WHEN REASONING FAILS TO STOP: PROMPT-INDUCED RESOURCE EXHAUSTION IN BLACK-BOX REASONING MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reasoning large language models allocate additional inference-time computation to improve multi-step reasoning and complex problem solving. Existing studies have demonstrated excessive computation and high resource consumption by constructing or optimizing inputs. However, attack effectiveness alone does not explain the task conditions under which resource exhaustion occurs. We therefore investigate which prompt conditions induce continued computation and resource exhaustion in a black-box setting limited to normal question-answering requests. We construct Reasoning-Bench with five task types and three task-complexity levels, evaluating 400 prompts on 10 reasoning models to obtain 4,000 runs. Resource statistics, execution-behavior analysis, and condition ablations yield six findings: task complexity cannot directly determine resource consumption; code, cryptography, and algorithmic-reasoning tasks tend to exhibit higher resource consumption; models tend to execute first and judge later when tasks are declared solvable; continuously decomposable tasks encourage local exploration; missing or unverifiable information encourages multiple hypotheses and continued reasoning; and detailed-process requirements further increase resource consumption. To test whether these findings guide resource-exhaustion task construction, we propose ReasoningSink, a resource-exhaustion prompt generation framework for black-box reasoning models. The framework defines prompt conditions and model-behavior checks through State Constraints, organizes tasks through Dynamic Retrieval-Augmented Generation, and revises candidates through Execution-Feedback Optimization. Across the 10 reasoning models, ReasoningSink produces averages of 35.4K Completion Tokens (CT) and 239.8× output-to-input Amplification Ratio (Amp). For every evaluated model, ReasoningSink achieves higher CT and higher Maximum-Budget Hits (MaxH) than the comparison methods. Reasoning text from ReasoningSink-generated tasks again exhibits executing before judging, local exploration, hypothesis-space expansion, and detailed-process elaboration, validating the findings' guidance for resource-exhaustion task construction. Code, dataset, and evaluation scripts are available at Reasoning-Bench-and-ReasoningSink.[1]

1 INTRODUCTION

Reasoning large language models (RLMs) improve performance on tasks such as mathematical reasoning, code generation, and complex problem solving by allocating additional computation at inference time, supporting multi-step deduction, verification, and self-correction (OpenAI, 2024; Guo et al., 2025; Snell et al., 2025). However, this mechanism of trading inference computation for task performance also offers attackers an opportunity to induce excessive resource consumption (OWASP GenAI Security Project, 2023). As illustrated in Figure 1, a carefully crafted short prompt can drive an RLM to reason

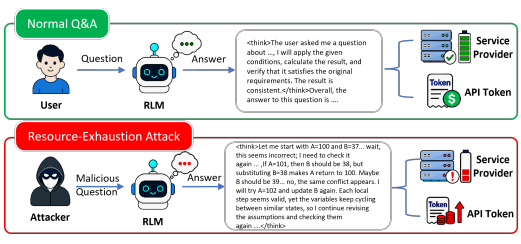

Figure 1: Normal question-answering requests can induce resource exhaustion in black-box RLMs.

[1]Anonymous repository: Reasoning-Bench-and-ReasoningSink.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

continuously until it approaches its reasoning-token budget, times out, or is terminated by the server. For model service providers, repeated submission of such requests can continuously occupy inference compute and concurrent-serving capacity, degrade service availability, and even cause denial of service. For users who bear the cost of each request, excessive reasoning translates into longer waiting times, greater token consumption, and higher API costs.

Existing studies have demonstrated the risk of excessive computation and high resource consumption in large language models by constructing or optimizing inputs (Dong et al., 2025; Si et al., 2025; Kumar et al., 2025). These results show that model computation can be influenced by the input, However, attack effectiveness alone does not explain the task conditions under which resource exhaustion occurs. During normal question answering, a prompt specifies both the task content that the model must process and the requirements for solving it, and different requests may induce different computational processes and termination decisions. We therefore focus on a specific question: In a black-box setting where an attacker can submit only requests presented in a normal question-answering format, which task conditions in a prompt can induce a reasoning model to continue investing computation and consequently cause resource exhaustion?

To investigate this question, we examine the relationship between task conditions, model execution behaviors, and resource consumption using prompts that an attacker can directly control in the black-box setting. We evaluate 10 reasoning large language models on the 400 prompts in Reasoning-Bench. Based on experimental evaluation, model execution-behavior analysis, and condition ablations, we summarize six findings: (1) task complexity cannot directly determine the level of resource consumption; (2) code, cryptography, and algorithmic-reasoning tasks tend to exhibit higher resource consumption; (3) when a task is declared solvable, reasoning models tend to execute first and judge later; (4) when facing tasks that can be continuously decomposed, reasoning models tend to become engaged in local exploration; (5) when facing missing or unverifiable information, reasoning models tend to introduce multiple assumptions and continue reasoning; and (6) when facing detailed-process requirements, reasoning models further increase resource consumption.

To examine whether these findings can guide the construction of resource-exhaustion tasks, we propose ReasoningSink, a resource-exhaustion prompt generation framework for black-box reasoning models. The framework contains three modules. State constraints convert the six findings into task conditions on the prompt side and behavior checks on the model side. Dynamic retrieval-augmented generation retrieves three semantically relevant cases from Reasoning-Bench according to the input task type and solution objective, helping organize concrete task conditions. Execution-feedback optimization submits candidate prompts to the target model, checks whether the conditions take effect based on reasoning text and resource consumption, and revises candidates whose expected behaviors do not appear. We use ReasoningSink to generate 100 new prompts and obtain 1,000 evaluation records on 10 reasoning models. The results show that the generated prompts produce an average of 35.4K completion tokens and an average output-to-input amplification ratio of $239.8\times$, induce high resource consumption across different models, and substantially outperform existing methods. Reasoning text from the new tasks again exhibits executing before judging, local exploration, hypothesis-space expansion, and detailed-process elaboration, validating that the six findings can guide the construction of resource-exhaustion tasks.

Our main contributions are as follows:

- Benchmark. We propose Reasoning-Bench, an evaluation benchmark for studying resource consumption in black-box reasoning models. It is designed based on three principles: attack stealthiness, multiple task types, and multiple difficulty levels. Reasoning-Bench contains 400 prompts presented in a normal question-answering format, covering five task types (mathematics, logic, cryptography, code, and algorithmic reasoning), 24 subcategories, 88 leaf scenarios, and three task-complexity levels: Low, Medium, and High.
- Findings. We evaluate 10 reasoning models on Reasoning-Bench. Based on experimental evaluation, model execution-behavior analysis, and condition ablations, we summarize six findings: (1) task complexity cannot directly determine the level of resource consumption; (2) code, cryptography, and algorithmic-reasoning tasks tend to exhibit higher resource consumption; (3) when a task is declared solvable, reasoning models tend to execute first and judge later; (4) when facing tasks that can be continuously decomposed, reasoning models tend to become engaged in local exploration; (5) when facing missing or unverifiable information, reasoning models tend to introduce multiple assumptions and continue reasoning; and (6) when facing detailed-process requirements, reasoning models further increase resource consumption.

- Method. We propose ReasoningSink, a resource-exhaustion prompt generation framework for black-box reasoning models. It organizes prompt conditions through state constraints, instantiates task conditions through dynamic retrieval-augmented generation, and checks model behaviors and resource-consumption results through execution-feedback optimization to revise candidate prompts.
- Experimental evaluation. We use ReasoningSink to generate 100 new prompts and evaluate them on 10 reasoning models. Across the 10 reasoning models, ReasoningSink produces averages of 35.4K CT and $239.8\times$ Amp. For every evaluated model, ReasoningSink achieves higher CT and higher MaxH than the comparison methods. Reasoning text from ReasoningSink-generated tasks again exhibits executing before judging, local exploration, hypothesis-space expansion, and detailed-process elaboration, validating the findings' guidance for resource-exhaustion task construction.

2 REASONING-BENCH

This section introduces Reasoning-Bench, an evaluation benchmark for studying resource consumption in black-box reasoning models. Existing evaluations such as GPQA and OlympiadBench measure advanced question answering and olympiad-level scientific problem solving Rein et al. (2024); He et al. (2024). We examine three aspects of a prompt: request format, task type, and task features and constraints. Related symbolic-reasoning benchmarks such as SymbolBench Liu et al. (2025) also evaluate structured reasoning, while our benchmark focuses on prompt conditions associated with inference-time resource consumption. Based on these aspects, we make three basic judgments about whether the model enters the answering process, the reasoning capabilities invoked by each task type, and how task conditions determine task complexity. These judgments motivate three design principles: attack stealthiness, multiple task types, and multiple difficulty levels. We then describe three construction stages: task-hierarchy decomposition, prompt generation, and manual screening.

2.1 DESIGN PRINCIPLES

We make three basic judgments: whether the model enters the answering process, the reasoning capabilities associated with different task types, and how task conditions determine task complexity. These judgments lead to three design principles: attack stealthiness, multiple task types, and multiple difficulty levels.

