THAW: BENCHMARKING LLM CONTROLLERS UNDER INFERENCE DELAY

Anonymous authors
Paper under double-blind review

ABSTRACT

An agent's command can arrive too late when the environment continues to change during inference. Thaw is a benchmark for studying this timing problem in a deterministic outpost task: a large language model (LLM) directs simulated scientists to deliver supplies before shortages cause failure. We measure survival as the share of a fixed-duration episode completed before failure. To show that the task can be completed, we replay commands found by a planning program or fixed-rule controller. Every replay keeps the outpost operating for the entire fixed duration without any LLM calls.

We compare a clock that pauses during inference with one that advances proportionally to each model response's measured duration. In a task variant with brief gathering, processing and travel times, each evaluated LLM has lower survival when response duration advances the simulation. As the simulation advances more per second of inference, one LLM goes from completing the episode with the clock stopped to surviving less than half of it on average. In a separate timing test, each response advances the simulation by a delay sampled from that model's earlier response times, independently of the current response's duration. As we increase how far each sampled delay advances the simulation, survival falls for every model. Benchmarks that pause the environment during inference can miss this source of control failure.

1 INTRODUCTION

Many agent benchmarks update the environment after a controller returns an action. They measure the choice of action while leaving response time outside the task. In a running process, the state can change during inference, so a sound command can arrive too late. A failed episode can reflect poor commands, late commands or a task that could not have been completed in the first place.

In real-time control, a process continues to evolve while the controller selects an action (Ramstedt & Pal, 2019; Thodoroff et al., 2022). In tests of large language models (LLMs) under time pressure, a faster smaller model can outperform a slower larger one (Kang et al., 2025). Recent LLM work also exposes timing directly: Real-Time Reasoning Gym advances games during inference, while AgileThinker combines a fast reactive thread with planning (Wen et al., 2026).

We introduce Thaw, a benchmark for LLM control of a deterministic outpost task. A controller directs simulated scientists across a map. They gather and process resources before depositing supplies at an outpost. Deposits replenish the outpost's buffers, which drain on every simulation tick. When a buffer empties, part of the outpost shuts down and its health measure, called integrity, falls. The episode fails when integrity reaches zero. Each episode ends at failure or at a fixed horizon, and the **survival fraction** is the proportion of that horizon completed. Figure 1 shows a replay view of the outpost and its surrounding map.

Each evaluated task condition sets demand, supply, routes and the time needed to deliver resources. We replay a successful command trace for each condition through the LLM controllers' command interface. An independent planning program supplies a trace where it can; a fixed-rule scripted controller supplies one otherwise. Each replay is a **solvability witness**: the task reaches the horizon under those commands.

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: A replay view of Thaw, with scientists working around the outpost and resource sites.

We test three ways of advancing task time during inference. The stopped clock leaves the task unchanged during inference. Under the own-latency clock, each measured model response advances task time. Under the assigned-latency clock, each turn advances by a delay sampled from that model's responses in prior episodes, regardless of the current response's duration. The **charge rate** converts seconds of measured or assigned delay into simulation ticks. Figure 2 depicts the scientists' routes and when these clocks advance. The **command gap** is the median number of simulation ticks between consecutive commands, regardless of which scientist receives them. We also run scripted controllers at different command gaps, without model inference. Their survival curves provide a timing reference for the LLM results.

Empirical findings. One condition, **Short loop**, compresses gathering, processing and travel. All three LLMs lose survival when measured response time advances the simulation; survival also falls as independently assigned delay rises. Both frontier LLMs complete Short loop with the clock stopped but survive less than half of it on average at the highest charge rate. The open-weight LLM has the lowest stopped-clock survival on every primary condition, and the models differ in survival even at a matched command gap. Thaw thus distinguishes decision quality as well as response timing.

Our contributions are:

- A deterministic LLM-control task with a successful command trace for every evaluated condition.
- Timing protocols that compare survival with inference delay excluded, measured or assigned independently of the current response.
- Results for LLM controllers across task conditions. We report survival, command timing, supplies carried per trip and time spent observing rather than commanding.

2 Task and Evaluation Design

Thaw is controlled through structured-text observations and a fixed command set. The simulation advances in discrete ticks. A controller assigns two scientists to gather, process and deposit three resources while the corresponding outpost subsystems consume them.

2.1 Task and Outcome

Each subsystem has a resource buffer that drains every tick and refills after a matching deposit. Its **buffer cap** discards any deposit above the cap. An empty buffer disables the subsystem, which reduces the outpost's integrity.

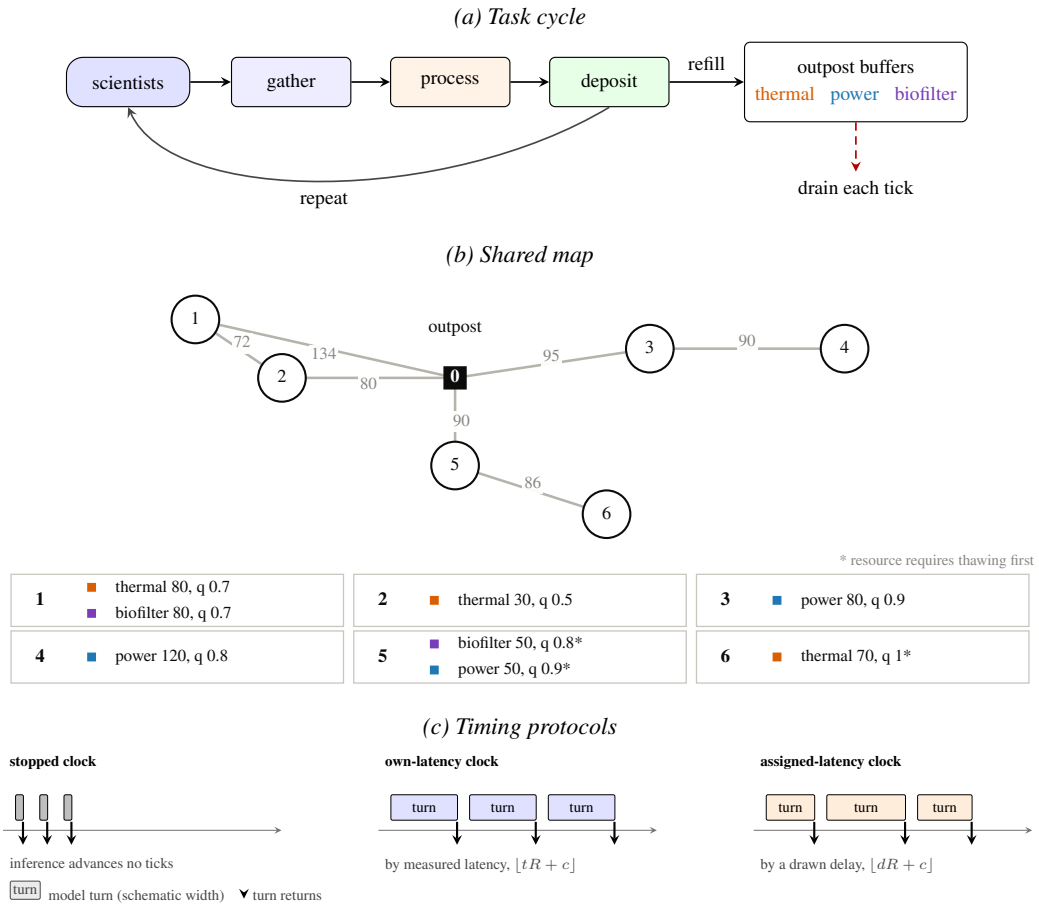

Figure 2: The Thaw task. *(a)* Scientists gather, process and deposit resources to refill buffers that drain each tick; the episode ends at zero integrity or the fixed horizon. *(b)* The fixed zone graph, with the outpost at zone 0. Its key gives resource type, maximum quantity and quality q, which affects processed yield. An asterisk marks resources that require thawing; only the condition with a thaw ramp unlocks them. *(c)* The stopped clock advances no ticks during inference. Running clocks charge rate R times measured duration t or sampled duration d, carrying the fractional remainder c between turns. A turn can command or observe.

The episode ends when integrity reaches zero or the fixed horizon is reached. Although the **simulator** can schedule random events, the evaluated conditions use fixed event schedules. The **task state** includes resource stocks and outpost integrity. With the same simulator executable, the task state follows the same trajectory for a fixed condition and command stream. The seed supplied for each episode can change the estimated resource quantities and qualities shown in observations.

A scientist's travel, gather, process and deposit sequence is the **maintenance loop**. Its duration is the condition's **loop length**. A **full-load trip** fills every **inventory slot** before the scientist returns to the outpost. **Trip load** is the number of filled slots.

2.2 Command timing

At the start of each maintenance loop and before each gather, the **reactive baseline** ranks the subsystems by buffer stock as a fraction of buffer cap, ascending. At the start of a loop, it sends each scientist to the nearest node holding the first-ranked subsystem's resource. Each gather takes that resource when the zone holds it, and otherwise the zone's first-listed resource. This scripted controller provides the timing reference for each condition.

The command gap is the median number of simulation ticks between consecutive commands to either scientist. It is the latency axis for both scripted and LLM controllers. A **scripted tolerance** is the interpolated command gap where the jittered reactive baseline's survival falls below 0.9 and stays below it. Curves with sustained earlier failures or later recovery have no such value (Appendix A). The shared command-gap axis places this task reference beside a model's survival. That comparison alone does not attribute a model failure to delay; the timing protocols provide that test. A scripted controller's **requested gap** (K) is the fixed interval at which it attempts to issue commands. **Gap jitter** is the coefficient of variation (CV) applied to that interval; it describes scripted controllers only, since a model controller has no requested gap.

For model controllers, **turn cost** is the median ticks advanced per model turn, including observe turns. The charge rate cannot serve as the latency axis: it is a knob set on the model side and has no scripted counterpart. A model controller's command gap combines two components, the turn cost and the share of turns spent observing rather than commanding: slower turns widen the gap, and so does spending more turns on observation before a command. Turn cost describes the per-turn advance, while Section 3.4 reports the fraction of episode ticks spent observing. The same command gap can arise from slow turns or repeated observations. Appendix A defines the descriptive crossing statistic used in those reference curves.

2.3 Solvability witnesses

The evaluated conditions keep the same goal and command interface, but vary demand, supply, routes, loop duration, buffer capacity and horizon (Table 1). Each evaluated condition has a successful command trace, showing that it can reach the horizon on the evaluated binary. Where a planner constructs a schedule independently of the evaluated LLMs, we replay its commands through the same interface. A round assigns both scientists to trips between the start and end ticks recorded in the schedule. It overruns when its last planned dispatch lands after the end tick. The schedule provides a separate check when replay reaches the horizon with no round overrun. This check does not establish optimality or bound model performance. Where no planned schedule is available, a reactive baseline supplies the successful trace. Appendix A describes both kinds of evidence.

