)	program assertions	164	●	●	●	●	○	○
CLEVER (Thakur et al., 2025)	Lean 4	161	●	○	–	–	●	○
Poiroux et al. (2025)	Lean 4, statement only	990	●	●	–	–	●[h]	●
FM-Bench (Cao et al., 2025)	5 languages	175[i]	●	●	●	○	○	○
Mao et al. (2025)	Tamarin	18	●	●	●	○	○	○
CrypFormBench (Li et al., 2026)	7 languages	700	●	●	●	○	○	○

Static validity requires a parser or a type checker to accept the output. *Execution* requires a tool to run the output on a declared configuration. *Selected properties* requires the evaluator to check the output against named properties. *Nontrivial output* requires a check to reject an output that passes for an empty or a vacuous reason. *Semantic equivalence* requires the evaluator to compare the output with a hidden reference and accept only equal meaning. A work that satisfies equivalence also receives full credit for each criterion that equivalence entails. *Contamination control* requires the work to measure or limit training contamination of its reference artefacts (Appendix A).
[g] 291 of the 403 configurations name a property; the other 112 name the specification alone.
[h] Each statement must be provable from the other, decided by a bounded automatic proof search; the search is incomplete, so the check can miss an equivalent pair.
[i] The TLA+ tasks only. The whole benchmark holds about 4k tasks in five languages.

Findings. The results support three conclusions. First, execution is necessary but not sufficient for successful formalisation. An executable, nontrivial specification is weak evidence of a correct formalisation. Second, different agents solve similar sets of tasks. They succeed and fail on substantially overlapping task sets. Third, more attempts rarely resolve modelling errors. Codex + GPT-5.6 Sol solves 1 of 11 selected tasks in 10 further attempts per task. These findings show a capability gap for frontier AI agents in protocol formalisation.

2 Related work

This section compares EQUINOX with other natural-language-to-formal-specification benchmarks and summarises other benchmarks with external verification.

2.1 Natural language to TLA+

Evaluation of natural-language-to-TLA+ generation progressed from validity checks to property checks. FormalLM separates parser acceptance from finite execution (Bisharat et al., 2026a). TLA-Prover adds a mutation check that requires TLC to reject a weakened invariant (Spencer et al., 2026). TLA+-Bench applies these checks to 403 reference specifications. It explores the reachable state space, rejects weak properties, and tests whether each named property affects the result (Bisharat et al., 2026b).

EQUINOX changes the meaning of success from property satisfaction to equivalence with a hidden reference. A property set constrains behaviour but does not define the full transition relation. TLA+-Bench states this limitation: its decision rule tests property satisfaction for one configuration, not equivalence. EQUINOX checks both inclusions between the generated and reference behaviour sets. Thus, the progression is from execution to selected properties to bounded trace equivalence. Table 1 compares these criteria without ranking the systems that use them.

2.2 Natural language to other formal languages

Other benchmarks evaluate translation from natural language to different formal targets. nl2spec uses an LTL parser and expert review (Cosler et al., 2023). nl2postcond executes assertions on hidden

tests and uses incorrect programs to reject weak postconditions (Endres et al., 2024). FM-Bench uses verifier acceptance across five languages (Cao et al., 2025). Security-protocol work compares selected properties or security verdicts with a reference (Mao et al., 2025; Li et al., 2026).

EQUINOX borrows three established ideas from this wider field. It adopts the separation between task input and hidden evaluation data from code benchmarks (Chen et al., 2021; Jimenez et al., 2024). It adopts two-way equivalence from statement autoformalisation (Liu et al., 2025; Poiroux et al., 2025; Thakur et al., 2025). It also keeps validity checks as diagnostic stages before the semantic criterion. EQUINOX contributes a TLA+-specific method: TLC checks both behaviour inclusions at a finite scope, and the private evaluator contains the published reference.

2.3 Evaluation with external verification

Executable benchmarks test programs or interactive actions against concrete outcomes (Jimenez et al., 2024; Just et al., 2014; Trivedi et al., 2024; Xie et al., 2024). Proof and verification benchmarks instead supply a claim and ask a system to prove it or implement it (Loughridge et al., 2024; Lin et al., 2024; Zheng et al., 2022; Liu et al., 2023). In each case, an external system checks evidence specific to the generated output. External verification provides stronger evidence than a model that judges its own output, but only when the check matches the intended meaning.

EQUINOX applies this principle to protocol specifications. The relevant meaning is the allowed set of observable behaviours, so syntax or selected properties are incomplete evaluation targets. Bounded trace equivalence aligns the acceptance criterion with that meaning. EQUINOX therefore evaluates protocol specifications by comparing their behaviour.

3 Benchmark design

A task asks an AI agent to write one TLA+ module from a protocol description. The benchmark has three design goals. It tests semantic correctness, keeps the target behaviour outside the generation environment, and derives each target from an executable published source. Figure 1 gives the full architecture. The benchmark has three components: instructions to AI agents, an agent executor, and an evaluator. The instructions define the task, the agent executor runs one attempt, and the evaluator scores the saved submission after the attempt ends. Section 3.1 defines these components. Section 3.2 explains how curators derive the instructions and hidden evaluation data from a published specification.

3.1 Benchmark components

Instructions to AI agents. Each task gives the agent a protocol description, an observable interface, finite configurations, and submission requirements. The instructions contain no TLA+ fragment, citation, protocol name, or distinctive source identifier. This rule makes the task a formalisation problem rather than a transcription problem. Appendix B.2 describes the prompt controls and shows one task.

Agent executor. The agent executor creates an anonymous writable workspace outside the repository. It stages copies of the public task, a TLA+ quick reference, and the pinned TLA+ Tools. A request service runs SANY and TLC in a separate network-disabled container. The executor invokes one agent, supervises its wall-clock, token, and tool-event budgets, and records its events. The agent cannot read the repository or the hidden evaluation data. After the agent exits, the executor saves the workspace and identifies the submission for evaluation. The execution record describes the attempt but does not establish correctness.

Evaluator. The evaluator applies bounded trace equivalence to transition systems at a declared finite scope. Let $\mathcal{S}_{\mathrm{src}}$ and $\mathcal{S}_{\mathrm{out}}$ be the behaviours of the hidden source and the output after projection onto the observable variables. Acceptance requires $\mathcal{S}_{\mathrm{src}} = \mathcal{S}_{\mathrm{out}}$. The inclusion $\mathcal{S}_{\mathrm{src}} \subseteq \mathcal{S}_{\mathrm{out}}$ fails when the output omits required behaviour. The inclusion $\mathcal{S}_{\mathrm{out}} \subseteq \mathcal{S}_{\mathrm{src}}$ fails when the output allows extra behaviour. Either direction alone can accept a system that allows extra behaviour or omits required behaviour.

```
---- MODULE BehaviorSourceToAgentOutput ----
VARIABLES <observable>, behaviorViolation

observableVars == <<observable>>
allVars == <<observable, behaviorViolation>>

Source == INSTANCE <source>      WITH ...
Target == INSTANCE <agentOutput> WITH ...

Init ==
  /\ Source!Init
  /\ behaviorViolation = ~Target!Init

Next ==
  /\ (Source!Next \/ UNCHANGED observableVars)
  /\ behaviorViolation' = (behaviorViolation \/
       ~(Target!Next \/ UNCHANGED observableVars))

Spec == Init /\ [][Next]_allVars
BehaviorAgreement == ~behaviorViolation
====
```

Figure 2: One inclusion wrapper. `Source` drives; `Target` matches observable steps. This checks G4; swapping the instances checks G5. Task metadata supplies aliases, substitutions, and scope.

```
dataset/tasks/<task-id>/
├── task-input/ ..... agent input
│   ├── prompt.md
│   ├── interface.yaml
│   └── configurations/
├── hidden-eval-data/ ..... hidden evaluation data
│   ├── manifest.yaml
│   ├── source-adapter.yaml
│   └── behavior.cfg
├── upstream/ ..... unchanged scoring source
│   ├── <source module>.tla
│   └── LICENSE
├── analysis/ ..... admission evidence
│   └── bounded-execution.yaml
└── README.md ..... curator rationale
```

Figure 3: One task package. The agent and evaluator read disjoint halves. Hidden data identifies the byte-identical source; aliases hide source names.

Five checks implement this criterion. G1 validates the package and interface. G2 executes the output with TLC, and G3 requires at least two distinct reachable states. These checks establish nontrivial execution, not semantic agreement. The required-behaviour check G4 requires $\mathcal{S}_{\text{src}} \subseteq \mathcal{S}_{\text{out}}$, and the no-extra-behaviour check G5 requires $\mathcal{S}_{\text{out}} \subseteq \mathcal{S}_{\text{src}}$. G4 and G5 independently check the same outputs that pass G1–G3.

Figure 2 shows how the evaluator checks both directions. First, the source takes a step, and the output must match its observable effect. The evaluator then swaps their roles to find extra output behaviour. For specifications with internal state, it compares finite graphs that contain only visible state. A submission passes only if both comparisons complete without a mismatch. Passing both checks guarantees agreement on every visible property that ignores repeated states at this scope. Passing selected properties alone does not guarantee equivalence.

After generation ends, an isolated evaluator scores the saved submission against the hidden source (Appendix B.3). Public records omit private paths, hidden source names, and TLC traces. Positive and negative controls test both behaviour checks, and Appendix A reports their coverage. A submission earns an equivalence pass only when all five checks pass. We report the equivalence-pass rate and equivalence passes per million recorded tokens. Check survival remains a diagnostic and does not measure correctness.

3.2 Constructing benchmark tasks

EQUINOX builds each task around an existing executable specification. This use of existing specifications follows executable software benchmarks (Just et al., 2014; Tomassi et al., 2019; Silva et al., 2024). The benchmark never writes a new transition system from prose.

Published source and validation (Appendix B.4). EQUINOX selects tasks from public repositories that maintain TLA+ specifications. For each task, we run the model checker at a finite scope and confirm that it reaches every declared protocol action. We include only tasks that pass this check. We may add adapters and finite settings, but we do not add or repair protocol transitions. The evaluator checks whether each submission has the same behaviour as the imported specification. Appendix B.1 describes the full task curation process.

We collected 207 task packages from 52 repositories. Of these packages, 118 satisfy bounded execution. The evaluator then excludes one package without a complete private name map and two whose bounded behaviour comparisons fail. This leaves 115 evaluation tasks from 36 repositories.

