BACKBEAT: LANGUAGE AGENTS KEEP WORKING WHILE TOOLS RUN

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-using language agents commonly suspend reasoning until a call returns. Launching calls in parallel removes serialization but still waits for the slowest final result, even when stable partial output already determines the next action. We introduce BACKBEAT, a model-agnostic execution interface that represents tool calls as typed futures with `launch`, `observe`, `resolve`, and `cancel` operations. Tools declare which fields are stable before completion; the agent can reason over these observations, finish when its decision predicate is satisfied, and cancel irrelevant tails. We evaluate BACKBEAT on PENDINGBENCH: 144 fresh tasks across six tool families, three model families, and 1,728 controlled trajectories, followed by complete agent sessions, generated concurrent pytest suites, and unmodified public-package tests. Against parallel-final execution, BACKBEAT reduces mean virtual evidence-ready latency from 35.21 to 21.56 seconds and raises deadline-correct completion from 25.7% to 93.8%, while ordinary accuracy remains 95.1% versus 95.4%. Even an optimistic adaptive final-only replay remains 11.69 seconds slower and 53.5 deadline points lower. The mechanism is visible in the schedule: 104/144 tasks place the deadline inside a stable tail averaging 13.65 seconds. In complete agent sessions at duration scale 1.0, BACKBEAT saves 14.47 seconds [11.05, 18.53] and wins all 24 paired runs. Through generated suites executed by pytest 9.1.1, it saves 2.24 session seconds [0.08, 4.38] and 5.46 seconds of executed suite service, with every task answer preserved. On 12 upstream SciPy and scikit-learn files, stable target-test reports precede full-suite completion in all 12 pairs, saving 0.50 tool seconds [0.16, 1.06] at zero API cost. The 1,980-trajectory campaign costs $2.56. Pending calls provide a productive test-time reasoning substrate.

1 INTRODUCTION

Consider an agent choosing one of four experiment configurations. It launches four probes, each taking 13–32 seconds to finish. The score and resource fields needed for the decision stabilize earlier, after 4–25 seconds, while the probes continue writing diagnostics. A sequential agent waits roughly 96 seconds. A conventional concurrent agent waits 35 seconds for all final results. An agent that can observe declared-stable fields decides in 22 seconds and stops more than two redundant tool-seconds per call set. All three see the same decision evidence; they differ in when the interface lets the model use it.

This timing distinction is largely absent from the standard reason–act loop. ReAct and its descendants alternate model output with completed observations (Yao et al., 2023; Schick et al., 2023). Agent benchmarks make the environment interactive but usually present each tool response atomically (Liu et al., 2024; Jimenez et al., 2024; Yao et al., 2024). Parallel planners and speculative execution launch independent work together (Kim et al., 2024; Ye et al., 2026; Guan et al., 2026), yet a node still becomes useful only at completion. Recent asynchronous function-calling systems expose complete futures (Feng et al., 2026; Shi et al., 2026). The missing interval is the period in which a call is unfinished but already informative.

BACKBEAT makes that interval explicit. A future emits monotone lifecycle events. An `OBSERVE` event may expose only fields whose values will not be retracted; a `RESOLVE` event closes the call; and `CANCEL` terminates a tail that can no longer affect the decision. The interface is deliberately small.

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.

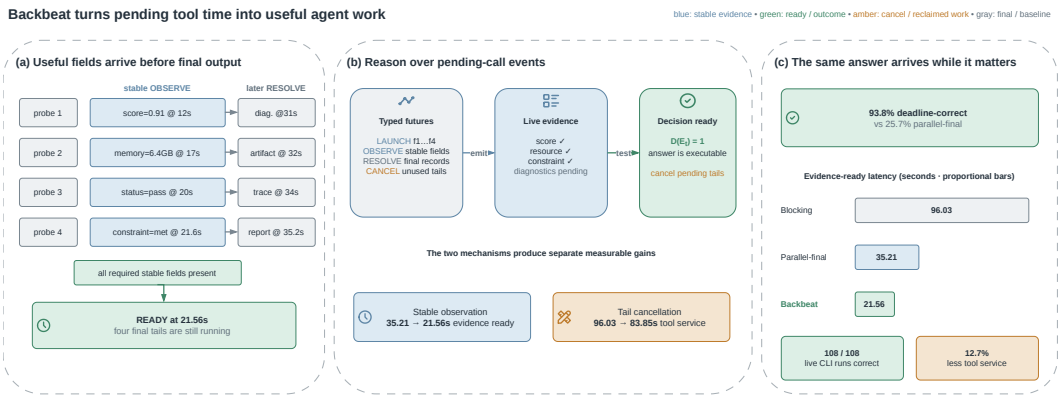

Figure 1: BACKBEAT exposes the informative interval between call launch and final resolution. Stable observations update live evidence, readiness ends the decision, and cancellation reclaims irrelevant final tails. Bar lengths in panel (c) are proportional to virtual evidence-ready latency.

