

REAL-TIME REASONING AGENTS IN EVOLVING ENVIRONMENTS

000
001
002
003
004
005
006
007
008
009
010
011
012
013
014
015
016
017
018
019
020
021
022
023
024
025
026
027
028
029
030
031
032
033
034
035
036
037
038
039
040
041
042
043
044
045
046
047
048
049
050
051
052
053
Anonymous authors
Paper under double-blind review

ABSTRACT

Agents in the real world must make not only logical but also *timely* judgments. This requires continuous awareness of the dynamic environment: hazards emerge, opportunities arise, and other agents act, while the agent’s reasoning is still unfolding. Despite advances in language model reasoning, existing approaches fail to account for this dynamic nature. We introduce *real-time reasoning* as a new problem formulation for agents in evolving environments and build **Real-Time Reasoning Gym** to demonstrate it. We study two paradigms for deploying language models in agents: (1) reactive agents, which employ language models with *bounded reasoning computation for rapid responses*, and (2) planning agents, which allow *extended reasoning computation for complex problems*. Our experiments show that even state-of-the-art models struggle with making logical and timely judgments in either paradigm. To address this limitation, we propose **AgileThinker**, which simultaneously engages *both reasoning paradigms*. AgileThinker consistently outperforms agents engaging only one reasoning paradigm as the task difficulty and time pressure rise, effectively balancing reasoning depth and response latency. Our work establishes real-time reasoning as a critical testbed for developing practical agents and provides a foundation for research in temporally constrained AI systems, highlighting a path toward real-time capable agents.

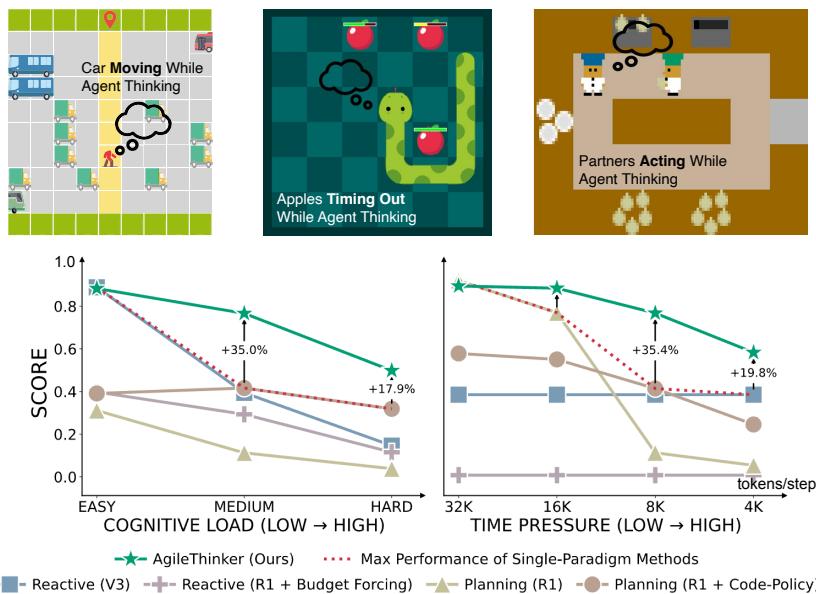


Figure 1: Upper: Three real-time games, *Freeway*, *Snake*, and *Overcooked*. Lower: Under cognitive load and time pressure, AgileThinker, engaging both reactive and planning reasoning, consistently outperforms agents that engage either of them. Scores are averaged across different games.

054

1 INTRODUCTION

055
 056 Remember your first highway drive? White-knuckling the wheel, fixated on the car ahead. Then sud-
 057 denly your exit appears and you’re three lanes over. This captures how our brains work under pres-
 058 sure: reacting intuitively keeps us safe moment-to-moment, but deliberate planning gets us where
 059 we need to go. This example highlights a fundamental challenge: real-time reasoning (Stanovich
 060 & West, 2000; Kahneman, 2011; Evans & Stanovich, 2013). Under time pressures, agents must si-
 061 multaneously support timely reactions and cognitively demanding long-term planning. Remarkably,
 062 humans excel at this balancing act: shifting seamlessly between instinct and analysis as situations
 063 demand. Expert drivers eventually handle both tasks effortlessly.

064 However, current Large Language Model (LLM)-based agents fail to live up to this challenge. Most
 065 existing work assume that the environments only change when the agents issue an action, ignoring
 066 the *dynamic* nature of the world, which evolves in parallel to the agent’s computation. As a result,
 067 despite great effort in improving agent planning with LLM reasoning, including Yao et al. (2022);
 068 Gou et al. (2023); Putta et al. (2024); Ferrag et al. (2025), under this assumption, how to evaluate
 069 and improve the capability to make timely decisions is still an open question.

070 To bridge this gap, we introduce **Real-Time Reasoning Gym**, the first environment for language
 071 agents to reason in dynamic environments (§2). Our gym consists of three real-time games: *Freeway*,
 072 inspired by the Atari game under the same name, *Snake*, an adaptation of a popular game, and
 073 *Overcooked*, a two-player version of the collaborative video game. In each game, the state updates at
 074 a fixed rate regardless of whether the agent finishes its reasoning, and if no action is produced by the
 075 agent, a default action is used, simulating reasoning and acting in a real-time world. They challenge
 076 agents with different aspects of a dynamic environment: *Freeway* features dynamic hazards with
 077 moving cars, *Snake* involves dynamic opportunities as food appears and disappears, and *Overcooked*
 078 requires coordination with dynamic partners who act on their own. Real-Time Reasoning Gym is
 079 useful for studying different agent designs for real-time tasks.

080 To compare different design choices of real-time reasoning agents, we study two paradigms: reactive
 081 agents and planning agents (§3). Reactive agents ensure responsiveness by limiting computation,
 082 while planning agents are allowed to perform more extensive thinking. However, neither of them
 083 is perfect: planning agents cannot easily react to changes in the environment, and reactive agents
 084 fail to make strategic decisions. We propose **AgileThinker** (§3), a simple yet effective method
 085 that combines the strengths of *both paradigms*. Unlike agents with one paradigm that must choose
 086 between speed and accuracy, AgileThinker runs two LLMs in two parallel threads: a *planning thread*
 087 performs extended reasoning over frozen game states, and a *reactive thread* outputs timely decisions
 088 within environmental update time. Specifically, the reactive thread can reference partial reasoning
 089 traces from the ongoing planning process, enabling informed real-time decisions without waiting for
 090 complete analysis. This also differs from prior dual-system methods (Zhang et al., 2025; Liu et al.,
 091 2024; Christakopoulou et al., 2024), where either two systems operate independently, or one must
 092 wait for another to complete before accessing its outputs.

093 In this paper, we study the following research questions:

094 **RQ1** How do environment factors affect performance of agents in Real-Time Reasoning Gym?

095 **RQ2** How to balance reaction and planning resources in AgileThinker?

096 **RQ3** How well do the results we get with simulation in Real-Time Reasoning Gym match real-
 097 world walltime experiments?

098 To study these questions, we manipulate the cognitive load and time pressure of the games, facilitating
 099 systematic evaluation across both dimensions. We evaluate different design choices for reactive
 100 and planning agents, including budget forcing (Muennighoff et al., 2025) for reactive agents, code-
 101 as-a-policy (Liang et al., 2022) for planning agents. For fair comparison across agent designs, we
 102 use one model family, focusing on DeepSeek V3 and R1 because they are open-source and provide
 103 transparent reasoning trajectories required by AgileThinker. To assess generality, we also exper-
 104 iment with proprietary models (App. § C.3) and observe similar performance trend. Our results
 105 demonstrate that AgileThinker consistently outperforms single-paradigm methods by effectively
 106 balancing reactive and planning processes. This advantage is confirmed to translate to real-world
 107 scenarios through wall-clock time experiments. Ultimately, our work establishes a foundation for

108 developing language agents capable of complex reasoning under time constraints, bringing AI sys-
 109 tems closer to human-level performance in dynamic environments.
 110

111 **2 REAL-TIME REASONING GYM**
 112

113 To evaluate the real-time reasoning capabilities of agents, we need an environment that is (1) dy-
 114 namic: the environment state continuously updates without waiting for the agent’s decision; (2)
 115 cognitively challenging: the tasks should be challenging enough so that logical reasoning is needed;
 116 (3) reproducible: simulated environments to avoid non-negligible noise from hardware factors.
 117

118 To achieve these three desiderata, we consider a new formulation of decision-making problem. Un-
 119 like conventional turn-based environments, where the environment steps only after the agent finishes
 120 thinking and produces an action (Figure 2 left), in Real-Time Reasoning Gym, the environment steps
 121 forward at a fixed rate, even when the agent has not finished thinking. If no action is produced in
 122 time, a default action is applied (Figure 2 right). This simulates the real-world situation where the
 123 environment does not delay or accelerate according to the agent’s computation time.
 124

125 **Games** In order to control the
 126 dynamics of the environment for
 127 evaluating real-time reasoning,
 128 we use real-time games in our
 129 gym. We created three games to
 130 capture different challenges that
 131 a dynamic environment brings:
 132 maintaining safety when haz-
 133 ards happen, seizing transient
 134 opportunities, and coordinating
 135 with partners (Tab. 1).
 136

137 In Freeway, the agent traverses multiple lanes of bidirectional traffic, which requires constant moni-
 138 toring for oncoming cars while planning future trajectories to avoid becoming trapped mid-road. In
 139 Snake, the agent eats apples which are only available for a short period of time, creating opportu-
 140 nities that must be seized quickly, while greedy food collection might lead to positions where the
 141 growing snake traps itself. In Overcooked, the agent collaborates with a scripted partner following
 142 a non-stationary policy (App. § A). Efficient dish preparation requires not only planning a sequence
 143 of actions, but also coordinating effectively with the partner.
 144

145 **Cognitive Load** To systematically control how challenging the games are, we make the difficulty of
 146 each game tunable through a cognitive load factor (Tab. 1). In Freeway, difficulty is determined by
 147 the minimum number of steps required to traverse the road, since longer paths typically introduce
 148 more detours and require deeper planning horizon. In Snake, we vary the density of obstacles,
 149 increasing route complexity and the need for look-ahead. In Overcooked, complexity is controlled
 150 by the length of an internal kitchen counter, as a longer counter expands navigation complexity and
 151 stretches temporal windows for high-level goals, creating larger discrepancies in long-term planning
 152 and immediate execution. For each game, we design 3 levels of difficulties, easy, medium, and hard;
 153 the corresponding ranges of each level can be found in App. Tab. 5.
 154

155 **Time Pressure** To enable reproducible and hardware-agnostic evaluation, we use token count as a
 156 time proxy to simulate the games, leveraging the fact that LLM decoding time scales almost linearly
 157 with output length through time-per-output-token (TPOT), while prefilling time becomes negligible
 158 for long sequences. This yields decoding time $T = N_T \times \text{TPOT}$, where N_T is the generated token
 159

160 **Table 1: Different Games in Real-Time Reasoning Gym.**
 161

Game	Dynamic Aspect	Cognitive Load Factor	Evaluation Metrics
Freeway	Hazards	Min steps to finish: S	#Steps the agent takes to get to the other side
Snake	Opportunities	#Obstacles: N	#Apples the agent eats before collision
Overcooked	Partners	Kitchen Counter Len.: L	#Orders completed cooperatively