Contamination control (Appendix B.4). All source specifications are public. A model might have seen them during training. We report tasks from the TLA+ example repository separately from all other tasks. This comparison indicates potential contamination. It cannot show whether a model saw a specific source.

Protocol coverage. Curators assign each task to one protocol family from a fixed taxonomy. The full collection contains 207 packages across 12 families. The 115 evaluation tasks cover 11 families. Table 12 in Appendix D.2 gives the number of tasks in each family.

Public and hidden task data. Each task package has public files for the agent and hidden files that only the evaluator can access (Figure 3). The public files describe the protocol, the observable variables, and the names that the submission must define. The benchmark replaces source-specific names with neutral names. The agent therefore receives the required interface but not the original names or source implementation. The hidden files contain the imported specification, a map from neutral names to original names, and the finite comparison settings.

Following TLA+-Bench (Bisharat et al., 2026b), curators use AI agents to draft prompts from source documentation, such as a README, a design document, or a paper. The curator checks each prompt against its source-to-sentence mapping and applies the prompt release checks. The imported executable specification remains the reference solution.

4 Experimental evaluation

This section evaluates five agents and examines task characteristics associated with their success. It also reports one focused confirmation experiment.

4.1 Research questions

RQ1 (Capability). *How capable are current AI agents at natural-language-to-TLA+ formalisation under semantic evaluation?* Pass@3 measures whether one of three attempts records an equivalence pass. Attempt-level check survival shows which checks reject the outputs.

RQ2 (Task characteristics). *Which observable benchmark characteristics are associated with agent success or failure?* We analyse success multiplicity, pairwise solved-set overlap, differences between ever-solved and never-solved tasks, and protocol family.

RQ3 (Repeated-attempt robustness). *Do selected failures persist across new attempts, and which behaviour check rejects the outputs?* We run a focused experiment with one agent, Codex + GPT-5.6 Sol (§4.2). The experiment contains 11 tasks that Codex + GPT-5.6 Sol did not solve in the baseline experiment. For each selected task, the evaluator accepts a submission built directly from the published specification (Appendix D.3). We run 10 new attempts for each task and measure task success, outcome stability, and behaviour-check failures.

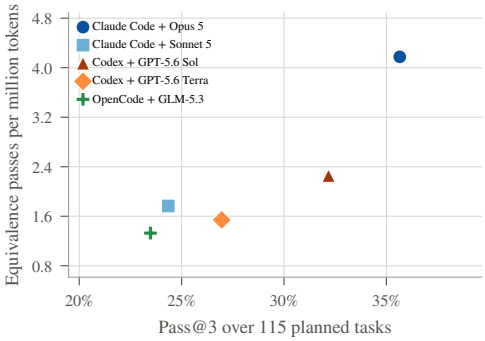
4.2 Shared experimental setup

Agents. We test Codex + GPT-5.6 Sol, Codex + GPT-5.6 Terra, Claude Code + Opus 5, Claude Code + Sonnet 5, and OpenCode + GLM-5.3. The two Codex agents use `max` effort. The other three agents use `high` effort.

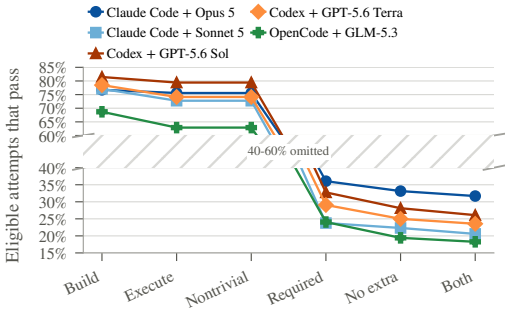
Each agent makes 3 attempts on each of the 115 tasks. This design gives 1725 planned attempts and 575 planned agent–task groups. We measure pass@3: a task passes a given check when at least one attempt passes that check. The executors do not set a temperature or expose a controlled sampling seed.

For RQ3, Codex + GPT-5.6 Sol uses `max` effort with the fixed evaluator. This experiment tests whether repeated attempts overcome earlier failures.

Limits and attempt policy. Each generation receives 5,400 seconds and 300 agent tool calls (Table 5). A 20,000,000-token safety limit stops a generation with abnormal token use. Each TLA+



(a) Pass@3 over all 115 planned tasks versus equivalence passes per million recorded tokens. Token totals include eligible failures. Compare token efficiency only within a provider.



(b) Check survival over eligible attempts. The broken y-axis expands 15–40% and 60–85% and omits 40–60%. G1–G3 form a chain. G4 and G5 independently test the same outputs that pass G1–G3. An equivalence pass satisfies all five checks.

Figure 4: RQ1: Semantic success remains low. The behaviour checks cause the largest attempt-level loss.

tool and scoring container receives 8 CPUs and 16 GB of memory. A generation timeout, tool limit, or invalid submission remains a failed attempt (Appendix C gives the retry policy).

Measures. An equivalence pass satisfies all five benchmark checks. We report each agent separately and do not pool them into one capability rate. Attempt-level check survival uses eligible attempts. All point estimates are descriptive statistics for this fixed benchmark. We report each pass@3 value with a 95% Wilson interval over the 115 planned task groups. The interval states the sampling uncertainty of one rate. We run no significance test between agents.

4.3 Results and Analysis

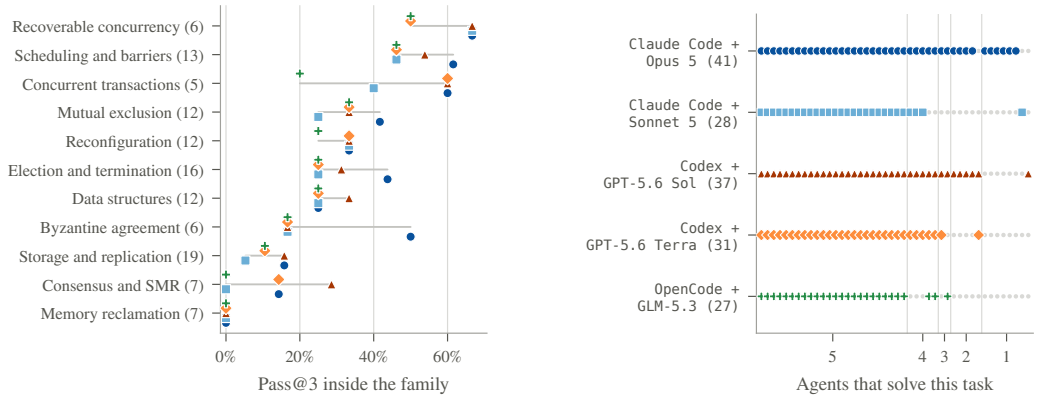
The public records include all 1725 planned attempts from the five agents. The attempt-level analysis includes 1723 eligible attempts. Of these attempts, package validation rejects 101 submissions, and 414 record an equivalence pass.

4.3.1 RQ1 (Capability)

End-to-end semantic success remains low at pass@3. Figure 4 reports pass@3 and attempt-level check survival for all five agents.

Codex + GPT-5.6 Sol solves 37 of 115 task groups (32.2%). Codex + GPT-5.6 Terra solves 31 (27.0%). Claude Code + Opus 5 solves 41 (35.7%). Claude Code + Sonnet 5 solves 28 (24.3%). OpenCode + GLM-5.3 solves 27 (23.5%). The five 95% Wilson intervals overlap, so this sample does not separate the agents (Table 6). Figure 4a also reports equivalence passes per million recorded tokens for each agent. The analysis includes 206,171,303 recorded tokens, and Appendix D.1 gives the values.

The behaviour checks cause the largest measured loss. G1 accepts 1318 attempts. G2 and G3 accept 1257 executable, nontrivial outputs. G3 rejects 0 attempts: every output that runs already reaches two or more states. Both behaviour checks pass for 414 attempts. Among the attempts that pass G1–G3, 728 both omit required behaviour and allow extra behaviour. Another 88 allow extra behaviour only, and 27 omit required behaviour only. Thus, nontrivial execution accepts 73.0% of eligible attempts, while bounded trace equivalence accepts 24.0%. Most outputs that fail the behaviour checks fail both directions. If the two directions failed for unrelated reasons, 490 attempts would fail both. The observed count is 728. A failing output therefore describes a different protocol, not a limited mismatch in one direction.



(a) Pass@3 by protocol family. Marks are agents; grey lines span them. Parentheses give task counts. Each denominator includes all planned tasks in that family.

(b) Solved-task overlap at pass@3. Columns are the 44 ever-solved tasks; filled marks denote success.

Figure 5: RQ2: Success concentrates on shared tasks. Pass@3 values differ, but their intervals overlap and the failures fall on shared tasks.

RQ1 answer. Pass@3 ranges from 23.5% for OpenCode + GLM-5.3 to 35.7% for Claude Code + Opus 5, with overlapping intervals. Most outputs that reach the behaviour checks do not match the hidden reference. A criterion that stops after nontrivial execution substantially overstates semantic success.

4.3.2 RQ2 (Task characteristics)

Success concentrates on a shared subset of tasks. Figure 5 shows protocol-family rates and task-level success overlap. Appendix D.2 reports success multiplicity, pairwise solved-set overlap, and observable task properties. The agents solve 44 distinct tasks in total. No agent solves 71 tasks. Exactly one agent solves 8 tasks, two agents solve 5, three agents solve 2, and four agents solve 5. All five agents solve 24 tasks.

Solved tasks have smaller observable interfaces. These shared outcomes also correspond to differences in observable task features. The median ever-solved task has 3 observable variables, 3 observable actions, and 10.5 required operators. The never-solved tasks have corresponding medians of 5, 4, and 12.

Potential contamination. Public sources create a risk of training-data contamination. Tasks from the TLA+ example repository form 68.2% of the ever-solved tasks and 35.2% of the never-solved tasks. These example-repository tasks have a pooled pass@3 rate about twice that of other tasks. Appendix D.2 provides details.

RQ2 answer. The agents succeed and fail on substantially overlapping task sets, and their pass@3 intervals overlap. No agent solves most tasks, and all five agents solve most of the solved tasks.

4.3.3 RQ3 (Repeated-attempt robustness)

Repeated attempts rarely solve the failed tasks. Of 575 agent–task groups, 411 have no successful attempt, and 298 of these fail the same check first in all three attempts. The confirmation experiment tests whether these failures persist across new attempts. Figure 6 shows the outcome of every confirmation attempt. Codex + GPT-5.6 Sol solves 1 of the 11 selected tasks in 110 new attempts.