It changes neither model weights nor the agent's task policy. It changes the temporal granularity of the evidence available to that policy.

We study the abstraction with a paired evaluation at three levels. First, a controlled matrix holds tasks and evidence fixed while changing execution lifecycle, enabling exact deadline and service accounting over 1,728 model–task decisions. Second, the same protocols run as actual CLI tools in isolated workspaces, measuring full agent session time rather than a simulated scheduler. Third, two duration-shift validations, a generated pytest task adapter, and unmodified public-package tests probe transfer.

Our contributions are:

- **A pending-call reasoning interface.** BACKBEAT combines typed futures, declared-stable partial observations, decision readiness, and tail cancellation in a model-agnostic runtime contract.
- **A paired asynchronous evaluation.** PENDINGBENCH contains six task families, three model families, 1,980 controlled and live trajectories, generated and upstream pytest workloads, exact labels, task-cluster uncertainty, ablations, and complete runtime/cost accounting.
- **A measured timing law with real-tool transfer.** Stable observations cut useful-evidence latency by 38.8%; the benefit becomes decisive as tool duration grows (14.47 seconds per complete session at scale 1.0) and transfers to pytest (2.24 seconds per session), while preserving task accuracy.

2 REASONING OVER PENDING CALLS

2.1 FUTURE LIFECYCLE AND READINESS

For a task, let $F = \{f_1, \ldots, f_m\}$ be launched tool futures. Future f_j starts at ℓ_j, first exposes its declared-stable fields at p_j, and resolves at r_j, with $\ell_j \le p_j \le r_j$. At time t, the agent has an evidence set

$$E_t = \bigcup_j \mathrm{stable}(f_j, t),$$

where $\mathrm{stable}(f_j, t)$ is empty before p_j, contains the stable partial fields on $[p_j, r_j)$, and contains the final record after r_j. A task-specific predicate $D(E_t)$ reports whether the available fields determine an executable answer. BACKBEAT returns at

$$T_{\mathrm{BB}} = \inf\{t : D(E_t) = 1\}, \qquad S_{\mathrm{BB}} = \sum_j \min(T_{\mathrm{BB}} - \ell_j, r_j - \ell_j),$$

then cancels each unresolved f_j whose remaining output is not required. T measures decision latency; S measures executed tool service. Parallel-final uses $T = \max_j r_j$, while sequential blocking accumulates all $r_j - \ell_j$.

Event	Runtime meaning	Agent affordance
LAUNCH	call begins in background	continue independent work
OBSERVE	declared fields stabilize	update live decision state
RESOLVE	final record is available	close the future
CANCEL	final tail is unnecessary	reclaim tool service

Table 1: The BACKBEAT lifecycle. Events are append-only and scoped by future id.

The runtime loop is: launch independent futures; work on the decision rule; observe new stable events; update E_t; return when $D(E_t)$ holds; cancel irrelevant tails. No learned scheduler is required. The model already reasons about the task; BACKBEAT lets that reasoning overlap external computation.

2.2 STABLE PARTIAL OUTPUT IS A CONTRACT

OBSERVE is a committed stable projection rather than an arbitrary token prefix. Each tool adapter names fields that become semantically stable before the process finishes: completed query rows before formatting, compiler diagnostics before artifact upload, simulation scores before trace compression, or document metadata before indexing. Once exposed, these values are monotone. A tool may withhold an unstable field until RESOLVE; it may not revise an observed value. This contract gives the agent a compact form of partial observability (Kaelbling et al., 1998) with the event-order discipline used by stream processors (Lamport, 1978; Akidau et al., 2015).

Cancellation follows readiness. BACKBEAT operates only on calls committed by the existing agent policy, waits until the required fields are present, and cancels remaining diagnostic tails. This separates it from speculative action systems and makes service savings directly auditable.

2.3 COMPARED EXECUTION POLICIES

We compare five interfaces with identical questions and evidence values:

1. **Blocking**: execute four probes sequentially and consume final records.
2. **Parallel-final**: launch all probes concurrently and wait for every final record, a strong static-DAG baseline.
3. **Adaptive final-only**: consume an early final record when it certifies the answer; otherwise wait for all finals. This optimistic replay reuses the parallel-final answers and gives final-only execution its task-specific stopping rule.
4. **BACKBEAT**: consume raw, timestamped OBSERVE/RESOLVE events and cancel tails after readiness.
5. **Ready-view**: materialize all currently stable fields into one compact state before model readout. This tests whether an extra aggregation layer helps.