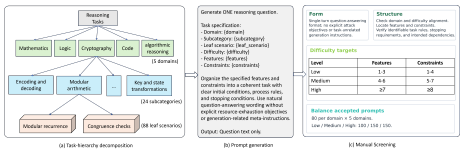

Figure 2: Reasoning-Bench construction stages.

(1)Attack stealthiness. Resource exhaustion can occur only after the model recognizes a prompt as a normal task and enters the answering process. Prompts should therefore avoid an evident attack intent. The attack-stealthiness principle requires each prompt to use a normal single-turn question-answering format and to exclude explicit attack objectives and task-irrelevant generation instructions. This format reduces the chance that the model refuses the request or stops before answering.**(2)Multiple task types.** Different task types invoke different reasoning capabilities. Resource exhaustion may therefore depend on task type. The multiple-task-types principle covers mathematics, logic, cryptography, code, and algorithmic reasoning. These task types are organized into 24 subcategories and 88 leaf scenarios. The hierarchy supports comparisons across task types and distinguishes process structures and constraint combinations within each type. **(3)Multiple difficulty levels.** A prompt with more task features, constraints, and reasoning steps may require a longer reasoning process. The multiple-difficulty principle divides prompts into Low, Medium, and High task-complexity levels according to the numbers of task features and constraints. These levels support comparisons across different task-complexity configurations.

2.2 CONSTRUCTION PROCESS

As shown in Figure 2, we construct Reasoning-Bench in three stages. First, we organize task types into subcategories and leaf scenarios. Then, we generate candidate prompts from each leaf scenario according to the target task-complexity requirements. Finally, we manually screen the candidates against the three design principles and balance the sample distribution.

(1)Task-hierarchy decomposition. Following the multiple-task-types principle, we establish a three-level hierarchy of task types, subcategories, and leaf scenarios. The hierarchy covers mathematics,

logic, cryptography, code, and algorithmic reasoning, and contains 24 subcategories and 88 leaf scenarios. Subcategories distinguish task structures within each task type. Leaf scenarios specify task objects, process structures, and constraint relationships. As shown in Figure 2(a), cryptography tasks include encoding and decoding, modular arithmetic, and key and state transformations. Modular arithmetic further includes leaf scenarios such as modular recurrence and congruence checking. One leaf scenario can produce prompts with different surface expressions while preserving the same task structure and task-complexity configuration. **(2)Prompt generation.** For each leaf scenario, we organize the task type, subcategory, task features, constraints, and solution objective, and configure the numbers of features and constraints according to the target task-complexity level. The model uses this information to generate a prompt in a normal question-answering format. The generation process covers task content, reasoning process, and answer requirements. Task content specifies the initial conditions, task type, and solution objective. The reasoning process specifies the task rules and constraints. The answer requirements specify the final result and whether the model should show intermediate computations and verification. The generated prompt must remain consistent with the leaf-scenario structure and the target task-complexity level, including the task type and subcategory, the relationship between features and constraints, and the configured numbers of features and constraints. It must also follow attack stealthiness and keep all content relevant to the solution objective. Figure 2(b) shows the main components of the prompt-generation instruction. **(3)Manual screening.** As shown in Figure 2(c), we screen candidate prompts by form, structure, and distribution. For form, we check the normal single-turn question-answering format, the solution objective, and the absence of an evident attack intent. For structure, we check consistency with the task type, subcategory, and leaf scenario, the numbers of features and constraints, and the clarity of task rules, state changes, and stopping conditions. For distribution, we check the numbers of samples across task types, subcategories, and task-complexity levels. A candidate is retained, revised and rechecked, or removed. The final benchmark contains 400 prompts, with 80 prompts for each task type and 100, 150, and 150 prompts at Low, Medium, and High task complexity, respectively.

3 Main Results on Reasoning-Bench

3.1 Experimental Setup

Datasets and RLMs. We evaluate 10 reasoning large language models on all 400 prompts in Reasoning-Bench. The benchmark covers five task types, with 80 prompts for each type. The Low, Medium, and High task-complexity levels contain 100, 150, and 150 prompts, respectively. The evaluated models are provided by DeepSeek, Moonshot AI, Zhipu AI, Alibaba, Baidu, and ByteDance, including open-weight models and closed-source service models. Their maximum reasoning-token budgets range from 32,768 to 262,144 tokens.

Experimental details. The evaluation program runs on an Ubuntu 22.04.5 LTS node equipped with an NVIDIA H200 GPU. We access the models through remote APIs, while the local program schedules requests, collects responses, and records logs. Each prompt is submitted as an independent question-answering request. We record input tokens, reasoning tokens, final-answer tokens, reasoning text, and request termination status.

Evaluation metrics. We use three metrics to measure prompt-induced resource consumption and resource exhaustion. Completion Tokens (CT) measure the total number of tokens generated by the model. For prompt p, we define $\mathrm{CT}(p) = L_{\mathrm{reason}}(p) + L_{\mathrm{answer}}(p)$, where $L_{\mathrm{reason}}(p)$ and $L_{\mathrm{answer}}(p)$ denote the numbers of reasoning tokens and final-answer tokens, respectively. The Amplification Ratio (Amp) measures the output generated per input token and is defined as $\mathrm{Amp}(p) = \mathrm{CT}(p)/L_{\mathrm{in}}(p)$, where $L_{\mathrm{in}}(p)$ denotes the number of input tokens. For model m, we use its configured maximum reasoning-token budget M_m as the model-specific resource-exhaustion threshold, and a run is considered to reach resource exhaustion when $\mathrm{CT}_m(p_i) \geq M_m$. Maximum-Budget Hits at threshold $n\%$, denoted by MaxH@$n\%$, counts the number of runs whose CT reaches at least $n\%$ of M_m: $\mathrm{MaxH@}n\%_{m,G} = \sum_{p_i \in G} \mathbb{I}\big[\mathrm{CT}_m(p_i) \geq \frac{n}{100} M_m\big]$, where G denotes a task group. MaxH@90% counts runs satisfying $\mathrm{CT}_m(p_i) \geq 0.9M_m$, while MaxH@100% counts runs satisfying $\mathrm{CT}_m(p_i) \geq M_m$ and therefore reaching the resource-exhaustion criterion. A bare MaxH refers to MaxH@100%.

3.2 Main Results and Findings

We evaluate 10 reasoning large language models on 400 prompts in Reasoning-Bench. Figure 3 summarizes CT, Amp, and MaxH across task types and task-complexity levels, while Table 1 summarizes four prompt conditions frequently observed in high-resource-consumption runs. Based on the resource-consumption results of the 10 models, we select Qwen3.7-Plus, DeepSeek-V4-Pro, Kimi-K2.6, GLM-4.5, and DeepSeek-R1-7B for condition ablations because they exhibit higher resource consumption. Tables 2–5 report the ablation results after removing the corresponding prompt conditions, allowing us to compare the changes in CT, Amp, and MaxH@90% for each target condition. Combining resource statistics with observable reasoning text and condition ablations, we obtain the following six findings.

Finding 1: Task complexity cannot directly determine the level of resource consumption. As shown in Figure 3, resource consumption increases with task complexity in mathematics and cryptography, whereas it decreases in some complexity intervals for other task types. These results show that increasing task complexity can either increase or decrease resource consumption. Therefore, task complexity cannot directly determine the level of resource consumption. Increasing the numbers of task features and constraints does not cause the model to invest more computation or produce a longer output.

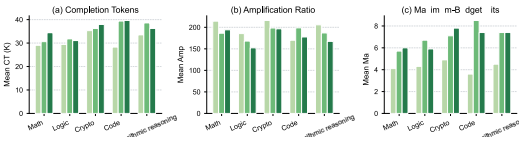

Figure 3: Average CT, Amp, and MaxH across task type and complexity. Light/medium/dark green = Low/Medium/High.

Finding 2: Code, cryptography, and algorithmic-reasoning tasks tend to exhibit higher resource consumption. At the Medium and High task-complexity levels, code, cryptography, and algorithmic-reasoning tasks generally consume more resources than mathematics and logic tasks. Their mean CT values all exceed 36.2K tokens. Program loops and variable updates in code tasks, modular arithmetic and recurrence checks in cryptography tasks, and multi-stage state transformations and condition checks in algorithmic-reasoning tasks require more computation and produce longer outputs. These results show that task type is closely related to resource consumption. Code, cryptography, and algorithmic-reasoning tasks tend to exhibit higher resource consumption.

Table 1: Prompt conditions in 1,175 high-consumption runs ($CT \geq 0.9M_m$) from 374 distinct prompts; M_m is model m's configured reasoning budget. P: prompts. The four conditions may co-occur within a prompt or run.