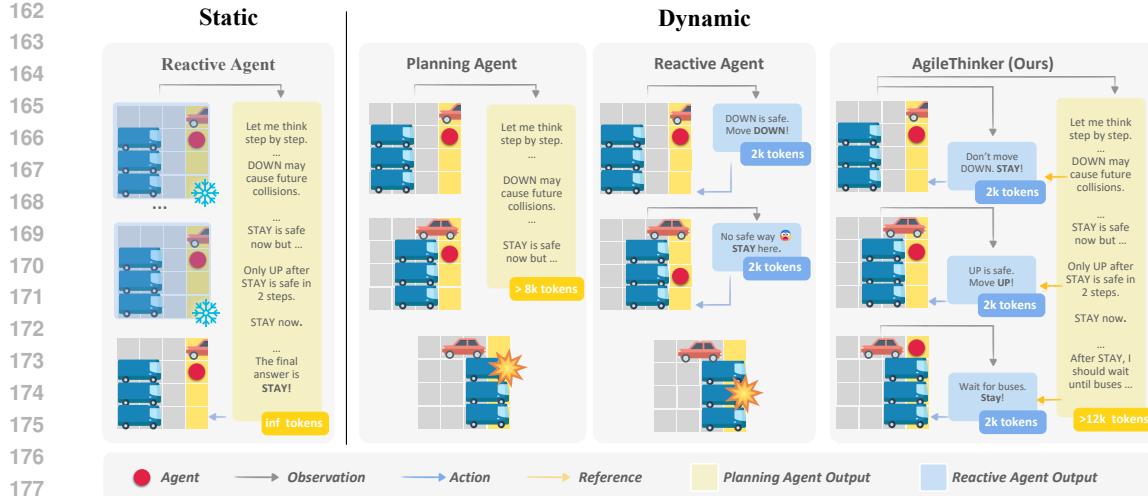


Figure 3: Existing evaluation setups for LLM Agents often assume a *static* setting, where the environment halts while the agent completes reasoning with unlimited computation. In Real-Time Reasoning Gym, environments are *dynamic*, evolving regardless of agents’ computation state. As illustrated in the *Freeway* setting, **Planning Agent**, which performs extended reasoning without interruption, fails to act timely while **Reactive Agent**, which performs reasoning strictly within environment update period, lacks foresight and collides. **AgileThinker** combines both timely reaction and long-term planning to navigate such environments effectively.

count, allowing fair comparison across deployment scenarios. We impose time pressure by letting the environment step every N_{T_E} , or T_E in Fig. 2, tokens generated by agents. When the agent cannot produce a valid action, we let the environment step with a `DEFAULT_ACTION` (Fig. 2). In Freeway and Snake, the default action is moving in the same direction as before, and in Overcooked, the default action is to stay idle. We consider four different time pressure levels, 32k, 16k, 8k, and 4k tokens per step, from low to high pressure. As shown in Figure 3, unlike existing static evaluations, the introduction of time pressure simulates the real world dynamic environments where the world does not freeze during agent reasoning.

Evaluation Evaluation metric differs for each game. In Freeway, we evaluate the number of steps the agent takes to get to the other side, while the agent is reset to the origin every time it gets hit by a car; in Snake, we count the number of apples that the agent eats before a collision; and in Overcooked, we use the number of orders that the agent and the partner completed in total. These evaluation metrics represent the capability of the agents to solve the tasks not only logically but also timely. For each game, we normalize the scores by the highest score the agent could get in that game, so we always have a score between 0 and 1. As cognitive load and time pressure increase, we expect the scores decrease. However, the scores of an agent with strong real-time reasoning capabilities should decrease slower. It is worth noting that our gym is used to evaluate *design choices of agent systems* when the model or model family (e.g. DeepSeek-V3 and R1) is fixed. Therefore, cross-model comparisons may be unfair due to their different tokenizers and underlying architectures.

3 REAL-TIME REASONING AGENTS

To address the real-time reasoning problem, we consider two solution paradigms: (1) reactive, where the agent produces a new action at every environment step, and (2) planning, where the agent reasons across multiple steps to generate an action plan, which is then executed until the agent resumes reasoning. In the following, we discuss how to create agents following each of the two paradigms, and how these two paradigms are engaged in AgileThinker.

Reactive agents We constrain reactive agents by a token budget N_i , ensuring they can respond within each environment update when $N_i \leq N_{T_E}$. We consider two kinds of language models

216 for reactive agents: (1) non-thinking models¹ that produce limited tokens for each response; and (2)
 217 thinking models that produce extended reasoning which is cut off at the token budget N_i with budget
 218 forcing (Muennighoff et al., 2025). In both cases, the agent produces one action per environment
 219 step, enabling immediate reaction to any change. This reactive approach is commonly used in agent
 220 systems these days.

221

222 **Planning agents** While reactive agents ensure timeliness, their token budget restricts the com-
 223 plexity of reasoning they can perform within a single step. In contrast, planning agents can consider
 224 their plans through multiple environment steps. We consider two formats of plans: (1) multi-step
 225 actions where a thinking model is used to generate a sequence of multiple actions to be executed;
 226 and (2) code plans where a thinking model is used to generate a code snippet that automatically
 227 produces actions based on observation input (Liang et al., 2022; Zhang et al., 2025). Although (1)
 228 is often easier to generate, (2) is more adaptive to potential changes. Both formats allow for more
 229 deliberate, long-horizon decision-making by leveraging extended reasoning.

230

231 **AgileThinker** All agents in-
 232 troduced above must complete
 233 their reasoning process before
 234 taking any action. To overcome
 235 this limitation, we propose Ag-
 236 ileThinker, which employs two
 237 parallel threads to achieve both
 238 timely action generation and un-
 239 interrupted deep planning. The
 240 planning thread \mathcal{P} runs an LLM
 241 that streams the thinking pro-
 242 cess for a multi-step action plan.
 243 Upon initialization, a reason-
 244 ing process begins that contin-
 245 ues until the execution of a plan.
 246 \mathcal{P} cannot keep up with environ-
 247 mental changes (e.g. Steps 1-2
 248 in Fig. 4). However, since its
 249 planning focuses on long-term
 250 objectives, many generated insights remain useful over extended horizons. In contrast, the reac-
 251 tive thread \mathcal{R} runs a separate LLM under strict time constraints $T_{\mathcal{R}} \leq T_{\mathcal{E}}$, making decisions based
 252 on the latest observation and the (partial) output of \mathcal{P} (see gray and yellow arrows in Fig. 4).

253

254 The coordination between the two threads \mathcal{R} and \mathcal{P} follows a time-sharing protocol: during each
 255 environment step, \mathcal{P} operates continuously while \mathcal{R} activates only in the final $T_{\mathcal{R}}$ time units. The
 256 hyperparameter $T_{\mathcal{R}}$ controls the resource trade-off between the two threads. With a larger $T_{\mathcal{R}}$, the
 257 reactive thread can be more adaptive, but there will be less reasoning from the planning thread to
 258 refer to. Effectively balancing planning and reaction resources is the key to success in AgileThinker,
 259 which will be discussed in §5.

260

261

4 IS SINGLE PARADIGM ENOUGH FOR REAL-TIME REASONING?

262

263 **Evaluation Setup:** To investigate how **cognitive load** and **time pressure** affect the performance
 264 respectively, we conduct two series of experiments. (1) Cognitive load varies (Easy, Medium, Hard)
 265 while time pressure is fixed at 8k tokens/step—lenient enough for non-thinking models to complete
 266 their responses, yet restrictive for thinking models. The intrinsic bound N_i (see Section 3) for
 267 reactive agent is set to 8k. (2) Time pressure varies ($N_{T_{\mathcal{E}}} \in \{4k, 8k, 16k, 32k\}$) with medium
 268 cognitive load. Here, N_i is set to 4k to ensure it remains lower than time pressure budget. We
 269 evaluate each agent 32 times (8 game seeds \times 4 LLM sampling seeds) under each setting and report

270 ¹Thinking models are the LLMs trained with reinforcement learning to incentivize reasoning before generat-
 271 ing answers (DeepSeek-AI et al., 2025), while non-thinking models are the LLMs that have not been specifically
 272 trained to generate long reasoning.

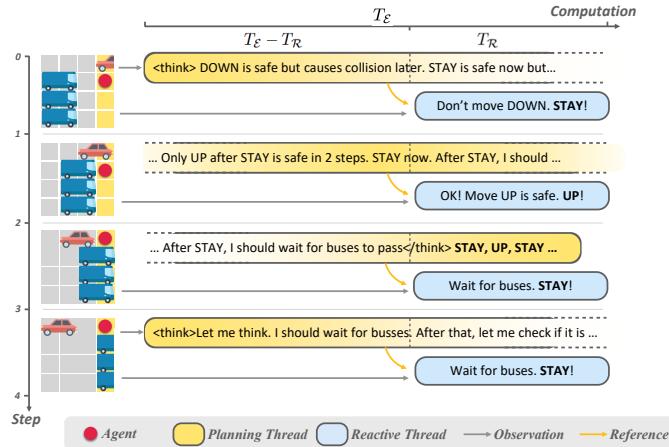


Figure 4: Two parallel threads in AgileThinker

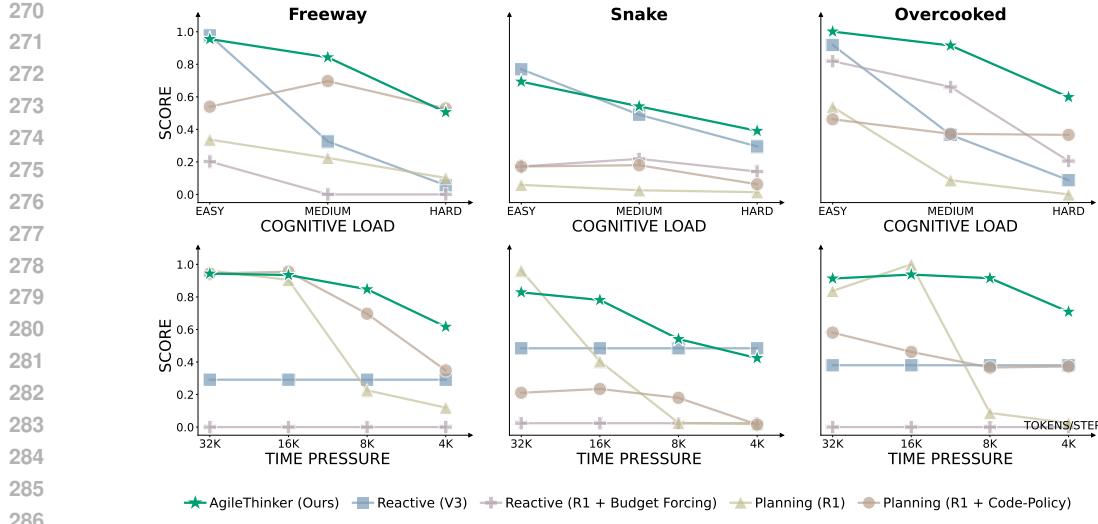


Figure 5: Performance of reasoning agents in Real-Time Reasoning Gym under varying cognitive loads and time pressures. Upper: we fix time pressure at 8k tokens per step and vary cognitive load. Lower: we fix cognitive load at medium level and vary time pressure. Full data at App. § C.

the average score of these samples. Details of the environments and score calculation can be found in App. § A, and prompts are provided in App. § B.

