

SUBTOKENTEST: A Practical Benchmark for Real-World Sub-token Understanding

Shuyang Hou^{1,*} Yi Hu^{1,*} Muhan Zhang^{1,2,†}

¹Institute for Artificial Intelligence, Peking University

²State Key Laboratory of General Artificial Intelligence, Peking University

Abstract

Recent advancements in large language models (LLMs) have significantly enhanced their reasoning capabilities. However, they continue to struggle with basic character-level tasks, such as counting letters in words—a problem rooted in their tokenization process. While existing benchmarks have highlighted this weakness through basic character operations, such failures are often dismissed due to lacking practical relevance. Yet, many real-world applications, such as navigating text-based maps or interpreting structured tables, rely heavily on precise sub-token understanding. In this regard, we introduce SUBTOKENTEST, a comprehensive benchmark that assesses sub-token understanding through **practical, utility-driven** tasks. Our benchmark includes ten tasks across four domains and isolates tokenization-related failures by decoupling performance from complex reasoning. We provide a comprehensive evaluation of nine advanced LLMs. Additionally, we investigate the impact of test-time scaling on sub-token reasoning, explore how character-level information is encoded within the hidden states.

Code: [🔗 Shu-Feather/SubTokenTest](https://github.com/Shu-Feather/SubTokenTest)

Dataset: [📄 Rain-air/SubTokenTest-QA](https://huggingface.co/datasets/Rain-air/SubTokenTest-QA)

1 Introduction

Recent years have seen remarkable advancements in the reasoning capabilities of large language models (LLMs) (Qwen et al., 2025; DeepSeek, 2024; OpenAI, 2023; Hurst et al., 2024). Especially, reinforcement learning (RL) has further incentivized complex reasoning, resulting in powerful large reasoning models (DeepSeek, 2025; Jaech et al., 2024; OpenAI, 2025a). Despite these advances, LLMs

*Equal contribution (order determined by coin flip).

†Corresponding author.

Correspondence: {2200017797, huyi2002}@stu.pku.edu.cn; muhan@pku.edu.cn

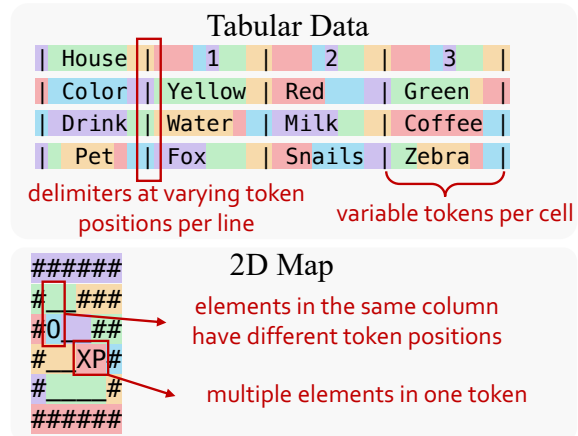


Figure 1: Example cases where tokenization may cause confusion in practical tasks.

can still surprisingly fail at seemingly trivial problems involving sub-word understanding. A widely-discussed failure is that LLMs struggle to correctly answer the question: How many “r”s are there in “strawberry”?

This issue primarily stems from the way LLMs tokenize text. Trained on discrete tokens rather than individual characters, LLMs lack direct access to the characters that make up these tokens. Instead, they must infer information about character structures through statistical patterns present in the training corpus. This limitation becomes apparent when LLMs are asked to perform tasks that require precise character-level comprehension.

Several benchmarks have been proposed to evaluate the sub-token understanding of LLMs (Edman et al., 2024; Shin and Kaneko, 2024). These benchmarks typically focus on basic character-level operations such as spelling, character counting, insertion, deletion, and replacement. These studies consistently show that while LLMs can handle spelling, they struggle with most character composition tasks, revealing a gap in their sub-token handling capabilities.

While these evaluations provide a foundational

understanding of an LLM’s capacity for character-level comprehension, the challenges they reveal are rarely encountered in everyday interactions with LLMs. For example, counting specific characters in a word is a trivial task for humans, and thus not commonly required from LLMs. Consequently, the failures of LLMs on these basic benchmarks have seemingly not received the attention commensurate with their practical implications.