2.4 Controller interface and timing protocols

A **runner** connects a controller to the simulator, applies the timing protocol and records the episode. An **LLM controller** receives the current observation and returns exactly one tool call per turn. An **observe turn** requests a new observation without commanding a scientist. The reactive baseline provides timing reference curves. A planned schedule supplies an independent replay check where one exists.

Under the **stopped clock**, model inference advances no simulation time. The controller can advance time through two additional **clock tools**: one advances a chosen number of ticks, and the other waits for an event. Neither commands a scientist. A dispatch advances the simulation by one tick, so the stopped clock does not remove all time between commands.

Under the **own-latency clock**, each model turn advances the simulation by the measured inference duration times the charge rate, retaining fractional ticks for later turns. A monotonic timer starts before message preparation and ends when the provider returns; the duration therefore includes client work, network transit and provider queueing, not just token generation. Completed context-summary requests also advance the simulation; failed summary requests are logged without advancing it. No correction removes network variation from a completed request. Under the **assigned-latency clock**, the turn advances by a delay sampled from a **recorded pool**. Each model's pool contains inference durations recorded from its own run at charge rate 8, the reference operating point for the running-clock evaluation. The own-latency clock does not identify the effect of delay: a controller with long turns has a larger command gap, but those turns may be long because the policy is worse. The assigned-latency clock breaks that link by drawing each per-turn advance from a fixed recorded pool independently of the current turn, leaving the model otherwise unchanged. Neither running protocol exposes the clock tools.

Table 1: Benchmark conditions. Changes are relative to Base. Each has a successful command trace; only Flooded route lacks a planned replay. Appendix A gives detailed evidence.

Condition	Defining change	Loop
Base (BASE)	the standard economy at the base time scale, thaw held at zero	Long
Tight economy (ECON-1)	every drain raised by a third, every buffer cap halved	Long
Tight economy with more supply (ECON-2)	every drain raised by three fifths, every node's supply by three tenths, every buffer cap halved	Long
Rising drain (RAMP)	Tight economy with increasing drain and delayed access to distant zones	Long
Flooded route (FLOOD)	Tight economy with the direct power route flooded three times, a longer bypass open	Long
Short loop (SHORT)	a short maintenance loop from much faster gather, process and travel, with all buffer caps lowered from Base	Short
Short loop, tight caps (SHORT-B)	Short loop with the thermal cap just above its starting stock and the biofilter cap at its starting stock, limiting advance deposits	Short

The recorded pools come from source episodes that end before the horizon. Their delays may therefore underrepresent the later contexts of an episode that survives longer. Appendix B reports their coverage.

The stopped-clock comparison alone cannot isolate the effect of delay: it gives the controller two additional clock tools, and each dispatch advances the simulation by one tick. The assigned-latency protocol also permits the charge rate to be chosen separately for each model so that their command gaps can be matched. Figure 2 shows how the three protocols advance the same simulation clock.

3 Experiments and analysis

3.1 Experimental setup

We evaluate three LLMs: GPT-5.6 Terra, the **larger frontier LLM**; GPT-5.6 Luna, the **smaller frontier LLM**; and NVIDIA Nemotron 3 Super, the **open-weight LLM**. **Reasoning effort** is a graded setting for the two frontier LLMs and an on-or-off setting for the open-weight LLM, so we compare effort within a model, never across the two vendors.

The seven conditions vary demand, supply, routes, loop duration and buffer capacity. Table 1 gives the defining changes; detailed settings and scripted reference curves are in Appendix A. The primary stopped-clock comparison uses Tight economy, Tight economy with more supply, Rising drain, Flooded route and Short loop. The running-clock experiments use Rising drain, Short loop and Short loop, tight caps.

An **episode** is one seeded evaluation that ends at the horizon or at zero integrity. A **run group** contains episodes with the same model, condition, timing protocol, charge rate and reasoning effort. Tables report the episode count n. A run group whose retained episodes have identical survival is marked *identical* and receives no confidence interval. The label describes observed variation, not the controller's outcome distribution.

3.2 Performance without inference delay

With inference delay removed, the LLMs still differ in survival (Figure 3; exact values in Appendix Table 4). The larger frontier LLM reaches or approaches the horizon on every primary condition. The smaller frontier LLM survives no longer than the larger one on any of them, and the open-weight LLM has the lowest survival throughout. On Tight economy with more supply, all three have the same command gap of 16.0 ticks, yet survival spans 0.255 to 0.971. The matched median gap does not account for the survival difference. Flooded route has no defined reactive tolerance, so the figure omits that reference point.

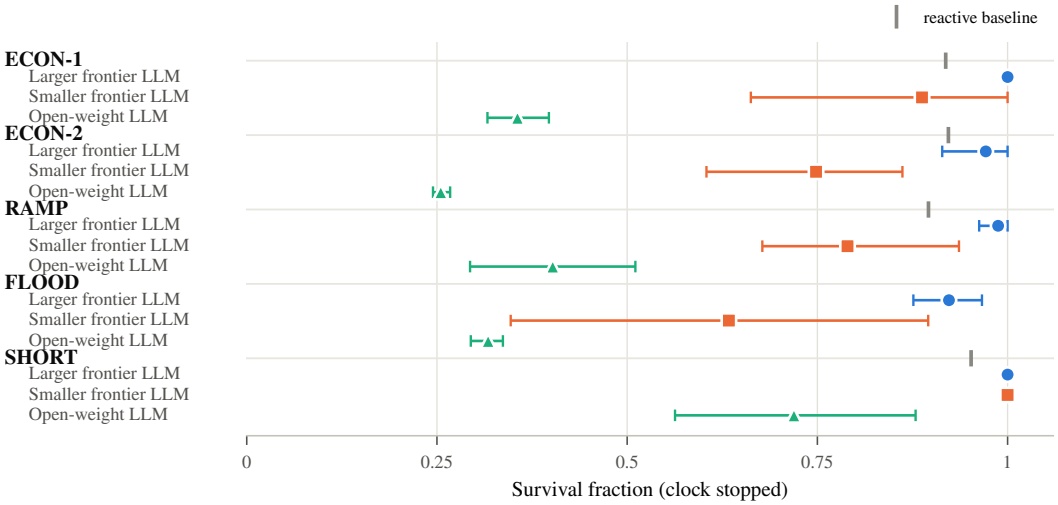

Figure 3: Stopped-clock survival by condition and LLM. Bars are percentile-bootstrap intervals except the Flooded-route smaller frontier LLM's observed range; no bar means identical observed survival. Grey ticks mark reactive-baseline survival at the largest measured gap within scripted tolerance (Table 2). Flooded route has a successful reactive trace but no defined tolerance. Exact values and effort comparisons appear in Appendix Tables 4 and 6.

Higher reasoning effort improves survival where a model otherwise falls short. On Tight economy with more supply, the smaller frontier LLM rises from 0.748 to 1.000; every higher-effort episode reaches the horizon. On Tight economy, the open-weight LLM rises from 0.356 to 0.535, with nonoverlapping intervals. Appendix Table 6 gives the episode counts and intervals for both within-model comparisons.

3.3 Effects of inference delay

Measured inference delay. On Short loop, all three LLMs have lower survival under the own-latency clock than with the clock stopped (Appendix Table 5). At the highest evaluated charge rate, the larger frontier LLM survives 0.413 of the horizon, down from 1.000 with the clock stopped. The smaller frontier LLM falls from 1.000 to 0.370, and the open-weight LLM from 0.719 to 0.292.

The larger frontier LLM's survival falls steeply as command gap grows (Figure 4). It reaches the horizon at 9.0 ticks, but survives 0.769 at 13.2 ticks and 0.413 at 24.2 ticks. The reactive baseline's scripted tolerance is 9.9 ticks [9.0, 10.2]. The tolerance lies between the model's full-survival observation and its next, lower-survival observation.

Assigned inference delay. Assigning longer delays also lowers survival for every LLM on Short loop. The larger frontier LLM reaches the horizon at a command gap of 8.0 ticks, but survives only 0.413 at 25.0 ticks. Each assigned delay is sampled independently of the current response duration, so this comparison tests the effect of applied delay. Both run groups use the same tool set, without the clock tools available when task time is paused.

At similar command gaps, the smaller frontier LLM reaches the horizon in all 5 episodes at 12.0 ticks. The larger frontier LLM survives 0.772 at 13.0 ticks, and the open-weight LLM survives 0.513 at 12.5 ticks. With longer assigned delays, the smaller frontier LLM falls to 0.380 and the open-weight LLM to 0.392.

Variation across task configurations. Rising drain and Short loop differ in demand, supply and buffers as well as action times.

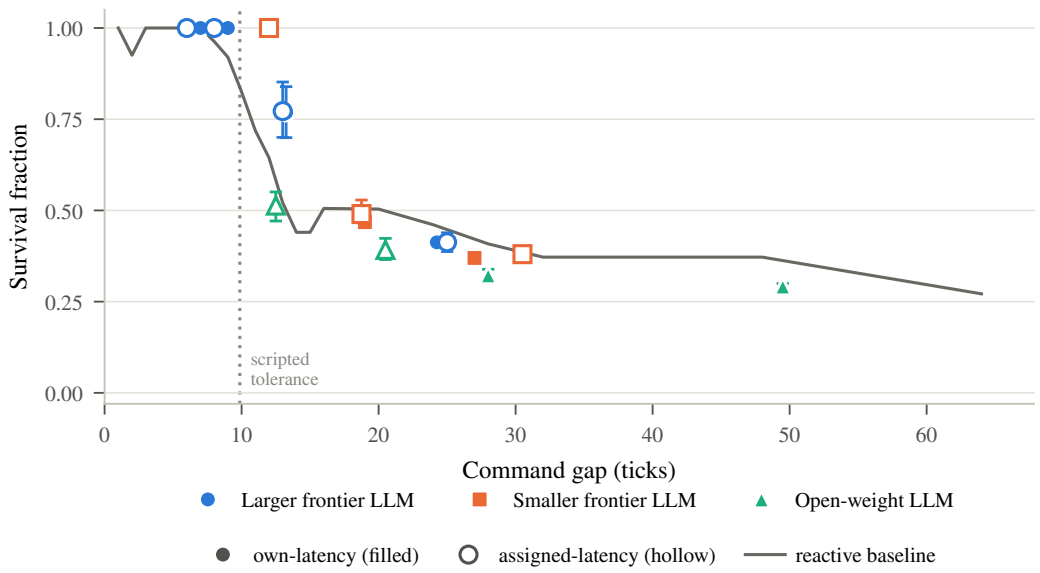

Figure 4: Short-loop survival against command gap. Filled markers show own-latency run groups; hollow markers show assigned-latency run groups. Bars are percentile bootstrap intervals; no bar means identical observed survival. The fixed-gap curve (gap jitter CV 0, one seed per point) and vertical tolerance rule (gap jitter CV 0.5, ten seeds) use different sweeps. Command gap: ticks between consecutive commands, to either scientist. Scripted tolerance: where the jittered reactive baseline first crosses below 0.9, not the largest gap it survives.

