Interruptible and Time-Aware Cognitive Loops for LLM Agents in Dynamic Environments

GPT5

November 27, 2025

Abstract

Large Language Models are increasingly deployed as autonomous agents that must perceive, reason, and act within environments that evolve during inference. Standard turn-based pipelines treat reasoning as a blocking procedure, which encodes a frozen-world assumption and leads to brittle behavior under mid-episode updates. This proposal develops a principled framework for interruptible, intra-turn metacognition. We formalize the generation process as a stochastic dynamical system under exogenous control and introduce an Interruptible Cognitive Loop (ICL) that operates on semantic Cognitive Blocks rather than raw tokens. A scheduler observes time budgets and external events, estimates hazard, and decides when to continue, soft-pause, abort, or summarize. We provide Bayesian resumption methods to maintain semantic coherence after interrupts, a block-wise streaming runtime to create safe preemption points, and an evaluation protocol that perturbs static benchmarks with temporal and interactive dynamics. The research yields theory, algorithms, and an open-source runtime kernel, together with new benchmarks for robustness, responsiveness, and reasoning quality under interruption.

Keywords: LLM agents, interruptible reasoning, real-time AI, metacognition, hazard modeling, scheduling, Bayesian resumption

1 Introduction and Motivation

1.1 From Chats to Continual Agency

Contemporary deployments of LLM agents in software engineering, scientific discovery, and embodied control require intra-episode perception, planning, and action across variable hori-

zons. These deployments invalidate the assumption that the world and the specification remain fixed during generation, which is still implicit in most agent stacks.

1.2 Failures of Frozen-World Pipelines

Turn-based agents typically enter a monolithic reasoning phase that cannot be preempted. This yields temporal incompetence (no graceful tradeoff between depth and latency), information obsolescence (premises drift during long generations), and interaction rigidity (users cannot correct mid-thought).

1.3 Scientific Challenge and Thesis

Empirical observations show that mid-generation context changes induce cognitive collapse in current models, including hallucination and inconsistent self-reference. We propose that reasoning should be treated as an interruptible stochastic process with explicit control, where semantic safety is preserved across preemption and resumption. The central thesis is that a time-aware, hazard-sensitive scheduler over semantic Cognitive Blocks can maintain coherence and performance under dynamic conditions.

Research Question: How can we transform monolithic Chain-of-Thought into a fluid, interruptible, and time-aware cognitive loop that preserves semantic invariants under aggressive scheduling constraints?

2 Background and Related Work

Event-driven architectures decouple tool latency from generation, and AIOS-like systems manage KV cache and memory at the systems layer, but they largely treat the model as an opaque process and do not provide a cognitive control loop within inference. Time-aware prompting and reflective memory streams improve inter-turn reasoning but do not provide intra-turn perception of time passage, budgeted depth control, or resumable state serialization at preemption points. Deliberation structures such as Trees of Thought improve problem solving while assuming stability within a single episode. Our framework nests such structures inside an interruptible controller.

3 Problem Formulation

3.1 Cognitive State

Let the cognitive state at continuous time t be

$$\mathcal{S}_t = \langle C_t, \mathcal{M}_t, \tau_{\text{rem}}(t), \mathcal{G}_t \rangle, \tag{1}$$

where C_t is the partial Chain-of-Thought, $\mathcal{M}_t$ is a structured working memory, $\tau_{\text{rem}}(t)$ is the remaining time budget, and $\mathcal{G}_t$ is the goal stack.

3.2 Cognitive Blocks as Atomic Units

Autoregressive tokens are too fine-grained for safe preemption. We define Cognitive Blocks B_k as semantically complete micro-steps (for example one completed sentence, one proof step, or one atomic code edit). The LLM parameterized by θ induces

$$B_{k+1} \sim P_\theta(B \,|\, \mathcal{S}_k, \mathcal{I}_{\text{ext}}) \,, \tag{2}$$

where $\mathcal{I}_{\text{ext}}$ are real-time exogenous inputs.

3.3 Interruptible Control

At block boundaries a scheduler selects an action

$$a_k \in \{\texttt{CONTINUE}, \texttt{SOFT\_PAUSE}, \texttt{HARD\_ABORT}, \texttt{SUMMARIZE}\} \tag{3}$$

to optimize discounted return

$$J(\pi) = \mathbb{E}\left[\sum_{k=0}^{H} \gamma^k \Big(\alpha \, \text{Utility}(\mathcal{S}_k) - \beta \, \text{Cost}_{\text{delay}}(\mathcal{E}_k) - \lambda \, \text{Cost}_{\text{switch}}(k) \Big) \right]. \tag{4}$$

3.4 Event Hazard Modeling

Each pending event e has hazard

$$\lambda_e(t) = \sigma\Big(w_t \, (T_{\text{deadline}} - t)^{-1} + w_c \, \mathbb{I}(e \perp C_t) + w_p \, \text{Priority}(e) \Big), \tag{5}$$

which captures temporal pressure, epistemic conflict with current premises, and task priority.