3 PENDINGBENCH

3.1 CONTROLLED TASK MATRIX

PENDINGBENCH instantiates 144 fresh, four-option decisions in six equal families: repository diagnosis, analytics, simulation selection, document compatibility, workflow draining, and experiment selection. Each task has four independent probes. Probe durations are sampled without replacement from 13–32 virtual seconds; stable fields appear earlier, and a 4–9 second cross-check completes the decision. Deadlines are sampled from 27–36 seconds. The answer is computed directly from the generated records, giving exact labels without a model judge.

We freeze every task world and render all four lifecycle views. Three model families then answer every task: DeepSeek V4 Flash, GLM 5.3 Flash, and GPT-5.6 Luna, through the same OpenCode runner and JSON answer contract. This yields 1,728 model–task decisions. Prompts batch 12 independent tasks for cost efficiency; all task-level answers, raw runner streams, session ids, token/cost records, and lifecycle metrics are retained.

3.2 METRICS AND UNCERTAINTY

Accuracy is exact option accuracy. *Deadline-correct* requires both the correct option and $T \leq d$ for task deadline d. *Virtual evidence-ready latency* is the time at which sufficient evidence is available under the controlled interface. It is computed from each frozen tool schedule and excludes model-service and orchestration overhead; the live CLI study separately measures complete session latency. *Tool service* sums time executed across all probes and the cross-check. *Cancelled calls* counts unresolved tails stopped after readiness.

All primary comparisons are task paired. For pooled intervals, we average each condition difference over the three models for one task, then bootstrap the 144 task clusters 5,000 times. The same task-cluster procedure is used for live wall-clock contrasts. A deadline sweep from 20 to 100 seconds tests persistence across operating thresholds.

We also measure the opportunity independently of model behavior. For every frozen schedule, *stable-ready* is the latest required stable field plus cross-check time; *final-ready* replaces that field time with final completion. A deadline can benefit from the interface exactly when it lies between these two values. This exposes the benchmark's timing structure directly rather than inferring it from agent scores. As a stronger final-only counterfactual, an adaptive oracle stops on a qualifying final record for repository and document tasks and conservatively waits for all finals in the other four families. It replays the saved parallel-final answers, so its timing is optimistic and adds no model calls.

3.3 LIVE CLI EVALUATION

Controlled transcripts isolate representation and scheduling, while live execution tests the complete interaction. We select two tasks per family and instantiate each as an isolated command-line workspace with a hidden task world. Agents may use only documented `bbtool` commands. Blocking calls `run-all`; parallel-final calls `launch-many` then `wait-all`; BACKBEAT calls `launch-many`, reasons while calls progress, and invokes `observe` until the required fields are stable. Twelve tasks, three models, and three policies yield 108 real trajectories.

The default time scale compresses tool durations to keep the study inexpensive. We therefore decompose measured sessions into tool and non-tool time, sweep the duration scale, and run two fixed validations at scales 0.50 and 1.0 with two model families, 12 tasks, and both BACKBEAT and parallel-final (96 additional trajectories). Neither validation is extended after observing results.

Finally, we implement the contract through pytest 9.1.1. The complete-agent study uses deterministically generated suites: each of four independent processes writes an immutable critical-test report before an unrelated tail test ends. Parallel-final waits for all processes; BACKBEAT consumes completed critical reports and terminates remaining tails. Twelve disjoint tasks, two models, and two policies produce 48 trajectories. A separate zero-API adapter check runs the first eight collected tests from each of 12 unmodified upstream SciPy 1.17.1 and scikit-learn 1.8.0 files. The first targeted test is decision-bearing; seven tests form the regression tail. Raw streams, subprocess reports, event times, termination records, and costs are retained. The entire pilot and evaluation costs $2.56.

4 RESULTS

4.1 STABLE PARTIAL OBSERVATIONS TRANSFORM DEADLINE UTILITY

Table 2 shows the central result. Parallel-final execution removes most serialization, reducing mean virtual evidence-ready latency from 96.03 to 35.21 seconds, but only 25.7% of answers arrive correctly by their task deadline. BACKBEAT uses the same probe set and answer rule yet reaches 93.8%: a paired gain of 68.1 points with interval [60.2, 75.2]. Ordinary accuracy is effectively unchanged relative to the strong parallel baseline: 95.1% versus 95.4%, a -0.2-point difference [-2.8, 2.1]. Task-specific stopping improves final-only execution to 40.3% and 33.25 seconds, but BACKBEAT remains 53.5 deadline points [44.9, 62.0] higher and 11.69 seconds [10.65, 12.71] earlier. Waiting for every final therefore explains only a small part of the gap; the larger gain comes from decision-bearing fields that stabilize before any answer-certifying final record. Sequential blocking reaches 98.8% accuracy, 3.7 points above BACKBEAT [1.4, 6.3]; this reflects answer extraction from full final records