Almost every output that reaches the behaviour checks allows extra behaviour. In the confirmation experiment, 88 attempts pass G1–G3 and reach the behaviour checks. The remaining attempts include 3 invalid submissions, 10 build failures, and 9 execution failures. Of the attempts that reach the behaviour checks, 87 fail the no-extra-behaviour check. Among these failures, 48 also fail the required-behaviour check, while 39 fail only the no-extra-behaviour check. This concentration

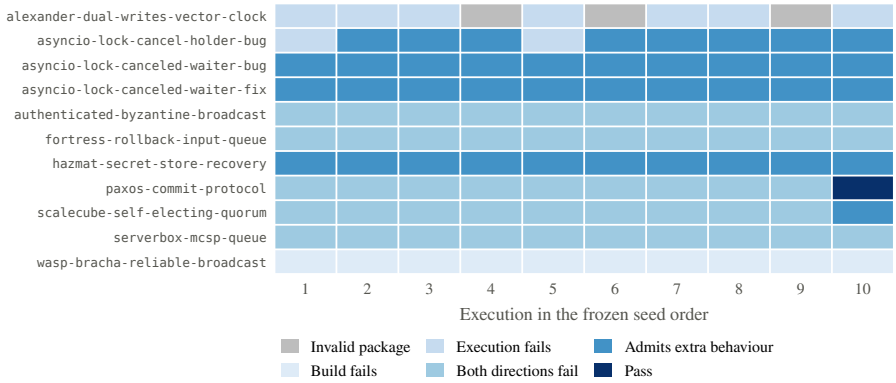


Figure 6: RQ3: Repeated attempts rarely solve the selected failed tasks. Each cell is one confirmation execution, grouped by task and frozen seed order. Colour gives the rejecting check; darker colours indicate more checks passed. One execution passes. Of the 88 executions that reach G4 and G5, 87 fail G5.

identifies control of extra behaviour as the dominant measured weakness. Appendix D.3 reports these results by the stability of the first failed check in the baseline experiment.

RQ3 answer. Repeated attempts rarely solve the failed tasks: the agent solves 1 of 11 selected tasks with 10 new attempts per task. Sampling variance in this experiment therefore does not fully explain the measured limit. This result applies only to this experiment and one agent.

5 Discussion and conclusion

This paper presents EQUINOX, a benchmark of 115 natural-language-to-TLA+ tasks derived from published executable specifications. A private evaluator accepts an output only when it satisfies bounded trace equivalence with the hidden reference. Three conclusions follow from the study.

Execution is necessary but not sufficient for successful formalisation. Nontrivial execution accepts 73.0% of eligible attempts, but only 24.0% satisfy bounded trace equivalence. Most executable, nontrivial outputs still omit required behaviour and allow extra behaviour. Execution-only checks therefore overstate semantic success. A benchmark or training loop needs a direct behaviour comparison to measure semantic equivalence. Future evaluations can add larger finite scopes, refinement mappings for hidden state, and liveness checks.

Different agents solve similar sets of tasks. At least one agent solves 44 tasks, and all five agents solve 24 of these tasks. The agents have overlapping pass@3 intervals and solve largely the same tasks. These results do not support a clear capability ranking between the agents. The never-solved tasks also expose more state and actions, but these task features vary together.

More attempts rarely resolve modelling errors. Codex + GPT-5.6 Sol solves 1 of 11 selected tasks in 110 new attempts. Among the 88 outputs that reach the behaviour checks, 87 allow extra behaviour. This result shows a capability gap for frontier AI agents in protocol formalisation.

AI USE STATEMENT

We used generative AI tools for three tasks in this work. First, AI agents drafted the natural-language prompts of the benchmark tasks from public source documentation. Second, AI agents wrote and refactored parts of the harness code and the analysis code. Third, AI agents drafted and edited parts of this paper.

We did not use generative AI tools to develop the acceptance test, to formulate the research questions, to design the experiments, or to interpret the results. The AI agents did not write or modify the imported specifications that define the scoring reference. We generated no synthetic data: every task comes from a public, human-written specification.

We reviewed all AI-assisted work. A curator reviewed each prompt draft against its source mapping. Appendix A states what that review does and does not establish. Automated tests and human review cover the harness code, and every number in this paper comes from an audited record set rather than from a language model. We take responsibility for the final content of this work, including the text, the claims, and the released artefacts.

ETHICS STATEMENT

Each task records its external source, revision, file hashes, and licence. We release all 115 evaluation tasks. We release 201 of 207 imported packages with complete licence records and withhold the other six. None of the six is an evaluation task. The benchmark team does not modify imported protocol transitions.

This work uses no human subjects or personal data. Each AI agent run has a 90-minute generation limit. Exhaustive TLC runs on one 8-core host use most of the compute. The evaluator runs untrusted submissions without network access, with a read-only root filesystem, reduced permissions, and resource limits. All sources are public, so the paper reports potential contamination as a limitation rather than claiming contamination-free tasks.

REPRODUCIBILITY STATEMENT

We designed the benchmark so that a reader can rebuild every reported number. Section 4 states the agents, the budgets, and the host that produced the runs. Appendix C states the pinned toolchain and the sandbox that isolates the evaluator. The supplementary archive holds the harness, the task packages, the audited run records, and the analysis code that writes each figure, table, and printed number. It also holds a claim-to-command map that names the exact command behind each claim. Each task package records the source repository, the pinned revision, the file paths, the file hashes, and the licence, so a reader can re-import the sources and repeat the derivation.

REFERENCES

Arslan Bisharat, Brian Ortiz, Eric Spencer, Khushboo Bhadauria, TaiNing Wang, George K. Thiruvathukal, Konstantin Laufer, and Mohammed Abuhamad. Can LLMs write correct TLA+ specifications? evaluating natural-language-to-TLA+ generation, 2026a. URL https://arxiv.org/abs/2606.05792. In Proceedings of the 21st International Conference on Software Technologies (ICSOFT 2026), pages 39–50.

Arslan Bisharat, Eric Spencer, Brian Ortiz, Khushboo Bhadauria, Mujtaba Nazari, Beatriz Santos, Anisa Ramos, TaiNing Wang, George K. Thiruvathukal, Konstantin Laufer, and Mohammed Abuhamad. TLA+-Bench: An execution-grounded benchmark and dataset for natural-language to TLA+ specification generation, 2026b. URL https://arxiv.org/abs/2607.23425.

Jialun Cao, Yaojie Lu, Meiziniu Li, Haoyang Ma, Haokun Li, Mengda He, Cheng Wen, Le Sun, Hongyu Zhang, Shengchao Qin, Shing-Chi Cheung, and Cong Tian. From informal to formal: Incorporating and evaluating LLMs on natural language requirements to verifiable formal proofs. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*,

pp. 26984–27003, 2025. doi: 10.18653/v1/2025.acl-long.1310. URL https://aclanthology.org/2025.acl-long.1310/.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. nl2spec: Interactively translating unstructured natural language to temporal logics with large language models. In *Computer Aided Verification (CAV)*, Lecture Notes in Computer Science, pp. 383–396. Springer, 2023. doi: 10.1007/978-3-031-37703-7_18. URL https://arxiv.org/abs/2303.04864.

Madeline Endres, Sarah Fakhoury, Saikat Chakraborty, and Shuvendu K. Lahiri. Can large language models transform natural language intent into formal method postconditions? *Proceedings of the ACM on Software Engineering*, 1(FSE):1889–1912, 2024. doi: 10.1145/3660791.

Thomas Hubert, Rishi Mehta, Laurent Sartran, et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651:607–613, 2025. doi: 10.1038/s41586-025-09833-y. URL https://doi.org/10.1038/s41586-025-09833-y.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

René Just, Darioush Jalali, and Michael D. Ernst. Defects4J: A database of existing faults to enable controlled testing studies for Java programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA)*, pp. 437–440, 2014. doi: 10.1145/2610384.2628055. URL https://doi.org/10.1145/2610384.2628055.

Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, 1994. doi: 10.1145/177492.177726.

Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002. ISBN 9780321143068.

Leslie Lamport. The PlusCal algorithm language. In *Theoretical Aspects of Computing (ICTAC)*, Lecture Notes in Computer Science, pp. 36–60. Springer, 2009. doi: 10.1007/978-3-642-03466-4_2.

Zhaoxuan Li, Qionglu Zhang, Hengyuan Liu, Xiaoyan Gu, Xianhui Lu, Hongbo Liu, Bingzheng Wang, Haihui Fan, Ziming Zhao, Rui Zhang, and Li Zhou. CrypFormBench: Benchmarking formal analysis capability of large language models for cryptographic schemes. *Proceedings of the ACM on Software Engineering*, 3(FSE):4025–4047, 2026. doi: 10.1145/3808184. URL https://doi.org/10.1145/3808184.

Xiaohan Lin, Qingxing Cao, Yinya Huang, Haiming Wang, Jianqiao Lu, Zhengying Liu, Linqi Song, and Xiaodan Liang. FVEL: Interactive formal verification environment with large language models via theorem proving. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2406.14408.

Mingjie Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. VerilogEval: Evaluating large language models for Verilog code generation. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–8. IEEE, 2023. doi: 10.1109/ICCAD57390.2023.10323812.

Qi Liu, Xinhao Zheng, Xudong Lu, Qinxiang Cao, and Junchi Yan. Rethinking and improving autoformalization: Towards a faithful metric and a dependency retrieval-based approach. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=hUb2At2DsQ.

Chloe Loughridge, Qinyi Sun, Seth Ahrenbach, Federico Cassano, Chuyue Sun, Ying Sheng, Anish Mudide, Md Rakib Hossain Misu, Nada Amin, and Max Tegmark. DafnyBench: A benchmark for formal software verification, 2024. URL https://arxiv.org/abs/2406.08467.

Ziyu Mao, Jingyi Wang, Jun Sun, Shengchao Qin, and Jiawen Xiong. LLM-aided automatic modeling for security protocol verification. In *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering*, pp. 642–654, 2025. doi: 10.1109/ICSE55347.2025.00197. URL https://doi.org/10.1109/ICSE55347.2025.00197.

Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. How Amazon web services uses formal methods. *Communications of the ACM*, 58(4):66–73, 2015. doi: 10.1145/2699417.

Auguste Poiroux, Gail Weiss, Viktor Kunčak, and Antoine Bosselut. Reliable evaluation and benchmarks for statement autoformalization. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025. URL https://arxiv.org/abs/2406.07222.