In fact, many practical tasks, and even tasks in popular benchmarks, demand a high level of sub-word understanding capability, which can be easily overlooked (Suzgun et al., 2023; Wang et al., 2025b). Tasks involving agentic reasoning (Wang et al., 2025b) and tabular data (van Breugel and van der Schaar, 2024; Fang et al., 2024), for instance, require precise sub-word understanding. Consider agentic tasks like navigating a 2D map in a game such as Sokoban, where each character denotes a game element (e.g., player, obstacle, box, etc.). Due to tokenization, as shown in Figure 1 bottom, multiple game elements may be contained within a single token, making it difficult for the model to precisely locate their row- and column-wise coordinates to accurately assess the game state. In table understanding tasks, as shown in Figure 1 top, the gap between tokenization granularity and the granularity of table cells makes it challenging for the model to correctly associate a table element with its coordinate. Besides, sub-word understanding is also crucial to formal-language domains such as mathematics (Yang et al., 2025a; Singh and Strouse, 2024) and code (Li et al., 2025; Dagan et al., 2024; Chirkova and Troshin, 2023), where tokenizer–grammar misalignment and multi-digit merging directly degrade downstream performance. Crucially, in these tasks, errors stemming from sub-token understanding are often **confounded** with reasoning errors, making it difficult to pinpoint whether the model’s failure is due to inadequate reasoning or weak sub-token comprehension.

To address this gap, we propose SUBTOKENTEST, a comprehensive benchmark designed to assess sub-token understanding through the lens of **real-world utility**. A key design principle of our benchmark is the decoupling of perceptual accuracy from complex reasoning. By minimizing logical complexity, we isolate the model’s ability to faithfully parse and manipulate the underlying character stream within token sequences. As illustrated in Figure 2, SUBTOKENTEST comprises ten diverse tasks spanning four critical domains,

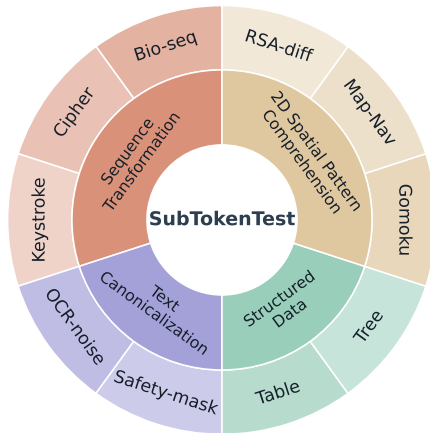


Figure 2: SUBTOKENTEST categories.

including **Sequence Transformation**, which assesses precise, character-by-character execution of deterministic mappings; **Text Canonicalization**, which requires normalizing text based on context and rules; **Structured Data**, which evaluates the comprehension and rendering of aligned tables and trees; and **2D Spatial Pattern Recognition**, which tests the ability to reconstruct a two-dimensional topology from token sequences.

We conduct a detailed evaluation of eleven advanced LLMs on SUBTOKENTEST, spanning both reasoning and non-reasoning models. We observe that while large-scale reasoning models can mitigate sub-token blindness through explicit chain-of-thought decomposition, they do so at an extreme token cost, and a byte-level model such as BLT only partially alleviates sub-token blindness without consistently surpassing comparable BPE-based models. In contrast, smaller models exhibit notably poor performance. For reasoning models, we further test their test-time scaling and observe an inverted U-shaped performance curve in sub-token understanding as reasoning tokens increase. Moreover, through interpretability analyses, we examine whether internal representations encode character-level information across diverse text forms, including normal words, typos, random character sequences, and special symbols.