Interface	Accuracy	Deadline-correct	Evidence ready (s)	Tool service (s)
Blocking	98.8	0.0	96.03	96.03
Parallel-final	95.4	25.7	35.21	96.03
Adaptive final-only	95.4	40.3	33.25	95.00
Ready-view	93.5	92.1	21.56	83.85
BACKBEAT	**95.1**	**93.8**	**21.56**	**83.85**

Table 2: Pooled controlled results over 144 tasks and three models. Accuracy and deadline-correct columns are percentages.

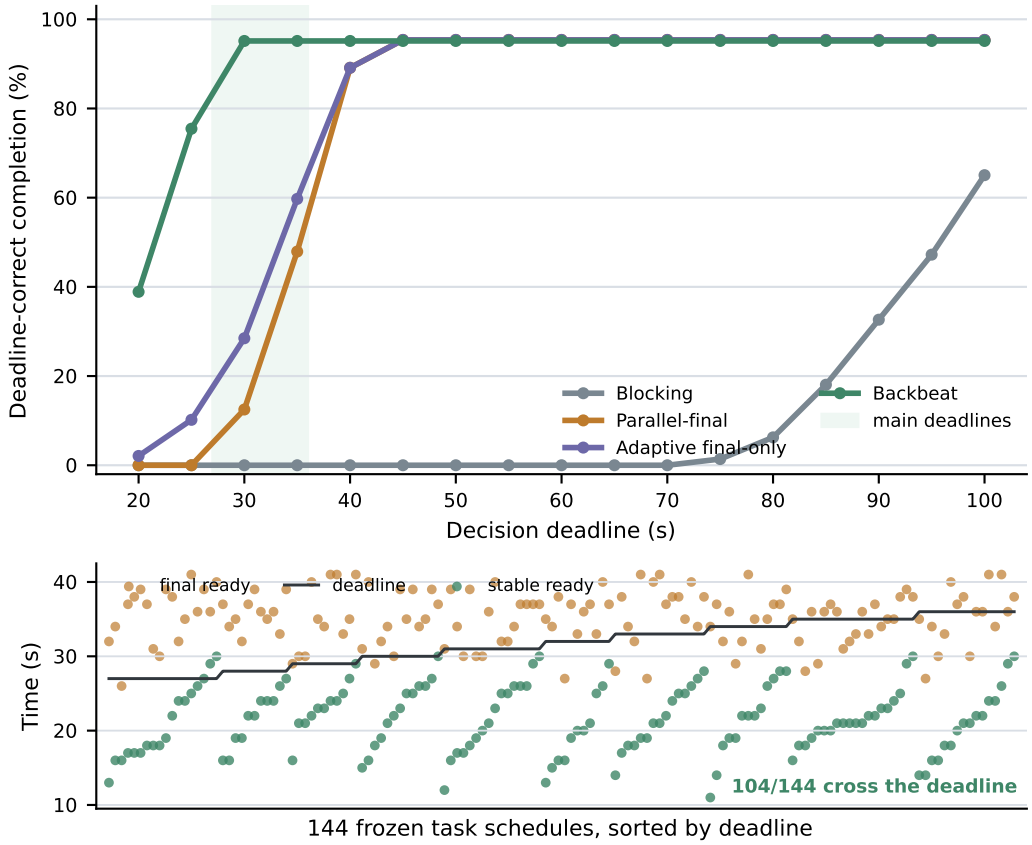

Figure 2: **Top:** deadline-correct completion across thresholds; the shaded band contains the 27–36s task deadlines and the adaptive curve uses only final records. **Bottom:** every frozen schedule's stable time, deadline, and final time. In 104/144 schedules, stable evidence arrives before the deadline and final output after it.

rather than the timing mechanism. The main temporal comparison therefore uses parallel-final, which holds concurrent execution and answer accuracy approximately matched.

The gain appears in every model family. Deadline-correct completion is 93.1% for DeepSeek, 93.8% for GLM, and 94.4% for Luna under BACKBEAT, versus 25.7%, 25.0%, and 26.4% under parallel-final. It also holds across all six task families, ranging from 90.3% in analytics to 95.8% in repository and workflow tasks. The shared temporal interface, rather than one model or domain, drives the result.