André Silva, Nuno Saavedra, and Martin Monperrus. GitBug-Java: A reproducible benchmark of recent Java bugs. In *Proceedings of the 21st International Conference on Mining Software Repositories (MSR)*, pp. 118–122, 2024. doi: 10.1145/3643991.3644884. URL https://doi.org/10.1145/3643991.3644884.

Eric Spencer, Arslan Bisharat, Brian Ortiz, Khushboo Bhadauria, Mujtaba Nazari, TaiNing Wang, George K. Thiruvathukal, Konstantin Laufer, and Mohammed Abuhamad. TLA-Prover: Verifiable TLA+ specification synthesis via preference-optimized low-rank adaptation, 2026. URL https://arxiv.org/abs/2606.06133. In Proceedings of the 21st International Conference on Software Technologies (ICSOFT 2026), pages 627–636.

Amitayush Thakur, Jasper Lee, George Tsoukalas, Meghana Sistla, Matthew Zhao, Stefan Zetzsche, Greg Durrett, Yisong Yue, and Swarat Chaudhuri. CLEVER: A curated benchmark for formally verified code generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://arxiv.org/abs/2505.13938.

David A. Tomassi, Naji Dmeiri, Yichen Wang, Antara Bhowmick, Yen-Chuan Liu, Premkumar T. Devanbu, Bogdan Vasilescu, and Cindy Rubio-González. BugSwarm: Mining and continuously growing a dataset of reproducible failures and fixes. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)*, pp. 339–349, 2019. doi: 10.1109/ICSE.2019.00048. URL https://doi.org/10.1109/ICSE.2019.00048.

Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625:476–482, 2024. doi: 10.1038/s41586-023-06747-5. URL https://doi.org/10.1038/s41586-023-06747-5.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.850. URL https://aclanthology.org/2024.acl-long.850/.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2404.07972.

Yuan Yu, Panagiotis Manolios, and Leslie Lamport. Model checking TLA+ specifications. In *Correct Hardware Design and Verification Methods (CHARME)*, Lecture Notes in Computer Science, pp. 54–66. Springer, 1999. doi: 10.1007/3-540-48153-2_6.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. miniF2F: A cross-system benchmark for formal olympiad-level mathematics. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://arxiv.org/abs/2109.00110.

A LIMITATIONS

Acceptance scope. The criterion checks equivalence with one published specification, at one finite configuration, over one observable interface. It does not check larger scopes, hidden state, fairness, termination, or every reasonable interpretation of the prompt. A pass establishes bounded agreement, not unbounded correctness. A failure can also reflect an ambiguous description.

Prompt sufficiency. Two factors can cause a task failure, and this study does not separate them. The agent may lack the capability, or the prompt may underdetermine the protocol. The positive controls test the evaluator with a rewritten source, so they establish that a correct submission passes. They do not establish that the prompt alone determines that submission. A curator reviews the prompt and its source mapping when no submission passes a task, but the paper reports no measured upper limit on task difficulty. An expert baseline or an oracle run that also reads the hidden source would bound this explanation. We leave that measurement to future work, so every failure count in this paper is an upper bound on agent incapability.

Dataset and association scope. Every source is public, so the benchmark cannot establish that a model did not see a reference during training. Protocol family, interface size, domain, and source group vary together. The observed splits measure associations in this task set and do not identify causes.

Experimental scope. The confirmation result applies to 11 tasks. Codex + GPT-5.6 Sol made three initial attempts on each task and passed none of them. For each task, the evaluator accepts a submission built directly from the published specification. The confirmation experiment uses one agent (Codex + GPT-5.6 Sol), one evaluator, and one set of limits. The run seed identifies an attempt. It does not control model sampling. The result does not estimate the success rate for all tasks. It does not isolate one system component or rank model families. It identifies the failed public check, not the source-level modelling error.

Evaluator coverage. Bounded equivalence is complete only for the declared finite state space and limits. A timeout or evaluator error is not evidence of protocol mismatch. The attempt-level analysis keeps a resource timeout and excludes an evaluator infrastructure error. The pass@3 analysis retains all 575 planned agent–task groups. It records no success for an errored attempt. Thus, the 2 evaluator errors can lower pass@3, but they cannot shrink its 115-task denominator. A rerun could increase a rate. Positive and negative controls test both behaviour directions, and they cover about two thirds of the tasks. Of 115 positive controls, 79 reach both behaviour checks, and all 79 pass. The source rewriter produces no verdict for the other 36 positive controls. The evaluator rejects all 158 decided negative controls and accepts 0. The same rewriter limit produces no verdict for the other 72 of 230 negative controls. The rewriter therefore decides 68.7% of the negative controls. A no-verdict control is a rewriter limit, not an evaluator failure, but it leaves the remaining tasks without control evidence.

B ADDITIONAL DETAIL FOR BENCHMARK DESIGN

This section provides supporting detail for §3. It describes task admission, source independence, and evaluator isolation.

B.1 DATASET ADMISSION AND CURATION

Every task starts from an external project. A curator imports a published root module, its dependencies, its documentation, its configurations, and its properties. The curator can add a generated namespace, an adapter, and a finite configuration that transcribes a source scope. The curator cannot add a new bound to make a source run and cannot add or repair protocol transitions.

Curation uses three different execution checks. The source screening run applies a 120-second limit to the selected source module and performs 13 legal, execution, scope, and integrity checks. The dataset diagnostic then runs every committed package with 600 seconds per configuration and records the `bounded_execution` property. An evaluation task uses the separate per-task limits in its manifest. A package can satisfy one check and fail another, so the records keep these results separate.

The 13 source checks group into four families:

Legal. The exact revision has an explicit licence record with its identifier, path, URL, and file hash.

Execution. The package has a complete executable TLA+ root. The root can contain a PlusCal algorithm and its TLA+ translation. The package also has a complete dependency graph with unique module names, a source-selected finite scope, a meaningful property beyond `TypeOK`, an immutable revision, and a successful bounded source run.

Scope. The source models concurrent or distributed behaviour, has an independent history, and has sufficient documentation for a standalone prompt that contains no TLA+ fragment or hidden assumption.

Integrity. The benchmark does not repair the protocol, and the source has auditable independent authorship. A source fails when an evaluated model family authored its TLA+ or configuration commits or when a fresh mirror hides the original path history.

Each failed check receives a stable defect code. The inventory retains the negative result. Helper, adapter, and refinement modules do not qualify as behaviour roots.

The three property states keep different results separate. `satisfied` means that a current hash-linked analysis supports a property. `not_satisfied` means that such an analysis refutes it. `unknown` means that the available record cannot settle it. Table 2 gives the complete property matrix. A total of 118 packages satisfy bounded execution. One lacks a complete private source-state alias map. The campaign excludes two more because their bounded behaviour comparison is `not_satisfied`. It therefore selects 115 tasks.

Table 2: Suitability states for 207 packages; each row sums to 207. The 15 satisfied behaviour-comparison records include 11 from the confirmation experiment. Evaluator filters yield 115 tasks.

Property	Satisfied	Not sat.	Unknown
`source_record`	201	0	6
`bounded_execution`	118	81	8
`bounded_behavior_comparison`	15	2	190
`task_definition`	205	1	1

The diagnostic uses the pinned TLA+ Tools 1.7.4, a 60-second SANY timeout, eight TLC workers, 16 GB of memory, and 600 seconds of TLC per configuration. Of the 89 packages that do not satisfy bounded execution, 81 have a measured refutation and eight remain unknown. Six unknown packages declare no observable interface. One reaches its TLC timeout before that check. One has a stale bounded record after its source import changed.

Table 3 applies the diagnostic causes in order and counts each package once. The 32 timeouts depend on the declared budget and machine. The remaining 57 outcomes do not change with a larger timeout. Seventeen packages attempt no TLC configuration: 14 declare no finite configuration, and 3 fail at SANY first. The fourth SANY-failure package attempts its declared TLC configurations, so it is not among the seventeen.

Table 3: First cause that blocks bounded execution for 89 sources. The deadlock/property row contains three deadlocks and one safety violation.

First blocking cause	Packages
TLC timeout at 600 s	32
Required action not reached	16
No observable interface declared	6
No finite configuration to run	14
Entry-module or configuration error	12
Deadlock or property violation	4
SANY failure	4
Stale bounded-execution record	1
Total	89

A positive control does not determine whether a task enters the benchmark. A minimal `INSTANCE` wrapper can prevent the control rewriter from building an agent output. An incorrect initial-state or next-state operator in the task record can instead make the rewritten source disagree with itself. The first case is a rewriter limit, and the second is a dataset defect. Scoring uses the pinned source directly and does not use the rewriter.

B.2 PROMPT INDEPENDENCE

A prompt author uses an AI agent to draft a prompt from source documentation. The source can include a README, a design document, or a paper. The author reviews the draft and records a source-to-sentence mapping for each prompt requirement. The author does not copy TLA+ text, comments, module identifiers, citations, protocol names, or distinctive source identifiers into the scored prompt. Generated aliases hide source names while preserving the observable contract. These checks establish prompt independence from the hidden specification. They cannot establish that a public source was absent from model training data.

A scored prompt uses nine sections: Model, State, Communication, Transitions, Faults, Fairness, Properties, Observable Interface, and Deliverables. Figure 7 shows one evaluation task. The release check normalises identifiers, removes names and citations, and compares short prompt phrases with source documentation, papers, public specifications, and public TLA+ datasets. A curator performs this comparison manually.

```
## Model
Model a nonempty unidirectional ring of N positions with distinct natural identities. Any subset may initiate
election; the smallest initiating identity must win.

## State
Expose set inboxes V2, per-node V3, Boolean V1, and election V4 in candidate/lost/won. Initial inboxes are empty;
initiators are candidates and other nodes lost.

## Transitions
Initiators first send themselves. A lost node forwards received identities. A candidate forwards smaller identities
and loses, discards larger ones, and wins on receiving itself.

[. . . ]

## Observable Interface
Constants: Id, N. Variables: V1–V4. Actions: A1, A2(self), A3(self), A4(self). Operators: Init, Next, O1–O8,
O9(n), Spec.
```

Figure 7: Excerpt from an aliased nine-section prompt. It contains no TLA+ fragment, protocol name, or source identifier. The hidden reverse map restores source names during evaluation.

Prompt sufficiency is a separate release question. If no submission passes a task, the curators examine the prompt and its documentation mapping before they classify the task as difficult. Reporting potential contamination as a limitation does not replace these prompt controls.