Figure 1 reports the average scores over three games, while Figure 5 provides a breakdown per game. The results show that reactive and planning agents fail to balance decision quality and efficiency, whereas our AgileThinker achieves robust performance under varying conditions.

Reactive agent sacrifices decision quality for efficiency. By design, the reactive agent restricts computation time less than $T_{\mathcal{E}}$ and maintains consistent performance across all time pressures. However, the limit on test-time scaling also causes a dramatic performance drop as cognitive load increases (scores falling from 0.89 to 0.15, versus 0.88 to 0.50 for AgileThinker). This drop stems from its inability to consider future consequences of a move carefully. As exemplified in the case study in Figure 6, the reactive agent greedily pursues immediate rewards, falling into predictable traps while AgileThinker avoids by considering long-term survival requirements.

Planning agent optimizes for decision quality but suffers under time pressure. Planning agent excels under relaxed time constraints but suffers from dramatic degradation when time pressure increases (scores dropping from 0.92 to 0.05, versus 0.90 to 0.58 for AgileThinker). Its fundamental flaw is obliviousness to environmental changes, executing plans based on outdated observations. As illustrated in the case study, the agent is unaware that the snake has moved forward during reasoning, hence it crashes into a wall by following the obsolete plan. However, reactive thread in AgileThinker is informed of the latest state, thus able to adjust the output of planning thread.

The variants in §3 fail to balance speed and quality. R1 with budget forcing severely degrades performance ($0.01 < 0.39$ vs. V3), as forced truncation typically results in no-ops. Even advanced budget-aware methods (Figure 9) cannot prevent such drops. Meanwhile, R1 with code-policy succeeds only in simple algorithmic tasks like Freeway. It underperforms in complex scenarios where contextual reasoning (e.g., Theory-of-Mind) cannot be effectively compressed into code (see App. §C.4).

5 HOW TO MANAGE RESOURCES BETWEEN REACTION AND PLANNING?

Effective coordination between reactive thread \mathcal{R} and planning thread \mathcal{P} requires careful time management to determine when to invoke \mathcal{R} within each environment step. We analyze how the token budget $N_{T_{\mathcal{R}}}$ allocated to \mathcal{R} affects the performance of AgileThinker. We set $N_{T_{\mathcal{E}}}$ to 8k and vary the token budget $N_{T_{\mathcal{R}}}$ from 0.5k to 8k tokens. Figure 7 presents agent scores under different $N_{T_{\mathcal{R}}}$, where we also plot the cumulative distribution function (CDF) of \mathcal{R} 's token usage across all game trajectories without constraints to understand its inherent computational requirements.

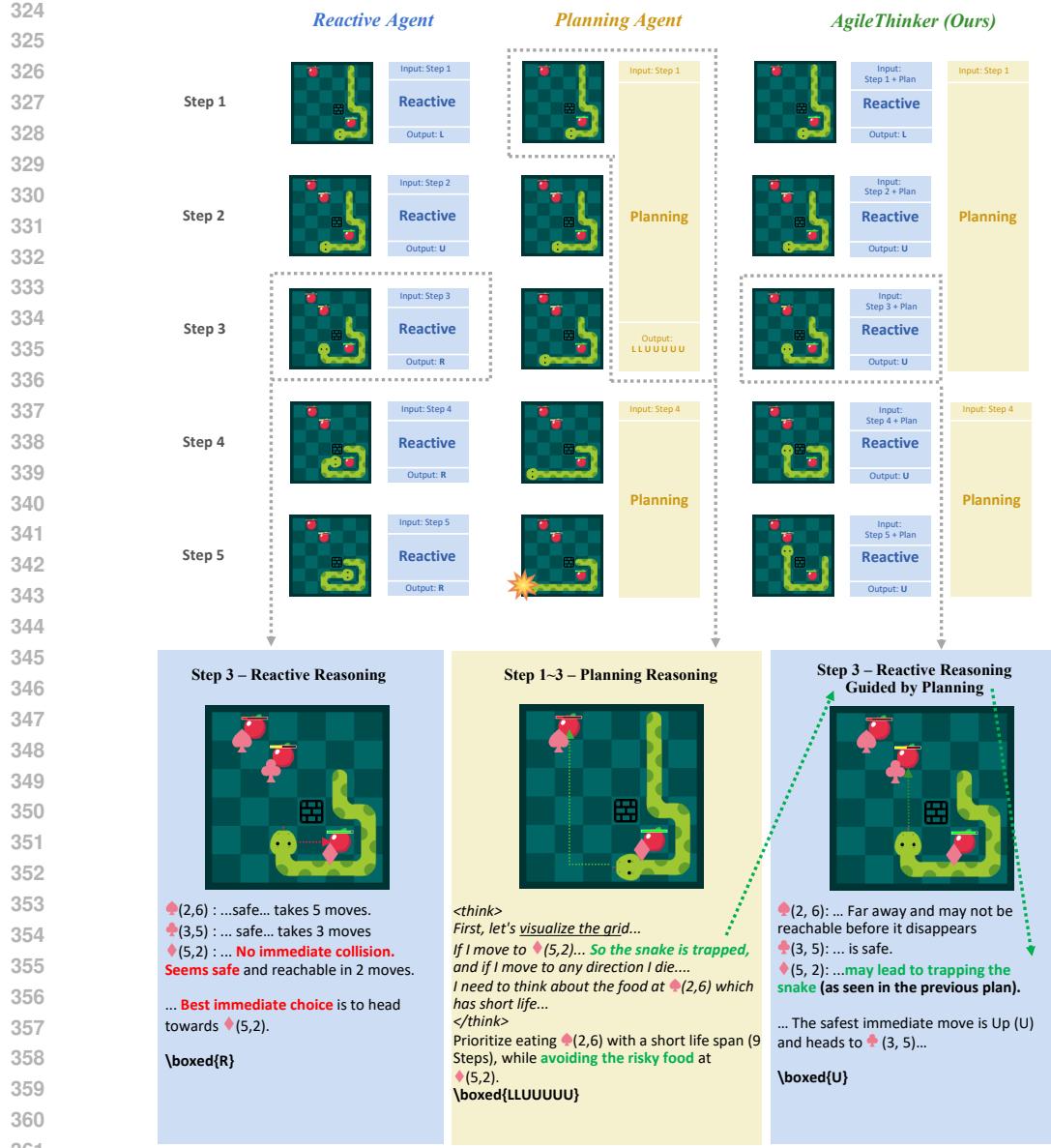


Figure 6: **Thinking trajectories of different paradigms at critical steps** At step 3, **Reactive Agent** (V3) greedily pursues the nearest food and collides inevitably after three steps. **Planning Agent** (R1), still reasoning over the outdated step-1 state, defaults left. However, it correctly identifies that eating the nearest food would result in a future collision, and that its lifespan is sufficient to delay consumption. Guided by the reasoning of Reactive Thread, Planning Thread in the **AgileThinker** anticipates the trap and chooses to move upward toward a safer food target.

We can see that setting N_{T_R} too small (e.g., 0.5k) leads to low scores, as \mathcal{R} doesn't have enough time to process strategic guidance from \mathcal{P} and generate well-reasoned actions. Conversely, setting N_{T_R} too large creates idle periods where \mathcal{R} has completed action generation but \mathcal{P} continues productive reasoning. Empirically, performance peaks when N_{T_R} approximates the natural token upper bound of \mathcal{R} , as indicated by the CDF of \mathcal{R} 's token usage. This suggests that \mathcal{R} benefits from fully utilizing its allocated time without truncation or extended idling.

It is worth noting that the optimal time budget varies across environments and requires empirical tuning. However, AgileThinker consistently outperform single-system baselines across broad budget ranges, suggesting that rough upper bound estimations are sufficient. In App. § E, we further propose a dynamic adjustment mechanism for N_{T_R} at test time, which is also effective.

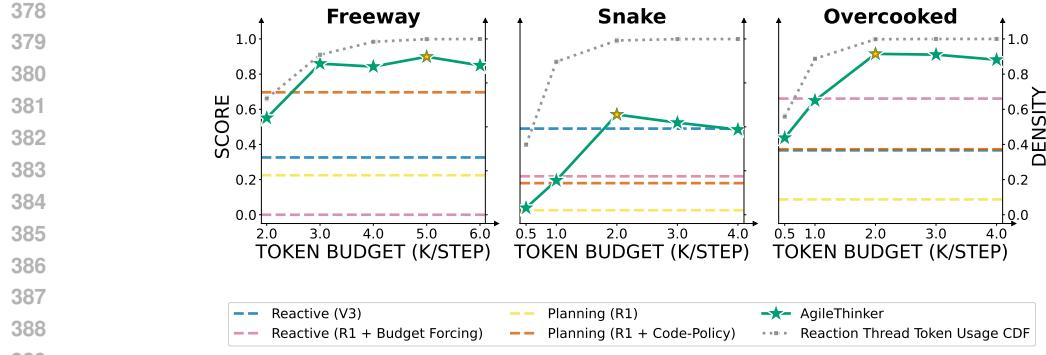


Figure 7: Performance of AgileThinker under different reactive thread token budgets N_{TR} . The cumulative distribution function (CDF) shows the natural token usage of \mathcal{R} across all game trajectories when generation is not truncated, indicating inherent computational requirements of \mathcal{R} .

6 PERFORMANCE IMPROVEMENT UNDER WALL-CLOCK TIME

To validate the practical applicability of our token-based simulation, we conduct experiments using actual wall-clock time with official API of DeepSeek. Our results show token count has strong linear correlation with physical inference time. Specifically, we model this relationship as $T = \alpha N + \beta$, where

T represents total runtime and N represents generated tokens. Least squares estimation on all experiment trajectories (plotted in Figure 10) yields $\alpha = 0.0473$ s/token, $\beta = 334.55$ s, with $R^2 = 0.9986$. This near-perfect correlation validates our token-based temporal abstraction and confirms its practical relevance for real-world deployments.

We also conduct experiments to verify that the advantage of AgileThinker remains when the game is simulated in wall-clock time. Using the derived TPOT of 0.047 s/token, we evaluate agent systems with environment evolution intervals of $T_E = 6$ minutes, corresponding to approximately 8,000 tokens per step. Table 2 shows that AgileThinker consistently outperforms both Reactive and Planning Agents in physical time. These results establish that our framework’s benefits extend beyond theory to practical applications, demonstrating applicability of our architecture for agent deployments on intelligence-demanding, real-time tasks.

7 RELATED WORK

Evaluation Environments for LLM Agents: Existing evaluation setups for LLM agents mostly focus on *static* environments where nothing changes during episodes (Yang et al., 2024; Zhou et al., 2024b) or state pauses during LLM reasoning (Zhou et al., 2024a; Shi et al., 2025). This unrealistic assumption risks performance drop and even safety hazards when applying LLM agents in latency-sensitive applications (Sinha et al., 2024; Zheng et al., 2025). Prior work has modeled computation delays through Delay-Aware MDPs (Chen et al., 2020), sticky-action schemes (Mahmood et al., 2018), and asynchronous interactive MDPs (Travnik et al., 2018; Riemer et al., 2024), but the scope of these works is limited to traditional reinforcement learning. Although some works (Liu et al., 2024; Zhang et al., 2025) do adopt LLM agents in wall-clock time, our work is the first to formalize real-time reasoning problem for LLM Agents. In particular, we measure elapsed token count as a hardware-agnostic temporal unit, enabling fair and reproducible comparison across agent systems.