Figure 2 exposes the full utility curve. At 20 seconds, BACKBEAT has already completed 38.9% of tasks correctly, versus 2.1% for adaptive final-only and zero for parallel-final and blocking. At 25 seconds it reaches 75.5% versus 10.2% for adaptive final-only; at 30 seconds the comparison is 95.1%

Variant	Accuracy	Deadline-correct	Evidence ready (s)	Service (s)
Parallel-final / no partial	95.4	25.7	35.21	96.03
BACKBEAT/ no cancel	95.1	93.8	21.56	96.03
BACKBEAT/ ready-view	93.5	92.1	21.56	83.85
BACKBEAT	**95.1**	**93.8**	**21.56**	**83.85**

Table 3: Component ablation pooled across models. Partial observations determine latency; cancellation separately reduces executed service.

Live policy	Success	Mean session (s)	Median (s)	Mean cost
Blocking	100.0	24.33	24.03	0.0055
Parallel-final	100.0	14.16	13.86	0.0068
BACKBEAT	**100.0**	15.93	**13.64**	0.0072

Table 4: Full OpenCode sessions over 12 executable CLI tasks and three models. Success is percentage; cost is USD per trajectory.

versus 28.5% and 12.5% for adaptive and parallel-final. The curves converge only after 45 seconds, when the slowest parallel finals have arrived. Thus the main threshold samples the broad interval in which stable evidence is useful rather than selecting one favorable point. The direct schedule analysis explains why: stable fields precede final fields by 13.65 seconds on average (median 13.00), and 104/144 tasks (72.2%) place the deadline inside that interval. All six families exhibit this opportunity in at least two thirds of their schedules.

4.2 OBSERVATION REMOVES TAIL LATENCY; CANCELLATION REMOVES TAIL SERVICE

The ablation in Table 3 cleanly separates the two gains. Exposing stable observations reduces virtual evidence-ready latency by 38.8% and adds 68.1 deadline points even when every final tail continues running. Enabling cancellation leaves the answer and decision time unchanged but reduces mean executed service from 96.03 to 83.85 seconds, a 12.7% saving, by cancelling 2.08 calls per task on average. This is the familiar tail-at-scale opportunity (Dean & Barroso, 2013), applied inside one agent decision.

Raw typed events also outperform the materialized ready-view by 1.6 accuracy points, with a paired interval spanning zero. The extra state layer is therefore optional: models can consume the small append-only event stream directly. This negative ablation keeps the core interface minimal.

4.3 REAL AGENTS REPRODUCE THE TEMPORAL EFFECT

All 108 live trajectories answer correctly. Relative to blocking, BACKBEAT saves 8.40 seconds per session on average, with task-cluster interval [6.36, 10.01], and reduces the median from 24.03 to 13.64 seconds. At the deliberately short default tool scale, parallel-final has the lower mean because additional observe steps are comparable to the remaining tail; BACKBEAT is nevertheless slightly lower at the median (13.64 versus 13.86 seconds). This produces a testable prediction: as external computation grows, partial readiness should overtake final-result waiting.

Table 5 and Figure 3 confirm that prediction. Scale 0.50 produces the expected positive direction: 1.54 seconds saved, with interval [-0.78, 4.00]. At the preregistered scale 1.0, the effect is decisive: BACKBEAT saves 14.47 seconds per complete session (median 11.97), the task-cluster interval is [11.05, 18.53], and all 24 paired runs favor BACKBEAT. Both policies make the correct semantic choice on every task. One BACKBEAT trace returns the correct bare letter rather than the requested JSON; strict format-and-answer compliance is therefore 95.8% versus 100.0%, while the timing comparison is unchanged.

The generated-suite study tests complete agent sessions through an unmodified runner. Across 48 pytest trajectories, BACKBEAT preserves every answer and saves 2.24 complete-session seconds on average (median 2.59), with task-cluster interval [0.08, 4.38]. It shortens observed tool span by 2.97 seconds and executed suite service by 5.46 seconds.

Live setting	Traj.	Task acc. P/B	Saved (s)	95% task CI	B wins
CLI, scale 0.50	48	100/100	1.54	[-0.78, 4.00]	62.5%
CLI, scale 1.00	48	100/100	**14.47**	**[11.05, 18.53]**	**100.0%**
pytest 9.1.1	48	100/100	**2.24**	**[0.08, 4.38]**	62.5%

Table 5: Fixed follow-up studies against parallel-final (P). B denotes BACKBEAT; "saved" is complete agent-session time, not a scheduler projection.