At the same charge rate, the larger frontier LLM survives 0.978 of Rising drain at a command gap of 26.0 ticks, but only 0.769 of Short loop despite a smaller gap of 13.2 ticks (Figure 5).

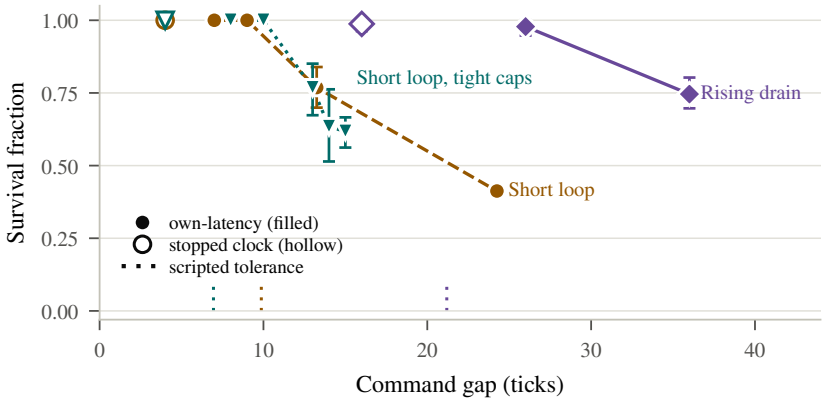

Figure 5: Larger frontier LLM survival across three conditions and two loop lengths. Hollow points show the stopped clock; lines join own-latency run groups. The short-loop variants share a stopped-clock point. Colored dotted ticks mark scripted tolerances. Command gap: ticks between consecutive commands, to either scientist. Scripted tolerance: where the jittered reactive baseline first crosses below 0.9, not the largest gap it survives.

The reactive scripted tolerance falls from 9.9 ticks on Short loop to 7.0 on Short loop, tight caps. Yet the larger frontier LLM reaches the horizon at a command gap of 10.0 ticks, beyond the tighter condition's scripted tolerance. The lower scripted tolerance therefore does not locate the LLM's failure threshold.

3.4 CONTROLLER BEHAVIOR

On Short loop, the controllers differ in how they carry resources and spend time between commands (Figure 6). These trace features do not enter the survival score.

With the clock stopped, the open-weight LLM carries 1.05 inventory slots per trip, fewer than either frontier LLM. A fuller trip spreads fixed travel time over more resources. For the **binding subsystem**, the one with the smallest resupply margin, the open-weight LLM delivers 0.404 of demand, compared with 0.944 and 1.118 for the frontier LLMs.

Under the own-latency clock, **observe share** measures the fraction of episode ticks attributable to observation-only turns. Those turns advance the simulation without commanding a scientist. At the highest evaluated charge rate, the open-weight LLM spends 0.443 of episode ticks observing, compared with 0.089 for the larger frontier LLM; the smaller frontier LLM lies between them.

On Short loop, the open-weight LLM combines shorter supply trips with more observation time than the larger frontier LLM.

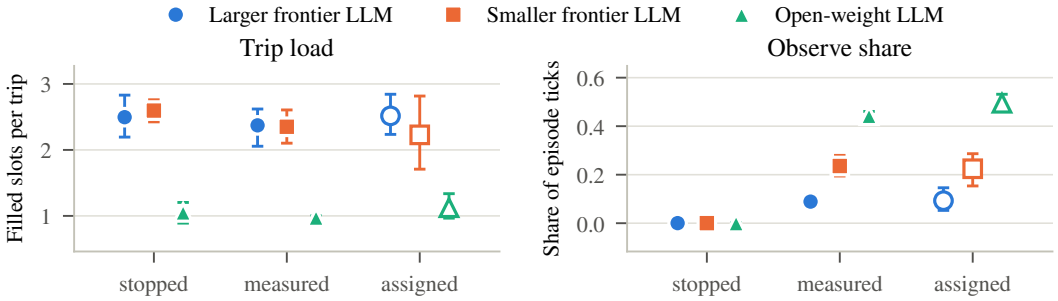

Figure 6: Trip load and observe share on Short loop. Running-clock points use each model's highest charge rate with retained behavior data; stopped-clock points have no charge rate. Bars show available intervals; hollow markers show assigned latency. Axis labels measured and assigned mean own-latency and assigned-latency clocks.

4 RELATED WORK

Real-time reinforcement learning lets the state evolve while a controller selects an action (Ramstedt & Pal, 2019). Two lines of work formalize the delay itself. Concurrent control has the controller select its next action while the previous one is still executing (Xiao et al., 2020), and delayed-action formulations study environments where a command takes effect only after a fixed delay (Derman et al., 2021). Thodoroff et al. (2022) evaluate continuous-control algorithms as their decision cost changes. For LLM controllers, Kang et al. (2025) report that a faster small model can outperform a slower large one under a real-time deadline, and select model size and quantization according to real-time demand. The `llm-colosseum` project reports API latency as one cause of poor play in real-time Street Fighter III (OpenGenerativeAI/llm-colosseum). Neither line of evidence supplies a latency axis shared across conditions and controller types. Thaw's axis is the command gap, the median number of ticks of world time between consecutive commands to either scientist.

Many LLM-facing benchmarks score artifacts or states without advancing the environment during inference. AgentBench spans interactive tasks, SWE-bench scores repository patches, and WebArena scores task completion (Liu et al., 2024; Jimenez et al., 2024; Zhou et al., 2024). Robotouille instead tests asynchronous planning with overlapping actions and time delays (Gonzalez-Pumariega et al., 2025). Zheng et al. (2025) convert inference delay into simulation frames and evaluate a controller that pairs immediate reflex actions with asynchronous LLM reflection. Real-Time Reasoning Gym uses generated tokens per game step for its main time-pressure sweep and also reports a wall-clock test of AgileThinker (Wen et al., 2026). Thaw compares the stopped, own-latency and assigned-latency clocks on one task with replayed solvability witnesses. The assigned-latency clock draws from a fixed pool independently of the current turn, separating applied delay from that turn's inference duration.

Test-time computation can vary with the problem. Snell et al. (2025) allocate reasoning across prompts by difficulty, and Wang et al. (2025) select models and compute budgets across subtasks of a multi-stage task. Other methods reduce inference cost through early exit, model cascades or routing (Schuster et al., 2022; Chen et al., 2023; Ong et al., 2025). These methods evaluate answer quality against computation or serving cost. Under its own-latency clock, Thaw advances the simulation during inference and scores survival. It can therefore test whether a policy's extra computation improves its commands enough to offset their later arrival, although this evaluation uses fixed effort settings.

5 Discussion

Successful command traces show that every evaluated condition can reach the horizon. With inference delay removed, the models still differ in survival; giving a model more reasoning effort can help when it otherwise falls short. On Short loop, survival declines when measured response time advances the simulation. It also declines for every model as assigned delay rises independently of the current response. The assigned-delay result identifies a delay effect within that protocol.

Rising drain and Short loop respond differently to a running clock, but their demand, supply and buffer settings also differ. The tighter-cap Short loop condition shows that a scripted controller's tolerance does not determine the LLM's survival curve. We did not test higher reasoning effort under a running clock, where any improvement in the next command would have to offset the time spent producing it.

Future work. An adaptive controller could choose its model, reasoning effort, observation timing or how far ahead to plan from the current buffer levels. It could be compared with fixed policies under the same assigned-delay distribution, using survival and command gap as outcomes. New configurations can vary demand, routes and buffer capacity; a successful command trace should again establish that each can reach the horizon. An independently planned replay provides an additional check where one is available.

6 Conclusion

Thaw tests whether LLM controllers can keep a simulated outpost supplied while its resources drain during inference. Survival is the share of a fixed episode completed before failure. Command traces from a planning program or fixed-rule controller reach the fixed horizon in every evaluated task configuration when replayed without LLM calls. On Short loop, where gathering, processing and travel are brief, the larger frontier LLM reaches that endpoint when the simulation clock pauses during inference. When measured response times advance the clock, its mean survival falls to 0.413. In a separate test, we advance the clock by delays sampled from each model's responses in prior episodes, independently of the current response. As these assigned delays increase, survival falls for all three LLMs. At each model's longest tested delay on Short loop, none reaches the endpoint.

AI Use Statement

We used generative AI tools to help write and edit the manuscript text, to help write the analysis and simulation code and its tests, and to review the manuscript for readability and comprehension. We did not use them to design the methodology, generate research hypotheses, produce or clean the experimental data, translate text, or construct proofs. The authors reviewed all AI-assisted work and take responsibility for the final content.

Reproducibility Statement

Every episode runs on a CPU-only simulator. With the same simulator executable, the task state follows the same trajectory for a fixed condition and command stream, but the runner's per-episode seed can change estimated resource quantities and qualities in observations. Each condition includes a successful command trace. Six also include a schedule constructed independently of the evaluated

LLMs. The code will be open sourced. Tables and figures are regenerated from the recorded episode results, and the appendix gives the commands that take a clean checkout to a finished episode and replay a condition's witness.

Ethics Statement

This work evaluates language models controlling a synthetic supply-chain simulation. It involves no human subjects, no personal or sensitive data, and no deployment in a consequential setting. We reached the models through a hosted provider under its terms of use. The benchmark measures controller performance when inference delay advances simulation time. It does not measure safety, and a result here does not describe how a model behaves away from this task. All authors have read and acknowledge the Code of Ethics.

References

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance, 2023. URL https://arxiv.org/abs/2305.05176v1.

Esther Derman, Gal Dalal, and Shie Mannor. Acting in delayed environments with non-stationary Markov policies, 2021. URL https://arxiv.org/abs/2101.11992v4.

Gonzalo Gonzalez-Pumariega, Leong Yean, Neha Sunkara, and Sanjiban Choudhury. Robotouille: An asynchronous planning benchmark for LLM agents. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 83757–83792, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/d033407a1a4b7a75cd5f9e4575ad9fb5-Paper-Conference.pdf.

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 B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/edac78c3e300629acfe6cbe9ca88fb84-Paper-Conference.pdf.

Hao Kang, Qingru Zhang, Han Cai, Weiyuan Xu, Tushar Krishna, Yilun Du, and Tsachy Weissman. Win fast or lose slow: Balancing speed and accuracy in latency-sensitive decisions of LLMs, 2025. URL https://arxiv.org/abs/2505.19481v1.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/e9df36b21ff4ee211a8b71ee8b7e9f57-Paper-Conference.pdf.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 34433–34448, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/5503a7c69d48a2f86fc00b3dc09de686-Paper-Conference.pdf.

OpenGenerativeAI/llm-colosseum. OpenGenerativeAI/llm-colosseum, n.d. URL https://github.com/OpenGenerativeAI/llm-colosseum.