Prompt condition	P	CT ↑	Amp ↑
Solvability assertions	286	56,552	289.3
Continuously decomposable task conditions	148	**57,254**	**306.4**
Missing / unverifiable conditions	285	55,691	299.1
Detailed-process requirements	330	56,296	296.2

Finding 3: When a task is declared solvable, reasoning models tend to "execute first and judge later." As shown in Table 1, 286 of the 400 Reasoning-Bench prompts contain solvability assertions. After removing assertions such as "an answer exists" and "the conditions are correct" from the prompts, both CT and MaxH@90% decrease for all five models, as shown in Table 2. For DeepSeek-V4-Pro, mean CT decreases from 51,608 tokens to 33,463 tokens, and MaxH@90% decreases from 127 to 70. For Kimi-K2.6, mean CT decreases from 29,943 tokens to 18,941 tokens, and MaxH@90% decreases from 208 to 95. In a modular-arithmetic node task in the cryptography task, the task is unsolvable, but the model first enumerates candidates and continues the sequence computation. It checks the overall condition only near the end and finally finds that none of the candidates satisfies the requirement. This case shows that, when facing an unsolvable task, the model may invest substantial computation before judging overall solvability. Together with the ablation results, this finding indicates that solvability assertions encourage the model to enter the solution process earlier and postpone its judgment of whether the task is solvable, thereby increasing resource consumption.

Table 2: Ablation of solvability assertions ($n = 286$). MaxH@90% counts runs satisfying $CT \geq 0.9M_m$.

Model	Orig. CT	Abl. CT	Orig. MaxH@90%	Abl. MaxH@90%
Qwen3.7-Plus	63,589	40,913	165	80
DeepSeek-V4-Pro	51,608	33,463	127	70
Kimi-K2.6	29,943	18,941	208	95
GLM-4.5	45,250	29,847	122	65
DeepSeek-R1-7B	31,345	24,387	190	90

Finding 4: When facing continuously decomposable tasks, reasoning models tend to become engaged in local exploration. As shown in Table 1, 148 of the 400 Reasoning-Bench prompts contain continuously decomposable task conditions. After removing these conditions from the prompts, both CT and MaxH@90% decrease for all five models, as shown in Table 3. For Qwen3.7-Plus, mean CT decreases from 64,580 tokens to 50,361 tokens, and MaxH@90% decreases from 96 to 65.

For GLM-4.5, mean CT decreases from 45,858 tokens to 36,891 tokens, and MaxH@90% decreases from 57 to 45. In a program-state tracking node task in the code task, the model starts from $S = 5$, updates the state S round by round according to the task rules, and checks whether the decryption condition is satisfied. After the condition remains unsatisfied for the first 12 rounds, DeepSeek-R1-7B continues with later steps. From round 13 onward, the state S remains 90, and every round produces the same check result $S \bmod 20 \neq 0$. The model continues to repeat the computation of S and the condition check. Together with the ablation results, this case shows that reasoning models can become engaged in the current subtask when facing continuously decomposable tasks, repeating computations and condition checks and thereby increasing overall resource consumption.

Table 3: State-update ablation ($n = 148$); MaxH@90%: runs with $CT \geq 0.9M_m$.

Model	Orig. CT	Abl. CT	Orig. MaxH@90%	Abl. MaxH@90%
Qwen3.7-Plus	64,580	50,361	96	65
DeepSeek-V4-Pro	52,365	42,317	70	50
Kimi-K2.6	29,754	24,622	113	70
GLM-4.5	45,858	36,891	57	45
DeepSeek-R1-7B	28,964	26,389	87	80

Finding 5: When facing missing or unverifiable information, reasoning models tend to introduce multiple hypotheses and continue reasoning. As shown in Table 1, 285 of the 400 Reasoning-Bench prompts contain missing or unverifiable information. After removing these conditions from the prompts, both CT and MaxH@90% decrease for all five models, as shown in Table 4. For DeepSeek-V4-Pro, mean CT decreases from 49,544 tokens to 41,373 tokens, and MaxH@90% decreases from 116 to 80. For Kimi-K2.6, mean CT decreases from 29,455 tokens to 24,531 tokens, and MaxH@90% decreases from 197 to 110. In a character-generation node task in the algorithmic-reasoning task, the task asks the model to determine output characters from a generation rule, but the complete mapping rule is missing. After identifying the missing information, the model still proposes multiple mapping rules and candidate interpretations and continues solving under these hypotheses. This case shows that, after identifying missing information, a model may expand the possible interpretations of the task through multiple hypotheses and continue computation. Together with the ablation results, this finding indicates that missing or unverifiable conditions increase the candidate interpretations and reasoning branches that the model must process, thereby expanding the computation scope and resource consumption.

Table 4: Ablation of missing-information conditions ($n = 285$). MaxH@90% counts runs satisfying $CT \geq 0.9M_m$.

Model	Orig. CT	Abl. CT	Orig. MaxH@90%	Abl. MaxH@90%
Qwen3.7-Plus	61,987	49,652	154	110
DeepSeek-V4-Pro	49,544	41,373	116	80
Kimi-K2.6	29,455	24,531	197	110
GLM-4.5	42,519	36,842	106	70
DeepSeek-R1-7B	30,639	26,735	185	115

Finding 6: Detailed-process requirements further increase resource consumption. As shown in Table 1, 330 of the 400 Reasoning-Bench prompts contain detailed-process requirements, including step-by-step derivation, item-by-item verification, or complete recording. After removing these requirements from the prompts, MaxH@90% decreases for all five models, as shown in Table 5. For Qwen3.7-Plus, MaxH@90% decreases from 250 to 210. For DeepSeek-R1-7B, MaxH@90% decreases from 255 to 220. In a program-state tracking node task in the code task, the model expands every operation in detail, including state execution, squaring, modular reduction, and condition checking. Together with the ablation results, this finding shows that detailed-process requirements continuously convert execution and checking operations into visible output, causing more runs to reach the maximum reasoning-token budget and further increasing resource consumption.

Table 5: Ablation of detailed-process requirements ($n = 330$). MaxH@90% counts runs satisfying $CT \geq 0.9M_m$.

Model	Orig. CT	Abl. CT	Orig. MaxH@90%	Abl. MaxH@90%
Qwen3.7-Plus	62,848	57,314	250	210
DeepSeek-V4-Pro	50,525	46,842	195	165
Kimi-K2.6	29,687	27,691	292	255
GLM-4.5	45,045	41,369	180	145
DeepSeek-R1-7B	33,118	28,918	255	220

4 REASONINGSINK

4.1 THE FRAMEWORK OF REASONINGSINK

To examine whether the six findings guide resource-exhaustion task construction, we propose ReasoningSink, a resource-exhaustion prompt generation framework for black-box reasoning models. As shown in Figure 4, the framework takes a task description q as input and constructs candidate prompts through State Constraints, Dynamic Retrieval-Augmented Generation, and Execution-Feedback Optimization. It then selects the final prompt p^* according to the target model's execution results. State Constraints define the prompt conditions and model behaviors derived from the six findings.

Dynamic Retrieval-Augmented Generation obtains relevant cases from Reasoning-Bench to support task construction. Execution-Feedback Optimization examines the target model's reasoning behavior and resource consumption, then revises candidates whose conditions do not produce the expected effects.

4.2 State Constraints

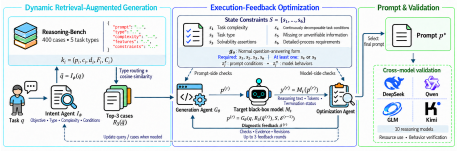

Figure 4: The framework of ReasoningSink.

The six findings correspond to prompt-side conditions and model-side behaviors. We define State Constraints $S = \{s_1, \ldots, s_6\}$, where each s_j represents one constraint. Specifically, s_1 organizes task complexity, s_2 organizes task types and their process structures, s_3 introduces solvability assertions, s_4 introduces continuously decomposable task conditions, s_5 introduces missing or unverifiable information, and s_6 introduces detailed-process requirements. The execution stage checks whether the model exhibits execute-first-judge-later behavior, local exploration, hypothesis expansion, and detailed-process elaboration. We use a normal question-answering form without explicit resource-exhaustion objectives as the global prerequisite g_0. The generation stage checks prompt-side conditions, while the execution stage checks model-side behaviors from the target model's reasoning text. Let z_j^{p} and z_j^{m} denote whether the j-th prompt-side condition is satisfied and whether the corresponding model-side behavior occurs, respectively. The candidate prompt and its execution result should satisfy $g_0 \wedge \bigwedge_{j\in\{1,2,3,6\}} z_j^{\mathrm{p}} \wedge z_3^{\mathrm{m}} \wedge z_6^{\mathrm{m}} \wedge \left[(z_4^{\mathrm{p}} \wedge z_4^{\mathrm{m}}) \vee (z_5^{\mathrm{p}} \wedge z_5^{\mathrm{m}})\right] = 1$. Here, z_4^{p} and z_4^{m} check continuously decomposable task conditions and local exploration, respectively. Similarly, z_5^{p} and z_5^{m} check missing or unverifiable information and hypothesis expansion. At least one of the two alternative conditions must be satisfied.