B.3 EVALUATOR ISOLATION

The executor stages only the prompt, public task files, tool wrappers, and the pinned TLA+ Tools JAR in a randomly named workspace outside the repository. Generation ends before the evaluator mounts the hidden source. The agent process cannot read or write the repository.

The G1 build check generates an interface module for the agent output. It asks SANY to resolve every required action and operator at its declared arity. The same interface defines the observable projection for both behaviour directions. An agent output can therefore use a different internal representation without changing the comparison contract.

Every TLC invocation uses its declared timeout. The execution check uses the host worker setting. Each two-model behaviour comparison uses one TLC worker to avoid contention for local TLC resources. The container launcher derives an outer deadline from the three declared TLC limits and a cleanup margin. The outer deadline cannot preempt an inner stage that remains within its deadline.

A package validator rejects symbolic links, path traversal, undeclared files, an incorrect entry module, and filename collisions with the evaluator. The scoring container has no network access, drops Linux

capabilities, and bounds its processes, CPUs, and memory. Its image fixes Python 3.12.11 and Java 21 by image hash. Public results retain tool versions, image hashes, limits, and verdicts. They remove source module names, private paths, hidden expressions, TLC traces, and counterexample states.

B.4 Source projects and task packages

The 207 committed packages import 52 repositories. The 115 evaluation tasks come from 36 of those repositories. Each manifest records the repository URL, the immutable revision, every imported path and SHA-256 hash, and the licence record. Validation recomputes the hashes.

The TLA+ example repository supplies 83 packages. All three official TLA+ example repositories together supply 90 packages. The top three repositories supply 102 packages, and most repositories supply at most two packages each. A total of 37 packages import a separately licensed external source, and each dependency has its own revision, file list, hashes, and licence record. Section D.2 reports the narrower source split over the 115 evaluation tasks.

Table 4 reports the scale of the imported specifications and the prompts. The root can be a minimal model-checking adapter, so the table reports both root-module lines and all primary-source lines.

Table 4: Task-package scale. Prompt statistics cover the 205 packages with a satisfied task definition. Source statistics exclude separately licensed external modules.

Measure per package	Median	Maximum
Root-module lines	192	2,293
All primary-source lines	360	4,467
Primary-source bytes	14,876	218,918
Prompt words	420	1,621

The collection contains 114,225 primary-source TLA+ lines.

Of the 207 packages, 195 have a permissive or dual-permissive licence, and six have another recorded licence. The other six packages come from a published archive with no licence record. The release withholds those six, and none enters the evaluation set. The source table retains their archive SHA-256 so that the private source record remains reproducible.

Table 15 lists each source repository and pinned revision. Table 16 lists the shared TLA+ module libraries. Table 17 lists every committed task package, its root module, its protocol family, and whether it satisfies bounded execution. A reader can therefore map a public task record to its source repository and exact revision.

C Reproducibility details

This section states the attempt policy, analysis units, token measure, and released evidence for the experiment in §4.

C.1 Attempt resources and retries

Table 5: Per-attempt limits. The token limit stops abnormal generation; it does not define success.

Limit	Value
Generation wall clock	5,400 s
AI agent tool calls	300
Token safety limit	20,000,000
TLA+ tool or scoring container	8 CPUs, 16 GB

Claude Code reports usage between model calls. Codex reports usage in its terminal event.

The runner retries a provider or command-line failure that occurs before the final generation record. The retry uses the same run identity and does not create another sample. A generation timeout, tool limit, or invalid submission remains the final attempt.

C.2 Analysis units and denominator

The attempt-level analysis uses the 1723 attempts that return an evaluator verdict. An evaluator infrastructure error produces no attempt-level verdict. The record keeps each error and its attempt identity.

Pass@3 uses one planned agent–task group as its unit. It retains all 115 groups for each agent. A group is solved when any of its three attempts records an equivalence pass. An errored attempt records no success, but it does not remove its group. Thus, every agent rate has the same denominator of 115.

C.3 Token-efficiency measure

For agent a, let S_a be its equivalence passes and T_a be the recorded tokens from all of its eligible attempts. Token efficiency is

$$E_a = 10^6 \frac{S_a}{T_a}.$$

The total includes fresh input, cache creation, and output. It excludes cache reads. Failed and invalid eligible attempts remain in T_a. We report this measure for each agent. We do not combine token efficiency across providers.

C.4 Released evidence

The public records store all 1725 planned run records and a hash for each scored input. They also store the sanitised score process, control, analysis, and audit records for the four Codex and Claude Code agents. The OpenCode + GLM-5.3 record stores its run matrix and generation checkpoint. These records pin the task and submission digests, tool versions, container image, limits, usage, and public verdicts. They omit private TLC traces, hidden module names, private evaluator paths, and credentials. A reader can score a saved agent output again without a new generation run. The paper analysis uses all planned attempts.

The archived run rows also retain old derived token fields for source-record continuity. This paper derives success from the five public checks. It does not use those old fields. Appendix D gives the supporting result tables.

D Supporting results

This section follows the three research questions in §4. It first reports agent outcomes, then task characteristics, and then repeated-attempt outcomes. Every pass@3 rate uses 115 planned tasks.

D.1 RQ1: End-to-end capability

Agent outcomes. Table 6 gives the attempt counts and pass@3 result for each agent. Each agent makes three attempts on each task. The "Solved" column therefore has the fixed denominator 115.

Table 6: Pass@3 and attempt outcomes for the five agents. The interval is a 95% Wilson interval over the 115 planned task groups.

Agent	Runs	Invalid	Eval. errors	Equivalence passes	Solved/115	Pass@3	95% interval
Codex + GPT-5.6 Sol	345	4	0	90	37/115	32.2%	[24.3, 41.2]
Codex + GPT-5.6 Terra	345	3	1	81	31/115	27.0%	[19.7, 35.7]
Claude Code + Opus 5	345	23	1	109	41/115	35.7%	[27.5, 44.7]
Claude Code + Sonnet 5	345	23	0	71	28/115	24.3%	[17.4, 32.9]
OpenCode + GLM-5.3	345	48	0	63	27/115	23.5%	[16.7, 32.0]

The five intervals overlap, so this sample does not separate the agents. The five agents record 414 equivalence passes from 1725 attempts. Codex + GPT-5.6 Sol produces 2.26 such passes per million recorded tokens. Codex + GPT-5.6 Terra produces 1.54. Claude Code + Opus 5 produces 4.17. Claude Code + Sonnet 5 produces 1.77. OpenCode + GLM-5.3 produces 1.33. Provider token records use different reporting conventions.

Check survival. Table 7 gives the cumulative attempt counts at the main evaluation stages. The behaviour checks cause the largest measured loss.

Table 7: Attempt-level check survival for the five agents.

Cumulative criterion	Codex + GPT-5.6 Sol	Codex + GPT-5.6 Terra	Claude Code + Opus 5	Claude Code + Sonnet 5	OpenCode + GLM-5.3
G1 (package builds)	281	270	264	266	237
G1–G3 (executes, multiple states)	274	255	260	251	217
G1–G5 (equivalence pass)	90	81	109	71	63

D.2 RQ2: Task characteristics

The evaluation set has 115 tasks in 11 families. It contains 55 concurrent tasks and 60 distributed tasks. The following tables report shared difficulty and observable task properties. Figure 5a gives the secondary protocol-family view.

Success concentration. Table 8 counts each task once according to the number of agents that solve it.

Table 8: Tasks by number of solving agents.

Successful agents	0	1	2	3	4	5
Tasks	71	8	5	2	5	24

Pairwise overlap. Table 9 gives the intersection and Jaccard similarity for every pair of solved task sets.

Table 9: Solved-set overlap. Each cell gives the intersection and Jaccard similarity.

	Codex + GPT-5.6 Sol	Codex + GPT-5.6 Terra	Claude Code + Opus 5	Claude Code + Sonnet 5	OpenCode + GLM-5.3
Codex + GPT-5.6 Sol	37 (100%)	31 (83.8%)	35 (81.4%)	27 (71.1%)	27 (73.0%)
Codex + GPT-5.6 Terra		31 (100%)	30 (71.4%)	27 (84.4%)	26 (81.2%)
Claude Code + Opus 5			41 (100%)	27 (64.3%)	27 (65.9%)
Claude Code + Sonnet 5				28 (100%)	24 (77.4%)
OpenCode + GLM-5.3					27 (100%)

Observable properties. Table 10 compares tasks that at least one agent solves with tasks that no agent solves.

Table 10: Properties of ever-solved and never-solved tasks. Values are medians unless shown as a count and rate.

Property	Ever solved (44)	Never solved (71)
Observable variables	3	5
Observable actions	3	4
Required operators	10.5	12
From the example repository	30 (68.2%)	25 (35.2%)
Concurrent task	25 (56.8%)	30 (42.3%)

Potential contamination. Every source is public, so the benchmark reports potential contamination instead of claiming that the tasks are contamination-free. Table 11 splits the tasks by source group and by interface size. Pass@3 pooled over the five agents is 38.9% for the TLA+ example repository and 19.0% for the other sources, a gap of 19.9 points. Interface size gives a larger gap of 29.0 points. The example-repository tasks also tend to have smaller interfaces, so the two features are related. The last four rows compare source groups within each interface-size group. Within these groups, the source gap falls from 17.1 points for small tasks to 7.2 points for large tasks. The interface-size gap is larger than the source-group gap. This task set cannot isolate the two effects.

Table 11: Pass@3 by source group and interface size, pooled over the five agents. A small interface declares at most four observable variables. The last four rows hold interface size fixed.

Task group	Tasks	Pass@3
Example repository	55	38.9%
Other source	60	19.0%
Small interface	57	43.2%
Large interface	58	14.1%
Small, example repository	36	49.4%
Small, other source	21	32.4%
Large, example repository	19	18.9%
Large, other source	39	11.8%

Protocol-family context. The complete collection has 207 packages in 12 protocol families. The evaluation set has 115 tasks in 11 families. The family inventory keeps collection counts, bounded-execution counts, and the evaluation set distinct. Each family rate in Figure 5a uses all planned tasks in that family for one agent.

Table 12: The collection's 12 protocol families. `Domain` is distributed (`dist.`) or concurrent (`conc.`). `Committed` counts imported packages; `Bounded` counts packages that satisfy bounded execution. Further filters yield 115 evaluation tasks. Table 17 names each package's family.