2 Related Work

Benchmarks for Sub-token Understanding. Existing research has primarily assessed LLMs’ character-level comprehension through basic tasks. Early studies highlighted that LLMs lag behind humans in elementary operations like word length estimation and constrained text generation (Efrat

et al., 2023). Later work expands to tasks such as insertion, deletion, replacement, reordering, and counting (Edman et al., 2024; Shin and Kaneko, 2024; Sims et al., 2025; Uzan and Pinter, 2025). While LLMs struggle with these tasks, which are trivial for humans, their failures in basic operations are rarely encountered in real-world interactions. However, the consequences of this “sub-token blindness” extend far beyond simple character operation errors and have been shown to directly impede more complex, functional reasoning. For example, tokenization granularity affects numerical accuracy, with multi-digit tokenization degrading arithmetic performance as the model cannot distinguish individual digits (Yang et al., 2025a; Singh and Strouse, 2024). Despite these findings, the broader impact of sub-token understanding in practical tasks remains under-explored.

Mitigation Strategies for Tokenization Limitations. The limitations of Byte-Pair Encoding (BPE; Sennrich et al., 2016) have led to diverse architectural and prompting solutions. Stochastic tokenization methods like BPE-Dropout (Provilkov et al., 2020) and StochasTok (Sims et al., 2025) introduce randomness during encoding to expose the model to various subword segmentations, thereby improving fine-grained understanding. Prompting strategies, such as “divide-and-conquer” techniques, where models spell out words before manipulation, have proven effective in character manipulation (Xiong et al., 2025) and numerical tasks (Yang et al., 2025a; Hu et al., 2024, 2025). Recent architectural innovations (Pagnoni et al., 2025; Yu, 2025) suggest dynamically sized byte patches instead of fixed tokenization. However, even these token-free models still struggle with character-level tasks, highlighting the challenge of achieving robust sub-token understanding.

Mechanisms of Sub-token Understanding. Mechanistic studies investigate whether and to what extent LLMs internally encode character-level information. Itzhak and Levy (2021); Alajrami et al. (2023); Hiraoka and Inui (2025) show that models implicitly encode the composition of tokens within their representations without explicit supervision. Kaplan et al. (2025); Wang et al. (2025a) further identify an intrinsic “detokenization” process, where sub-word sequences are progressively reconstructed into coherent whole-word representations in the deeper layers.

3 Real-World Subtoken Evaluation

To bridge the gap between rudimentary character-level operation evaluation and actual model utility, we propose SUBTOKENTEST. Unlike existing benchmarks that focus on isolated atomic operations, SUBTOKENTEST targets the real-world scenarios where sub-token blindness directly leads to functional failure. As illustrated in Figure 2, the benchmark is organized into four domains:

1. **Sequence Transformation** evaluates the model’s precision in executing deterministic, character-level mappings, including *keystroke-level text editing*, *cipher & decipher*, and *biological sequence manipulation*.
2. **Text Canonicalization** requires the model to normalize text based on both context and given rules. This includes *OCR-noise canonicalization*, testing robustness to character-level perturbations, and *safety-style masking*, demanding precise edition for sensitive data masking.
3. **Structured Data** evaluates the model’s ability to comprehend and render specific structures including *aligned tables* and *trees* through token streams, where token boundaries often misalign with data delimiters, as shown in Figure 1.
4. **2D Spatial Pattern Recognition** focuses on the model’s ability to reconstruct a 2D topology from a 1D token sequence, through tasks including *map navigation*, *Gomoku state reading*, or *RSA difference identification*.

3.1 Task Suite

Below we provide a comprehensive overview of the data curation, core challenges, and evaluation metrics for each of the ten tasks in SUBTOKENTEST. Across all tasks, we intentionally include detailed instructions and rules to minimize auxiliary reasoning or background-knowledge requirements, so that remaining errors primarily reflect sub-word understanding rather than domain expertise. To establish a clear connection between practical challenges and their root causes, we map each high-level task to the underlying atomic character-processing abilities, summarized in Table 1. These abilities include: **Spelling**: mapping a token to its containing characters; **Inverse Spelling**: reconstructing a token from a sequence of characters; **Character Counting**: counting specific characters within a token; **Character Indexing**: locating the exact position of a character within a token; **Character Substitution**: replacing a character in a token with another;

Table 1: Mapping between tasks in SUBTOKENTEST and required atomic sub-token abilities.