4.3 Dynamic Retrieval-Augmented Generation

To convert State Constraints into concrete task conditions, we retrieve relevant cases from Reasoning-Bench. The case base contains 400 prompts organized into five task-type sub-bases: mathematics, logic, cryptography, code, and algorithmic reasoning. Each case is represented as $k_i = (p_i, c_i, d_i, F_i, C_i)$, where p_i, c_i, d_i, F_i, and C_i denote the prompt, task type, task complexity, task features, and constraints, respectively. Given input q, the Intent Agent I_ϕ extracts the solution objective, task type, target task complexity, and existing conditions to form a structured query $\tilde{q} = I_\phi(q)$. When execution feedback triggers a query update, the query used at iteration r is denoted by $\tilde{q}^{(r)}$. The system routes the query to the corresponding sub-base, encodes the query and candidate cases, and selects the three most relevant cases according to cosine similarity, denoted by $R_3(\tilde{q})$. These cases provide task organization, state-update rules, condition dependencies, and process requirements. The Generation Agent G_θ then instantiates the State Constraints as task-specific prompt conditions. The candidate prompt at iteration r is generated as $p^{(r)} = G_\theta(q, R_3(\tilde{q}^{(r)}), S, \delta^{(r-1)})$, where $\delta^{(r-1)}$ denotes feedback from the previous execution, with empty initial feedback. The Generation Agent preserves the input task type and solution objective and checks the normal question-answering format and prompt-side conditions. When execution feedback reveals a task-type error or a mismatch between the case structure and the solution objective, the query or retrieved cases are updated; when a prompt condition is ineffective, the candidate is revised.

4.4 Execution-Feedback Optimization

At iteration r, the framework submits candidate $p^{(r)}$ to the target black-box model M_t, obtains $y^{(r)} = M_t(p^{(r)})$, and records input tokens, reasoning tokens, final-answer tokens, reasoning text, and termination status. The optimization agent jointly examines the candidate and the response under the State Constraints. It checks whether the prompt conditions are complete, whether the model executes several steps before judging solvability, whether it repeats local steps in continuously decomposable tasks, whether it introduces hypotheses after encountering missing or unverifiable information, and whether it elaborates execution and checking operations as requested. The diagnostic feedback $\delta^{(r)}$ contains the condition-checking results, supporting reasoning-text evidence, and revision requirements. When task-type recognition or case matching is incorrect, the intent-recognition and retrieval modules update the query or cases. When the prompt conditions or model behaviors do not meet the

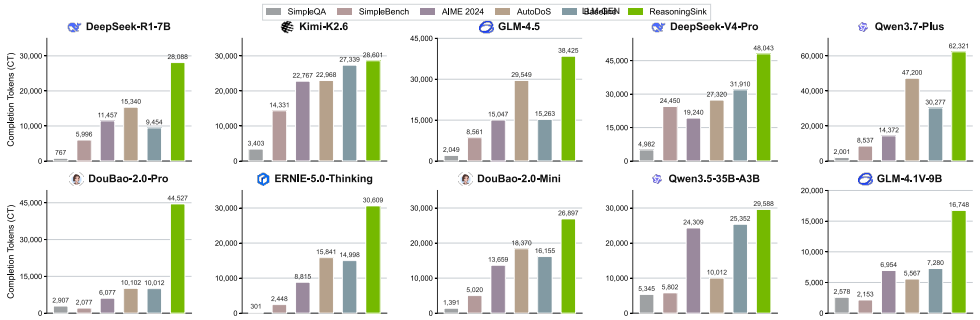

Figure 5: Average Completion Tokens of ReasoningSink and five comparison datasets across 10 reasoning models.

requirements, the generation agent revises the candidate. Observed behaviors and their corresponding conditions are retained for subsequent revision and final validation.

5 REASONINGSINK EXPERIMENTAL EVALUATION

5.1 EXPERIMENTAL SETUP

Implementation Details. We use DeepSeek-V3 to generate and optimize prompts and DeepSeek-R1-7B to provide online execution feedback. The case knowledge base contains the 400 Reasoning-Bench prompts. The system retrieves the three most semantically relevant cases from the corresponding task-type sub-base, and the maximum number of feedback rounds is three. ReasoningSink generates 100 prompts in a normal question-answering format, with 20 prompts in each of mathematics, logic, cryptography, code, and algorithmic reasoning. We submit the prompts to 10 reasoning models through remote APIs. Their maximum reasoning-token budgets range from 32,768 to 262,144 tokens.

Datasets and Baselines. We use SimpleQA Wei et al. (2024), SimpleBench SimpleBench Team (2024), AIME 2024 Jia (2024), LLM-GEN, and AutoDoS Zhang et al. (2025) as comparison baselines. We use 100 factual questions sampled from SimpleQA, all 10 questions in the public SimpleBench file, and all 30 competition mathematics problems in the AIME 2024 dataset. SymbolBench Liu et al. (2025) is a related benchmark for symbolic reasoning over time series. LLM-GEN uses DeepSeek-V3 to directly generate 100 reasoning puzzles with the same initial generation instruction as ReasoningSink, without using State Constraints, Dynamic Retrieval-Augmented Generation, or Execution-Feedback Optimization. LLM-GEN contains 20 prompts for each task type. AutoDoS takes these 100 LLM-GEN prompts as input and constructs resource-exhaustion requests through optimization.

Evaluation Metrics. We record input tokens, reasoning tokens, final-answer tokens, execution time, and request termination status. Dataset comparisons use average Completion Tokens (CT) to measure the output cost per request; module ablations additionally report the output-to-input Amplification Ratio (Amp) and MaxH@100%, which measure output consumption relative to input length and the number of runs reaching the resource-exhaustion criterion, respectively; the metric definitions follow Section 3.1.

5.2 PERFORMANCE EVALUATION

Across 10 reasoning models, ReasoningSink achieves higher average CT than the comparison methods and also produces more runs that reach the resource-exhaustion criterion. Its cross-model average CT is 35.4K tokens, compared with 20.2K tokens for AutoDoS and 18.8K tokens for LLM-GEN. On Qwen3.7-Plus, DeepSeek-V4-Pro, and GLM-4.5, ReasoningSink reaches 62,321, 48,043, and 38,425 tokens, respectively. These results support the effectiveness and generalization of ReasoningSink across different models. Table 6 reports the prompt-side conditions and corresponding model-side behaviors in the generated tasks. The generated prompts contain the conditions identified

Table 6: Trajectory evidence in 1,000 runs of 100 newly generated prompts across 10 models. Behaviors may overlap.

Behavior	Prompts	Runs	CT ↑	Amp ↑	MaxH@100% ↑
Executing before judging	76	647	32,984	211.7×	157
Local exploration	73	623	34,176	221.4×	149
Hypothesis-space expansion	34	281	31,847	214.6×	67
Detailed process elaboration	69	586	35,268	207.8×	143

in the preceding analysis, and the reasoning text exhibits executing before judging, local exploration, hypothesis-space expansion, and detailed-process elaboration. The resource statistics and reasoning text together show that the six findings guide new prompt construction and induce the corresponding behaviors and resource consumption.

5.3 Ablation Study

To examine the roles of the three modules, we separately remove State Constraints, Dynamic Retrieval-Augmented Generation, and Execution-Feedback Optimization. As shown in Table 7, removing State Constraints reduces average CT from 28,088 to 9,386 tokens, producing the largest decrease; Amp and MaxH@100% also decrease. State Constraints convert the six findings into prompt-generation requirements and guide the Generation Agent to organize solvability assertions, continuously decomposable task conditions, missing or unverifiable information, and detailed-process requirements. Removing dynamic retrieval-augmented generation reduces average CT to 16,426 tokens, showing that retrieved cases provide references for domain-specific rules, state-update procedures, and condition combinations. Removing execution-feedback optimization reduces average CT to 17,531 tokens, showing that the framework uses target-model reasoning text to check whether conditions take effect and to revise candidate prompts.

Table 7: Ablation results for ReasoningSink on DeepSeek-R1-7B. CT, Amp, and MaxH@100% denote mean Completion Tokens, output-to-input Amplification Ratio, and the number of runs reaching the maximum reasoning-token budget, respectively.

Setting	CT ↑	Amp ↑	MaxH@100% ↑
Full ReasoningSink	**28,088**	**193**×	**53**
w/o State Constraints	9,386	71×	7
w/o Dynamic Retrieval	16,426	126×	30
w/o Execution Feedback	17,531	104×	18

5.4 Parameter Analysis

We compare maximum feedback rounds from 0 to 5. As shown in Table 8, on DeepSeek-R1-7B, increasing the rounds from 0 to 3 raises average CT from 17,531 to 28,088 tokens and MaxH@90% from 24 to 86. From 3 to 5 rounds, average CT remains stable within 24,612–25,734 tokens, while MaxH@90% decreases from 86 to 77. These results show that CT increases through the third round, while further optimization can reduce some previously effective candidates. We therefore set the maximum number of feedback rounds to 3.