Budget Control for Reasoning Models: Test-time compute improves LLM performance but increases inference time, with overthinking behaviors commonly observed in current reasoning models (Chen et al., 2025). Budget control aims to maximize LLM performance under fixed budgets, and popular methods include early truncation (Muennighoff et al., 2025), prompting (Pu et al., 2025)

432 and training (Aggarwal & Welleck, 2025; Team et al., 2025; Gemini Team, 2025). These methods
 433 are effective to a certain extent, but still struggle with precise control over generated token count
 434 (Alomrani et al., 2025) and performance drops when budgets are far from adequate (Han et al.,
 435 2025). This suggests that existing techniques are inadequate for handling both loose and tight bud-
 436 get constraints within a single model. The results on Real-Time Reasoning Gym demonstrate that
 437 SOTA budget control methods cannot effectively balance reaction and planning, necessitating dual
 438 LLM architectures for real-time environments.

439 **Real-Time AI and Dual-Process Architectures:** The trade-off between accuracy and latency is
 440 a foundational problem in computer science, extensively studied ranging from real-time heuristic
 441 search to robotics (Kim et al., 2024). Classic approaches address this by dynamically allocating
 442 compute resources, such as using cascade classifiers for efficient detection (Chen & Yuille, 2005)
 443 or frame-skipping strategies in video tracking (Luo et al., 2018; Wang et al., 2020). While these
 444 paradigms are well-established in traditional AI, they are being revisited in the context of Large
 445 Language Models (LLMs), where variable inference costs introduce new challenges for real-time
 446 interaction (Ginart et al., 2024). Recent works have explored real-time capabilities in embodied
 447 agents (Kim et al., 2024) and asynchronous tool usage (Ginart et al., 2024), yet a unified architectural
 448 framework for balancing reasoning depth and reaction speed remains an open question.

449 To address this, researchers often draw inspiration from the dual process theory, which posits that
 450 human cognition operates through two distinct components: *System 1* (fast and intuitive) and *System*
 451 2 (slow and deliberate) (Evans, 2013; Kahneman, 2011). This cognitive model has motivated various
 452 dual-agent designs, combining fast modullessuch as finite-state machines (Zhang et al., 2025), vision
 453 transformers (Cui et al., 2025), or small language models (Liu et al., 2024)with powerful, computa-
 454 tionally intensive modules like LLMs with tools (Christakopoulou et al., 2024) or Large Reasoning
 455 Models (LRMs) (Zhang et al., 2025). However, most existing architectures treat these systems as
 456 separate stages (e.g., cascading) or isolated parallel processes. AgileThinker distinctively advances
 457 this paradigm by allowing the *System 1* (Reactive Thread) to access the real-time partial reasoning
 458 traces of *System 2* (Planning Thread). This shared-context mechanism enables informed decision-
 459 making with minimal delay, effectively bridging the gap between classical real-time efficiency and
 460 modern LLM reasoning capabilities.

460

461

462 8 CONCLUSION

463

464 In this work, we identified and formalized **real-time reasoning**, a fundamental challenge faced by
 465 agents in real-world deployment. We introduced **Real-Time Reasoning Gym**, the first gym for
 466 evaluating LLM agents in continuously evolving environments. It supports independent control of
 467 cognitive load and time pressure, using token count as a hardware-independent temporal measure.
 468 Our evaluation revealed critical shortcomings of existing reasoning paradigms (reactive and plan-
 469 ning reasoning). To address this gap, we proposed **AgileThinker**, which engages two reasoning
 470 paradigms in parallel. Experiments demonstrate that our method consistently outperforms all base-
 471 lines, with advantages growing as cognitive load increases and time constraints tighten. Future work
 472 can extend our gym to more realistic scenarios, improve coordination mechanisms between two
 473 threads, or leverage our gym to train urgency-aware LLM agents.

474

475

476

477 9 LIMITATIONS

478

479 Our method investigates the real-time reasoning in LLM agents. Although our formulation is gen-
 480 eral, we only conducted experiments on DeepSeek models due to two reasons: (1) open source
 481 models perform poorly in general, thus the difference between different systems is not significant
 482 enough, and (2) other commercial model providers, including OpenAI, Google and Anthropic, do
 483 not provide reasoning traces which are crucial in our evaluation. We also try our best to make sure
 484 the readers understand that we do not have any empirical evidence showing that the dual system
 485 implemented in AgileThinker is precisely modeling human dual systems. The connection and dif-
 ference require more rigorous evaluation.

486 10 USE OF LANGUAGE MODELS
487488 We used large language models to assist with paraphrasing and improving the readability of this
489 paper. All research ideas, designs, experiments, analyses, and conclusions are entirely our own.
490491 11 REPRODUCIBILITY STATEMENT
492493 We are committed to ensuring the reproducibility of our results. The implementation of Real-Time
494 Reasoning Gym and AgileThinker, including all necessary code and dependencies, will be made
495 publicly available on GitHub upon paper publication. The code can be run on any platform sup-
496 porting DeepSeek V3 and R1 models, which we use for our experiments. Detailed instructions on
497 how to set up and reproduce our results will be provided in the repository, including any additional
498 dependencies and environment configurations.
499500 REFERENCES
501502 Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with
503 reinforcement learning, 2025. URL <https://arxiv.org/abs/2503.04697>.504 Mohammad Ali Alomrani, Yingxue Zhang, Derek Li, Qianyi Sun, Soumyasundar Pal, Zhanguang
505 Zhang, Yaochen Hu, Rohan Deepak Ajwani, Antonios Valkanas, Raika Karimi, Peng Cheng,
506 Yunzhou Wang, Pengyi Liao, Hanrui Huang, Bin Wang, Jianye Hao, and Mark Coates. Reasoning
507 on a budget: A survey of adaptive and controllable test-time compute in llms, 2025. URL <https://arxiv.org/abs/2507.02076>.
508509 Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and
510 Wojciech Zaremba. Openai gym, 2016.511 Baiming Chen, Mengdi Xu, Liang Li, and Ding Zhao. Delay-aware model-based reinforcement
512 learning for continuous control, 2020. URL <https://arxiv.org/abs/2005.05440>.513 Xiangrong Chen and A.L. Yuille. A time-efficient cascade for real-time object detection: With
514 applications for the visually impaired. In *2005 IEEE Computer Society Conference on Computer
Vision and Pattern Recognition (CVPR'05) - Workshops*, pp. 28–28, 2005. doi: 10.1109/CVPR.
515 2005.399.516 Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu,
517 Mengfei Zhou, Zhuseng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Do not
518 think that much for 2+3=? on the overthinking of ol-like llms, 2025. URL <https://arxiv.org/abs/2412.21187>.519 Konstantina Christakopoulou, Shibl Mourad, and Maja Matari. Agents thinking fast and slow: A
520 talker-reasoner architecture, 2024. URL <https://arxiv.org/abs/2410.08328>.521 Can Cui, Pengxiang Ding, Wenxuan Song, Shuanghao Bai, Xinyang Tong, Zirui Ge, Runze Suo,
522 Wanqi Zhou, Yang Liu, Bofang Jia, Han Zhao, Siteng Huang, and Donglin Wang. Openhelix: A
523 short survey, empirical analysis, and open-source dual-system vla model for robotic manipulation,
524 2025. URL <https://arxiv.org/abs/2505.03912>.525 DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu,
526 Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu,
527 Zhibin Gou, Zhihong Shao, Zhushu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao
528 Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan,
529 Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao,
530 Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding,
531 Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang
532 Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai
533 Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang,
534 Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang,
535 Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang,
536

540 Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang,
 541 R. J. Chen, R. L. Jin, Ruyi Chen, Shanghai Lu, Shangyan Zhou, Shanhua Chen, Shengfeng
 542 Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing
 543 Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanja Zhao, Wen
 544 Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong
 545 Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu,
 546 Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaoqin Shen, Xi-
 547 aoshua Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia
 548 Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng
 549 Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong
 550 Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong,
 551 Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou,
 552 Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying
 553 Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda
 554 Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu,
 555 Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu
 556 Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforce-
 557 ment learning, 2025. URL <https://arxiv.org/abs/2501.12948>.

558 Jonathan St. B. T. Evans and Keith E. Stanovich. Dual-process theories of higher cognition:
 559 Advancing the debate. *Perspectives on Psychological Science*, 8(3):223–241, 2013. doi:
 560 10.1177/1745691612460685. URL <https://doi.org/10.1177/1745691612460685>.
 561 PMID: 26172965.

562 Jonathan St BT Evans. Dual-process theories of reasoning: Contemporary issues and developmental
 563 applications. *Developmental Review*, 33(2):145–170, 2013.

564 Mohamed Amine Ferrag, Norbert Tihanyi, and Merouane Debbah. From llm reasoning to au-
 565 tonomous ai agents: A comprehensive review. *arXiv preprint arXiv:2504.19678*, 2025.

566 Gemini Team. Gemini 2.5: A multimodal ai model. https://storage.googleapis.com/deepmind-media/gemini/gemini_v2_5_report.pdf, January 2025. Technical Re-
 567 port, Google DeepMind.

568 Antonio A. Ginart, Naveen Kodali, Jason Lee, Caiming Xiong, Silvio Savarese, and John Em-
 569 mons. Asynchronous tool usage for real-time agents, 2024. URL <https://arxiv.org/abs/2410.21620>.

570 Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and
 571 Weizhu Chen. Tora: A tool-integrated reasoning agent for mathematical problem solving. *arXiv*
 572 *preprint arXiv:2309.17452*, 2023.

573 Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-
 574 budget-aware llm reasoning, 2025. URL <https://arxiv.org/abs/2412.18547>.

575 Daniel Kahneman. *Thinking, fast and slow*. Farrar, Straus and Giroux, 2011.

576 Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair,
 577 Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Ben-
 578 jamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn.
 579 Openvla: An open-source vision-language-action model, 2024. URL <https://arxiv.org/abs/2406.09246>.

580 Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and
 581 Andy Zeng. Code as policies: Language model programs for embodied control. In *arXiv preprint*
 582 *arXiv:2209.07753*, 2022.

583 Jijia Liu, Chao Yu, Jiaxuan Gao, Yuqing Xie, Qingmin Liao, Yi Wu, and Yu Wang. Llm-
 584 powered hierarchical language agent for real-time human-ai coordination, 2024. URL <https://arxiv.org/abs/2312.15224>.

585 Hao Luo, Wenxuan Xie, Xinggang Wang, and Wenjun Zeng. Detect or track: Towards cost-effective
 586 video object detection/tracking, 2018. URL <https://arxiv.org/abs/1811.05340>.