Simon Ramstedt and Chris Pal. Real-time reinforcement learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.,

2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/54e36c5ff5f6a1802925ca009f3ebb68-Paper.pdf.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. In *Advances in Neural Information Processing Systems 35*, pp. 17456–17472. Neural Information Processing Systems Foundation, Inc. (NeurIPS), 2022. doi: 10.52202/068431-1269. URL https://doi.org/10.52202/068431-1269.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 10131–10165, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/1b623663fd9b874366f3ce019fdfdd44-Paper-Conference.pdf.

Pierre Thodoroff, Wenyu Li, and Neil D. Lawrence. Benchmarking real-time reinforcement learning. In Samuel Albanie, João F. Henriques, Luca Bertinetto, Alex Hernández-García, Hazel Doughty, and Gül Varol (eds.), *NeurIPS 2021 Workshop on Pre-registration in Machine Learning*, volume 181 of *Proceedings of Machine Learning Research*, pp. 26–41. PMLR, 13 Dec 2022. URL https://proceedings.mlr.press/v181/thodoroff22a.html.

Fali Wang, Hui Liu, Zhenwei Dai, Jingying Zeng, Zhiwei Zhang, Zongyu Wu, Chen Luo, Zhen Li, Xianfeng Tang, Qi He, and Suhang Wang. AgentTTS: Large language model agent for test-time compute-optimal scaling strategy in complex tasks. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 98396–98433. Curran Associates, Inc., 2025. doi: 10.52202/085713-3288. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/8d9bbba8cac9cabb54e85ee7f21441c8-Paper-Conference.pdf.

Yule Wen, Yixin Ye, Yanzhe Zhang, Diyi Yang, and Hao Zhu. Real-time reasoning agents in evolving environments. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 126266–126292, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/ccbe16043125599293b01dd467c260f3-Paper-Conference.pdf.

Ted Xiao, Eric Jang, Dmitry Kalashnikov, Sergey Levine, Julian Ibarz, Karol Hausman, and Alexander Herzog. Thinking while moving: Deep reinforcement learning with concurrent control, 2020. URL https://arxiv.org/abs/2004.06089v4.

Yangqing Zheng, Shunqi Mao, Dingxin Zhang, and Weidong Cai. Reflex first, reflect later: Latency-aware embodied LLM agents for dynamic response, 2025. URL https://arxiv.org/abs/2506.07223v2.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/4410c0711e9154a7a2d26f9b3816d1ef-Paper-Conference.pdf.

A Conditions and scripted reference curves

Every condition uses the map in Figure 2. The conditions vary the economy and horizon; only the Flooded route changes the map. It adds a route from zone 5 to zone 3. During each flood, the direct route from the outpost to zone 3 closes, leaving the bypass through zone 5 open.

Table 1 gives the defining change, loop class, and solvability witness for each condition. Table 2 reports the detailed settings and scripted evidence.

Table 2: Condition settings and scripted evidence. Horizon and idle loss are computed from the episode file. Idle loss is the tick at which integrity reaches zero when no command is issued. The reactive tolerance is the command gap at which the jittered reactive baseline first falls below 0.9 after reaching it; brackets give the seed-bootstrap interval and n gives the seed count. The jittered baseline also falls below 0.9 at a smaller command gap on ECON-2 (at command gap 11.5). A curve for which the rule returns no tolerance is marked *not defined*. The stretched planned schedule uses a fixed requested gap and does not re-plan. Command gap: median ticks of world time between consecutive commands, to either scientist.

Condition	Horizon (ticks)	Idle loss (tick)	Reactive tolerance	Stretched planned tolerance
BASE	18000	5065	55.1 [52.6, 56.1], n 10	50.6, n 1, no interval
ECON-1	18000	4011	25.5 [21.7, 27.3], n 10	43.1, n 1, no interval
ECON-2	18000	3384	16.0 [15.0, 17.0], n 10	39.3, n 1, no interval
RAMP	18000	3963	21.2 [19.0, 22.4], n 10	34.9, n 1, no interval
FLOOD	18000	4011	*not defined*	*no schedule*
SHORT	3500	949	9.9 [9.0, 10.2], n 10	14.1, n 1, no interval
SHORT-B	3500	949	7.0 [6.5, 7.6], n 10	11.4, n 1, no interval

A.1 What each condition changes

Base uses the standard economy and time scale with the thaw ramp held at zero. The other conditions modify one or more parts of this configuration.

Tight economy raises all subsystem drains by a common factor and halves all buffer caps. The map, resource nodes, horizon, scientists and inventory slots remain unchanged.

Tight economy with more supply raises the same three drains by a larger common factor and halves the same three caps. It also raises the maximum quantity and regeneration rate of every resource node by one common factor. No other condition changes the available supply.

Rising drain is Tight economy with the thaw ramp switched on. The ramp rate and acceleration are the only additional changes; they make demand nonstationary and make the two distant zones available during the episode.

Flooded route is Tight economy with three timed disasters added. Each floods the direct route to the power resource for the same number of ticks, while the longer bypass through zone 5 remains open. This condition adds that bypass to the shared map.

Short loop lowers all three buffer caps from Base and raises all three drains by a common factor. It shortens each maintenance loop: gathering falls from 15 ticks to 3, processing from 30 to 5, and a scientist's movement speed rises from 0.5 to 9, and scientist energy regenerates faster. Table 1 reports the shorter horizon.

Short loop, tight caps is Short loop with its thermal and biofilter caps lowered again and the power cap untouched. The biofilter cap equals its starting stock, while the thermal cap retains a small amount of headroom. These caps limit advance deposits to either subsystem.

A.2 Planned-schedule construction

An independently planned schedule undergoes three checks. Static feasibility verifies that the reachable supply covers all subsystem demands over the horizon. A planner then searches assignment sequences exhaustively and maximizes the minimum subsystem buffer. It records the sequence, the location and time of the minimum, the planned deposits per subsystem, and whether the search completed. Finally, schedule replay issues the sequence to the simulator with the command gate open at every tick. The replay qualifies as an independent check only if the episode reaches the horizon and no planned dispatch lands after its round's recorded end tick. Each condition in the main stopped-clock comparison except Flooded route meets these requirements (Section 2.3). Base and Short loop, tight caps also have planned schedules. Base is used for calibration; Short loop, tight caps is used for the held-out comparison in Section 3.3.

A successful replay establishes that the task is solvable through the benchmark command interface on the evaluated binary. It does not set a performance ceiling. The planner synchronizes its rounds and uses a full trip load on every trip. Its round model omits the simulator's one-tick advance on each dispatch and models deposits as slightly slower than they are in the simulator. In calibration, its predicted outcome does not exceed the simulator outcome. A model controller can therefore outperform the planned schedule.

The planner has no schedule for the Flooded route because it does not model a scientist stopped part way along a flooded route. A reactive-baseline run does reach the horizon. This trace proves feasibility on the evaluated binary, but does not provide the separate planned-schedule check.

The Flooded route's stopped-clock results appear in Figure 3 and Appendix Table 4. Retained running-clock groups exist for the smaller frontier and open-weight LLMs but not for the larger frontier LLM. These groups remain in the released ledger; the principal running-clock comparison uses Short loop.

A.3 The two scripted reference curves

Both curve families use the same binary. The **patient baseline** uses the reactive baseline's action rule but checks again at a fixed interval if a gather or process remains pending at its expected completion. The reactive and patient controllers run on all seven conditions over a grid of requested gaps that extends beyond each published tolerance. Where a schedule exists, the planned controller uses a coarser grid, producing wider brackets. The fixed curve uses one seed per grid point; the jittered curve uses nine or ten. At the smallest requested gap, which schedule replay also uses, the command gate remains open. Gap jitter is therefore inactive, and the ten jittered seeds produce one deterministic episode.

Both sides of every comparison in this paper are measured the same way, on the command gap. The scripted command gap is the median of the differences between consecutive commands in each episode's own dispatch log, merged across the two scientists and restricted to the commanding tools. That is the quantity and the filter used for model traces. The simulator's per-tick record counts busy-action onsets instead, omitting zero-duration deposits and overestimating the gaps. All published tolerances use dispatch logs.

A.4 Fixed-gap aliasing

At gap jitter CV 0, the reactive baseline has a nonmonotonic outcome on four of the seven conditions. It can reach the horizon at a larger requested gap after failing at a smaller one. A crossing on such a curve does not define a stable tolerance. The jittered reactive curves do not show this fixed-phase alternation.

Table 3 reports one of three outcomes for every curve. A *crossing* falls below the threshold after reaching it and does not recover; only this outcome yields a published tolerance. An isolated point below the threshold before the curve first reaches it does not remove the crossing. This occurs on Tight economy with more supply under jitter. An *aliased* curve climbs back above the threshold after its first crossing. On fixed curves, this is the alternation described above. Two jittered curves also recover after crossing, without fixed-phase alternation. A *sub-threshold* curve has two or more consecutive grid locations under the threshold before the crossing. Neither an aliased nor a sub-threshold curve yields a tolerance.

Of the seven conditions only the Flooded route has a sub-threshold jittered reactive curve. Table 1 therefore reports no tolerance for this condition. The baseline succeeds only within a band of command gaps, with failures both above and below it.

All published tolerances use the reactive baseline. Under jitter, the patient baseline has no tolerance on two conditions, so the table does not combine the two controllers in one tolerance column. Four conditions are aliased on the fixed reactive curve: Tight economy, Tight economy with more supply, Rising drain, and Flooded route. Each retains Base's time scale and horizon while halving its buffer caps and raising its drains. Base has the same long maintenance loop at the standard economy and is not aliased on any curve. Its fixed reactive curve stays at or above the threshold until the top of the grid, then crosses without recovery. The ledger reports the fixed-gap value alongside the jittered

Table 3: Tolerance-rule outcomes for the scripted curves. A crossing remains below the threshold after first reaching it and yields a tolerance. An aliased curve later recovers above the threshold. A sub-threshold curve has at least two consecutive locations under the threshold before the crossing. Published tolerances appear in Table 2. The Flooded route has no planned schedule or planned curve.

	Reactive baseline		Patient baseline		Planned
Condition	CV 0	CV 0.5	CV 0	CV 0.5	CV 0
Base	crossing	crossing	crossing	crossing	crossing
Tight economy	aliased	crossing	aliased	crossing	crossing
Tight economy, more supply	aliased	crossing	aliased	aliased	crossing
Rising drain	aliased	crossing	aliased	crossing	crossing
Flooded route	aliased	sub-threshold	aliased	aliased	no schedule
Short loop	crossing	crossing	crossing	crossing	crossing
Short loop, tight caps	crossing	crossing	aliased	crossing	crossing

Table 4: Survival fraction with the clock stopped on five conditions, the exact values Figure 3 plots. Four have an independently planned schedule; Flooded route has a successful reactive trace but no planned schedule or defined tolerance. Where defined, the reactive baseline is shown at the largest measured gap at or below its jittered tolerance (Table 2). Model columns use the lower reasoning effort; Table 6 reports the effort comparison. Except the Flooded-route smaller frontier LLM's observed range, brackets are percentile-bootstrap intervals; their absence means identical observed survival.