Protocol family	Domain	Committed	Bounded
Consensus and SMR	dist.	23	8
Byzantine agreement	dist.	17	7
Election and termination	dist.	27	16
Storage and replication	dist.	34	19
Reconfiguration	dist.	25	12
Mutual exclusion	conc.	17	13
Data structures	conc.	21	12
Scheduling and barriers	conc.	14	13
Memory reclamation	conc.	9	7
Recoverable concurrency	conc.	8	6
Weak memory	conc.	6	0
Concurrent transactions	conc.	6	5
Total	—	207	118

D.3 RQ3: Repeated-attempt outcomes

Table 13 counts agent–task groups by the number of successful attempts. Each row sums to 115.

Table 13: Agent–task groups by number of successful attempts.

Agent	0	1	2	3
Codex + GPT-5.6 Sol	78	8	5	24
Codex + GPT-5.6 Terra	84	3	6	22
Claude Code + Opus 5	74	4	6	31
Claude Code + Sonnet 5	87	3	7	18
OpenCode + GLM-5.3	88	8	2	17

For Codex + GPT-5.6 Sol, the first rejection is stable in 68 of the 78 zero-pass groups. For Codex + GPT-5.6 Terra, it is stable in 66 of the 84 zero-pass groups. The corresponding count is 55 of 74 for

Claude Code + Opus 5. It is 58 of 87 for Claude Code + Sonnet 5. For OpenCode + GLM-5.3, the count is 51 of 88. These counts use all 115 planned task groups for each agent. The record reports evaluator errors separately.

Confirmation experiment. Before the confirmation experiment, we build a submission directly from each published specification. We include a task only if the evaluator accepts this submission with the fixed evaluator image and limits. Table 14 reports the two baseline groups of the confirmation experiment. The stable-check group contains 8 tasks with one baseline rejecting check. The mixed-check group contains 3 tasks with different baseline rejecting checks.

Table 14: Confirmation outcomes by baseline group.

Baseline group	Tasks	Evaluable	Consistent check
Stable check	8	8	6
Mixed check	3	3	2

The stable-check count requires every failed confirmation attempt to stop at the original baseline check. The mixed-check count requires every failed confirmation attempt to stop at the same check. Both counts require at least two failed confirmation attempts. Across both groups, 10 tasks never pass and 1 tasks pass at least once.

Across both groups, 88 attempts reach both behaviour checks. Of these, 87 fail the no-extra-behaviour check, and 48 also fail the required-behaviour check.

The fixed plan records each task package digest. The public confirmation rows lack a verifiable task digest for 3 attempts. The result record reports this count instead of assuming that each row contains a digest.

D.4 Record status

The released record contains the public source commit and the raw-result commit. It contains the complete process, retry, control, analysis, and audit records for the four Codex and Claude Code agents. The OpenCode + GLM-5.3 record contains the complete public run matrix and generation checkpoint. The paper analysis validates the five run matrices together. The separate RQ3 result record contains the hashes of the fixed plan, the baseline record, and the confirmation record.

Table 15: Sources for 207 packages: 52 projects at 52 pinned revisions. The table has 53 rows because one revision has two licenses. Links identify Git commits; the unlinked 64-character value hashes a published archive. Packages counts imports. Manifests pin paths, file hashes, and license URLs.

Project	Pinned revision	Packages	License
aio-libs/aiorwlock	62c1978c406fba5517b26f1b12d346b7579104c4	1	Apache-2.0
Alexander-N/tla-specs	e17070a97205ced3b306bfb80268517781578b7c	4	MIT
aneoconsulting/ArmoniK.Spec	167ed97711558100a6ba23ef94a7d213f2c2ad14	5	Apache-2.0
apache/zookeeper	53a78e36f99602162643303ab330377fc383b41f	3	Apache-2.0
Azure/unbounded	158bc99c9084edfb9135fc25375a623c2e46e00b	5	MIT
cactusdynamics/cactus-rt	1ac871c818d4698684461e9fdbb6555c59cf549b	1	MPL-2.0
cartesi/sequencer	6dc4f59615830943f9f56e01005ca8f062e81171	2	Apache-2.0
cometbft/cometbft	8715446eae412602c6e5fccd96317452d2943929	6	Apache-2.0
cwi-swat/tla-ci	da10d932270879f5e9d21eb3ed192b2d57d99109	1	BSD-2-Clause
danwt/demo-chain-replication	fafaf3581629a9675396a4d0a3bca2cb02aa3478	1	GPL-3.0-only
danwt/demo-primary-backup-replication	6859293ce7c1fcd2df0960323ac0cf3a86f96e1c	1	GPL-3.0-only
deepcausa/safeatomic	4905a74334eb7fd0c93e292866d754047718f010	2	MIT
dfinity/formal-models	09151cf5ea555afe62c5055b4188cfc54c7f98bf	10	Apache-2.0
dgpv/SASwap_TLAplus_spec	82674da3f5d5d8c06a9e1ea8c79a23f25191566d	1	CC-BY-SA-4.0
dredozubov/hazmat	34e57a3e5ab3edad462a70b191c5032832c2d206	1	MIT
Entrolution/bengal-stm	ef6222500e4ffa62a7361aed74cd03bd514e0ce4	1	Apache-2.0
etcd-io/raft	26647d57a34382bc60457ad508cea835ed9b52da	1	Apache-2.0
Gerold103/serverbox	1845eb09c528f131e25b7c250ef5d390a2078701	2	MIT
iotaledger/wasp-legacy	856bf7f214ebd3d95c737cfdf45db9d16afd02f3	3	Apache-2.0
JoshuaRowePhantom/Phantom.Coroutines	d2cb5da4abd55d360c619928fd5704bf74ca3325	1	MIT
JuliaLang/julia	e9f1de4f6903c359ca0ecd506cbab7ae12998aa1	1	MIT
lamport.org/tla/wildfire-challenge.zip	d169d5b1e541741bba275f40de4b68b1adf10857 9f8c0a4b8b5d94116153d69a	6	license_pending_private_only
LeaveSHY/Evolv-BFT-Artifact	6847f6fc062c0c9f05bb2d173d8c0b23c92941a0	1	Apache-2.0
lemmy/BlockingQueue	68715979666344cbcc66fa621b27039f281bdd5c	4	MIT
linkedin/venice	2534e6acd6076ac000a8fc9b2b6d98f7e5724bd5	1	BSD-2-Clause
lorin/snapshot-isolation-tla	5a6cd3d3b72f063784f8847731e28f291208231e	1	MIT
lorin/tla-prophecy	764fd974441d78594b86aaa32a6870eb58e0ca2c	2	Apache-2.0
microsoft/CCF	ded6766afb0dbe0604f70a1e67c5ac29a158a162	4	Apache-2.0
microsoft/CCF	ded6766afb0dbe0604f70a1e67c5ac29a158a162	1	CC-BY-4.0
mongodb/mongo	ddcda7a3b670c3d3b7cbc08f912723fa3d2ba7af	1	CC-BY-3.0-US
nikolskiy/learn-tla	d0ae80af8ed900fa3fb10fab35b401383d605e93	1	MIT
northriv/KAME	0cd6abf5a65baa5d99267e6cb7b268e0a120dccc	7	Apache-2.0 OR GPL-2.0-or-later
obsidiandynamics/spire-tla	bd241558169cb70f95bb97aed620c9160b932733	2	BSD-3-Clause
oxarbitrage/zcash-p2p-spec	c0bc981c64af39a978d0b4d709d27c5c3cca8172	1	MIT
pduarte21/tla-plus-by-example	e95ed0ca7925fd307a3b7dc3475519e5190d0ec6	2	MIT
peterzeller/TLA-broadcast-algorithms	78fc568257491910df9ad35909c989938b094f64	1	Apache-2.0
pingcap/tla-plus	8ae629ddf9f5ad8efc763bff93e3e43ce0a3585d	5	Apache-2.0
plasma-umass/scalene	c2e9e17b2ae499cdcdf549a0f12862d8307634f5	2	Apache-2.0
polikow/bully_election	84a9ff5da170ca4966fc0e0e3fa1a02098e55e44	1	MIT
rs3lab/kernel-tla	c8aff149b069ec39e1f676166bcd6285b222c227	1	GPL-2.0
scalecube/scalecube-prism	8c78f56f1bae7b6876be7b8eb2d180306518ded9	2	Apache-2.0
stephanfr/minimalstdlib	2602e4eb394ce4644a4d50c786228a18ffe01a3b	1	BSD-3-Clause
tlaplus/azure-cosmos-tla	2cfd1c34cd2937f8ba2344f9285936bbb793a4a4	1	MIT
tlaplus/Examples	47b0e2cc0268836b89f5ce451f38e5df5f1cf773	83	MIT
Tuntii/KayaDB	4fcf14761e803a53223f40a36ac131dce831cb9c	1	Apache-2.0 OR MIT
tursodatabase/turso	5e17d8ea0287de98f08b3cee11afbeab78530348	1	MIT
ubisoft/task-scheduler	77e74e7b3e3c783d9c47955deadf158fcc183f39	2	MIT
unodb-dev/unodb	7f2e05b3a1d14b9fd2742a92b86778c1a5bf04f8	1	Apache-2.0
Vanlightly/bookkeeper-tlaplus	fe199dc717b5723f5308aca74eda6ab3b03917e5	1	MIT
Vanlightly/kafka-tlaplus	823615e8a0962d0160c11d9affb2576c56b7c054	4	MIT
Vanlightly/raft-tlaplus	69ad9d6092f1ff32536726e4dd945e306bc1b73d	9	MIT
Vanlightly/vsr-tlaplus	7566e8af7bb4f5e323bb5601340c7d8192a1fab8	3	MIT
wallstop/fortress-rollback	224f325bb22405c59c4826d1fb276d77b39d5429	1	MIT

Table 16: Shared TLA+ libraries imported beyond the primary sources. Revision links identify Git commits. Packages counts imports, so counts overlap with Table 15.

Project	Pinned revision	Packages	License
tlaplus/CommunityModules	a20cc8eb2ee6e7140848ca511040cf283e9d7256	15	MIT
tlaplus/CommunityModules	c2f2c77b1a399e56875d52afd92f4ffd1acb43f2	18	MIT
tlaplus/tlaplus	30cc3601321c3fc02e044d0ecb5c58d8921e18df	4	MIT
tlaplus/tlapm	fd3988fc4e930565e0fa118158055448673dbcd2	11	BSD-2-Clause

Table 17: The 207 packages, grouped by source project. `Root module` gives the file name; manifests pin full paths and dependency hashes. `Protocol family` also fixes the domain. `Bounded` marks the 118 packages that satisfy bounded execution. Behavior-suitability and alias-map filters yield 115 evaluation tasks. A dash marks a recorded defect. Table 15 gives source revisions and licenses. Continued over 4 pages.