594 A. Rupam Mahmood, Dmytro Korenkevych, Gautham Vasan, William Ma, and James Bergstra.
595 Benchmarking reinforcement learning algorithms on real-world robots, 2018. URL <https://arxiv.org/abs/1809.07731>.
596

597 Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke
598 Zettlemoyer, Percy Liang, Emmanuel Cands, and Tatsunori Hashimoto. s1: Simple test-time
599 scaling, 2025. URL <https://arxiv.org/abs/2501.19393>.
600

601 Xiao Pu, Michael Saxon, Wenyue Hua, and William Yang Wang. Thoughtterminator: Bench-
602 marking, calibrating, and mitigating overthinking in reasoning models, 2025. URL <https://arxiv.org/abs/2504.13367>.
603

604 Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and
605 Rafael Rafailov. Agent q: Advanced reasoning and learning for autonomous ai agents. *arXiv*
606 preprint [arXiv:2408.07199](https://arxiv.org/abs/2408.07199), 2024.
607

608 Matthew Riemer, Gopesh Subbaraj, Glen Berseth, and Irina Rish. Enabling realtime reinforcement
609 learning at scale with staggered asynchronous inference, 2024. URL <https://arxiv.org/abs/2412.14355>.
610

611 Jiajun Shi, Jian Yang, Jiaheng Liu, Xingyuan Bu, Jiangjie Chen, Junting Zhou, Kaijing Ma, Zhou-
612 futu Wen, Bingli Wang, Yancheng He, Liang Song, Hualei Zhu, Shilong Li, Xingjian Wang, Wei
613 Zhang, Ruibin Yuan, Yifan Yao, Wenjun Yang, Yunli Wang, Siyuan Fang, Siyu Yuan, Qianyu
614 He, Xiangru Tang, Yinghui Tan, Wangchunshu Zhou, Zhaoxiang Zhang, Zhoujun Li, Wenhao
615 Huang, and Ge Zhang. Korgym: A dynamic game platform for llm reasoning evaluation, 2025.
616 URL <https://arxiv.org/abs/2505.14552>.
617

617 Rohan Sinha, Amine Elhafsi, Christopher Agia, Matthew Foutter, Edward Schmerling, and Marco
618 Pavone. Real-time anomaly detection and reactive planning with large language models, 2024.
619

620 Keith E. Stanovich and Richard F. West. Individual differences in reasoning: Implications for
621 the rationality debate? *Behavioral and Brain Sciences*, 23(5):645665, 2000. doi: 10.1017/
622 S0140525X00003435.

623 Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun
624 Xiao, Chenzhuang Du, Chonghua Liao, Chuning Tang, Congcong Wang, Dehao Zhang, Enming
625 Yuan, Enzhe Lu, Fengxiang Tang, Flood Sung, Guangda Wei, Guokun Lai, Haiqing Guo, Han
626 Zhu, Hao Ding, Hao Hu, Hao Yang, Hao Zhang, Haotian Yao, Haotian Zhao, Haoyu Lu, Haoze
627 Li, Haozhen Yu, Hongcheng Gao, Huabin Zheng, Huan Yuan, Jia Chen, Jianhang Guo, Jianlin
628 Su, Jianzhou Wang, Jie Zhao, Jin Zhang, Jingyuan Liu, Junjie Yan, Junyan Wu, Lidong Shi,
629 Ling Ye, Longhui Yu, Mengnan Dong, Neo Zhang, Ningchen Ma, Qiwei Pan, Qucheng Gong,
630 Shaowei Liu, Shengling Ma, Shupeng Wei, Sihan Cao, Siying Huang, Tao Jiang, Weihao Gao,
631 Weimin Xiong, Weiran He, Weixiao Huang, Weixin Xu, Wenhao Wu, Wenyang He, Xianghui
632 Wei, Xianqing Jia, Xingzhe Wu, Xinran Xu, Xinxing Zu, Xinyu Zhou, Xuehai Pan, Y. Charles,
633 Yang Li, Yangyang Hu, Yangyang Liu, Yanru Chen, Yejie Wang, Yibo Liu, Yidao Qin, Yifeng
634 Liu, Ying Yang, Yiping Bao, Yulun Du, Yuxin Wu, Yuzhi Wang, Zaida Zhou, Zhaoji Wang,
635 Zhaowei Li, Zhen Zhu, Zheng Zhang, Zhexu Wang, Zhilin Yang, Zhiqi Huang, Zihao Huang,
636 Ziyao Xu, Zonghan Yang, and Zongyu Lin. Kimi k1.5: Scaling reinforcement learning with llms,
637 2025. URL <https://arxiv.org/abs/2501.12599>.
638

638 Jaden B. Travnik, Kory W. Mathewson, Richard S. Sutton, and Patrick M. Pilarski. Reactive rein-
639 forcement learning in asynchronous environments. *Frontiers in Robotics and AI*, 5, June 2018.
640 ISSN 2296-9144. doi: 10.3389/frobt.2018.00079. URL <http://dx.doi.org/10.3389/frobt.2018.00079>.
641

642 Zhongdao Wang, Liang Zheng, Yixuan Liu, Yali Li, and Shengjin Wang. Towards real-time multi-
643 object tracking. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (eds.),
644 *Computer Vision – ECCV 2020*, pp. 107–122, Cham, 2020. Springer International Publishing.
645 ISBN 978-3-030-58621-8.
646

646 John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan,
647 and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering,
648 2024. URL <https://arxiv.org/abs/2405.15793>.
649

648 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao.
 649 React: Synergizing reasoning and acting in language models, 2022.
 650

651 Shao Zhang, Xihuai Wang, Wenhao Zhang, Chaoran Li, Junru Song, Tingyu Li, Lin Qiu, Xuezhi
 652 Cao, Xunliang Cai, Wen Yao, Weinan Zhang, Xinbing Wang, and Ying Wen. Leveraging dual
 653 process theory in language agent framework for real-time simultaneous human-ai collaboration,
 654 2025. URL <https://arxiv.org/abs/2502.11882>.

655 Yangqing Zheng, Shunqi Mao, Dingxin Zhang, and Weidong Cai. Llm-enhanced rapid-reflex async-
 656 reflect embodied agent for real-time decision-making in dynamically changing environments.
 657 *arXiv preprint arXiv:2506.07223*, 2025.
 658

659 Qinhong Zhou, Sunli Chen, Yisong Wang, Haozhe Xu, Weihua Du, Hongxin Zhang, Yilun Du,
 660 Joshua B. Tenenbaum, and Chuang Gan. Hazard challenge: Embodied decision making in dy-
 661 namically changing environments, 2024a. URL <https://arxiv.org/abs/2401.12975>.

662 Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng,
 663 Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic
 664 web environment for building autonomous agents, 2024b. URL <https://arxiv.org/abs/2307.13854>.
 665

668 A ENVIRONMENT DETAILS

669

- 670 • **Freeway:** The player navigates across parallel highways with moving cars. At each step
 671 the player can move to an adjacent lane or stay in place. Cars move forward at constant
 672 speeds and new cars may spawn on either side of the road. If the player is hit by a car, it
 673 will be reset to the starting position. The game terminates if the player crosses the road or
 674 if the step limit $M = 100$ is reached. The reward for a trajectory τ is computed as:
 675

$$676 R(\tau) = M - |\tau|$$

677

- 678 • **Snake:** The player controls a snake in a 2D rectangular grid with surrounding walls and
 679 internal obstacles. At each step, the snake can move one step left, right, up or down. If the
 680 snake head collides with an obstacle or its body segment, it dies. Foods spawn continuously
 681 in the map and disappears after a fixed number of steps; eating food increases the snake's
 682 length by 1 unit. The game terminates if the snake dies or the number of steps exceeds
 683 threshold $M = 100$. The reward is calculated as:
 684

$$R(\tau) = \text{Number of eaten food} - \mathbb{I}[\text{Dies in } M \text{ steps}]$$

685

- 686 • **Overcooked:** A fully observable two-player cooperative game where players must collect
 687 onions, cook them in a pot and serve the cooked soup for rewards. At each step, the players
 688 can move in 4 directions or use interact action to trigger some events, such as picking or
 689 placing an item depending on the game state. Since we focus on single-agent settings, we
 690 model the second player as part of the changing environment, controlled by a manually
 691 written script for simplicity and consistency. This agent randomly chooses one policy to
 692 follow: *deliver an onion into an arbitrary pot or a kitchen counter*. The game runs for
 693 $M = 100$ steps, and rewards are assigned for accomplishment of special events listed in
 694 Table 3. The game is implemented based on the repository https://github.com/HumanCompatibleAI/overcooked_ai

695 Event	696 Reward
697 Picking up a dish from the dispenser	3
698 Picking up a cooked soup from the pot	5
699 Serving the soup	20

700

701 Table 3: Rewards for different events in the Overcooked environment.

The reward is then normalized to $[0, 1]$ to get the final game score. Specifically, let R_{\min} and R_{\max} be the minimum and maximum rewards observed in all trajectories. The score S is computed as: $S = \frac{R - R_{\min}}{R_{\max} - R_{\min}}$. Empirically, the R_{\max} and R_{\min} of each environment are listed in Table 4.

Environment	R_{\min}	R_{\max}
Freeway	0	89
Snake	-1	15
Overcooked	0	56

Table 4: Minimum and maximum rewards for each environment.

Table 5: Game difficulty settings, showing ranges for easy, medium, and hard levels.

Game	Dynamic Aspect	Cognitive Load Factor	Easy	Medium	Hard
Freeway	Hazards	Min steps to finish: S	$S \leq 12$	$13 \leq S \leq 16$	$17 \leq S \leq 21$
Snake	Opportunities	#Obstacles: N	$N = 1$	$2 \leq N \leq 5$	$6 \leq N \leq 8$
Overcooked	Partners	Kitchen Counter Len.: L	$L = 0$	$L = 3$	$L = 4$

B PROMPT

Prompt for the Planning Agent

Now a player is playing a multi-turn game, and suppose current turn is $\{t_1\}$. Given the initial position $(0, y_{t_1})$ on a 2D grid (vertical axis $y = 0, 1, \dots, 9$), determine the minimal number of turns H and a sequence of actions $\{a_{t_1+t}\}_{t=0}^{H-1}$ to reach $(0, 9)$, avoiding collisions with cars on freeways $y = 1, \dots, 8$.

1. Game Dynamics:

- **Player update:** $y_{t+1} = y_t + \Delta y_t$, where

$$\Delta y_t = \begin{cases} +1 & \text{if } a_t = U \\ -1 & \text{if } a_t = D, \quad y_{t+1} \in [0, 9] \\ 0 & \text{if } a_t = S \end{cases}$$

- **Car update rules:**

For car k on freeway i , suppose its head is at h , tail is at τ at turn t_1 , and speed is s . Then at turn $T > t_1$, the car span becomes:

- Left-moving: $\text{Span}(t_1) = [h, \tau] \rightarrow \text{Span}(T) = [h - s(T - t_1), \tau - s(T - t_1)]$
- Right-moving: $\text{Span}(t_1) = [\tau, h] \rightarrow \text{Span}(T) = [\tau + s(T - t_1), h + s(T - t_1)]$

- Collision occurs at turn T only if $0 \in \text{Span}(T)$ for any car on freeway y_T .
- Note that if you decide to move to $y_{T+1} \neq y_T$ at turn T , you will **NOT** be considered to be on y_{T+1} at turn T , thus will **NOT** be collided by cars on y_{T+1} if $0 \in \text{Span}(T)$ but $0 \notin \text{Span}(T+1)$.

2. Task (Turn t_1):

Find a sequence of actions $\{a_{t_1+t}\}_{t=1}^{H-1}$ which minimizes H such that $y_{t_1+H-1} = 9$.