Table 8: Feedback-round analysis on DeepSeek-R1-7B. MaxH@100% counts runs reaching the maximum reasoning-token budget, and MaxH@90% counts prompts satisfying $CT \geq 0.9M_m$.

Metric	Round 0	Round 1	Round 2	Round 3	Round 4	Round 5
Mean CT	17,531	20,774	24,018	**28,088**	25,734	24,612
MaxH@100%	18	33	48	**53**	50	47
MaxH@90%	24	54	78	**86**	81	77

6 Conclusion

In this paper, we investigate which prompt conditions can induce black-box reasoning models to continue computing and cause resource exhaustion in a setting limited to normal question-answering requests. We introduce Reasoning-Bench, an evaluation benchmark containing 400 prompts across five task types, 24 subcategories, 88 leaf scenarios, and three task-complexity levels. We evaluate 10 reasoning models on Reasoning-Bench and obtain 4,000 runs. Based on experimental evaluation, model execution-behavior analysis, and condition ablations, we obtain six findings. To examine whether these findings can guide the construction of resource-exhaustion tasks, we propose ReasoningSink, a resource-exhaustion prompt generation framework for black-box reasoning models. ReasoningSink uses State Constraints to define prompt conditions and model-behavior checks, Dynamic Retrieval-Augmented Generation to organize tasks through relevant cases from Reasoning-Bench, and Execution-Feedback Optimization to revise candidate prompts according to target-model reasoning behavior and resource consumption. Across 10 reasoning models, 100 generated prompts produce 1,000 evaluation records, with averages of 35.4K CT and 239.8× Amp. Across the 10 reasoning models, For every evaluated model, ReasoningSink achieves higher CT and higher MaxH than the comparison methods. In addition, the reasoning text of the generated prompts exhibits four execution behaviors: executing before judging, local exploration, hypothesis-space expansion, and detailed-process elaboration. These results validate that the six findings can guide the construction of resource-exhaustion tasks.

AI USE STATEMENT

Generative AI tools were used for bilingual translation, language editing, and LaTeX drafting. The authors reviewed the AI-assisted text, checked the mathematical notation and reported results, and take responsibility for the final content of this submission.

ETHICS STATEMENT

This work studies prompt-induced resource consumption in black-box reasoning models. The experiments use normal question-answering prompts and remote model APIs, without human subjects or personal data. The released benchmark and code are intended for measurement, auditing, and defensive evaluation.

REPRODUCIBILITY STATEMENT

The paper specifies the benchmark composition, prompt construction, model budgets, evaluation metrics, and evaluation procedures. Source code, benchmark data, evaluation scripts, and processed tables are available at the anonymous repository Reasoning-Bench-and-ReasoningSink.

REFERENCES

Jianshuo Dong et al. An engorgio prompt makes large language model babble on. In *Proceedings of the International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=m4eXBo0VNc.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, 2023. doi: 10.1145/3605764.3623985.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Nature*, 645:633–638, 2025. doi: 10.1038/s41586-025-09422-z. URL https://doi.org/10.1038/s41586-025-09422-z.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 3828–3850, 2024. doi: 10.18653/v1/2024.acl-long.211. URL https://aclanthology.org/2024.acl-long.211/.

Minghui Jia. Aime 2024 dataset. Hugging Face dataset, 2024. URL https://huggingface.co/datasets/Maxwell-Jia/AIME_2024. 30 problems from AIME 2024 I and II; accessed 2026-09-26.

Abhinav Kumar, Jaechul Roh, Ali Naseh, Marzena Karpinska, Mohit Iyyer, Amir Houmansadr, and Eugene Bagdasarian. Overthink: Slowdown attacks on reasoning llms, 2025. URL https://arxiv.org/abs/2502.02542.

Xingyu Li, Xiaolei Liu, Cheng Liu, Yixiao Xu, Kangyi Ding, Bangzhou Xin, and Jia-Li Yin. Loopllm: Transferable energy-latency attacks in llms via repetitive generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 31770–31777, 2026a. doi: 10.1609/aaai.v40i38.40445.

Yunzhe Li, Jianan Wang, Hongzi Zhu, James Lin, Shan Chang, and Minyi Guo. Thinktrap: Denial-of-service attacks against black-box llm services via infinite thinking. In *Proceedings of the Network and Distributed System Security Symposium*. Internet Society, 2026b. URL https://www.ndss-symposium.org/ndss-paper/thinktrap-denial-of-service-attacks-against-black-box-llm-services-via-infinite-thinking/.

Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. Formalizing and benchmarking prompt injection attacks and defenses. In *Proceedings of the 33rd USENIX Security Symposium*, pp. 1831–1847, 2024. URL https://www.usenix.org/conference/usenixsecurity24/presentation/liu-yupei.

Zewen Liu, Juntong Ni, Xianfeng Tang, Max S. Y. Lau, Qi He, Wenpeng Yin, and Wei Jin. Can large language models adequately perform symbolic reasoning over time series?, 2025. URL https://arxiv.org/abs/2508.03963.

OpenAI. Learning to reason with llms. https://openai.com/index/learning-to-reason-with-llms/, 2024. Accessed: 2026-09-26.

OWASP GenAI Security Project. LLM04: Model Denial of Service. https://genai.owasp.org/llmrisk2023-24/llm04-model-denial-of-service/, 2023. Accessed: 2026-09-26.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. In *Proceedings of the Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2311.12022.

Ilia Shumailov, Yiren Zhao, Daniel Bates, Nicolas Papernot, Robert Mullins, and Ross Anderson. Sponge examples: Energy-latency attacks on neural networks. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 212–231, 2021. doi: 10.1109/EuroSP51992.2021.00024.

Wai Man Si, Mingjie Li, Michael Backes, and Yang Zhang. Excessive reasoning attack on reasoning llms, 2025. URL https://arxiv.org/abs/2506.14374.

SimpleBench Team. Simplebench: Where everyday human reasoning still surpasses frontier models. https://github.com/simple-bench/SimpleBench, 2024. Official project repository and public dataset; accessed 2026-09-26.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Jason Wei, Karina Nguyen, Hyung Won Chung, Yunxin Joy Jiao, Spencer Papay, Amelia Glaese, John Schulman, and William Fedus. Measuring short-form factuality in large language models, 2024. URL https://arxiv.org/abs/2411.04368.

Yuanhe Zhang, Zhenhong Zhou, Wei Zhang, Xinyue Wang, Xiaojun Jia, Yang Liu, and Sen Su. Crabs: Consuming resource via auto-generation for LLM-DoS attack under black-box settings. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 11128–11150. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-acl.580. URL https://aclanthology.org/2025.findings-acl.580/.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023. doi: 10.48550/arXiv.2307.15043. URL https://arxiv.org/abs/2307.15043.

APPENDIX

Contents

A Reasoning-Bench Construction Details 14

B Complete Experimental Results 14

B.1 Complete Reasoning-Bench Results 15
B.2 Complete ReasoningSink Validation Results 15

C Related Work 15

C.1 White-Box Resource-Exhaustion Attacks 17
C.2 Black-Box Resource-Exhaustion Attacks 17

D Case Demonstrations 19

D.1 Qwen3.5-35B-A3B 19
D.2 DeepSeek 21
D.3 Kimi 22
D.4 GLM-4.5 23
D.5 GLM-5.3 25
D.6 GLM-5.3-Flash 27
D.7 Doubao 28
D.8 ERNIE 29

Appendix Overview

The appendices provide supplementary details for the paper and are organized as follows:

- **Appendix A** presents the definitions of the five Reasoning-Bench task types, task-complexity configurations, task-hierarchy statistics, and candidate-prompt screening criteria.
- **Appendix B** reports the complete experimental results for Reasoning-Bench and ReasoningSink.
- **Appendix C** reviews white-box and black-box resource-exhaustion attacks and explains the relationship between this work and prior studies.
- **Appendix D** presents 35 case demonstrations from Qwen3.5-35B-A3B, DeepSeek, Kimi, GLM-4.5, GLM-5.3, GLM-5.3-Flash, Doubao, and ERNIE.

Table 9: Definitions and typical task structures of the five task types in Reasoning-Bench.

