interwhen: A Generalizable Framework for Verifiable Reasoning with Test-time Monitors

Vishak K Bhat[1] Prateek Chanda[2] Ashmit Khandelwal[1] Maitreyi Swaroop[3] Vineeth N. Balasubramanian[1]
Subbarao Kambhampati[4] Nagarajan Natarajan[1,†] Amit Sharma[1,†]

[1] Microsoft Research, India [2] IIT Bombay [3] Carnegie Mellon University [4] Arizona State University

arxiv

Motivation

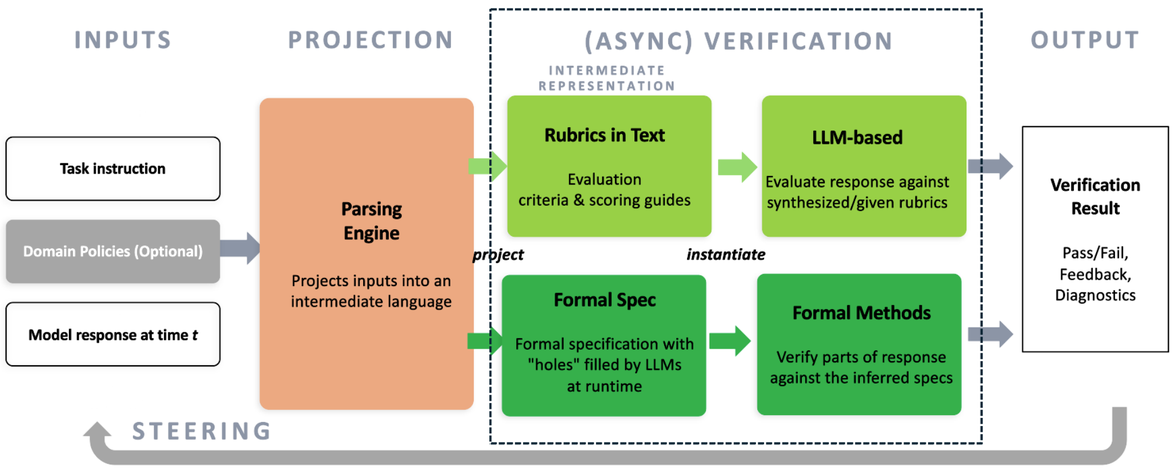
- **Verified reasoning** is critical for high-stakes scenarios: acting in the physical world, and in law & finance domains.
- **Existing approaches:**
  - **Generate-test**: Generate and verify only the final answer, provide feedback, and regenerate (LLM-Modulo [Kambhampati et al., 2024]).
  - **Structured decomposition**: Break tasks into sub-problems solved by LLMs (Tree of Thoughts [Yao et al., 2023]).
- **Key limitation**: These methods either verify only the *final answer* or *artificially constrain* the solution strategy.

★ interwhen enables process verification (LLM-process modulo) and steers a model's reasoning without constraining the solution strategy.

How to Build Process Verifiers

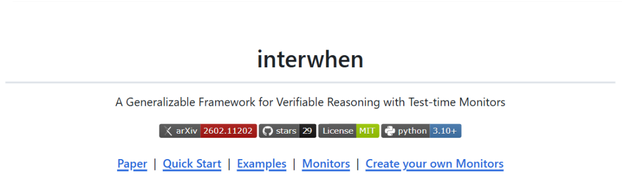
A **verifiable property** P is a predicate over an intermediate output that evaluates to pass/fail. *Examples:* "all maze moves lie on valid grid cells"; "the generated Lean4 code compiles"; "spatial claims are consistent with constraints".