Condition	Reactive	Larger frontier LLM	Smaller frontier LLM	Open-weight LLM
ECON-1	0.92, gap 24.0	1.000, n 8	0.888 [0.663, 1.000], n 4	0.356 [0.316, 0.397], n 9
ECON-2	0.92, gap 16.0	0.971 [0.914, 1.000], n 7	0.748 [0.604, 0.862], n 10	0.255 [0.245, 0.267], n 9
RAMP	0.90, gap 21.0	0.988 [0.963, 1.000], n 7	0.790 [0.678, 0.936], n 4	0.402 [0.294, 0.511], n 5
FLOOD	*no defined tolerance*	0.923 [0.876, 0.966], n 7	0.634 [0.347, 0.896], n 3	0.317 [0.294, 0.337], n 5
SHORT	0.95, gap 9.0	1.000, n 5	1.000, n 10	0.719 [0.563, 0.879], n 6

PROVENANCE n 99; seeds 201 to 218; stopped clock.

value in Table 1. Neither short-loop condition is aliased on the fixed reactive curve, although Short loop, tight caps is aliased on the fixed patient curve.

The alternation is consistent with an interaction between re-planning phase and limited buffer capacity, but the evaluated conditions do not separate their contributions. Base has a long maintenance loop without tightened buffer caps and does not alternate. Short loop, tight caps cannot hold an advance deposit in two subsystems and alternates under the patient controller. The reported tolerance is measured under jitter because a fixed-gap curve can recover after crossing the threshold.

B STOPPED-CLOCK, CLOCKED AND REASONING-EFFORT RESULTS

These tables report the complete run-group estimates behind the figures in Sections 3.2 and 3.3. Table 4 gives the exact stopped-clock survival fractions that Figure 3 plots. The frontier LLMs and the open-weight LLM expose reasoning effort differently, so the blocks of Table 6 must be interpreted separately.

Table 7 reports tail gaps for every retained Short loop low-effort running-clock run group. Unlike the run-group median command gap, its percentile pools individual gaps, so longer episodes contribute more intervals. It is a descriptive check for long pauses, not a claim that a particular pause caused failure. Table 8 shows the recorded-delay sources. Every source trace ends before the horizon, and their lengths differ across models. Assigned delays are sampled from these traces without matching

Table 5: Run groups on both short-loop conditions, Short loop (SHORT) and the held-out Short loop, tight caps (SHORT-B), under every clock setting. Command gap: ticks between consecutive commands, to either scientist. Charge rate: simulation ticks advanced per second of measured or assigned delay. Identical means the retained episodes have the same survival, so no interval is printed.

Condition	Clock setting	Charge rate	Command gap	n	Survival	Interval
Larger frontier LLM						
SHORT	stopped clock	stopped	4.0	5	1.000	identical
SHORT	own-latency clock	4 Hz	7.0	5	1.000	identical
SHORT	own-latency clock	6 Hz	9.0	10	1.000	identical
SHORT	own-latency clock	8 Hz	13.2	10	0.769	[0.699, 0.839]
SHORT	own-latency clock	16 Hz	24.2	10	0.413	[0.396, 0.429]
SHORT	assigned-latency clock	2 Hz	6.0	2	1.000	identical
SHORT	assigned-latency clock	4 Hz	8.0	5	1.000	identical
SHORT	assigned-latency clock	8 Hz	13.0	10	0.772	[0.700, 0.852]
SHORT	assigned-latency clock	16 Hz	25.0	10	0.413	[0.387, 0.439]
SHORT-B	stopped clock	stopped	4.0	3	1.000	identical
SHORT-B	own-latency clock	4 Hz	8.0	3	1.000	identical
SHORT-B	own-latency clock	6 Hz	10.0	5	1.000	identical
SHORT-B	own-latency clock	7 Hz	13.0	5	0.768	[0.673, 0.851]
SHORT-B	own-latency clock	7.5 Hz	14.0	5	0.634	[0.514, 0.762]
SHORT-B	own-latency clock	8 Hz	15.0	5	0.618	[0.562, 0.666]
Smaller frontier LLM						
SHORT	stopped clock	stopped	4.0	10	1.000	identical
SHORT	own-latency clock	8 Hz	19.0	39	0.468	[0.447, 0.491]
SHORT	own-latency clock	16 Hz	27.0	39	0.370	[0.356, 0.383]
SHORT	assigned-latency clock	4 Hz	12.0	5	1.000	identical
SHORT	assigned-latency clock	8 Hz	18.8	10	0.490	[0.452, 0.528]
SHORT	assigned-latency clock	16 Hz	30.5	10	0.380	[0.361, 0.399]
Open-weight LLM						
SHORT	stopped clock	stopped	1.0	6	0.719	[0.563, 0.879]
SHORT	own-latency clock	8 Hz	28.0	10	0.322	[0.306, 0.339]
SHORT	own-latency clock	16 Hz	49.5	10	0.292	[0.285, 0.300]
SHORT	assigned-latency clock	2 Hz	12.5	10	0.513	[0.471, 0.551]
SHORT	assigned-latency clock	4 Hz	20.5	10	0.392	[0.365, 0.423]

PROVENANCE n 252; seeds 201 to 240; stopped clock, own-latency clock, assigned-latency clock.

episode progress, so this intervention does not test matched late-episode delays. The open-weight pool predates the runner correction required for scored episodes. Those traces supply assigned delays, not the scored outcomes in Table 5.

Table 6: Reasoning effort by model, condition and clock setting. Each row compares lower and higher effort within a model. The frontier LLMs use graded effort; the open-weight LLM uses effort off or on, so the blocks are not compared with each other. Each entry gives survival fraction and sample size n. Intervals appear when outcomes vary within a run group. Without an interval, the entry states whether the group has identical observed survival or one episode.

Model	Condition	Clock setting	Lower effort	Higher effort
Graded effort (frontier LLMs)				
Larger frontier LLM	FLOOD	stopped clock	0.923 [0.876, 0.966], n 7	0.943 [0.881, 1.000], n 4
Smaller frontier LLM	ECON-2	stopped clock	0.748 [0.604, 0.862], n 10	1.000 identical, n 6
Effort off or on (open-weight LLM)				
Open-weight LLM	ECON-1	stopped clock	0.356 [0.316, 0.397], n 9	0.535 [0.491, 0.575], n 7

PROVENANCE n 43; seeds 201 to 210; stopped clock.

Table 7: Tail command gaps on Short loop at low reasoning effort. Each gap is the tick difference between consecutive dispatches to either scientist; the 95th percentile and maximum pool intervals over the retained episodes of each run group. The median is the existing median of episode medians. These descriptive tails are not failure thresholds.

Model	Clock	Hz	Episodes	Median	P95	Max
Smaller	Assigned	4	5	12.0	24.0	41
Smaller	Assigned	8	10	18.8	37.0	58
Smaller	Assigned	16	10	30.5	60.3	99
Smaller	Own	8	39	19.0	42.0	149
Smaller	Own	16	39	27.0	66.0	165
Open-weight	Assigned	2	10	12.5	27.8	440
Open-weight	Assigned	4	10	20.5	51.0	79
Open-weight	Own	8	10	28.0	93.4	138
Open-weight	Own	16	10	49.5	129.5	175
Larger	Assigned	2	2	6.0	16.0	75
Larger	Assigned	4	5	8.0	18.0	45
Larger	Assigned	8	10	13.0	28.0	55
Larger	Assigned	16	10	25.0	47.0	105
Larger	Own	6	10	9.0	19.0	75
Larger	Own	4	5	7.0	17.0	57
Larger	Own	8	10	13.2	26.0	97
Larger	Own	16	10	24.2	53.0	101

Table 8: Coverage of the assigned-delay source pools, taken from each model's own Short-loop run at the reference charge rate. All source episodes ended before the horizon. Durations are measured model turns, including observe turns; the latest tick is the latest source-trace tick, not a duration.

Model	Source episodes	Durations	Latest tick
Larger	7	1321	3348
Smaller	10	1186	2165
Open-weight	4	175	1180

C CONTROLLER INTERFACE AND REPRODUCTION PROCEDURE

This appendix documents the controller interface and a minimal procedure for running an episode. It reproduces one controller observation, the system prompt, and the tools available under each clock setting. The final tables map paper terminology to identifiers in the runner and data files.

C.1 ONE OBSERVATION

Each turn, the controller calls one tool and receives its result as an observation. The simulator can also render the world as text, but the runner does not pass that form to the controller. Instead, `render_observation_json` in `openreward/observation.py` serializes the same state as one line of JSON. Every tool that reports a scientist's state returns this form; `get_rules` alone returns prose.