Task package	Root module	Protocol family	Bounded
aio-libs/aiorwlock			*1 package*
aiorwlock-current-production	aiorwlock.tla	Mutual exclusion	—
Alexander-N/tla-specs			*4 packages*
alexander-dual-writes-vector-clock	dual_writes_vector_clock.tla	Election and termination	yes
asyncio-lock-cancel-holder-bug	step1.tla	Mutual exclusion	yes
asyncio-lock-canceled-waiter-bug	step2.tla	Mutual exclusion	yes
asyncio-lock-canceled-waiter-fix	step3.tla	Mutual exclusion	yes
aneoconsulting/ArmoniK.Spec			*5 packages*
armonik-object-processing-1	ObjectProcessing1.tla	Reconfiguration	yes
armonik-object-processing-2	ObjectProcessing2.tla	Reconfiguration	yes
armonik-object-processing-3	ObjectProcessing3.tla	Reconfiguration	yes
armonik-session-processing-1	SessionProcessing1.tla	Reconfiguration	yes
armonik-task-processing-1	TaskProcessing1.tla	Reconfiguration	yes
apache/zookeeper			*3 packages*
zookeeper-fast-leader-election	FastLeaderElection.tla	Election and termination	—
zookeeper-system-3-7	ZkV3_7_0.tla	Reconfiguration	—
zookeeper-zab	Zab.tla	Consensus and SMR	—
Azure/unbounded			*5 packages*
azure-bufferpool-singleflight-recovery	BufferpoolSingleflight.tla	Recoverable concurrency	—
azure-bufferpool-wake-before-release-wedge	BufferpoolSingleflight.tla	Recoverable concurrency	—
azure-cow-btree-crash-consistency	CowBtreeCrash.tla	Recoverable concurrency	yes
azure-engine-reclamation-recovery	EngineReclamation.tla	Recoverable concurrency	yes
azure-fabric-completion-recovery	FabricCompletion.tla	Recoverable concurrency	yes
cactusdynamics/cactus-rt			*1 package*
cactus-double-buffer	double_buffer.tla	Data structures	yes
cartesi/sequencer			*2 packages*
cartesi-optimistic-batch-recovery	optimistic.tla	Storage and replication	—
cartesi-preemptive-batch-recovery	preemptive.tla	Storage and replication	—
cometbft/cometbft			*6 packages*
cometbft-light-client-accountability	TendermintAcc_004_draft.tla	Byzantine agreement	—
cometbft-light-client-attacks	LCVerificationApi_003_draft.tla	Byzantine agreement	—
cometbft-light-client-detection	LCDetector_003_draft.tla	Byzantine agreement	—
cometbft-light-client-supervisor	supervisor_001_draft.tla	Election and termination	—
cometbft-light-client-verification	Lightclient_003_draft.tla	Byzantine agreement	—
cometbft-proposer-based-timestamp	TendermintPBT.tla	Byzantine agreement	—
cwi-swat/tla-ci			*1 package*
client-centric-two-phase-locking	ClientCentric2PL_MC.tla	Storage and replication	yes
danwt/demo-chain-replication			*1 package*
one-shot-chain-replication	spec.tla	Storage and replication	—
danwt/demo-primary-backup-replication			*1 package*
one-shot-primary-backup-replication	spec.tla	Storage and replication	—
deepcausa/safeatomic			*2 packages*
safeatomic-atomic-visibility	SafeAtomicSmoke.tla	Recoverable concurrency	yes
safeatomic-checksum-sidecar	SafeAtomicChecksum.tla	Recoverable concurrency	yes
dfinity/formal-models			*10 packages*
dfinity-ckbtc	Ck_BTC.tla	Storage and replication	—
dfinity-file-based-lease	Lease.tla	Reconfiguration	—
dfinity-governance-ledger	Governance_TLC.tla	Storage and replication	—
dfinity-ii-migration	II_Migration.tla	Reconfiguration	—
dfinity-ledger-archive	Ledger.tla	Storage and replication	—
dfinity-people-parties	Connections.tla	Reconfiguration	—
dfinity-sns-governance-ledger	Governance_TLC.tla	Storage and replication	—
dfinity-sns-swap	Swap.tla	Storage and replication	yes
dfinity-subnet-splitting-v1	Subnet_Splitting.tla	Reconfiguration	—
dfinity-subnet-splitting-v2	Subnet_Splitting_v2.tla	Reconfiguration	—
dgpv/SASwap_TLAplus_spec			*1 package*
saswap-succinct-atomic-swap	MC.tla	Storage and replication	yes
dredozubov/hazmat			*1 package*
hazmat-secret-store-recovery	MC_SecretStoreRecovery.tla	Concurrent transactions	yes
Entrolution/bengal-stm			*1 package*
bengal-stm-scheduler	Scheduler.tla	Concurrent transactions	—
etcd-io/raft			*1 package*
etcd-raft-model-checking	MCetcdraft.tla	Consensus and SMR	—
Gerold103/serverbox			*2 packages*
serverbox-mcsp-queue	MCSPQueue.tla	Data structures	yes

Table 18: Task packages by source project (continued, page 2 of 4).

Task package	Root module	Protocol family	Bounded
serverbox-task-scheduler	TaskScheduler.tla	Scheduling and barriers	yes
iotaledger/wasp-legacy			*3 packages*
wasp-asynchronous-verifiable-secret-sharing	WaspACSS.tla	Byzantine agreement	—
wasp-bracha-reliable-broadcast	BrachaRBC.tla	Byzantine agreement	yes
wasp-consensus-journal-recovery	WaspChainConsensusJournal.tla	Election and termination	yes
JoshuaRowePhantom/Phantom.Coroutines			*1 package*
phantom-read-copy-update-memory-safety	MCReadCopyUpdate_MemorySafety.tla	Memory reclamation	yes
JuliaLang/julia			*1 package*
julia-scheduler-wakeup	MCFixed.tla	Scheduling and barriers	yes
lamport.org/tla/wildfire-challenge.zip			*6 packages*
alpha-memory-barrier	InnerAlpha.tla	Weak memory	—
alpha-memory-full	InnerAlpha.tla	Weak memory	—
alpha-memory-llsc	InnerAlpha.tla	Weak memory	—
alpha-memory-llsc-barrier	InnerAlpha.tla	Weak memory	—
alpha-memory-load-store	InnerAlpha.tla	Weak memory	—
wildfire-cache-coherence	Wildfire.tla	Weak memory	—
LeaveSHY/Evolv-BFT-Artifact			*1 package*
evolv-bft-byzantine-core	EvolvbftByzantine.tla	Byzantine agreement	—
lemmy/BlockingQueue			*4 packages*
bounded-blocking-queue	BlockingQueue.tla	Data structures	yes
poison-apple-blocking-queue	BlockingQueuePoisonApple.tla	Data structures	—
poison-pill-blocking-queue	BlockingQueuePoisonPill.tla	Data structures	—
split-waitset-blocking-queue	BlockingQueueSplit.tla	Data structures	yes
linkedin/venice			*1 package*
venice-leapfrog-divergence-detection	MCleapfrog.tla	Storage and replication	yes
lorin/snapshot-isolation-tla			*1 package*
mvcc-transaction-termination	MCMVCC.tla	Storage and replication	yes
lorin/tla-prophecy			*2 packages*
herlihy-wing-prophecy-queue	HWQueueProphecy.tla	Data structures	—
lock-based-linked-list-queue	LLLQueue.tla	Data structures	—
microsoft/CCF			*5 packages*
ccf-abstract-consensus	MCabs.tla	Consensus and SMR	yes
ccf-consistency-generic	Consistency.tla	Storage and replication	—
ccf-multi-node-consistency	MCMultiNode.tla	Storage and replication	yes
ccf-raft-model-checking	MCccfraft.tla	Consensus and SMR	—
ccf-single-node-consistency	MCSingleNode.tla	Storage and replication	yes
mongodb/mongo			*1 package*
mongodb-range-deletion-secondary-safety	MCRangeDeletionsSecondaryNodes.tla	Storage and replication	yes
nikolskiy/learn-tla			*1 package*
mpmc-condition-variable-queue	queue.tla	Data structures	yes
northriv/KAME			*7 packages*
kame-atomic-shared-orphan-chain	OrphanChain_atomicshared.tla	Memory reclamation	—
kame-chunk-allocation	ChunkAlloc_microscopic.tla	Memory reclamation	yes
kame-chunk-claim-publication	ChunkClaim.tla	Memory reclamation	—
kame-chunk-recycle-microscopic	ChunkRecycle_microscopic.tla	Memory reclamation	yes
kame-orphan-chain-adoption	OrphanChain_adopt.tla	Memory reclamation	yes
kame-separate-counts-orphan-chain	OrphanChain_pathB.tla	Memory reclamation	yes
kame-thread-epoch-seqlock-reclamation	ChunkRecycle_threadepoch.tla	Memory reclamation	yes
obsidiandynamics/spire-tla			*2 packages*
spanning-privilege-multivalue-safety	SPSafe.tla	Consensus and SMR	yes
spire-single-value-safety	SpireSafe.tla	Consensus and SMR	yes
oxarbitrage/zcash-p2p-spec			*1 package*
zcash-pre-ai-block-synchronization	p2p.tla	Storage and replication	—
pduarte21/tla-plus-by-example			*2 packages*
pduarte-refined-gossip	RefinedGossip.tla	Reconfiguration	yes
pduarte-vector-clocks	VectorClocks.tla	Election and termination	yes
peterzeller/TLA-broadcast-algorithms			*1 package*
peter-zeller-reliable-broadcast	reliable_broadcast.tla	Byzantine agreement	yes
pingcap/tla-plus			*5 packages*
pingcap-collapse-rollbacks	CollapseRollbacks.tla	Storage and replication	yes
pingcap-distributed-transaction	DistributedTransaction.tla	Storage and replication	—
pingcap-percolator	Percolator.tla	Storage and replication	yes
pingcap-raft-merge	RaftMerge.tla	Reconfiguration	—
pingcap-resolved-ts	MC.tla	Storage and replication	yes
plasma-umass/scalene			*2 packages*

Table 19: Task packages by source project (continued, page 3 of 4).