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

771

772

773

774

775

776

777

778

779

780

781

782

783

784

785

786

787

788

789

790

791

792

793

794

795

796

797

798

799

800

801

802

803

804

805

806

807

808

809

Answer Format:

Turn t_1 : a_{t_1}
Turn $t_1 + 1$: a_{t_1+1}
\vdots

3. Current State (Turn t_1):**Current Turn:** $t_0 = 10$ **Player Position:** (0, 6)**Car State:**

Freeway k	Cars (head h , tail τ , direction d , speed s)
1	(48, 37, right, 12), (0, -11, right, 12)
2	(48, 1, right, 48)
...	...
8	(48, 37, right, 12), (0, -11, right, 12)

810 **Prompt for the Reactive Agent**
811812 **Prompt:**
813814 You are a player in a freeway game, starting at $(0, y_{t_0})$ on a 2D grid (vertical axis $y = 0, 1, \dots, 9$).
815 Your goal is to reach $(0, 9)$ while avoiding collisions with cars on freeways $y = 1, \dots, 8$.
816817 **1. Game Dynamics:**
818819 • **Player update:**
820

821
$$y_{t+1} = y_t + \Delta y_t, \text{ where}$$

822
$$\Delta y_t = \begin{cases} +1 & \text{if } a_t = U \\ -1 & \text{if } a_t = D, \quad y_{t+1} \in [0, 9] \\ 0 & \text{if } a_t = S \end{cases}$$

823

824 • **Car update rules:**
825826 For car k on freeway i , suppose its head is at h , tail is at τ at turn t_0 , and speed is s . Then at
827 turn $T > t_0$, the car span becomes:
828829

- Left-moving: $\text{Span}(t_0) = [h, \tau] \rightarrow \text{Span}(T) = [h - s(T - t_0), \tau - s(T - t_0)]$
- Right-moving: $\text{Span}(t_0) = [\tau, h] \rightarrow \text{Span}(T) = [\tau + s(T - t_0), h + s(T - t_0)]$

830831 • Collision occurs at turn T only if $0 \in \text{Span}(T)$ for any car on freeway y_T .
832833 • Note that if you decide to move to $y_{T+1} \neq y_T$ at turn T , you will **NOT** be considered
834 to be on y_{T+1} at turn T , thus will **NOT** be collided by cars on y_{T+1} if $0 \in \text{Span}(T)$ but
835 $0 \notin \text{Span}(T+1)$.
836837 **2. Guidance from a Previous Thinking Model (Turn $t_1 \leq t_0$):**
838839 Sometimes, you have access to a past output from a thinking model, computed at turn t_1 based on
840 then-current observations. This guidance may no longer perfectly match the current situation but can
841 still be valuable for decision-making. You can use this plan as a **strategic reference**, not a mandatory
842 instruction. Consider how much of the original strategy is still valid under the current dynamics.
843844 **3. Task (Turn t_0):**
845846 Choose **one** action $a_{t_0} \in \{U, D, S\}$ for the current turn, with the following considerations:
847848

- **Collision Avoidance:** Ensure the action avoids both immediate and near-future collisions.
- **Strategic Consistency (Optional):** Refer to the thinking model's prior strategy. If the current
849 environment still aligns with its assumptions, you may choose to continue along the same strategic direction. If not, adapt as needed.

850851 **Answer Format:**
852

853

a_{t_0}

854

855 **4. Current State (Turn t_0):**
856857 **Current Turn:** $t_0 = 10$
858859 **Player Position:** $(0, 6)$
860861 **Car State:**
862

863

864
865
866
867
868
869
870

Freeway k	Cars (head h , tail τ , direction d , speed s)
1	(48, 37, right, 12), (0, -11, right, 12)
2	(48, 1, right, 48)
3	(-48, -25, left, 24)
4	(-48, -59, right, 12), (-12, -23, right, 12), (24, 13, right, 12)
5	(-24, -35, right, 4)
6	(-24, -35, right, 4), (52, 41, right, 4), (16, 5, right, 4)
7	(48, 37, right, 12), (0, -11, right, 12)
8	(48, 37, right, 12), (0, -11, right, 12)

871

Guidance from a Previous Thinking Model: Turn $t_1 = 10$

872

Previous thinking process:

874

First, I need to get the player from $(0, 6)$ to $(0, 9)$ in as few turns as possible, avoiding collisions with cars on the freeways $y = 1$ to $y = 8$. The current turn is $t_1 = 10$.

876

The player can move up (U), down (D), or stay (S), changing y by $+1$, -1 , or 0 respectively, and y must stay between 0 and 9 inclusive.

879

Cars move on specific freeways, and I need to avoid being on a freeway when a car is at position $x = 0$ at that turn. The collision happens only if at turn T , the player is on freeway y_T , and for that freeway, there's a car where 0 is in its span at that exact turn.

880

Importantly, when I move to a new freeway, I'm not considered on that freeway until the next turn. So if I move from y to $y + 1$ at turn T , at turn T I'm still on y , and I only occupy $y + 1$ at turn $T + 1$. That means I don't collide with cars on $y + 1$ at turn T , only at turn $T + 1$ if I'm still there.

883

Now, the goal is to reach $y = 9$. Since we start at $y = 6$, and $y = 9$ is three levels up, but I have to go through $y = 7$ and $y = 8$, and avoid the cars there.

884

The minimal number of turns might be more than 3 because I might need to wait or something to avoid collisions.

886

 H is the number of turns, so from turn t_1 to $t_1 + H - 1$, so if I start at $t = 10$, and I take H turns, I end at turn $10 + H - 1$.

888

I need $y_{10+H-1} = 9$.

889

Since I start at $y = 6$, and I need to get to $y = 9$, which is $+3$, so minimum H would be 3 if I can go up three times without colliding, but I need to check if that's possible given the car positions.

891

But the cars are moving, so I need to see when they are at $x = 0$. First, let's understand the car movements.

892

The car spans are given at turn $t_1=10$.

893

For each car, I need to find when 0 is in its span for future turns.

894

The span changes linearly with time based on speed and direction.

895

Let me denote the time elapsed from $t_1=10$ as $\Delta t = T - 10$.

896

For a car, at turn T , span is:

897

If left-moving: $[h - s^* \Delta t, \tau - s^* \Delta t]$ since it moves left, decreasing x .

898

In the rule: for left-moving, $\text{Span}(T) = [h - s^*(T-t_1), \tau - s^*(T-t_1)]$

899

Similarly for right-moving: $\text{Span}(T) = [\tau + s^*(T-t_1), h + s^*(T-t_1)]$

900

And collision if 0 is in $\text{Span}(T)$ for the freeway we're on at turn T .

901

Now, since the player is at y , and cars are per freeway, I need for each freeway k , and for each car on that freeway, find the times when 0 is in its span.

902

Then, to avoid collision, when I'm on freeway k at turn T , none of the cars on k should have 0 in span at T .

903

Now, for the path to $y=9$.

904

Since $y=9$ is the top, and no cars above, but I need to pass through $y=7$ and $y=8$.

905

 $y=9$ has no cars since cars are on $y=1$ to 8 , so once I reach $y=9$, I'm safe.

906

But I need to get there.

907

Start at $y=6$ at $t=10$.

908

I need to go to $y=7$, then $y=8$, then $y=9$.

909

But I might stay or move down to avoid cars, but moving down might not help since I need to go up eventually.

910

Since the grid is small, y from 0 to 9, and I start at 6, going to 9, it's only 3 steps up, so H min is 3, but if not possible, I need more turns by staying or moving down and up.

911

... ...

918
919**Prompt for Code-As-Policy**

920

921

922

923

924

925

926

927

Prompt:

Now a player is playing a multi-turn game, and suppose current turn is $\{t_1\}$. Given the initial position $(0, y_{t_1})$ on a 2D grid (vertical axis $y = 0, 1, \dots, 9$). You are tasked with implementing a Python function that determines the optimal next action to avoid collisions with moving cars and progress towards the goal position $(0, 9)$ in minimal turns.

1. Game Dynamics:**• Player update:**

$y_{t+1} = y_t + \Delta y_t$, where

$$\Delta y_t = \begin{cases} +1 & \text{if } a_t = U \\ -1 & \text{if } a_t = D, \quad y_{t+1} \in [0, 9] \\ 0 & \text{if } a_t = S \end{cases}$$

• Car update rules:

For car k on freeway i , suppose its head is at h , tail is at τ at turn t_1 , and speed is s . Then at turn $T > t_1$, the car span becomes:

- Left-moving: $\text{Span}(t_1) = [h, \tau] \rightarrow \text{Span}(T) = [h - s(T - t_1), \tau - s(T - t_1)]$
- Right-moving: $\text{Span}(t_1) = [\tau, h] \rightarrow \text{Span}(T) = [\tau + s(T - t_1), h + s(T - t_1)]$

• Collision occurs at turn T only if $0 \in \text{Span}(T)$ for any car on freeway y_T .

- Note that if you decide to move to $y_{T+1} \neq y_T$ at turn T , you will **NOT** be considered to be on y_{T+1} at turn T , thus will **NOT** be collided by cars on y_{T+1} if $0 \in \text{Span}(T)$ but $0 \notin \text{Span}(T+1)$.

2. Task

You need to determine the best next action for the player by generating an **executable** Python function `next_action(json_state)` with the input `json_state` representing the current game state as a JSON object. The function should analyze the game state and return the next action, represented as a single character string:

- 'U' for moving up (to $y + 1$)
- 'D' for moving down (to $y - 1$)
- 'S' for staying in the current position

Notice that the code will be executed in a loop, so it should return the next action each time it is called with the current game state, which will **change after each action**.

Input Format

```

1 json_state = {
2     'player_states': current_y_position, # int: 0-9, 9 is the goal
3         position
4     'car_states': [ # list of tuples
5         (lane, head, direction, speed, span),
6         # lane: 1-8 (freeway number)
7         # head: int, position of the car's head
8         # direction: 'left' or 'right', tail = head + span if left,
9             head - span if right
10        # speed: int, speed of the car
11        # span: int, span of the car, defined as the absolute
12            difference between head and tail
13    ],
14    'turn': current_turn_number # int: current turn
15 }
```

972 **Output Format**

973

974

Generate two clearly labeled parts:

975

Part 1: Summary

976

One-sentence intent describing your strategy for the next actions

977

978

Part 2: Python Function

979

980

981

982

983

984

985

986

987

988

989

990

991

992

```

1  def next_action(json_state) -> str:
2      """
3          Returns one of the actions: 'U', 'D', or 'S' based on the
4          current game state.
5
6          Args:
7              json_state: The current game state as JSON object
8
9          Returns:
10             str: Single character representing the next action ('U', 'D'
11                 ', 'S')
12
13             # Your logic here
14             return action

```

993

994

995

Example Output

996

997

Part 1: Summary

998

999

Wait for the cars in lane 4 and 5 to pass, then move up to avoid collisions.