3.5 Bayesian Resumption

Upon interrupt, we summarize the partial trajectory and resume with a synthesized context

$$C'_{\text{new}} = \arg\min_{C} D_{\text{KL}}(P(Y \mid C, e_{\text{new}}) \,\|\, P(Y \mid \mathcal{S}_{\text{ideal}})) \,. \tag{6}$$

A specialized summarizer converts internal state into a compact state object with invariants such as facts, derived constraints, and frontier.

4 Methodology: Algorithms and Architecture

4.1 Block-wise Streaming and Safe Preemption

We design a *Cognitive Thread Manager* that enforces block boundaries through a combination of: stop sequences aligned with discourse markers (for example newlines and `[STEP]`), constrained decoding with logit processors that increase probability mass at boundary tokens under budget stress, and incremental syntax checks for code tasks to ensure no preemption within lexical entities. Design goals include never splitting numeric literals or variable names and never interrupting inside unmatched parentheses or brackets.

4.2 Event and Hazard Loop

An asynchronous loop ingests user edits, tool returns, sensor data, and clock ticks. It maintains hazard scores $\lambda_e(t)$ and raises interrupts when

$$\max_{e} \lambda_e(t) \geq \kappa(\mathcal{S}_k), \tag{7}$$

where κ is a learned inertia threshold dependent on task criticality and current plan confidence.

4.3 Scheduler Policies

We study three policy families:

1. Myopic thresholding: choose `SOFT_PAUSE` when hazard exceeds inertia; otherwise `CONTINUE`.
2. Model-predictive scheduling: simulate L candidate futures with coarse rollouts using a small model and choose actions that maximize predicted utility minus delay cost.

3. Learned meta-controller: train a light policy network on offline traces with reinforcement learning to map summary features to actions.

4.4 State Serialization and Summarization

We implement a *Hippocampus* module with four steps: Freeze (capture KV cache and block-wise text), Compress (use a small model to extract a structured tuple of goals, derived facts, assumptions, unresolved items, and draft artifacts), Validate (deterministic schema check and invariant enforcement), and Persist (store with priority and expiry).

4.5 Resumption Synthesis

A *Resumption Engine* constructs a re-orientation prompt with the state object, the description of new events and conflicts, and the explicit continuation target such as "resume at Step 3 given requirement change R". We include a short consistency probe that verifies updated premises before full decoding.

4.6 Time-Aware Depth Control

A budget-aware decoding controller switches between deep Chain-of-Thought (long blocks with beam size $b > 1$), shallow heuristics (short blocks with greedy decoding), and direct answer mode as $\tau_{\text{rem}} \to 0$. Switching is smooth via a continuous function of τ_{rem} and estimated task hardness.

4.7 Implementation Plan

We will implement a runtime kernel in Python with vLLM or TensorRT-LLM backends. Asynchronous orchestration uses structured concurrency. State objects are typed via JSON Schema with validation. Language-agnostic hooks for code tasks use tree-sitter to guarantee lexical safety at boundaries.

5 Experimental Design

5.1 Dynamic Benchmarks

We construct three perturbation suites over standard datasets.

Ticking Clock (math reasoning).
GSM8K and MATH with sampled deadlines. Budget stress signals appear as deadlines approach. Outcome: accuracy versus latency and graceful degradation.

Moving Target (code generation).
HumanEval and MBPP with mid-episode requirement changes. Measure ability to pivot without incoherence.

Interrupted Planner (embodied tasks).
WebShop or ALFWorld with exogenous state changes such as item availability flips. Measure plan regret and recovery time.

5.2 Metrics

Cognitive Resilience Ratio (performance under perturbation normalized by static performance), Interrupt Latency (wall-clock from hazard threshold crossing to generation halt), Panic Rate (frequency of loops or format breakage after interrupts), Information Integration Score (human rating for incorporating new information), Plan Regret (difference in cumulative reward between a counterfactual uninterrupted plan and the executed plan), and Edit Waste (for code, fraction of tokens later discarded after requirement changes).

5.3 Baselines and Controls

Baselines include turn-based agents without interrupts, asynchronous tool-use without cognitive interrupts, tree-search planners without hazard-aware control, and prompt-only time awareness without intra-turn updates.

5.4 Statistical Protocol

We use five random seeds with paired bootstrap confidence intervals. We pre-register ablations with Holm–Bonferroni correction and run power analysis for sample sizes in human evaluation.