Below is one such payload, copied from the trace of an episode on Short loop with the larger frontier LLM under the own-latency clock at charge rate 8.


```
# Short loop, larger frontier LLM, own-latency clock at charge rate 8.
# The payload is one line of JSON with no spaces, reformatted here for
# readability; nothing else is changed.
{
  "tick": 216, "sim_time": 108.0,
  "agent": {
    "id": 1, "zone": 0, "zone_terrain": "frozen", "zone_thaw_level": 0.0,
    "energy": 100.0, "max_energy": 100.0,
    "location": {
      "kind": "in_zone", "zone_id": 0, "commanded_target_zone": null
    },
    "inventory": [
      {"slot": 0, "item": null, "quantity": 0, "quality": 0.0},
      {"slot": 1, "item": "CryoCrystal", "quantity": 5, "quality": 0.9},
      {"slot": 2, "item": "CryoCrystal", "quantity": 5, "quality": 0.9},
      {"slot": 3, "item": "CryoCrystal", "quantity": 5, "quality": 0.9}
    ],
    "corruption_phase": "clean", "biosphere_exposure": 0.0,
    "last_action_outcome": {
      "action_submitted": {"action_type": "deposit", "target": 0},
      "action_executed": {"action_type": "deposit", "target": 0},
      "result": "success", "detail": "deposited 3 FuelRod",
      "outcome_tick": 216
    }
  },
  "available_actions": {
    "can_gather": [],
    "can_process": [
      {
        "station_id": 0, "position": {"x": 0.0, "y": 0.0},
        "open_slots": 2,
        "processable_inventory_slots": [1, 2, 3]
      }
    ],
    "can_deposit": [], "can_move": true, "can_repel": null
  },
  "base": {
    "integrity": 100.0, "integrity_state": "ok",
    "subsystems": {
      "thermal": {
        "buffer": 14.815964, "buffer_max": 30.0, "health": 100.0,
        "operational": true, "status": "low",
        "offline_effect": "regen at 25%"
      },
      "power": {
        "buffer": 19.49823, "buffer_max": 30.0, "health": 100.0,
        "operational": true, "status": "ok",
```

```
        "offline_effect": "process 2x slower"
      },
      "biofilter": {
        "buffer": 17.615417, "buffer_max": 30.0, "health": 100.0,
        "operational": true, "status": "ok",
        "offline_effect": "contamination rises"
      }
    },
    "n_offline": 0, "integrity_drain_floor_when_offline_per_tick": null,
    "integrity_drain_floor_is_lower_bound": false,
    "power_recovery_window": false
  },
  "team": [
    {
      "agent_id": 0, "energy": 100.0, "max_energy": 100.0,
      "corruption_phase": "clean", "zone": 0, "route_id": null,
      "action": "process"
    }
  ],
  "world": [
    {
      "zone_id": 0, "thaw_level": 0.0, "biosphere_activity": 0.0,
      "has_leviathan": false, "noise_level": 0.0, "resources": []
    },
    {
      "zone_id": 1, "thaw_level": 0.0, "biosphere_activity": 0.0,
      "has_leviathan": false, "noise_level": 0.500002,
      "resources": [
        {
          "node_id": 7, "type": "GeothermalOre", "quantity": 62.762825,
          "quality": 0.7003507, "quantity_max": 80.0,
          "regen_rate": 0.018,
          "accessible": true, "gathered_by": null
        },
        {
          "node_id": 8, "type": "XenofloraSample", "quantity": 79.92719,
          "quality": 0.69968146, "quantity_max": 80.0,
          "regen_rate": 0.0216,
          "accessible": true, "gathered_by": null
        }
      ]
    },
    {
      "zone_id": 2, "thaw_level": 0.0, "biosphere_activity": 0.0,
      "has_leviathan": false, "noise_level": 0.0,
      "resources": [
        {
          "node_id": 9, "type": "GeothermalOre", "quantity": 30.043962,
          "quality": 0.50036633, "quantity_max": 30.0,
          "regen_rate": 0.009,
          "accessible": true, "gathered_by": null
        }
      ]
    },
    {
      "zone_id": 3, "thaw_level": 0.0, "biosphere_activity": 0.0,
      "has_leviathan": false, "noise_level": 0.0,
      "resources": [
        {
          "node_id": 10, "type": "CryoCrystal", "quantity": 63.3879,
          "quality": 0.8993655, "quantity_max": 80.0,
          "regen_rate": 0.0216,
          "accessible": true, "gathered_by": null
        }
      ]
```

```
      },
      {
        "zone_id": 4, "thaw_level": 0.0, "biosphere_activity": 0.0,
        "has_leviathan": false, "noise_level": 0.0,
        "resources": [
          {
            "node_id": 11, "type": "CryoCrystal", "quantity": 119.62367,
            "quality": 0.7987456, "quantity_max": 120.0,
            "regen_rate": 0.024,
            "accessible": true, "gathered_by": null
          }
        ]
      },
      {
        "zone_id": 5, "thaw_level": 0.0, "biosphere_activity": 0.0,
        "has_leviathan": false, "noise_level": 0.0,
        "resources": [
          {
            "node_id": 12, "type": "XenofloraSample", "quantity": 49.8875,
            "quality": 0.7991, "quantity_max": 50.0, "regen_rate": 0.009,
            "accessible": false, "gathered_by": null
          },
          {
            "node_id": 13, "type": "CryoCrystal", "quantity": 49.978622,
            "quality": 0.8998076, "quantity_max": 50.0,
            "regen_rate": 0.012,
            "accessible": false, "gathered_by": null
          }
        ]
      },
      {
        "zone_id": 6, "thaw_level": 0.0, "biosphere_activity": 0.0,
        "has_leviathan": false, "noise_level": 0.0,
        "resources": [
          {
            "node_id": 14, "type": "GeothermalOre", "quantity": 69.92552,
            "quality": 0.99946797, "quantity_max": 70.0,
            "regen_rate": 0.015,
            "accessible": false, "gathered_by": null
          }
        ]
      }
    ],
    "visible_routes": [
      {"route_id": 0, "traversable": true, "terrain": "frozen"},
      {"route_id": 1, "traversable": true, "terrain": "frozen"},
      {"route_id": 2, "traversable": true, "terrain": "frozen"},
      {"route_id": 3, "traversable": true, "terrain": "frozen"},
      {"route_id": 4, "traversable": true, "terrain": "frozen"},
      {"route_id": 5, "traversable": true, "terrain": "frozen"},
      {"route_id": 6, "traversable": true, "terrain": "frozen"}
    ],
    "action": {
      "type": "deposit", "result": "success", "elapsed_ticks": 1,
      "tick_before": 215, "tick_after": 216
    }
  }
```


A payload opens with the world time and commanded scientist. The `agent` block records that scientist's zone, energy, position on the zone graph, inventory slots, and most recent action outcome. The `available_actions` block lists the actions currently possible and the resource node identifiers required by `gather`. Outpost integrity and each subsystem's buffer, buffer cap, health, operational status, and offline effect appear in `base`. The `team` block gives the other scientist's zone, energy, and current action. Under `world`, each zone contains its thaw level and resource nodes, including

their quantity, quality, and availability. Route traversal status appears under `visible_routes`, including changes caused by flooding. A turn that dispatches a command also includes its outcome under `action`; an observe turn omits this block because it dispatches no command.

The configuration omits constant mechanics from observations because the system prompt states them once. Empty sections are also omitted, which is why this payload contains no event, message, or fauna fields. If a payload exceeds the configured size, the runner removes optional sections in a fixed order, starting with events and ending with the map. The `_truncated` field records any removal. No observation in this episode was truncated.

Every observation gives absolute world time in `tick`. For a dispatched command, `action` also records `tick_before`, `tick_after`, and `elapsed_ticks`; the latched outcome records its resolution tick. The difference from the preceding observation therefore reveals the turn cost without an additional query. This calculation is most direct for an observe turn, which dispatches no command. The payload does not name the resulting interval. Although the simulator's text rendering states the ticks since the preceding command, the runner substitutes the JSON payload before delivery. No evaluated controller received that text field.

Context summarization is an exception to this one-turn calculation. When the conversation reaches its token budget, the runner requests a summary in a separate call that returns no observation. A completed summary call advances the simulation; a failed one is logged without advancing it. After a completed summary, the next observed difference includes elapsed time for both calls without marking their boundary.

Every payload also includes `sim_time`, which divides the tick by a runner setting rather than by the charge rate. In the example, this field implies a longer tick than the system prompt states. It also appears with the clock stopped, when the prompt states no rate. Under every evaluated setting, the runner advances the simulator rather than a wall clock. Thus `sim_time` is a display label, not a measure of task pace, and all reported quantities use ticks.

C.2 The prompt and the tools

The script `tools/baselines/llm_agent.py` connects an LLM controller to the simulator. Once per episode, it builds the tool schema (`build_tools`) and system message (`build_system_message`). It records both texts and their hashes in the trace header. The quotations below come from the retained Short loop episodes used in the preceding observation example.

The tools. With the clock stopped, the controller has nine tools. Under either running clock, it has seven: the same set without the two clock tools. The own-latency and assigned-latency settings have identical tool-schema hashes.

```
clock stopped, 9 tools:
  observe  advance_ticks  wait_for  move_to  gather
  process  deposit  repel  get_rules

own-latency clock and assigned-latency clock, 7 tools:
  observe  move_to  gather  process
  deposit  repel  get_rules
```

The two clock tools are available only with the clock stopped. They let the controller advance world time without commanding a scientist. Under either running clock, a turn advances the simulation by its measured or assigned delay, and these tools are unavailable. Their recorded schemas are:

```
[
 {
  "function": {
   "description": "Advance a paused non-blocking simulation by an
     explicit number of ticks, then observe one agent. Use this to
     progress multiple concurrent in-flight actions without replacing
     them.",
   "name": "advance_ticks",
```

```
    "parameters": {
     "properties": {
      "agent_id": {
       "description": "Agent to command (0-1)",
       "type": "integer"
      },
      "ticks": {
       "description": "Paused simulator ticks to advance (1-1000)",
       "maximum": 1000,
       "minimum": 1,
       "type": "integer"
      }
     },
     "required": [
      "ticks",
      "agent_id"
     ],
     "type": "object"
    }
   },
   "type": "function"
  },
  {
   "function": {
    "description": "Advance a paused non-blocking simulation until the
      selected agent's current action resolves, while every other agent
      also continues progressing.",
    "name": "wait_for",
    "parameters": {
     "properties": {
      "agent_id": {
       "description": "Agent to command (0-1)",
       "type": "integer"
      },
      "max_ticks": {
       "description": "Safety cap for paused ticks (default 1000)",
       "maximum": 1000,
       "minimum": 1,
       "type": "integer"
      }
     },
     "required": [
      "agent_id"
     ],
     "type": "object"
    }
   },
   "type": "function"
  }
]
```

Every other tool keeps the same name and parameters across settings. Its description changes only where it states the turn cost. The `observe` descriptions are:

```
clock stopped:
Observe an agent's current state. Submits no action and does not advance
the paused simulation or interrupt an in-flight action. Use this when
state is needed; avoid redundant polling.

own-latency clock:
Observe an agent's current state. Submits no action and does not
interrupt an in-flight action, but the model call's replayed latency
advances simulated time before the next turn. Use this when state is
needed; avoid redundant polling.
```

```
assigned-latency clock:
Observe an agent's current state. Submits no action and does not
interrupt an in-flight action, but the model call's replayed latency
advances simulated time before the next turn. Use this when state is
needed; avoid redundant polling.
```

One command per turn. Every setting accepts one tool call per turn. If a response contains more, the runner dispatches only the first and returns errors for the rest. A turn can therefore command at most one scientist under any clock setting. With the clock stopped, a dispatch advances the world by one tick; the controller can command the two scientists two ticks apart, then advance time with a clock tool. Under a running clock, each turn advances the simulation by its measured or assigned delay before the next command.