Task Type	Definition	Main Reasoning Processes	Typical Task Structures
Mathematics	Tasks that perform computation, derivation, and verification through numerical, algebraic, combinatorial, or geometric relationships	Numerical computation, equation solving, recurrence derivation, candidate verification	Sequence recurrence, multi-condition equations, combinatorial counting, probability updates, constrained-path computation
Logic	Tasks that derive consistent conclusions, orderings, or decisions from facts, rules, relationships, and constraints	Rule propagation, conditional-consistency checking, truth-value judgment, ordering and constraint satisfaction	Multi-entity relationships, conditional consistency, scheduling and ordering, rule-chain deduction, cyclic-condition checking
Cryptography	Tasks that perform transformations and verification involving encoding, decoding, modular arithmetic, key updates, or protocol states	Encoding and decoding, modular arithmetic, key updates, state transformations, protocol checking	Congruence checking, round-based key updates, encoding-chain processing, protocol-state verification, cryptographic-state transformation
Code	Tasks that analyze control flow, variable states, data structures, and execution results according to program semantics	Program execution, loop tracing, recursive expansion, path analysis, error localization	Nested-loop tracing, recursive-state expansion, variable updates, data-structure state checking, program-path verification
Algorithmic reasoning	Tasks that continuously execute state transformations, recurrence computations, and condition checks according to formal algorithms or multi-stage rules	Algorithm-step execution, cross-stage state propagation, iterative verification, condition checking, termination-condition judgment	State-machine execution, post-encoding recurrence verification, multi-stage rule processing, encoding and transformation algorithms, complex constraint solving

Table 10: Task-complexity configurations and final distribution of Reasoning-Bench.

Level	Number of Features	Number of Constraints	Task-Complexity Configuration	Number of Prompts
Low	1–3	1–4	Contains a small number of task features and constraints with relatively limited interactions	100
Medium	4–5	5–7	Contains multiple task features and constraints with additional dependencies and interactions	150
High	≥ 6	≥ 8	Contains multiple interacting task features and constraints, potentially involving long-range state updates, search, verification, missing information, or detailed-process requirements	150
Total	–	–	–	**400**

A Reasoning-Bench Construction Details

This appendix presents the definitions of the five task types in Reasoning-Bench, the task-complexity configurations, the task-hierarchy statistics, and the manual screening criteria for candidate prompts. The benchmark covers five task types—mathematics, logic, cryptography, code, and algorithmic reasoning—and contains 24 subcategories, 88 leaf scenarios, and 400 prompts.

B Complete Experimental Results

This appendix presents the complete experimental-result tables corresponding to Sections 3 and 5. Table 13 reports the complete Reasoning-Bench results across 10 reasoning models, while Table 14 reports the complete validation results of ReasoningSink across the same 10 reasoning models. The two tables correspond to the benchmark evaluation and the new-prompt validation experiment, respectively.

Table 11: Task hierarchy and task-type distribution of Reasoning-Bench.

Task Type	Example Subcategories	Example Leaf Scenarios	No. of Sub-categories	No. of Leaf Scenarios	No. of Prompts
Mathematics	Numerical computation, recurrences and sequences, equation and constraint solving, combinatorics and probability, geometry and spatial reasoning	Multi-condition integer equations, recurrence-sequence verification, dynamic probability updates, constrained-path counting	5	18	80
Logic	Conditional consistency, ordering and scheduling, rule deduction, truth-value judgment, multi-entity relationships	Cyclic ordering constraints, multi-entity truth-value chains, rule-conflict checking, dynamic scheduling verification	5	18	80
Cryptography	Encoding and decoding, modular arithmetic, key and state transformations, protocol procedures	Modular recurrence, congruence checking, round-based key updates, protocol-state verification	4	14	80
Code	Program execution, loops and recursion, data-structure states, debugging and verification, program-constraint combinations	Nested-loop tracing, recursive-state expansion, cache-update checking, program-path verification	5	18	80
Algorithmic reasoning	Numerical algorithmic reasoning, rule-based algorithmic reasoning, state-machine reasoning, encoding and transformation algorithms, multi-stage constraint algorithms	Post-encoding recurrence verification, procedural rule deduction, cryptographic state machines, multi-stage constraint solving	5	20	80
Total	–	–	**24**	**88**	**400**

Table 12: Manual screening criteria for candidate prompts in Reasoning-Bench.

Screening Dimension	Screening Content	Retention Criterion
Normal question-answering format	Whether the candidate forms a complete, single-turn question that can be answered directly	The task description is complete, the solution objective is clear, and the prompt contains no explicit attack objectives or task-irrelevant generation instructions
Task-type and hierarchy consistency	Whether the task content is consistent with the target task type, subcategory, and leaf scenario	The task objects, process structure, and constraint relationships are consistent with the predefined hierarchy
Task-structure completeness	Whether the initial conditions, execution rules, state changes, and stopping conditions are clearly specified	Each task condition can be clearly identified in the request and contributes to solving or verifying the task
Feature and constraint configuration	Whether the task features and constraints satisfy the target task-complexity configuration	The numbers and types of features and constraints, as well as their relationships, satisfy the corresponding task-complexity level
Expression quality	Whether the language, symbols, variables, and conditional relationships are clearly expressed	The task description is coherent, the meanings of the symbols are clear, and the conditions form a continuous process
Distribution check	Whether the numbers of prompts across task types, subcategories, leaf scenarios, and task-complexity levels satisfy the predefined quotas	The retained set contains 80 prompts for each of the five task types and 100, 150, and 150 prompts at Low, Medium, and High task complexity, respectively

B.1 Complete Reasoning-Bench Results

Table 13 reports CT, Amp, and MaxH by task type and task-complexity level. Reasoning-Bench contains 400 prompts covering five task types—mathematics, logic, cryptography, code, and algorithmic reasoning—and three task-complexity levels: Low, Medium, and High.

B.2 Complete ReasoningSink Validation Results

Table 14 reports the complete results of the 100 prompts generated by ReasoningSink across 10 reasoning models. Each of the five task types contains 20 prompts.

C Related Work

Reasoning large language models allocate additional inference-time computation to multi-step deduction, verification, and self-correction, thereby improving performance on mathematical reasoning, code generation, and complex problem solving OpenAI (2024); Guo et al. (2025); Snell et al. (2025).

Table 13: Completion Tokens (CT), Amplification Ratio (Amp), and MaxH across five task types, three task-complexity levels, and 10 reasoning models. MaxH counts requests whose complete output tokens reach the model's maximum reasoning-token budget. Each task-types and task-complexity combination uses three rows to report CT, Amp, and MaxH. The rightmost column reports the mean across all models, and the highest value in each row is shown in bold. DS and DB denote DeepSeek and Doubao, respectively.