6 Ablations and Analyses

We run ablations: remove Bayesian resumption (append raw text only), preempt at token level instead of block level, remove epistemic conflict from hazard, compare myopic versus model-predictive versus learned schedulers, boundary detectors (delimiter-only versus

syntax-aware), and budget controller schedules. We also measure cache size, context length, and summarizer model scale versus quality and latency.

7 Theoretical Results (work plan)

We target three results: (i) a lower bound on mutual information between resumed output and ideal uninterrupted output under block-level preemption with bounded switching cost, given summarizer fidelity; (ii) bounded interrupt frequency under Lipschitz hazard and inertia functions with a minimum block duration and deadline margin; (iii) a regret bound for myopic scheduling compared with an oracle controller when delay cost processes are submodular.

8 Applications

Applications include autonomous software engineering with live requirement changes during code synthesis, scientific assistants that continuously ingest literature and update hypotheses while drafting methods and analyses, and robotics with human-in-the-loop control where plans must update under strict timing constraints and sensor updates.

9 Safety, Ethics, and Governance

We will ensure transparency through inspectable state objects and resumption prompts, protect privacy with redaction and retention windows, provide human override for all interrupts, limit autonomous actions under low budgets or high hazard unless approvals are present, and disclose evaluation protocols for replicability.

10 Limitations and Risks

Summarizer errors may bias state objects; we mitigate with deterministic validators and consistency probes. Hazard estimation quality determines responsiveness; we will collect supervised labels for calibration. Block boundaries require task-specific tuning; syntax-aware detectors generalize across domains. Extreme interrupt frequency may still induce collapse; thrashing guards are added at the scheduler.

11 Project Plan

Milestones: M1 (months 1–2) kernel prototype with block-wise streaming and manual interrupts and initial dynamic benchmarks. M2 (months 3–4) hazard loop and myopic scheduler with summarizer v1 and early ablations. M3 (months 5–6) model-predictive and learned schedulers with syntax-aware boundaries and full benchmark release. M4 (months 7–8) theoretical results, user studies, and integration examples. M5 (months 9–10) paper, open-source release, and reproducibility package. Resources: four GPUs for training small controllers and summarizers, one multi-GPU node for evaluation, and human raters for integration scores.

12 Open-Source and Data Release

We will release the ICL runtime kernel under a permissive license, dynamic perturbation benchmarks that wrap GSM8K, MATH, HumanEval, MBPP, and a planning suite with event generators and deadline samplers, anonymized logs and traces for research on meta-control, and safety checklists for audit of interrupt policies.

13 Expected Contributions

We expect contributions in theory (interruptible reasoning as a controlled stochastic process with hazard-aware scheduling), algorithms (block-wise streaming, Bayesian resumption, and budgeted depth control with learned or model-predictive schedulers), systems (an open runtime with safe preemption and resumable prompts), benchmarks (dynamic perturbation suites and metrics for robustness), and practice (integration recipes and invariants).

14 References

1. Ginart, A. A., Narang, S., Zhou, S., et al. (2024). *Asynchronous Tool Usage for Real-Time Agents*. arXiv.
2. Mei, K., Zhu, X., Xu, W., et al. (2024). *AIOS: LLM Agent Operating System*. arXiv.
3. Wu, T. H., Miroyan, M., Chan, D. M., Darrell, T., Norouzi, N., & Gonzalez, J. E. (2025). *Are Large Reasoning Models Interruptible?* arXiv.
4. Cao, S., & Wang, L. (2022). *Time-Aware Prompting for Text Generation*. Findings of EMNLP.

5. Eldridge, L. J., & Abdelshehid, E. S. (2025). *Time-Injected Large Language Models*. TechRxiv.

6. Babenko, A., et al. (2025). *It's High Time: A Survey of Time-Aware LMs*. arXiv.

7. Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). *Generative Agents*. UIST.

8. Shinn, N., Labash, B., & Leahy, C. (2023). *Reflexion: Language Agents with Verbal Reinforcement Learning*. arXiv.

9. Yao, S., Yu, D., Zhao, J., et al. (2023). *Tree of Thoughts*. arXiv.

10. Qian, C., Zhang, R., Li, Y., et al. (2024). *Language Agent Tree Search*. arXiv.

11. Wu, Y., Zeng, W., Zhang, Y., et al. (2023). *AutoGen: Enabling Next-Generation LLM Applications via Multi-Agent Conversation*. arXiv.

12. Xie, Y., Xiang, T., et al. (2024). *AsyncTool: Asynchronous Tool Usage for Real-Time LLM Agents*. OpenReview.

13. Salesforce Research. (2025). *2024 in AI Research: Building Blocks for the Agentic Era Ahead*. Blog.

14. Patel, A. (2024). *Awesome AI Agents: A Curated List of 500+ AI Agent Projects and Use Cases*. GitHub.