Part 2: Python Function

1000

1001

1002

```

1  def next_action(json_state) -> str:
2      # Implementation...
3      return 'S' # default action if no immediate threat

```

1003

1004

Current State

1005

1006

1007

1008

1009

1010

1011

1012

1013

1014

1015

1016

1017

1018

1019

1020

1021

1022

1023

1024

1025

```

1  {
2      "player_states": 5,
3      "car_states": [
4          (1, -39, 'right', 3, 11), (2, 48, 'right', 12, 11),
5          (2, -36, 'right', 12, 11), (2, 0, 'right', 12, 11),
6          (3, 48, 'right', 12, 11), (3, -36, 'right', 12, 11),
7          (3, 0, 'right', 12, 11), (4, 48, 'right', 12, 11),
8          (4, -36, 'right', 12, 11), (4, 0, 'right', 12, 11),
9          (5, -18, 'right', 6, 11), (6, -18, 'right', 6, 11),
10         (7, -16, 'right', 4, 11), (8, -16, 'right', 4, 11)
11     ],
12     "turn": 8
13 }

```

1026 **C ADDITIONAL EXPERIMENT RESULTS**
10271028 **C.1 MAIN RESULTS**
10291030
1031 Table 6: Complete agent performance across various cognitive load levels (Easy, Medium, Hard)
1032 with time pressure fixed at 8k tokens/step.
1033

Cognitive Load	Code-as-Policy	Reactive (R1)	Reactive (V3)	Planning (R1)	AgileThinker
Easy	0.5393	0.2022	0.9775	0.3371	0.9551
Medium	0.6966	0.0000	0.3258	0.2247	0.8427
Hard	0.5281	0.0000	0.0562	0.1011	0.5056

1034 **Freeway**
1035

Cognitive Load	Code-as-Policy	Reactive (R1)	Reactive (V3)	Planning (R1)	AgileThinker
Easy	0.1719	0.1719	0.7694	0.0588	0.6931
Medium	0.1797	0.2188	0.4900	0.0256	0.5413
Hard	0.0625	0.1406	0.2950	0.0137	0.3906

1036 **Snake**
1037

Cognitive Load	Code-as-Policy	Reactive (R1)	Reactive (V3)	Planning (R1)	AgileThinker
Easy	0.4621	0.8193	0.9188	0.5379	1.0000
Medium	0.3724	0.6607	0.3664	0.0871	0.9152
Hard	0.3661	0.2054	0.0877	0.0000	0.5982

1038 **Overcooked**
1039

Cognitive Load	Code-as-Policy	Reactive (R1)	Reactive (V3)	Planning (R1)	AgileThinker
Easy	0.4621	0.8193	0.9188	0.5379	1.0000
Medium	0.3724	0.6607	0.3664	0.0871	0.9152
Hard	0.3661	0.2054	0.0877	0.0000	0.5982

1040
1041 Table 7: Complete agent performance across time pressure levels (4k to 32k tokens/step) with cognitive load fixed at Medium.
10421043 **Freeway**
1044

Tokens/Turn	Code-as-Policy	Reactive (R1)	Reactive (V3)	Planning (R1)	AgileThinker
32k	0.9438	0.0000	0.2911	0.9621	0.9431
16k	0.9551	0.0000	0.2911	0.9045	0.9347
8k	0.6966	0.0000	0.2911	0.2261	0.8469
4k	0.3483	0.0000	0.2911	0.1194	0.6166

1045 **Snake**
1046

Tokens/Turn	Code-as-Policy	Reactive (R1)	Reactive (V3)	Planning (R1)	AgileThinker
32k	0.2109	0.0238	0.4844	0.9629	0.8281
16k	0.2344	0.0238	0.4844	0.4043	0.7813
8k	0.1797	0.0238	0.4844	0.0254	0.5410
4k	0.0156	0.0238	0.4844	0.0176	0.4238

1047 **Overcooked**
1048

Tokens/Turn	Code-as-Policy	Reactive (R1)	Reactive (V3)	Planning (R1)	AgileThinker
32k	0.5804	0.0000	0.3800	0.8371	0.9129
16k	0.4621	0.0000	0.3800	1.0000	0.9375
8k	0.3661	0.0000	0.3800	0.0871	0.9152
4k	0.3724	0.0000	0.3800	0.0246	0.7087

1080
1081

C.2 SIGNIFICANCE TEST

1082
1083
1084
1085

We investigate the significance of advantage of AgileThinker over single-paradigm agents. We hypothesize that: (1) AgileThinker’s advantage over reactive agents (V3) becomes more significant as cognitive load increases, and (2) its advantage over planning agents (R1) becomes more significant as time pressure increases.

1086
1087
1088
1089

To validate these hypotheses, we perform experiments across 3 cognitive load levels (Easy, Medium, Hard) and three 3 pressures (High: 32k tokens/step, Medium: 8k tokens/step, Low: 4k tokens/step). Below, we formally describe the validation procedure (1) and (2) can be tested similarly.

1090
1091
1092
1093
1094

For each fixed cognitive load and time pressure condition, we analyze the paired score differences between AgileThinker and the reactive agent. Let μ_d denote the mean score difference between the paired observations (AgileThinker minus reactive agent). We pair observations that share the same environmental configuration seed, with each score calculated as the average across experimental runs within that configuration seed. We formulate the hypotheses as follows:

1095
1096
1097
1098
1099

- **Null Hypothesis:** $H_0 : \mu_d = 0$
- **Alternative Hypothesis** $H_1 : \mu_d > 0$

The test statistic for the paired t-test is calculated as:

1100
1101
1102
1103

$$t = \frac{\bar{d}}{s_d / \sqrt{n}}$$

1104
1105
1106

where \bar{d} is the mean score difference, s_d is the standard deviation of the differences, and n is number of environment configuration seeds. We estimate the p-value based on the t-statistic with $n - 1$ degrees of freedom.

1107
1108
1109
1110
1111

Figure 8 presents the p-values across different conditions, with statistical significance assessed at $\alpha = 0.05$. The results show that AgileThinker’s advantage generally becomes statistically significant as cognitive load and time pressure increase.

1130
1131
1132
1133

Figure 8: Statistical significance of AgileThinker’s advantage over single-paradigm agents. Upper: Advantage over reactive agent (V3). Lower: Advantage over planning agent (R1). Numbers represent p-values under varying cognitive loads and time pressures, with red indicating statistical significance ($p < 0.05$). The advantage of AgileThinker generally increases with both cognitive load and time pressure.

1134 Table 8: DeepSeek-V3.2 based agents’ performance across different time pressure levels (4k to 32k
 1135 tokens/step) with cognitive load fixed at High.

Freeway			
Tokens/Turn	Reactive (thinking off)	Planning (thinking on)	AgileThinker
32k	0.4659	0.8539	0.8889
16k	0.4659	0.4649	0.8328
8k	0.4659	0.1025	0.6352
4k	0.4659	0.0000	0.4466

Snake			
Tokens/Turn	Reactive (thinking off)	Planning (thinking on)	AgileThinker
32k	0.2556	0.7381	0.5700
16k	0.2556	0.3906	0.5350
8k	0.2556	0.0550	0.3456
4k	0.2556	0.0075	0.2738

Overcooked			
Tokens/Turn	Reactive (thinking off)	Planning (thinking on)	AgileThinker
32k	0.5246	0.8371	0.8550
16k	0.5246	0.4554	0.8438
8k	0.5246	0.0134	0.6563
4k	0.5246	0.0068	0.4486

1161 Table 9: DeepSeek-V3.2 based agents’ performance with time pressure of 350 seconds per step
 1162 and high cognitive load (Hard difficulty), confirming AgileThinker’s advantage persists in wall-time
 1163 constraints.

Environment	Reactive (thinking off)	Planning (thinking on)	AgileThinker
Freeway	0.3665	0.0	0.7022
Snake	0.3050	0.0863	0.6013
Overcooked	0.5246	0.0	0.6675

C.3 RESULTS OF OTHER MODELS

1174 Since AgileThinker relies on transparent reasoning trajectories, which are only available in open-
 1175 source models, our primary experiments are conducted using the state-of-the-art open-source
 1176 DeepSeek models. Besides using DeepSeek-V3 and DeepSeek-R1 as the reactive and planning
 1177 model, we also conduct experiments with DeepSeek-V3.2, where non-thinking mode is used as
 1178 reactive and thinking mode is used as planning. The results are shown in Tab. 8 and Tab. 9.

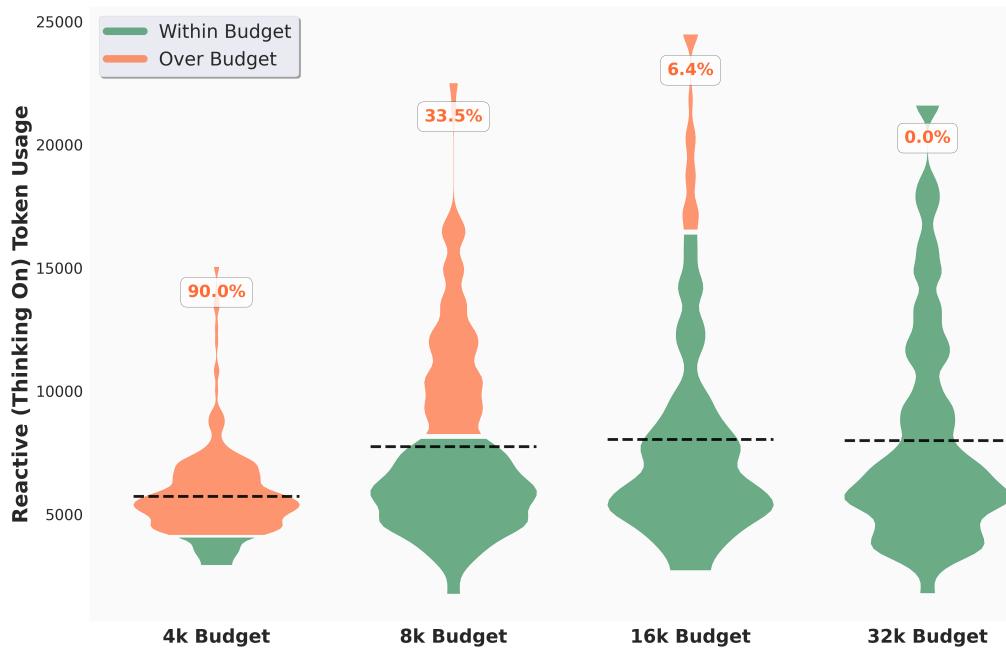
1180 To assess the generalizability of our approach, we also evaluate reactive and planning agents using
 1181 other models, such as Gemini-2.5-Flash, which features an intrinsic budget control function. For
 1182 reactive agents, we disable extended thinking, while for planning agents, thinking is enabled.

1183 However, we cannot directly implement AgileThinker with Gemini due to the lack of access to its
 1184 reasoning traces. Instead, we approximate the system using a reduced design: the reactive agent
 1185 references the *final* output of the planning agent after it completes reasoning, rather than accessing
 1186 *partial* reasoning traces during streaming. Despite this limitation, the results in Tab. 10 consistently
 1187 show that combining reactive and planning paradigms improves performance across all levels of
 time pressure.