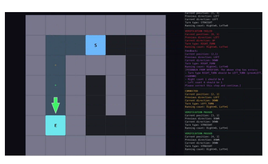
Pipeline: Problem constraints ⟶ extract verifiable properties ⟶ choose verifier:



Key operation: Automatically breaks down inputs into a list of verifiable properties in an intermediate language
Efficient: Verification is async, model execution is not stopped unless an error is found.
Plug-n-Play: Allows practitioners to plug-in existing verifiers

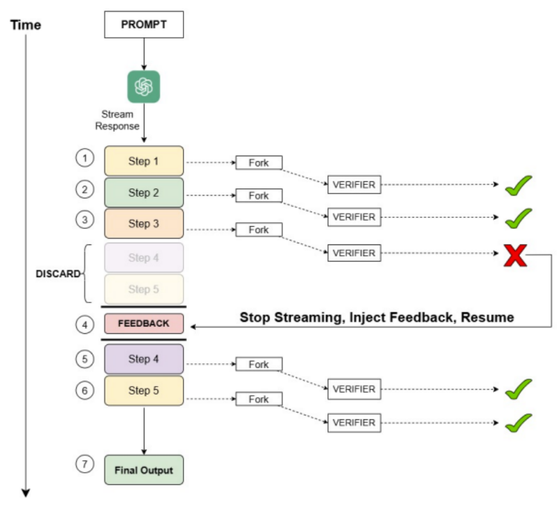
Open-Source Library





Maze demo

ZebraLogic demo



- **Plug-in verifiers:** Python, Z3, Lean4, LLM
- **3-line integration**, MIT Licensed

Scan for code

Datasets & Verifiers

Dataset	Verifiable Property	Verifier	Feedback
Maze [1]	Valid moves; correct turn counts	Python	Bad position / turn
SpatialMap [2]	Claims match constraints	Z3	Contradicts constraints
GameOf24 [3]	4 numbers used once; equals 24	Python	Wrong result / numbers
ZebraLogic [4]	Assignments satisfy clues	Z3	Violates clue
Verina [5]	Lean4 code compiles	Lean4	Compile error + hint

[1] Iyer et al., 2024. [2] Yamada et al., 2024. [3] Yao et al., 2023. [4] Lin & Ng, 2025. [5] Ye et al., 2025.

interwhen: Extract–Verify–Intervene

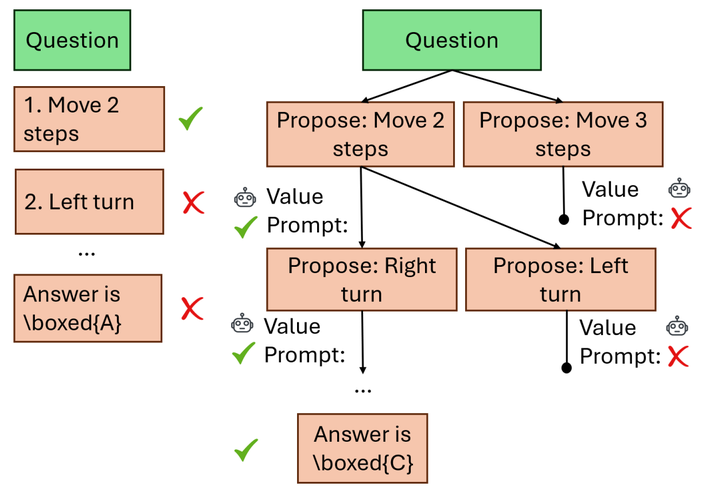
★ interwhen is a 3-function abstraction—extract, verify, intervene—that can realize any test-time scaling (TTS) strategy.

Mechanism (Fork-and-Verify): *We implement verification by periodically* ***forking*** *the reasoning trace,* `extract`*ing verifiable properties, and* `verify`*ing them asynchronously.*

- **Extract (when):** Fork after a few steps ⟶ obtain *verifiable properties*
- **Verify (how):** Check the extracted properties (python / z3 verifiers)
- **Intervene (what):** If incorrect: *rollback + feedback + resample*