Task package	Root module	Protocol family	Bounded
scalene-lock-queue-topology	Deadlock.tla	Mutual exclusion	yes
scalene-signal-snapshot-safety	SignalSafety.tla	Data structures	—
polikow/bully_election			*1 package*
bully-leader-election	bully.tla	Election and termination	—
rs3lab/kernel-tla			*1 package*
linux-qspinlock	qspinlock.tla	Mutual exclusion	yes
scalecube/scalecube-prism			*2 packages*
scalecube-self-electing-quorum	SelfElectingQuorum.tla	Reconfiguration	yes
scalecube-static-lease-election	LeaseElection.tla	Election and termination	yes
stephanfr/minimalstdlib			*1 package*
minimalstdlib-lockfree-skiplist-reclamation	LockfreeSkiplist.tla	Memory reclamation	yes
tlaplus/azure-cosmos-tla			*1 package*
cosmos-anomalies	AnomaliesCosmosDB.tla	Storage and replication	yes
tlaplus/Examples			*83 packages*
abstract-thread-barrier	APBarrier.tla	Scheduling and barriers	yes
asynchronous-byzantine-agreement	aba_asyn_byz.tla	Byzantine agreement	—
asynchronous-termination-abstraction	AsyncTerminationDetection.tla	Election and termination	yes
authenticated-byzantine-broadcast	bcastByz.tla	Byzantine agreement	yes
auxiliary-variable-lock	Lock.tla	Mutual exclusion	yes
bosco-byzantine-consensus	APbosco.tla	Byzantine agreement	—
bosco-one-step-core	bosco.tla	Byzantine agreement	yes
boulangerie-model-checker	MCBoulanger.tla	Mutual exclusion	yes
bounded-btree-operations	btree.tla	Data structures	yes
btree-key-value-store-core	kvstore.tla	Data structures	yes
byzantine-paxos-consensus	Consensus.tla	Byzantine agreement	yes
byzantine-paxos-consistency-proof	BPConProof.tla	Byzantine agreement	—
byzantine-paxos-vote-proof	VoteProof.tla	Byzantine agreement	yes
cf1s-one-step-crash-consensus	cf1s_folklore.tla	Consensus and SMR	yes
chameneos-concurrency-game	Chameneos.tla	Scheduling and barriers	yes
chang-roberts-apalache-instance	APChangRoberts.tla	Election and termination	yes
chang-roberts-election	MCChangRoberts.tla	Election and termination	—
chang-roberts-ring-election-checker	MCChangRoberts.tla	Election and termination	yes
cigarette-smokers-synchronization	APCigaretteSmokers.tla	Scheduling and barriers	yes
crash-fault-folklore-broadcast	APbcastFolklore.tla	Byzantine agreement	yes
dijkstra-mutex-four-processors	MC.tla	Mutual exclusion	—
dijkstra-mutual-exclusion	MC.tla	Mutual exclusion	yes
dijkstra-scholten-termination-checker	MCEWD687a.tla	Election and termination	—
dining-philosophers	DiningPhilosophers.tla	Mutual exclusion	yes
disk-paxos-hdisk-synod	MC_HDiskSynod.tla	Consensus and SMR	—
disruptor-mpmc	Disruptor_MPMC.tla	Data structures	yes
disruptor-mpmc-apalache	APDisruptor_MPMC.tla	Data structures	—
disruptor-spmc	Disruptor_SPMC.tla	Data structures	yes
disruptor-spmc-apalache	APDisruptor_SPMC.tla	Data structures	—
distributed-spanning-tree	APspanning.tla	Reconfiguration	yes
echo-wave-checker	MCEcho.tla	Election and termination	yes
ewd426-self-stabilizing-token-ring	TokenRing.tla	Reconfiguration	—
ewd687a-diffusing-termination	MCEWD687a.tla	Election and termination	—
ewd840-apalache-instance	APEWD840.tla	Election and termination	yes
ewd840-json-trace-export	EWD840_json.tla	Election and termination	—
ewd840-ring-termination	EWD840.tla	Election and termination	yes
lamport-bakery-model-checker	MCBakery.tla	Mutual exclusion	yes
lamport-clocked-mutual-exclusion	APLamportMutex.tla	Mutual exclusion	—
lamport-fast-mutual-exclusion	MCLamportMutex.tla	Mutual exclusion	yes
lock-history-prophecy-refinement	LockHS.tla	Mutual exclusion	yes
nbacg-failure-sensitive-atomic-commit	nbacg_guer01.tla	Storage and replication	yes
nonblocking-atomic-commit	ACP_NB_TLC.tla	Storage and replication	yes
nonblocking-atomic-commit-state-model	ACP_SB_TLC.tla	Storage and replication	yes
one-communication-step-consensus	APc1cs.tla	Consensus and SMR	—
parallel-misra-reachability	APParReach.tla	Scheduling and barriers	yes
paxos-commit-protocol	PaxosCommit.tla	Storage and replication	yes
paxos-model-checker	MCPaxos.tla	Consensus and SMR	yes
peterson-lock-refinement	Peterson.tla	Mutual exclusion	yes
practical-smr-multipaxos	MultiPaxos_MC.tla	Consensus and SMR	yes
priority-message-resource-pool	AllocatorImplementation.tla	Scheduling and barriers	—
readers-writers-fifo	MC.tla	Mutual exclusion	—

Table 20: Task packages by source project (continued, page 4 of 4).

Task package	Root module	Protocol family	Bounded
readers-writers-lock	MC.tla	Concurrent transactions	yes
resource-allocator-implementation	AllocatorImplementation.tla	Scheduling and barriers	yes
resource-allocator-refinement	AllocatorRefinement.tla	Scheduling and barriers	yes
reusable-thread-barrier-core	Barrier.tla	Scheduling and barriers	yes
ring-credit-termination	EWD998.tla	Election and termination	—
ringmaster-checkpoint-coordination	MCCheckpointCoordination.tla	Election and termination	yes
safra-asynchronous-termination	EWD998.tla	Election and termination	—
sailfish-dag-consensus-checker	TLCSailfish2.tla	Consensus and SMR	—
scheduling-resource-allocator	SchedulingAllocator.tla	Scheduling and barriers	yes
sdp-zero-trust-access-protocol	MC.tla	Reconfiguration	—
sequence-ring-multicast	Disruptor_MPMC.tla	Data structures	—
simple-resource-allocator	SimpleAllocator.tla	Scheduling and barriers	yes
simplified-fast-paxos	FastPaxos.tla	Consensus and SMR	yes
single-lane-bridge-coordination	MC.tla	Scheduling and barriers	yes
snapshot-isolation-key-value-store	MCKVS.tla	Storage and replication	—
spanning-algorithm-checker	MC_spanning.tla	Reconfiguration	yes
spanning-tree-all-topologies	SpanTreeTest.tla	Reconfiguration	yes
spanning-tree-topology	SpanTree.tla	Reconfiguration	yes
synchronous-termination-abstraction	SyncTerminationDetection.tla	Election and termination	yes
synchronous-termination-apalache-instance	APSyncTerminationDetection.tla	Election and termination	yes
tcp-connection-establishment	MCtcp.tla	Reconfiguration	yes
teaching-concurrency-simple-regular	SimpleRegular.tla	Concurrent transactions	yes
tlaplus-nano-blockchain-small	MCNano.tla	Storage and replication	yes
transaction-commit	TwoPhase.tla	Storage and replication	—
two-phase-commit-btm	2PCwithBTM.tla	Storage and replication	—
two-phase-commit-protocol	TwoPhase.tla	Storage and replication	yes
unreliable-failure-detector	EnvironmentController.tla	Election and termination	—
voucher-transaction-lifecycle	VoucherLifeCycle.tla	Concurrent transactions	yes
write-through-cache-refinement	MCWriteThroughCache.tla	Data structures	—
yoyo-election-all-graphs	YoYoAllGraphs.tla	Election and termination	yes
yoyo-election-with-pruning	MCYoYoPruning.tla	Election and termination	yes
yoyo-election-without-pruning	MCYoYoNoPruning.tla	Election and termination	yes
Tuntii/KayaDB			*1 package*
kayadb-wal-crash-recovery	WalCrash.tla	Recoverable concurrency	yes
tursodatabase/turso			*1 package*
turso-sqlite-wal-transactions	SqliteTx.tla	Concurrent transactions	yes
ubisoft/task-scheduler			*2 packages*
multiple-consumer-single-producer-queue	MCSPQueue.tla	Data structures	yes
ubisoft-task-scheduler	TaskScheduler.tla	Scheduling and barriers	yes
unodb-dev/unodb			*1 package*
unodb-olc-upsert-erase	OLCUpsertErase.tla	Data structures	yes
Vanlightly/bookkeeper-tlaplus			*1 package*
bookkeeper-ledger-recovery	BookKeeperProtocol.tla	Storage and replication	yes
Vanlightly/kafka-tlaplus			*4 packages*
kafka-consumer-group-kip-848	consumer_group_protocol_kip_848_v2.tla	Election and termination	—
kafka-kraft-kip-853	kraft_kip_853.tla	Consensus and SMR	—
kafka-kraft-kip-996	kraft_kip_996.tla	Reconfiguration	—
kafka-replication-kraft-v3-5	kafka_replication_v3_5.tla	Storage and replication	—
Vanlightly/raft-tlaplus			*9 packages*
flexible-raft	FlexibleRaft.tla	Consensus and SMR	—
kraft-pull-raft	KRaft.tla	Consensus and SMR	—
kraft-with-reconfiguration	KRaftWithReconfig.tla	Reconfiguration	—
pull-raft	PullRaft.tla	Consensus and SMR	—
pull-raft-variant-2	PullRaftVariant2.tla	Consensus and SMR	—
raft-and-fsync	RaftFsync.tla	Consensus and SMR	—
raft-reconfiguration-add-remove	RaftWithReconfigAddRemove.tla	Reconfiguration	—
raft-reconfiguration-joint-consensus	RaftWithReconfigJointConsensus.tla	Reconfiguration	—
standard-raft	Raft.tla	Consensus and SMR	—
Vanlightly/vsr-tlaplus			*3 packages*
vsr-revisited	VSR.tla	Consensus and SMR	—
vsr-view-change-assume-new-view-change	VR_ASSUME_NEWVIEWCHANGE.tla	Consensus and SMR	—
vsr-view-change-incremental-resend	VR_INC_RESEND.tla	Consensus and SMR	yes
wallstop/fortress-rollback			*1 package*
fortress-rollback-input-queue	InputQueue.tla	Data structures	yes