The system prompt. The system message opens with the initial prompt the environment builds (`format_initial_prompt` in `openreward/observation.py`), which carries the rules of the world, the map, the node directory and the first observation. The runner appends fixed guidance (`build_system_message`), a context-summarization notice, and the constant mechanics. After the first turn, user content consists only of a tool result and a one-line request for the next action. The complete message for Short loop with the clock stopped is shown below. It describes **corruption** and **fauna**, but both mechanics are disabled in every evaluated condition.

```
EXPERIMENT CONDITION: terra_LATSTRESS_paused
PROMPT VARIANT: default | CLOCK MODE: paused | DISPATCH MODE:
nonblocking

GAME SUMMARY:
You are controlling agents in Thaw, a survival benchmark on an alien ice
world. The simulation is paused while you think. Thinking time is free;
ticks advance only through dispatched actions or explicit clock tools.
Plan carefully and keep both agents working concurrently.

OBJECTIVE: Keep the outpost alive by maintaining its three subsystems
(thermal, power, biofilter). Each subsystem has a buffer (drains every
tick, refilled by depositing the matching processed good) and a health
value. A subsystem goes OFFLINE the moment its buffer hits 0 (its
offline effect applies even while health is still above 0 -- e.g. power
offline does not stop processing, it doubles the process duration); once
offline, its health decays each tick until you refill the buffer. The
outpost itself has integrity 100 which drains whenever one or more
subsystems are offline -- that is, whenever a subsystem buffer is empty
(the failure cascade), not only when health has reached 0. Win by
surviving 3500 ticks. Lose if integrity reaches 0. Per-tick rates are
scaled by tick_compression=3.0 in this config so the failure cascade is
reachable inside the shorter horizon.

RESOURCE CHAIN: Gather raw resources (GeothermalOre, CryoCrystal,
XenofloraSample) from nodes in zones. Each successful Gather
automatically uses one empty inventory slot and takes up to 5 raw units.
An agent has four slots and may Gather repeatedly before returning.
Process one raw slot per call at the outpost into goods (GeothermalOre
-> HeatCell at 0.8 yield; CryoCrystal -> FuelRod at 0.6; XenofloraSample
-> FilterCompound at 0.7). A full raw inventory can still be processed:
if no spare slot exists, the output replaces its input slot. Output
quantity is round(raw quantity * quality * yield), minimum 1; quality is
applied exactly once during processing and does not alter gather
quantity or deposit value. Deposit one processed slot per call; each
deposited good unit adds one point to its matching subsystem buffer,
capped at buffer_max. Processing takes 5 ticks (doubles to 10 if power
is offline). Gathering takes 3 ticks.

THAW: Zones start frozen and thaw over time. Each tick a zone's
thaw_level rises based on its distance to the configured thaw centers,
and the base rate accelerates linearly across the episode. thaw_level
```

```
rises from 0.0 toward 1.0. Some resources are gated by thaw level: a
node only becomes gatherable once its zone's thaw_level meets the
resource's requires_thaw_level. Some resources also degrade above an
upper thaw threshold. The NODE DIRECTORY (in the initial prompt) lists
thaw thresholds per resource so you can plan which lanes unlock when.
Per-zone thaw is shown live on each agent's observation as `thaw=N.NN`.

MOVEMENT: All movement is along routes between zones. Move accepts
either a target_zone (preferred -- the sim auto-pathfinds across the
graph one route at a time) OR a target_route + the destination zone of
that route. Same-zone Move = no-op. Once issued, the agent continues
moving every tick until the destination is reached -- you do NOT need to
reissue Move. A route edge is committed once entered: a new Move while
ON a route changes the final commanded destination, but the agent first
finishes the current edge and reroutes from its next endpoint; it cannot
reverse midway along an edge. Agents auto-idle between commands. While
ON a route, gather/deposit return FailedPrecondition -- the agent must
reach a zone first. Per-tick energy on a route = base_traversal_cost *
route_terrain_multiplier; flooded routes are impassable.

ACTIONS: Observe (no action; does not advance a paused simulation and
does not interrupt an in-flight action; lists every currently executable
gather target in the agent's zone with its resource_type, node_id, and
current quantity/quality -- no need to guess names or IDs), Move (graph
movement; auto-continues until destination), Gather (harvest from node,
in_zone only; takes 3 ticks), Process (raw -> goods at outpost; takes 5
ticks), Deposit (instant, refill buffers, in_zone only), Repel
(cooperative leviathan defense, needs 3 agents). Gather, Process, and
Repel block the agent for their listed tick count and continue
automatically until done; issuing any Move/Gather/Process/Repel command
at any tick replaces the in-flight one. Observe is the only command that
does not interrupt an in-flight action. Each agent has 4 inventory
slots.

ENERGY: Actions cost energy. Agents regenerate in the outpost zone
(throttled to 25% if thermal is offline). At 0 energy an agent is
exhausted and can only move at half speed.

SUBSYSTEMS: Thermal (refilled by HeatCell; offline throttles outpost
energy regen to 25%). Power (refilled by FuelRod; offline doubles
processing duration). Biofilter (refilled by FilterCompound; affects
contamination/exposure). A subsystem is offline whenever its buffer is
empty (or its health has reached 0); refilling the buffer before health
reaches 0 fully recovers it. Each offline subsystem adds 0.150/tick of
integrity drain (formula: 0.150 * n_offline^2, where n_offline counts
subsystems with an empty buffer or zero health), so two offline =
0.60/tick. That quadratic rate is a floor: cold damage (while thermal is
offline) and contamination (while biofilter is offline) add further
integrity drain on top.

CORRUPTION: Agents in high-biosphere zones accumulate exposure. Phases:
clean -> blending (actions may be deflected, messages unreliable) ->
assimilated (fully corrupted). Monitor teammate behavior for signs of
corruption.

FAUNA: Grazers (attracted by harvesting noise, pack attacks), Stalkers
(ambush stationary agents, hard to detect), Leviathans (zone-wide
threat, requires 3 agents to repel cooperatively).

EPISODE CONFIG:
  Mode: endurance | Clock: paused | Dispatch: nonblocking | Agents: 2 |
Zones: 7 | Max ticks: 3500 | Tick compression: 3.0x

MAP GEOMETRY:
  Outpost is in Zone 0.
```

```
  Action scale: move_speed=9.000 units/tick | gather=3 ticks | process=5
ticks (10 if power offline)
  Energy: max=100 | regen=0.875/tick in outpost (x0.25 if thermal
offline) | base_move_cost=0.14/tick | gather_cost=0.3 | process_cost=0.5
  Terrain multipliers (movement cost): frozen=1.0 | thaw_boundary=1.0 |
thawed=1.0 | overgrown=1.5 | flooded=impassable
  Subsystem drain: thermal=0.0240/tick | power=0.0394/tick |
biofilter=0.0110/tick
  Zones: Z0 | Z1 | Z2 | Z3 | Z4 | Z5 | Z6
  Routes: R0: Z0<->Z1 len=134.2 | R1: Z0<->Z2 len=80.0 | R2: Z1<->Z2
len=72.1 | R3: Z0<->Z3 len=94.9 | R4: Z3<->Z4 len=90.0 | R5: Z0<->Z5
len=90.0 | R6: Z5<->Z6 len=86.0
  Movement: speed=9.000 units/tick, cost=0.14 energy/tick (scaled by
route terrain multiplier; flooded routes are impassable).

NODE DIRECTORY (initial nodes per zone; thaw gates apply):
  Z1: GeothermalOre (always), XenofloraSample (always)
  Z2: GeothermalOre (always)
  Z3: CryoCrystal (always)
  Z4: CryoCrystal (always)
  Z5: XenofloraSample (thaw>=0.50), CryoCrystal (thaw>=0.50)
  Z6: GeothermalOre (thaw>=0.50)

INITIAL OBSERVATIONS:

== Tick 1 | Sim Time 0.5s ==

AGENT 0 (you commanded this agent):
  Zone 0 (frozen, thaw=0.00) | Energy 100/100
  Location: IN ZONE Z0
  Inventory: [0] empty | [1] empty | [2] empty | [3] empty
  Last Action: noop -> success
  Corruption: clean | Biosphere exposure: 0.00

TEAM STATUS:
  Agent 1: Zone 0 | Energy 100/100 | noop | clean

BASE: integrity=100 (ok) | Thermal: buf=20/30 (ok), health=100/100,
online | Power: buf=25/30 (ok), health=100/100, online | Biofilter:
buf=20/30 (ok), health=100/100, online


You control all 2 agents. Each tool call commands one agent. The
simulation remains paused while you decide on the next call. Long
actions use nonblocking dispatch.

EARLY GAME: Scout zones, identify accessible resources, then run gather
-> process -> deposit loops for the subsystem with the shortest runway
(buffer divided by its drain rate).

STRATEGY HINTS:
- Use observe to inspect an agent; it submits no action, does not
advance the paused simulation, and does not interrupt the current
action. Survey your team before deciding what to do.
- When in a zone with unknown nodes, call observe first -- it lists
every gatherable node with its resource_type, node_id, and current
quantity/quality. Don't guess names or IDs.
- Watch zone thaw_level. Some resources are thaw-gated (see the NODE
DIRECTORY in the initial prompt for thresholds); attempting to gather a
node in a zone whose thaw_level is below the gate just fails. Move
agents to higher-thaw zones first when waiting for early-game lanes to
unlock.
- Prioritize the subsystem with the shortest runway, not the smallest
raw buffer: compare buffer / configured drain_rate, then subtract the
time needed to gather, return, process, and deposit. Drain rises up to
```

1.5x as thaw progresses. Configured-runway references: thermal=buffer/0.024, power=buffer/0.03936, biofilter=buffer/0.01104.
- Subsystem failure is graceful: when a buffer empties, the subsystem stops being operational and health begins to decay; refilling the buffer before health hits 0 recovers it. Power buffer empty does NOT instantly fail the subsystem.
- Integrity drains 0 when all subsystems are operational and at least 0.150 * n_offline^2 per tick otherwise (cold damage and contamination add more); one offline subsystem is survivable for many ticks, two offline cascades fast.
- Movement is graph-constrained: each agent is either IN a zone or ON a route. While ON a route, gather/deposit will fail. Once Move is dispatched it auto-continues; do not reissue it. The current edge is committed: retargeting changes commanded_target_zone, but the agent reaches heading_to_zone before rerouting. Any non-Move action replaces movement.
- Assign agents to different resource types for parallelism.
- Each gather fills one empty slot with up to 5 raw units. Normally gather repeatedly to fill the four slots before returning because travel is expensive; return early only when a subsystem's runway is shorter than the extra gather plus return/process/deposit time.
- Process and deposit one slot per call. Yields are ore->HeatCell 0.8, crystal->FuelRod 0.6, xenoflora->FilterCompound 0.7, multiplied by raw quantity and quality then rounded (minimum 1). Each deposited good adds one buffer point; quality is applied only during processing.
- Keep agents near the outpost zone for energy regeneration. Thermal offline throttles regen to 25%, so prioritize thermal recovery if energy is bottlenecking the team.
- Power offline doubles processing duration; if power is offline, expect process tools to take ~10 ticks instead of 5.
- Watch for fauna threats and low energy.
- Call one tool at a time. Each tool call commands one agent.
- Focus on observe, move_to, gather, process, deposit, and get_rules.
- Do not invent messaging or coordination payloads. Communication tools are not part of this evaluation.

CONTEXT COMPACTION: your conversation history is compacted when it grows past a token budget. You will be asked to write a summary for your future self, and the earlier history is then replaced by that summary plus a state snapshot (each agent's zone or route, energy, inventory, last action and commanded target; base buffers; node quantities and zones; recent commands). Keep your plan and each agent's role explicit in your reasoning so the summary can carry them.