Key: Asynchronous verification ⟹ **no latency overhead if correct**

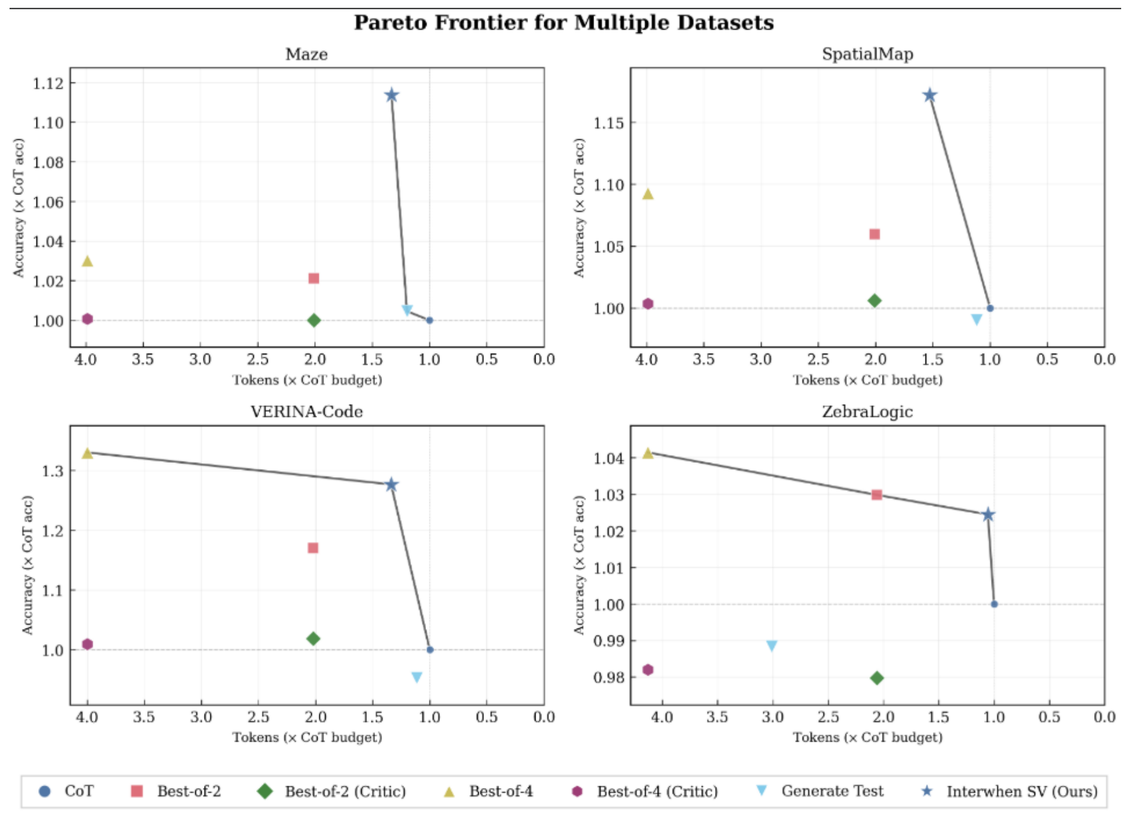
Comparison: Standard vs ToT vs interwhen



Standard Output | Tree of Thoughts | interwhen: Intervening with Textual Feedback

interwhen steers a **single trajectory** via fork-and-verify unlike ToT (multiple trajectories) or generate-test (final answer only).

Results: Accuracy vs Token Efficiency



- **Pareto-optimal** across all 7 datasets

Process Verification Also Improves Efficiency

Problem: Reasoning models "overthink" generating thousands of tokens after finding the answer. **EAT** [Wang et al., 2025] stops when EWMV of next-token entropy after `</think>` is low; **DEER** [Yang et al., 2025] stops when confidence after inserting `</think>` is high. Both achieve limited reduction.

k-Stable is a direct instantiation of Extract–Verify–Intervene:

Operation	k-Stable
Extract	Regex-match candidate answers from trace / LLM extracting the steps
Verify	Same answer for k consecutive extractions
Intervene	Inject `</think>` ⟶ force final answer

No new verifier needed the interwhen abstraction instantly solves early stopping, showing the *generalizable* nature of the framework. **Up to 55% token reduction** without accuracy loss, outperforming EAT and DEER.