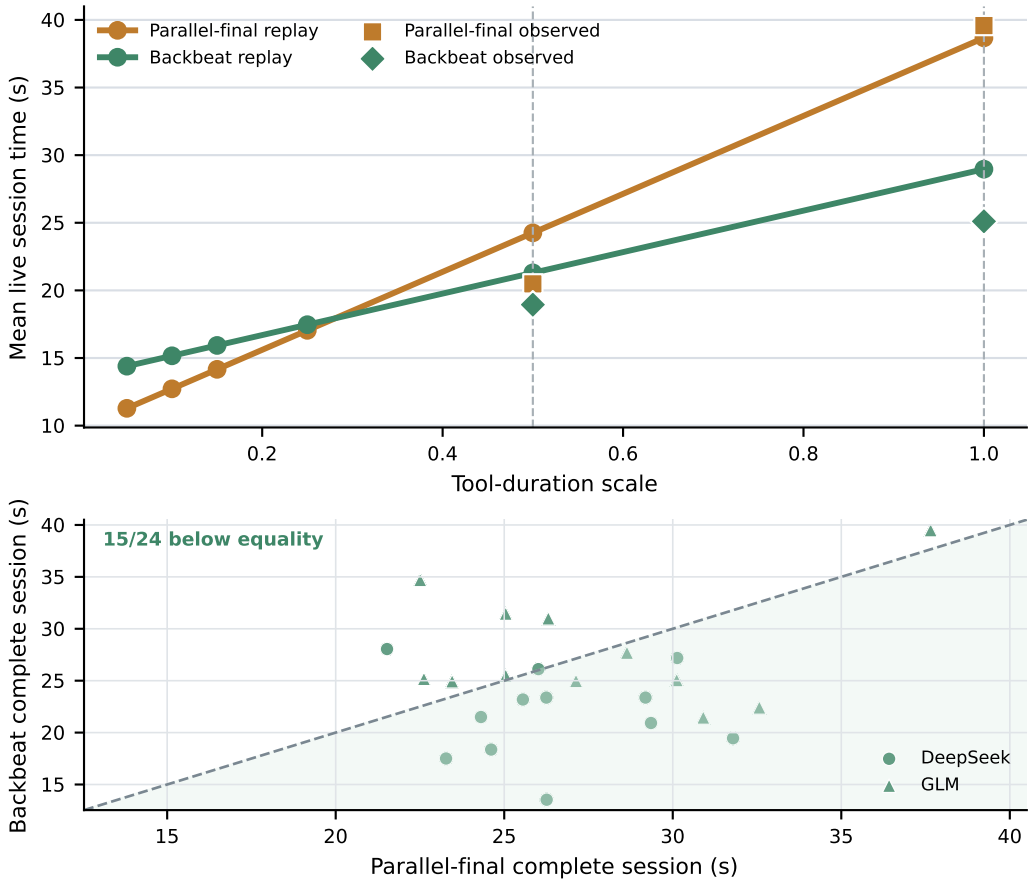

Figure 3: **Top:** replay predicts a duration crossover; squares and diamonds at scales 0.50 and 1.0 are observed complete sessions. **Bottom:** 24 paired sessions through generated suites on real pytest processes; points below equality favor BACKBEAT.

The model slices localize the crossover rather than hiding it in an aggregate. At the short default scale, DeepSeek and GLM incur 3.14s and 2.77s overhead while Luna saves 0.62s. At scale 1.0, DeepSeek and GLM save 14.46s and 14.49s, respectively. Through generated pytest, DeepSeek saves 4.64s while GLM is effectively tied at -0.16s; all six policy–model cells retain 100% semantic task accuracy. Appendix Table 7 reports every model-level contrast.

The public-suite check then removes generated sleep tests entirely. On all 12 unmodified SciPy and scikit-learn files, the targeted-test outcome is identical under full-suite and stable-report execution, and every regression tail is still running when cancelled. Stable reports save 0.50 process seconds on average (median 0.20), with file-paired interval [0.16, 1.06] and 12/12 positive pairs. This adapter-only result shows that the stable-before-final interval also occurs in natural public test workloads.

5 Related work

Tool agents and benchmarks. ReAct established the alternating reasoning/action interface (Yao et al., 2023); Toolformer learns when to call tools (Schick et al., 2023). AgentBench, SWE-bench, and τ-bench broaden interactive evaluation (Liu et al., 2024; Jimenez et al., 2024; Yao et al., 2024), and workflow benchmarks test multi-step construction (Qiao et al., 2025). These works evaluate which calls agents make. PendingBench instead holds the call set fixed and measures when an unfinished call becomes useful.

Parallel and speculative execution. LLMCompiler constructs parallel function-call DAGs (Kim et al., 2024). Speculative Actions and Dynamic Speculative Agent Planning execute predicted future branches (Ye et al., 2026; Guan et al., 2026). Backbeat predicts no branch and requires no rollback: all launched calls belong to the current policy, and only declared-stable fields enter the decision.