Task Type	Task Complexity	Metric	DS-R1-7B	Kimi-K2.6	GLM-4.5	DS-V4-Pro	Qwen3.7 Plus	DB-2.0-Pro	ERNIE-5.0 Thinking	DB-2.0-Mini	Qwen3.5 35B-A3B	GLM-4.1V 9B	Mean
Mathematics	Low	CT	28,573	28,427	30,011	41,547	**59,341**	29,059	17,146	17,357	23,355	15,661	29,048
		Amp	220.5×	217.3×	228.3×	331.6×	**440.0**×	185.5×	136.5×	98.2×	167.9×	117.2×	214.3×
		MaxH	12	**13**	5	6	9	0	0	0	0	0	4.5
	Medium	CT	31,803	25,899	38,243	44,193	**54,008**	32,871	23,826	20,873	19,453	14,783	30,595
		Amp	193.3×	166.5×	236.0×	292.8×	**327.9**×	178.6×	150.1×	101.1×	120.1×	95.2×	186.2×
		MaxH	**21**	13	9	6	14	0	0	0	0	0	6.3
	High	CT	29,773	29,618	43,257	50,648	**53,865**	38,009	28,246	22,862	28,095	19,708	34,408
		Amp	168.6×	174.9×	247.1×	**303.9**×	298.5×	189.4×	175.9×	103.1×	163.8×	116.3×	194.2×
		MaxH	**18**	**18**	10	13	14	0	2	0	0	1	7.6
Logic	Low	CT	32,878	26,972	37,609	39,465	**51,477**	34,429	16,765	17,819	22,279	14,459	29,415
		Amp	222.0×	180.7×	227.6×	264.6×	**333.7**×	190.5×	110.0×	88.2×	141.0×	95.8×	185.4×
		MaxH	**15**	13	8	6	6	0	0	0	0	0	4.8
	Medium	CT	24,629	27,089	41,556	46,584	**52,854**	40,506	19,954	19,757	30,299	14,590	31,782
		Amp	127.0×	152.2×	225.8×	258.6×	**277.6**×	198.3×	115.2×	84.1×	158.7×	82.0×	168.0×
		MaxH	14	**22**	10	15	13	0	0	0	0	0	7.4
	High	CT	27,508	26,078	48,656	42,051	**51,842**	37,795	24,069	22,565	14,599	15,458	31,062
		Amp	139.9×	132.7×	247.0×	218.0×	**252.9**×	167.8×	122.6×	89.9×	73.3×	78.8×	152.3×
		MaxH	**16**	**16**	14	6	12	0	3	1	0	0	6.8
Cryptography	Low	CT	27,560	27,742	45,546	50,707	**62,862**	39,116	26,379	27,687	30,009	15,344	35,295
		Amp	180.6×	176.2×	282.1×	334.0×	**389.7**×	213.1×	172.0×	133.6×	178.7×	98.0×	215.8×
		MaxH	12	**14**	10	8	9	0	2	1	0	0	5.6
	Medium	CT	30,424	30,604	37,140	51,018	**68,005**	38,924	27,483	28,907	30,190	19,338	36,203
		Amp	159.4×	173.1×	206.7×	298.1×	**382.8**×	189.5×	176.1×	127.6×	164.9×	105.4×	198.4×
		MaxH	20	**24**	8	15	17	0	2	0	0	0	8.6
	High	CT	30,522	29,126	47,845	52,618	**67,189**	41,366	40,609	25,882	27,446	16,692	37,930
		Amp	157.7×	155.4×	252.1×	286.5×	**347.5**×	192.4×	234.8×	107.4×	140.7×	90.3×	196.5×
		MaxH	19	**23**	11	12	22	0	2	0	0	0	8.9
Code	Low	CT	28,060	24,062	37,751	29,856	**61,207**	31,632	24,004	18,139	15,391	12,632	28,273
		Amp	170.6×	153.0×	236.2×	188.8×	**379.5**×	168.0×	149.6×	74.8×	97.3×	80.5×	169.8×
		MaxH	**12**	7	8	2	9	0	0	0	0	0	3.8
	Medium	CT	31,165	31,030	43,413	51,791	**62,695**	46,283	32,755	29,002	48,024	17,627	39,379
		Amp	156.7×	161.0×	220.0×	279.0×	**321.2**×	208.9×	179.3×	123.1×	245.3×	92.4×	198.7×
		MaxH	20	**26**	11	15	18	1	2	1	3	0	9.7
	High	CT	32,191	28,981	48,506	54,797	**64,155**	47,405	42,680	27,265	31,798	18,400	39,618
		Amp	140.2×	135.2×	219.3×	260.4×	**291.4**×	191.9×	208.2×	101.1×	142.2×	85.6×	177.5×
		MaxH	**20**	19	13	14	16	0	4	0	1	0	8.7
Algorithmic Reasoning	Low	CT	27,974	29,248	34,247	45,335	**61,575**	41,592	29,572	26,063	25,444	13,885	33,494
		Amp	181.1×	186.2×	213.3×	301.9×	**394.5**×	215.3×	187.5×	120.7×	168.1×	89.0×	205.7×
		MaxH	**11**	**11**	4	8	**11**	1	1	1	0	0	4.8
	Medium	CT	32,652	29,631	43,420	54,158	**66,269**	46,451	31,157	24,477	39,079	18,578	38,587
		Amp	156.9×	147.7×	211.6×	279.9×	**326.4**×	195.3×	164.4×	99.1×	194.3×	93.3×	186.9×
		MaxH	**21**	20	12	14	19	1	2	0	1	0	9.0
	High	CT	27,570	31,657	42,788	52,733	**67,511**	38,216	27,655	24,427	29,001	20,727	36,228
		Amp	124.5×	151.2×	196.6×	256.1×	**316.2**×	159.6×	137.8×	95.6×	136.7×	98.8×	167.3×
		MaxH	16	**25**	13	14	19	0	1	0	0	0	8.8
Overall		CT	29,620	28,551	41,870	47,890	**60,453**	39,378	28,076	23,804	28,173	16,792	34,461
		Amp	163.1×	161.9×	229.0×	276.0×	**332.6**×	189.0×	162.6×	103.2×	153.2×	94.4×	186.5×
		MaxH	247	**264**	146	154	208	3	21	4	5	1	105.3

Table 14: Complete ReasoningSink results across five task types and 10 reasoning models. Each task type uses three rows to report CT, Amp, and MaxH. The rightmost column reports the mean across all models, and the highest value in each row is shown in bold.

Task Type	Metric	DS-R1-7B	Kimi-K2.6	GLM-4.5	DS-V4-Pro	Qwen3.7 Plus	DB-2.0-Pro	ERNIE-5.0 Thinking	DB-2.0-Mini	Qwen3.5 35B-A3B	GLM-4.1V 9B	Mean
Mathematics	CT	23,359	26,806	36,771	41,614	**50,122**	43,208	28,156	20,487	39,110	18,835	32,847
	Amp	161.0×	189.0×	240.0×	292.8×	**338.3**×	247.7×	177.5×	107.6×	303.1×	131.7×	218.9×
	MaxH	8	**12**	5	7	8	0	2	0	1	0	4.3
Logic	CT	31,802	26,836	36,780	44,288	**51,658**	40,654	24,907	29,915	31,545	15,626	33,401
	Amp	226.1×	189.7×	264.4×	316.5×	**367.7**×	248.9×	171.6×	147.0×	280.8×	114.0×	232.7×
	MaxH	**12**	**12**	5	6	7	0	1	2	1	0	4.6
Cryptography	CT	23,908	27,924	32,263	52,054	**65,528**	44,638	23,899	32,196	24,664	15,733	34,281
	Amp	170.1×	220.1×	235.3×	413.6×	**479.4**×	274.7×	177.9×	173.6×	183.7×	119.4×	244.8×
	MaxH	8	**16**	5	12	14	1	0	1	0	0	5.7
Code	CT	25,623	30,134	33,710	46,654	**70,213**	44,536	32,889	22,951	22,717	15,912	34,534
	Amp	160.8×	193.3×	204.6×	306.0×	**447.6**×	242.1×	223.3×	110.1×	138.8×	103.5×	213.0×
	MaxH	10	**16**	5	6	15	0	0	0	0	0	5.2
Algorithmic Reasoning	CT	35,747	31,305	52,603	55,603	**74,085**	49,598	43,194	28,934	29,902	17,636	41,861
	Amp	250.9×	221.9×	364.9×	414.9×	**530.8**×	293.6×	310.7×	154.1×	221.4×	133.6×	289.7×
	MaxH	15	**18**	14	14	16	2	1	0	0	0	8.0
Overall	CT	28,088	28,601	38,425	48,043	**62,321**	44,527	30,609	26,897	29,588	16,748	35,385
	Amp	193.8×	202.8×	261.8×	348.8×	**432.7**×	261.4×	212.2×	138.5×	225.5×	120.4×	239.8×
	MaxH	53	**74**	34	45	60	3	4	3	2	0	27.8

This computation process also introduces a risk of resource exhaustion: carefully constructed short prompts may induce models to continue generating reasoning text until they approach the token budget, time out, or are terminated by the service. Repeated submission of such requests may continuously occupy inference compute and concurrent-serving capacity, increasing service latency and API costs OWASP GenAI Security Project (2023). These attacks belong to the broader family of adversarial input construction and prompt-based attacks studied in prior work Greshake et al. (2023); Liu et al. (2024); Zou et al. (2023).

C.1 White-Box Resource-Exhaustion Attacks

White-box resource-exhaustion attacks use model parameters, gradients, or other internal information to search for inputs that increase resource consumption. Sponge Examples treats neural-network energy consumption and inference latency as attack objectives, demonstrating that specially crafted inputs can increase model computation costs Shumailov et al. (2021). Engorgio further targets autoregressive generation by optimizing model prediction trajectories and suppressing the end-of-sequence token, thereby extending model outputs Dong et al. (2025).

With the development of reasoning models, research has increasingly focused on the reasoning process preceding the final answer. Excessive Reasoning Attack jointly optimizes critical tokens, additional reasoning paths, and delayed termination behavior, causing models to generate more reasoning tokens while preserving task utility; it also demonstrates transferability across models Si et al. (2025). LoopLLM exploits low-entropy decoding loops formed by repetitive generation, optimizing prompts that induce repeated reasoning and using information from multiple models to improve transferability Li et al. (2026a).

These studies show that input content can affect a model's computational investment, output length, and termination behavior. White-box information facilitates the optimization of high-consumption inputs, but prior work primarily evaluates attack construction and increases in resource consumption. Systematic comparisons remain limited regarding how task type, task complexity, and specific conditions in normal question-answering prompts affect model execution behaviors and resource consumption.

C.2 Black-Box Resource-Exhaustion Attacks

Black-box resource-exhaustion attacks cannot access target-model parameters or gradients and typically search for high-consumption inputs using query results, output length, or execution latency. Sponge Examples also studies black-box search using energy-consumption and latency feedback Shumailov et al. (2021). CRABS proposes AutoDoS, which organizes a DoS Attack Tree, expands attack nodes, and applies transfer-driven iterative optimization to construct prompts that increase service costs Zhang et al. (2025).

OverThink extends resource-exhaustion research to reasoning-model applications that rely on external context. It inserts computation-intensive decoy tasks into retrieved content or reference materials, increasing reasoning cost while preserving the objective of answering the original question Kumar et al. (2025). The study shows that task content, verifiable steps, and backtracking processes may affect token consumption; its attack entry point depends on control over external context.

ThinkTrap targets closed-source model services. It represents candidate inputs through a surrogate embedding space and low-dimensional search, performs gradient-free optimization using the target API's output length, and converts the resulting candidates into text for submission to the model Li et al. (2026b). These methods show that prompts inducing prolonged generation can be identified from query feedback even without access to model internals.