MECHANICS (constant for this episode, JSON):
{"base_gather_cost": 0.33, "base_move_cost": 0.145, "base_process_cost": 0.48, "deposit_buffer_units_per_item": 1.0, "energy_regen_rate": 0.875, "gather_duration_ticks": 3, "gather_units_per_success_max": 5, "integrity_drain_per_offline_unit": 0.15, "inventory_slots": 4, "max_energy": 100.0, "max_ticks": 3500, "movement_speed_units_per_tick": 9.0, "process_duration_ticks": 5, "process_duration_ticks_power_offline": 10, "processing_full_inventory_supported": true, "processing_recipes": {"CryoCrystal": {"output": "FuelRod", "yield_factor": 0.6}, "GeothermalOre": {"output": "HeatCell", "yield_factor": 0.8}, "XenofloraSample": {"output": "FilterCompound", "yield_factor": 0.7}}, "processing_yield_formula": "round(raw_quantity * quality * yield_factor), minimum 1", "resource_quality_effect": "applied exactly once during processing; does not change gather quantity or deposit value", "subsystem_base_drain_per_tick": {"biofilter": 0.01104, "power": 0.03936, "thermal": 0.024}, "subsystem_drain_multiplier_range_over_episode": [1.0, 1.5], "thermal_offline_energy_regen_multiplier": 0.25, "tick_compression": 3.0}

Under the own-latency clock, only the following lines change. The assigned-latency prompt changes only the charge rate. It does not tell the controller whether delays come from a recorded pool or from its own turns.

```
stopped clock:
EXPERIMENT CONDITION: terra_LATSTRESS_paused
own-latency clock at charge rate 8:
EXPERIMENT CONDITION: terra_LATSTRESS_replay_f3

stopped clock:
PROMPT VARIANT: default | CLOCK MODE: paused | DISPATCH MODE:
nonblocking
own-latency clock at charge rate 8:
PROMPT VARIANT: default | CLOCK MODE: latency_replay | DISPATCH MODE:
nonblocking

stopped clock:
You are controlling agents in Thaw, a survival benchmark on an alien ice
world. The simulation is paused while you think. Thinking time is free;
ticks advance only through dispatched actions or explicit clock tools.
Plan carefully and keep both agents working concurrently.
own-latency clock at charge rate 8:
You are controlling agents in Thaw, a survival benchmark on an alien ice
world. Inference latency is replayed at 8 base ticks/second (0.125s per
tick). Balance deliberation against the simulated time it consumes.

stopped clock:
ACTIONS: Observe (no action; does not advance a paused simulation and
does not interrupt an in-flight action; lists every currently executable
gather target in the agent's zone with its resource_type, node_id, and
current quantity/quality -- no need to guess names or IDs), Move (graph
movement; auto-continues until destination), Gather (harvest from node,
in_zone only; takes 3 ticks), Process (raw -> goods at outpost; takes 5
ticks), Deposit (instant, refill buffers, in_zone only), Repel
(cooperative leviathan defense, needs 3 agents). Gather, Process, and
Repel block the agent for their listed tick count and continue
automatically until done; issuing any Move/Gather/Process/Repel command
at any tick replaces the in-flight one. Observe is the only command that
does not interrupt an in-flight action. Each agent has 4 inventory
slots.
own-latency clock at charge rate 8:
ACTIONS: Observe (submits no action and does not interrupt an in-flight
action, but the model call's replayed latency advances simulated time;
lists every currently executable gather target in the agent's zone with
its resource_type, node_id, and current quantity/quality -- no need to
guess names or IDs), Move (graph movement; auto-continues until
destination), Gather (harvest from node, in_zone only; takes 3 ticks),
Process (raw -> goods at outpost; takes 5 ticks), Deposit (instant,
refill buffers, in_zone only), Repel (cooperative leviathan defense,
needs 3 agents). Gather, Process, and Repel block the agent for their
listed tick count and continue automatically until done; issuing any
Move/Gather/Process/Repel command at any tick replaces the in-flight
one. Observe is the only command that does not interrupt an in-flight
action. Each agent has 4 inventory slots.

stopped clock:
  Mode: endurance | Clock: paused | Dispatch: nonblocking | Agents: 2 |
Zones: 7 | Max ticks: 3500 | Tick compression: 3.0x
own-latency clock at charge rate 8:
  Mode: endurance | Clock: latency_replay | Dispatch: nonblocking |
Agents: 2 | Zones: 7 | Max ticks: 3500 | Tick rate: 8.0 Hz | Tick
compression: 3.0x

stopped clock:
```

```
You control all 2 agents. Each tool call commands one agent. The
simulation remains paused while you decide on the next call. Long
actions use nonblocking dispatch.
own-latency clock at charge rate 8:
You control all 2 agents. Each tool call commands one agent. Replayed
inference time advances the simulation before the next call. Long
actions use nonblocking dispatch.

stopped clock:
- Use observe to inspect an agent; it submits no action, does not
advance the paused simulation, and does not interrupt the current
action. Survey your team before deciding what to do.
own-latency clock at charge rate 8:
- Observe submits no action and does not interrupt current work, but the
model call's replayed latency advances simulated time; use it when state
is needed and avoid redundant polling. Survey your team before deciding
what to do.
```

C.3 From a checkout to one episode

Three commands produce a complete episode from a clean checkout. They build the simulator, install the runner's Python dependencies, and run the reactive baseline on Short loop to completion.

```
cargo build --release --bin thaw-sim

python3 -m pip install -r requirements-dev.txt

python3 tools/baselines/heuristic_agent.py \
    --config latstress_1x --synchronous --no-realtime \
    --seed 1 --max-steps 40000
```

A runner script looks for the simulator at `target/release/thaw-sim`, while the environment variable `THAW_SIM_BINARY` specifies another location. The third command omits a requested-gap gate, so the baseline commands as often as possible and reaches the horizon on the seed shown. A reference-curve run adds the requested gap, gap jitter, and gate scope:

```
python3 tools/baselines/heuristic_agent.py \
    --config latstress_1x --synchronous --no-realtime \
    --seed 1 --max-steps 40000 \
    --action-interval-ticks 8 --action-interval-jitter-cv 0.5 \
    --action-interval-scope global
```

For a condition with a solvability witness, the planned schedule is replayed at requested gap one. The replay script reads the corresponding schedule file and reports the overrun count, which must be zero:

```
python3 tools/baselines/dp_replay_agent.py \
    --config latstress_1x --synchronous --no-realtime \
    --seed 1 --max-steps 40000 \
    --schedule episodes/schedules/schedule_latstress_1x_exhaustive.json \
    --overrun-tolerance-ticks 0 \
    --action-interval-ticks 1 --action-interval-jitter-cv 0.5 \
    --divergence-policy gate
```

The model path requires provider credentials. For an offline check, the same runner can use a scripted client in place of the provider. This exercises the tool schema, prompt builder, and dispatch path:

```
python3 tools/baselines/llm_agent.py \
    --config latstress_1x --provider fake --model fake-model \
    --seed 1 --synchronous --no-realtime \
    --sync-tool-mode nonblocking --max-steps 200
```

The command stops at its action cap rather than at an outcome. The client cycles through the scientists without implementing a task policy, so this checks the execution path rather than controller performance. It also prints the tool-schema hash. For Short loop with the clock stopped, that hash matches every available trace header for the same condition and setting.

C.4 NAMES AND CODE IDENTIFIERS

A code identifier is the name a paper term carries in the runner, episode files, and data files. The tables below map these names to the corresponding paper terms.

Table 9 maps the conditions. Its configuration column gives the stem of a file under `episodes/`, whose full name ends in `.toml`. The final column gives the condition fragment within a run group identifier. This fragment links a ledger row to its condition.

Table 9: The seven conditions, their codes, the episode configuration each one loads, and the fragment each carries inside a run group identifier.

Condition	Code	Configuration	In an identifier
Base	BASE	`level0_1x`	`_L0_`
Tight economy	ECON-1	`econ_m133_b50_1x`	`_E133B50_`
Tight economy with more supply	ECON-2	`econ_m160_s130_b50_1x`	`_E160S130B50_`
Rising drain	RAMP	`thaw_econ133_1x`	`_THAW_`
Flooded route	FLOOD	`flood_r3bypass_1x`	`_FLOODR3_`
Short loop	SHORT	`latstress_1x`	`_LATSTRESS_`
Short loop, tight caps	SHORT-B	`latshort_1x`	`_LATSHORT_`

Table 10 maps the clock settings, the scripted controllers, and the paper term for a commanded scientist. Each clock setting appears as runner flags and as a label in trace headers and run group identifiers. The index in an own-latency label records only its batch position. The ledger's charge-rate column records the applied rate.

Table 10: Clock settings, scripted controllers and the commanded unit, against the flags and labels the code uses for them.

Term	Where it appears	Identifier
stopped clock	runner flags	`-synchronous -no-realtime -sync-tool-mode nonblocking`
stopped clock	trace, identifier	`clock_mode paused` `cadence_arm paused`
own-latency clock	runner flag	`-latency-replay-rate`
own-latency clock	trace, identifier	`clock_mode latency_replay` `cadence_arm replay_f` plus an index
assigned-latency clock	runner flags	`-latency-replay-rate` `-injected-latency-file`
assigned-latency clock	trace, identifier	`clock_mode injected_latency` `cadence_arm injected`
reactive baseline	script, data files	`tools/baselines/heuristic_agent.py` `heuristic`
patient baseline	data files	`heuristic_cw`
planned schedule	script, data files	`tools/baselines/dp_replay_agent.py` `dp_replay`
controller	runner script	`tools/baselines/llm_agent.py`
scientist	wire protocol, tool parameters, observation	`agent_id`, `agent.id`, `team` entries keyed by `agent_id`

Table 11 maps the terms of the boxed glossary and the table headings to their data columns. Most files listed there are under `paper/data/ledger/`; `episode_features.csv` is the per-episode feature table used for the behavior figures.

Table 11: Terms against the columns and flags that carry them.

Term	File or flag	Column or name
command gap	`cells.csv`, `scripted.csv`	`command_gap_median`
command gap, model side	`episode_features.csv`	`realized_gap_median_trace`
command gap, scripted side	`scripted.csv`	`gap_axis`, `merged_median_dispatch_log`
per-scientist command gap	`scripted.csv`, `episode_features.csv`	`command_gap_median_per_scientist`, `per_agent_gap_median_trace`
requested gap	`scripted.csv`, runner flag	`requested_gap`, `-action-interval-ticks`
gap jitter	`scripted.csv`, runner flag	`jitter_cv`, `-action-interval-jitter-cv`
turn cost	`cells.csv`, `episode_features.csv`	`turn_cost_median`, `latency_advanced_p50_trace`
charge rate	`cells.csv`, runner flag	`charge_rate_hz`, `-latency-replay-rate`
scripted tolerance	`tolerances.csv`	`tolerance_command_gap`
survival fraction	`cells.csv`	`mean_survival`
run group	`cells.csv`	`run_group_id`
clock setting	`cells.csv`	`clock_setting`
identical observed survival	`cells.csv`	`deterministic`

Two other columns must not be interpreted as the command gap. `realized_mean_gap` is a mean over intervals planned by the dispatch gate, rather than a median over dispatched commands. It is larger than the command gap on long-loop conditions. The per-tick metric log's `realized_gap_median` counts the onsets of actions that occupy a scientist and omits zero-duration deposits. Neither column is plotted, quoted, or used to compute a scripted tolerance.