Asynchrony and streaming. AsyncFC represents complete calls as futures, while AsyncTool evaluates multitasking with delayed function feedback (Feng et al., 2026; Shi et al., 2026). Gaia2 studies worlds that evolve independently of the agent (Froger et al., 2026). Backbeat is complementary: it turns the lifecycle *inside* one external call into observable agent state. Futures have long supported overlap in programming languages (Halstead, 1985); the contribution here is their stable-partial and decision-readiness contract for language-agent tool use.

6 Demonstrated Scope and Extensions

Backbeat applies when a tool can declare decision-relevant fields stable before final completion; atomic APIs reduce naturally to parallel-final execution. The controlled benchmark provides exact paired labels and schedules, complete CLI trajectories establish the same mechanism end to end, generated suites exercise the full agent–pytest path, and unmodified SciPy and scikit-learn files expose the timing opportunity on public workloads. Cancellation applies to side-effect-free tails; the no-cancel ablation shows that stateful actions retain the observation benefit. The evaluated agents span three model families and price points through one common runner. Database cursors, compiler pipelines, longer task horizons, and additional runners define direct extensions of the same runtime contract.

7 Conclusion

Tool latency need not be agent idle time. Backbeat exposes stable partial observations from typed futures, lets reasoning continue while calls run, and cancels final tails after the decision is ready. Across 1,728 controlled decisions it turns parallel-final's 25.7% deadline-correct completion into 93.8% while preserving answer accuracy. Complete agent sessions confirm the duration crossover; generated suites reproduce the latency and service gains through pytest, and unmodified public suites expose the same stable-before-final opportunity. Pending-call reasoning is a compact, measurable, and immediately useful test-time capability for language agents.

AI Use Statement

Generative AI systems assisted with implementation, experiment orchestration, analysis code, and language editing. The authors defined the research questions and evaluation, inspected saved prompts and outputs, verified all exact labels, and checked every reported number against the included artifacts.

References

Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, and Sam Whittle. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proceedings of the VLDB Endowment*, 8(12):1792–1803, 2015.

Jeffrey Dean and Luiz André Barroso. The tail at scale. *Communications of the ACM*, 56(2):74–80, 2013.

Guangyu Feng, Huanzhi Mao, Prabal Dutta, and Joseph E. Gonzalez. Concurrency without model changes: Future-based asynchronous function calling for LLMs. *arXiv preprint arXiv:2605.15077*, 2026.

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, et al. Gaia2: Benchmarking LLM agents on dynamic and asynchronous environments. *arXiv preprint arXiv:2602.11964*, 2026.

Yilin Guan, Qingfeng Lan, Fei Sun, Dujian Ding, Devang Acharya, Chi Wang, William Yang Wang, and Wenyue Hua. Dynamic speculative agent planning. In *International Conference on Learning Representations*, 2026.

Robert H. Halstead. Multilisp: A language for concurrent symbolic computation. *ACM Transactions on Programming Languages and Systems*, 7(4):501–538, 1985.

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*, 2024.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM compiler for parallel function calling. In *International Conference on Machine Learning*, 2024.

Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, et al. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, 2024.

Shuofei Qiao, Runnan Fang, Zhisong Qiu, Xiaobin Wang, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Benchmarking agentic workflow generation. In *International Conference on Learning Representations*, 2025.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023.

Kou Shi, Ziao Zhang, Shiting Huang, Avery Nie, Zhen Fang, Qiuchen Wang, Lin Chen, Huaian Chen, Zehui Chen, and Feng Zhao. AsyncTool: Evaluating the asynchronous function calling capability under multi-task scenarios. *arXiv preprint arXiv:2605.27995*, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Naimeng Ye, Arnav Ahuja, Georgios Liargkovas, Yunan Lu, Kostis Kaffes, and Tianyi Peng. Speculative actions: A lossless framework for faster agentic systems. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=P0GOk5wslg.

A TASK FAMILIES AND GENERATED DECISIONS

Each controlled task contains four named alternatives and a deterministic selection rule. Repository tasks select the unique failing non-retryable module; analytics tasks maximize a ratio; simulation tasks minimize energy subject to an accuracy threshold; document tasks select the latest compatible version; workflow tasks minimize drain time among dependency-clear stages; and experiment tasks maximize score subject to a runtime limit. Names, values, durations, stable times, and option order are sampled from the frozen seed. Every task JSON contains the exact answer before prompt rendering.

Family	Tasks	Backbeat accuracy	Deadline-correct	Evidence ready (s)
Analytics	24	94.4	90.3	21.08
Documents	24	91.7	91.7	21.46
Experiments	24	94.4	94.4	23.17
Repository	24	100.0	95.8	22.88
Simulation	24	94.4	94.4	21.08
Workflow	24	95.8	95.8	19.71

Table 6: Pooled BACKBEAT results by controlled task family.