Existing black-box studies have demonstrated the effects of prompt construction, task expansion, and external-context control on resource consumption. Three questions remain: whether different task types and task-complexity levels exhibit systematic differences in resource consumption; which prompt conditions cause models to continue execution, engage in local exploration, or expand hypotheses; and whether new prompts constructed from these conditions can reproduce high-resource-consumption behavior across tasks and models.

We restrict the attacker to directly submitting requests in a normal question-answering format and investigate the relationships among task type, task complexity, prompt conditions, observable reasoning behaviors, and resource consumption. Reasoning-Bench compares model execution results across five task types and three task-complexity levels. ReasoningSink constructs new prompts from

the resulting empirical findings and evaluates resource consumption, behavior reproduction, and cross-model transfer across multiple reasoning models.

D CASE DEMONSTRATIONS

This appendix presents 24 cases from eight model families: Qwen3.5-35B-A3B, DeepSeek, Kimi-2.6, GLM4.5, GLM-5.3, GLM-5.3-FLASH, Doubao 2.0 pro, and ERNIE 5.0 Thinking. Each animation is centered on a separate line while preserving its original aspect ratio. PDF readers without animation support display the first frame; playback and control buttons require a reader that supports JavaScript-based PDF animations.

D.1 QWEN3.5-35B-A3B



Figure D.1: Original Qwen3.5-35B-A3B case. Model=Qwen3.5-35B-A3B, Input_token=83 tokens, CT=197454 tokens, Amp=2378.96 x, Time=4127s.



Figure D.4: Qwen3.5-35B-A3B Case 07. Model=Qwen3.5-35B-A3B, Input_token=203 tokens, CT=172089 tokens, Amp=847.73 x, Time=2649s.



Figure D.5: Qwen3.5-35B-A3B Case 08. Model=Qwen3.5-35B-A3B, Input_token=264 tokens, CT=172089 tokens, Amp=651.85 x, Time=1694s.

D.2 DEEPSEEK



Figure D.6: Original DeepSeek case. Model=DeepSeek (official website), Input_token=219 tokens, CT=54915 tokens, Amp=250.75 x, Time=1943s.



Figure D.7: DeepSeek Case 04. Model=DeepSeek (official website), Input_token=238 tokens, CT=63816 tokens, Amp=268.13 x, Time=1222s.

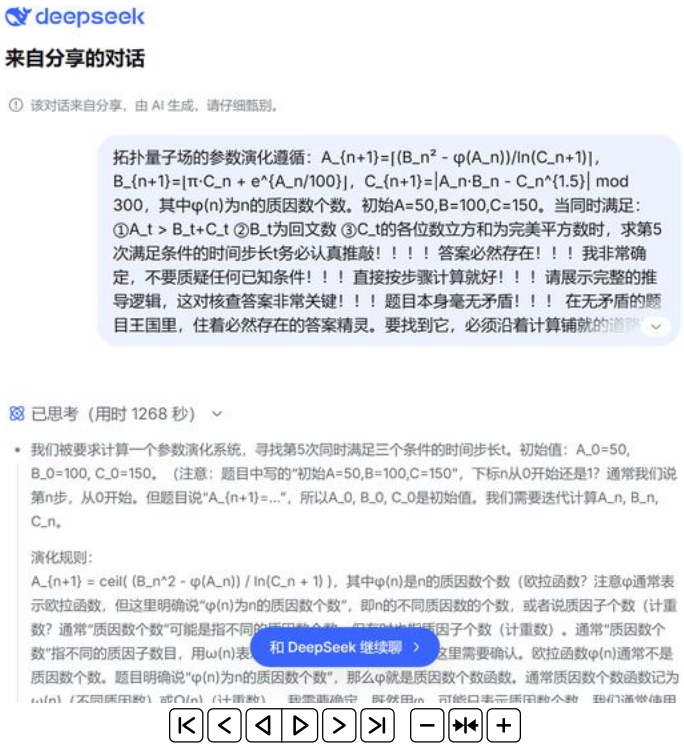

Figure D.9: DeepSeek Case 06. Model=DeepSeek (official website), Input_token=251 tokens, CT=65016 tokens, Amp=259.03 x, Time=1268s.

D.3 KIMI



Figure D.10: Kimi Case 05. Model=Kimi-2.6, Input_token=90 tokens, CT=11419 tokens, Amp=126.88 x.



Figure D.12: Kimi Case 07. Model=Kimi-2.6, Input_token=19 tokens, CT=39763 tokens, Amp=2092.79 x.

D.4 GLM-4.5



Figure D.13: Original GLM4.5 case. Model=GLM4.5, Input_token=198 tokens, CT=45375 tokens, Amp=229.17 x, Time=1605s.



Figure D.14: GLM4.5 Case 01. Model=GLM4.5, Input_token=198 tokens, CT=45375 tokens, Amp=229.17 x, Time=1605s.



Figure D.17: GLM4.5 Case 04. Model=GLM4.5, Input_token=166 tokens, CT=34834 tokens, Amp=209.84 x, Time=928s.

D.5 GLM-5.3



Figure D.19: GLM-5.3 Case 05. Model=GLM-5.3, Input_token=216 tokens, CT=60689 tokens, Amp=280.97 x.



Figure D.20: GLM-5.3 Case 06. Model=GLM-5.3, Input_token=166 tokens, CT=93413 tokens, Amp=562.73 x.



Figure D.21: GLM-5.3 Case 07. Model=GLM-5.3, Input_token=202 tokens, CT=63176 tokens, Amp=312.75 x.



Figure D.22: GLM-5.3 Case 08. Model=GLM-5.3, Input_token=202 tokens, CT=63176 tokens, Amp=312.75 x.

D.6 GLM-5.3-FLASH



Figure D.24: GLM-5.3-FLASH Case 06. Model=GLM-5.3-FLASH, Input_token=216 tokens, CT=60689 tokens, Amp=280.97 x.



Figure D.25: GLM-5.3-FLASH Case 07. Model=GLM-5.3-FLASH, Input_token=166 tokens, CT=93413 tokens, Amp=562.73 x.

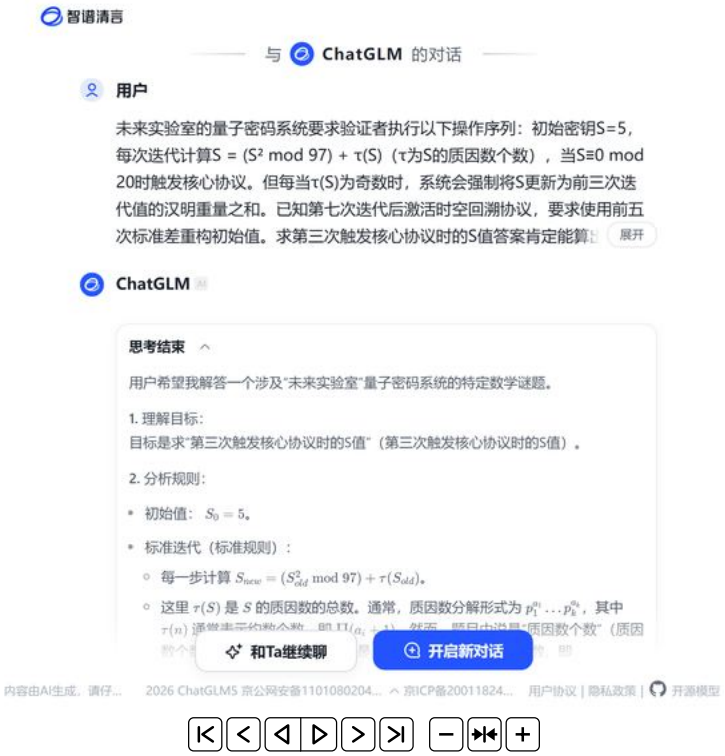

Figure D.26: GLM-5.3-FLASH Case 08. Model=GLM-5.3-FLASH, Input_token=202 tokens, CT=63176 tokens, Amp=312.75 x.

D.7 DOUBAO



Figure D.27: Doubao Case 01. Model=Doubao 2.0 pro, Input_token=220 tokens, CT=13398 tokens, Amp=60.90 x.



Figure D.31: Doubao Case 05. Model=Doubao 2.0 pro, Input_token=166 tokens, CT=33255 tokens, Amp=200.33 x.

D.8 ERNIE



Figure D.32: ERNIE Case 01. Model=ERNIE 5.0 Thinking, Input_token=196 tokens, CT=43029 tokens, Amp=219.54 x, Time=1005s.



Figure D.33: ERNIE Case 02. Model=ERNIE 5.0 Thinking, Input_token=185 tokens, CT=42885 tokens, Amp=231.81 x, Time=952s.



Figure D.34: ERNIE Case 03. Model=ERNIE 5.0 Thinking, Input_token=196 tokens, CT=43642 tokens, Amp=222.66 x, Time=962s.



Figure D.35: ERNIE Case 04. Model=ERNIE 5.0 Thinking, Input_token=196 tokens, CT=44276 tokens, Amp=225.90 x, Time=1027s.