1188 Additionally, we observe that Gemini-2.5-Flash’s built-in budget control struggles to precisely regulate token usage, often exceeding the allocated budget (Figure 9). This highlights the ongoing challenge that LLMs face in controlling computational costs during real-time reasoning.
 1189
 1190
 1191

1192 **Table 10: Performance of Gemini-2.5-Flash (Medium-difficulty Freeway) under various agent
 1193 designs.** We implement budget control by setting the thinking budget equal to the time pressure
 1194 budgets. Since Gemini-2.5-Flash’s internal reasoning traces are not public, we cannot apply Agile-
 1195 Thinker directly. Instead, we reduce Reactive + Planning to allowing reactive thread to reference the
 1196 *final* non-thinking tokens produced by planning thread after its reasoning is completed.
 1197

Tokens/Step	Reactive (thinking off)	Reactive (thinking on + budget control)	Planning (thinking on)	Reactive+ Planning
32k	0.12	0.93	0.93	0.92
16k	0.12	0.76	0.70	0.70
8k	0.12	0.09	0.25	0.31
4k	0.12	0.00	0.05	0.26



1227 Figure 9: To test the effectiveness of thinking budget control, we set the thinking budget of Gemini-
 1228 2.5-Flash at 4k, 8k, 16k and 32k to investigate the distribution of response length. The distribution
 1229 shows that even Gemini-2.5-Flash is aware of the budget, it cannot precisely regulate response token
 1230 count, often generating excessive tokens.
 1231
 1232

C.4 CODE-AS-POLICY

1234 We analyze generated code to understand the limitations of code-as-policy approaches. Below are
 1235 representative examples from each environment. All code is directly generated by DeepSeek-R1
 1236 with some helper functions omitted for brevity.
 1237

FREEWAY

```
1239 1  def next_action(json_state) -> str: # Extract game state # Helper
1240 2  functions
1241 2  ... t0, y0, cars = extract_game_state(json_state) max_time = t0 + 50
1242 2  # Reasonable
```

```

1242 3     time horizon visited = set() queue = deque() start = (t0, y0, []) # (
1243 4             current_turn,
1244 5             y, path_actions) visited.add((t0, y0)) queue.append(start)
1245 6
1246 7     while queue: # BFS path finding ... # BFS found no path: choose
1247 8             safest
1248 9             immediate next action ...
1249
1250
1251
1252
1253
1254

```

Freeway has a simple, efficient solution: breadth-first search (BFS) to explore all possible movement sequences across lanes, and the state space is very small (time \times player position). Therefore, LLMs can implement naive search strategies to get optimal performance.

SNAKE

```

1255 1     def next_action(json_state) -> str: # Extract game state # Helper
1256 2             functions
1257 3             ... # Main logic: choose best move for current state safe_moves =
1258 4                     generate_safe_moves(json_state)
1259 5             if not safe_moves: return 'U' # no safe move
1260 6
1261 7             best_move = None best_score = -float('inf') depth = 5 # Limited depth
1262 8                     search
1263 9
1264 10            for move in safe_moves: next_state, immediate_reward = simulate_move(
1265 11                    json_state,
1266 12                    move) future_reward = search(next_state, depth - 1) total_reward =
1267 13                    immediate_reward
1268 14                    + future_reward if total_reward > best_score: best_score =
1269 15                    total_reward
1270 16            best_move = move
1271
1272 17            return best_move
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295

```

Snake does not have an algorithmic solution with bounded complexity, forcing LLMs to resort to heuristic approaches. The generated code typically implements limited-depth search (5 turns in this example) to evaluate immediate rewards, but this constrained horizon proves insufficient for complex scenarios. Long-term consequences - such as distant food opportunities or collision risks that emerge beyond the search depth - remain invisible to the policy.

OVERCOOKED

```

1279 1     def next_action(json_state: dict) -> str: # Extract game state # Helper
1280 2             functions
1281 3             ... alice_held_obj = alice.get('held_object')
1282 4
1283 5             # Decision logic if alice_held_obj and alice_held_obj.get('name') ==
1284 6                     'onion':
1285 7             # Find available pots and add ingredients ...
1286 8
1287 9             elif alice_held_obj and alice_held_obj.get('name') == 'plate': # Pick
1288 10                     up
1289 11             ready_soup ... elif not alice_held_obj: # Priority 1: Get plate for
1290 12                     ready
1291 13             soups ... # Priority 2: Get ingredients for cooking ...
1292
1293 1     def next_action(json_state: dict) -> str: if alice_held_obj.get('name'
1294 2                     ) == 'onion':
1295 3             pot = find_pot() add_ingredient(pot) elif alice_held_obj.get('name')
1296 4                     == 'plate':
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1599

```

```

1296 3     soup = find_ready_soup() pickup_ready_soup() elif not alice_held_obj:
1297 4             # Priority
1298 4     1: Get plate for ready soups # Priority 2: Get ingredients for
1299 4             cooking ... return
1300 5             'S' # Stay as default
1301
1302
1303

```

Overcooked has complex game context, demanding that players infer partner intentions through action history and plan coordinated responses based on current state. However, the generated code considers only limited context (e.g., what Alice is currently holding). This narrow focus leads to suboptimal decisions. For instance, when the agent holds an onion but all pots are occupied (perhaps by Bob), it simply stays idle rather than dropping the onion to pick up a plate and serve ready soup.

```

1309
1310 1     def next_action(json_state: dict) -> str: # Extract game state #
1311 1             Helper functions
1312 2             ...
1313 3
1314 4     pots = [tuple(pos) for pos in layout['Pot']] ready_soups = [obj for
1315 5             obj in objects
1316 5             if obj.get('name') == 'soup' and obj.get('is_ready', False)]
1317 6             cooking_soups =
1318 6             [obj for obj in objects if obj.get('name') == 'soup' and obj.get(
1319 7                 'is_cooking',
1319 7                 False)]
1320 8
1321 9     alice_held_obj = alice.get('held_object')
1322 10
1323 11     # Priority 1: Serve ready soup if ready_soups: if alice_held_obj and
1323 11             alice_held_obj.get('name')
1324 12             == 'soup': # Deliver soup to serving counter ... elif alice_held_obj
1324 12             and
1325 13             alice_held_obj.get('name') == 'plate': # Pick up ready soup from pot
1325 13             ... else:
1326 14             # Get plate from dispenser ...
1327 15
1328 16
1329 16     # Priority 2: Handle cooking soups elif cooking_soups: soup =
1329 16             cooking_soups[0]
1330 17     if soup['remaining_cooking_tick'] <= 1 and not alice_held_obj: # Get
1331 17             plate for
1332 18             soon-to-be-ready soup ... # Priority 3: Start cooking (add
1333 18             ingredients to empty
1334 19             pot) idle_pot = next((pot for pot in pots if not any(obj.get('
1334 19             position') == list(pot)
1335 20             for obj in objects)), None) if idle_pot: if held_obj and held_obj.get(
1336 20             ('name')
1337 21             == 'onion': # Add ingredient to pot ... else: # Get ingredient from
1337 21             dispenser
1338 22             ...
1339 23
1340 24     return 'S' # Stay as default
1341
1342
1343

```

Beyond ignoring broad game context, the generated code exhibits inconsistent goal prioritization that leads to oscillatory behavior. Consider a scenario where Alice holds an onion and ready soup is available: the first code prioritizes adding the onion to an empty pot, while the second prioritizes serving the ready soup. These conflicting objectives cause the agent to alternate between incompatible actions. This demonstrates a fundamental limitation of code-as-policy approaches: while they can encode local heuristics effectively, they struggle to maintain coherent long-term strategies in environments requiring dynamic coordination and context-aware decision making.

C.5 PERFORMANCE UNDER LIMITED THROUGHPUT

Game	Parallel Threads	Concurrent Threads	Reactive (V3)	Planning (R1)
Freeway	0.84	0.72	0.32	0.22
Snake	0.54	0.58	0.49	0.03
Overcooked	0.92	0.85	0.37	0.09

Table 11: Performance of AgileThinker with abundant (parallel threads) and limited (concurrent threads) throughput.

While our main experiments assume parallel execution with independent throughput for each subsystem, practical deployments may face resource constraints. In this section, we investigate whether AgileThinker remains effective when LLM and LRM share computational resources through concurrent switching rather than parallelism. Specifically, we evaluate the performance of agent systems under equivalent token throughput, implementing the AgileThinker as alternating inference between reactive and planning thread (concurrent generation) rather than simultaneous generation.

Figure 11 compares concurrent AgileThinker against parallel AgileThinker and single-model baselines. Even with equivalent throughput, concurrent AgileThinker significantly outperform both reactive agents and planning agents. While parallel execution offers modest performance improvements over concurrent execution, the gains are relatively small compared to the fundamental advantage of AgileThinker. This indicates that the primary benefit stems from cognitive specialization rather than computational resources, and AgileThinker remain effective for resource-constrained deployments.

D WALLTIME EXPERIMENTS

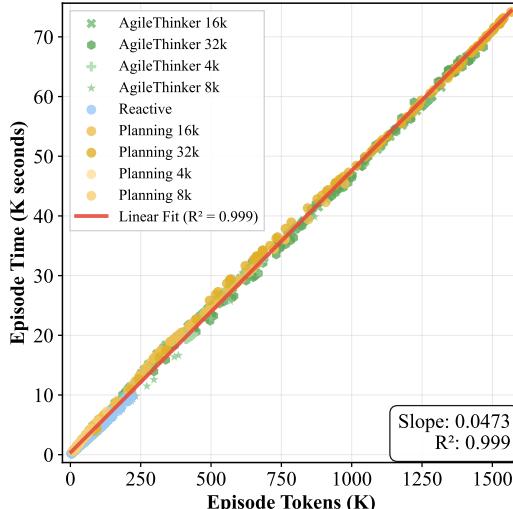


Figure 10: Almost linear correlation between generated token count and wall-clock time using DeepSeek official API, demonstrating the validity of our token-as-time abstraction. Here the numbers after agent methods, e.g. 4k, 8k, refer to the corresponding environment time pressure budgets.

E DYNAMIC ADJUSTMENT ALGORITHM FOR INTERNAL BUDGET

Inspired by the AIMD algorithm², we conduct additional experiments to show that thread resource can be allocated adaptively through a simple dynamic adjustment mechanism for N_{T_n} . Specifically,

²https://en.wikipedia.org/wiki/Additive_increase/multiplicative_decrease

1404 we initialize N_{T_R} to a small value (1k tokens). If the reactive thread fails to generate a complete
 1405 response within the allocated tokens, we increase N_{T_R} by 1k tokens. Conversely, if the reactive
 1406 thread uses fewer than tokens for 3 consecutive responses, we decrease N_{T_R} by 0.5k tokens. This
 1407 approach eliminates the need to predefine a fixed N_{T_R} , allowing the model to adapt it based on
 1408 query complexity.

1409 In fact, the advantage of AgileThinker is still significant with this adaptive algorithm, as we can see
 1410 in Tab. 12. The experiments are conducted on DeepSeek-V3.2 under **Hard** cognitive load and **8k**
 1411 **tokens/step** time pressure.

Algorithm 1 Dynamic Budget Adjustment

```

1:  $N_{T_R} \leftarrow 1024$ , efficient_streak  $\leftarrow 0$ 
2: while game not terminates do
3:   Execute AgileThinker with internal budget  $N_{T_R}$ 
4:   if reactive thread exceeds budget then
5:      $N_{T_R} \leftarrow N_{T_R} + 1024$ 
6:     efficient_streak  $\leftarrow 0$ 
7:   else
8:     efficient_streak  $\leftarrow$  efficient_streak + 1
9:   if efficient_streak  $\geq 3$  then
10:     $N_{T_R} \leftarrow \max(1024, N_{T_R} - 512)$ 
11:    efficient_streak  $\leftarrow 0$ 
12:  end if
13: end if
14: end while

```

Environment	Reactive (Non-thinking)	Planning (Thinking)	AgileThinker (fixed, optimal N_{T_R})	AgileThinker (dynamic N_{T_R})
Freeway	0.4659	0.1025	0.6352 (3k)	0.6279
Snake	0.2556	0.0550	0.3456 (1k)	0.4166
Overcooked	0.5246	0.0134	0.6563 (2k)	0.6741

1435 Table 12: Scores of AgileThinker under dynamic internal budget.
 1436

1437
 1438
 1439
 1440
 1441
 1442
 1443
 1444
 1445
 1446
 1447
 1448
 1449
 1450
 1451
 1452
 1453
 1454
 1455
 1456
 1457