B LIVE INTERACTION PROTOCOLS

The live workspace prompt prevents direct inspection of source, process state, environment variables, or the hidden task world. It exposes only the documented commands. The three policy instructions are:

- Blocking: `bbtool run-all probe_1 ... probe_4`.
- Parallel-final: `bbtool launch-many ...`, then `bbtool wait-all`.
- BACKBEAT: `bbtool launch-many ...`; reason about the decision rule; call `bbtool observe`; answer when required fields are stable.

All policies finish with the same one-object JSON answer contract. Each trajectory retains the raw OpenCode stream, normalized answer, event timestamps, session wall time, tool span, runner version, session id, and accounting records.

For the pytest transfer, each probe is a separate pytest process containing a named critical test followed by an unrelated tail test. The critical test atomically writes its immutable report. Parallel-final consumes reports after all suite processes end; BACKBEAT consumes the same reports when all critical tests are stable and terminates remaining process groups. The 12 pytest tasks are disjoint from the primary live slice.

C COMPLETE-SESSION MODEL SLICES

Setting	Model	Pairs	Task acc. P/B	Mean saved (s)	B wins (%)
CLI default	DeepSeek	12	100.0/100.0	-3.14	41.7
CLI default	GLM	12	100.0/100.0	-2.77	41.7
CLI default	GPT-5.6 Luna	12	100.0/100.0	0.62	66.7
CLI scale 0.50	DeepSeek	12	100.0/100.0	1.09	58.3
CLI scale 0.50	GLM	12	100.0/100.0	1.99	66.7
CLI scale 1.00	DeepSeek	12	100.0/100.0	14.46	100.0
CLI scale 1.00	GLM	12	100.0/100.0	14.49	100.0
Generated pytest	DeepSeek	12	100.0/100.0	4.64	83.3
Generated pytest	GLM	12	100.0/100.0	-0.16	41.7

Table 7: Every complete-session comparison by model. Positive time means BACKBEAT is faster; P/B denotes parallel-final/BACKBEAT semantic task accuracy. The default scale exposes the short-tail boundary, both models cross decisively at scale 1.0, and the generated pytest aggregate combines a strong DeepSeek gain with a near-zero GLM mean.

The upstream-suite adapter check is complementary to Table 7: it uses no model calls. It runs unmodified SciPy and scikit-learn tests, reports the first target test through a pytest hook, and cancels seven still-pending regression tests. All 12 target outcomes agree with full-suite execution and all 12 pairs save process time; the frozen node ids and both policies' stdout, stderr, reports, and timestamps are included in the artifact.

D ADDITIONAL PER-MODEL RESULTS

Model	Interface	Accuracy	Deadline	Wall	Service
DeepSeek	Parallel-final	95.1	25.7	35.21	96.03
DeepSeek	BACKBEAT	94.4	93.1	21.56	83.85
GLM	Parallel-final	96.5	25.0	35.21	96.03
GLM	BACKBEAT	95.1	93.8	21.56	83.85
GPT-5.6 Luna	Parallel-final	94.4	26.4	35.21	96.03
GPT-5.6 Luna	BACKBEAT	95.8	94.4	21.56	83.85

Table 8: Strong-baseline comparison for each model family.

E REPRODUCTION MAP

```
python3 scripts/build_main.py
python3 scripts/run_main.py
python3 scripts/analyze_main.py
python3 scripts/build_interactive_main.py
python3 scripts/run_interactive_main.py
python3 scripts/analyze_interactive_main.py
python3 scripts/run_duration_validation.py
python3 scripts/analyze_duration_validation.py
python3 scripts/run_duration_validation.py --time-scale 1.0 \
  --output-subdir duration_validation_scale100_runs
python3 scripts/analyze_duration_validation.py \
  --input-subdir duration_validation_scale100_runs \
  --output-prefix duration_validation_scale100 --time-scale 1.0
python3 scripts/analyze_timing_opportunity.py
python3 scripts/analyze_adaptive_final.py
python3 scripts/build_pytest_stream_validation.py
python3 scripts/run_pytest_stream_validation.py
python3 scripts/analyze_pytest_stream_validation.py
python3 scripts/run_public_pytest_transfer.py
python3 scripts/analyze_public_pytest_transfer.py
python3 scripts/analyze_live_stratified.py
python3 scripts/render_paper_assets.py
python3 -m pytest -q
```

The package preserves every task world, prepared prompt, raw model stream, normalized answer, lifecycle trace, task-level score, bootstrap input, figure source table, and cost record. All tasks and outputs were generated for this project; no result from an earlier paper is reused.