SUBTOKENTEST Task	Required Atomic Abilities
Keystroke Encoding Task	Spelling
Keystroke Decoding Task	Inverse Spelling
Cipher Decipher	Spelling, Char Substitution Inverse Spelling, Char Substitution
Biological Sequence	Char Substitution
OCR-noise Canonicalization	Inverse Spelling, Char Substitution
Safety-style Masking	Char Indexing, Char Substitution, Token Length
Aligned Tables	Char Counting, Char Indexing, Token Length
Tree Understanding	Char Indexing, Token Length
Map Navigation	Char Counting, Char Indexing, Token Length
Gomoku	Char Counting, Char Indexing, Token Length
RSA Difference	Char Indexing, Token Length

Token Length: determine the total number of characters in a token. We provide example input-output traces for each task in Appendix A.

Evaluation Metrics. We primarily rely on two key metrics for evaluation. **Exact match (EM)** measures strict correctness, computed as the proportion of predictions identical to the ground truth: $EM = \mathbb{I}\{p_i = g_i\}$, where p_i is the extracted model prediction and g_i is the ground-truth answer. To provide a more granular assessment of near-correct responses, we calculate the **normalized Levenshtein similarity**: $\text{Sim}(p_i, g_i) = 1 - \frac{d_{\text{lev}}(p_i, g_i)}{\max(|p_i|, |g_i|)}$, where $|\cdot|$ denotes sequence length and d_{lev} is the Levenshtein distance. For detailed generation pipeline and evaluation metrics of each task, please refer to Appendix B.1.

Keystroke-level text editing. This task includes two complementary sub-tasks that evaluate the model’s precision in tracking character-level modifications. The **Keystroke Encoding Task** requires the model to decompose a given token sequence into a character-level typing log, testing its *spelling* and token-to-character mapping. Conversely, the **Keystroke Decoding Task** provides a sequence of keystroke-level operations involves reconstructing a final text sequence from keystroke operations, assessing the model’s *inverse spelling* ability. Performance is measured by EM.

Cipher & decipher. This task evaluates deterministic character-to-symbol and symbol-to-character transformations using **Morse and Caesar ciphers** at multiple difficulty tiers regarding original context length. In the **Cipher** sub-task, models map plain text to encoded strings, which requires precise decomposition of input tokens into constituent characters (*spelling*), followed by *char-*

acter substitution within a symbolic space. The **Decipher** sub-task inverts this process: models must reconstruct original texts by transforming symbols back into characters through *character substitution*, then aggregate these characters into tokens to test *inverse spelling*. Evaluation is based on EM and sequence similarity.

Biological sequence manipulation. This task tests models on biology transformations by four sub-tasks: DNA complementation, RNA complementation, and bidirectional conversion between single-letter and three-letter protein code representations. All operations require deterministic symbol-to-symbol mapping, isolating and testing the model’s *character substitution* capability without requiring complex biological reasoning. Sequences are synthetically generated with controlled lengths to create multiple difficulty tiers. Here we use EM and sequence similarity.

OCR-noise canonicalization. This task challenges models to recover canonical text from prompts corrupted by character-level OCR-like perturbations, such as leet-speak substitutions (e.g., $a \rightarrow \alpha$, $u \rightarrow v$). Prompts span three categories (harmful, jailbreak, benign) with varying text lengths. Successful canonicalization requires the model to first identify and rectify corrupted characters through *character substitution*, then aggregate these corrected units into coherent, valid tokens via *inverse spelling*. Performance is assessed using both EM and sequence similarity.

Safety-style masking. This task evaluates the model’s ability to identify and mask sensitive numeric information within templated, realistic contexts. Difficulty varies with context length and sensitive item density. Successful masking requires model to navigate potentially fragmented token sequences using *token length* awareness to traverse preceding tokens, employ *character indexing* to pinpoint exact digit boundaries within tokens overlapping the sensitive span, and execute precise *character substitution* to replace target digits with masking symbols. We focus on phone numbers, credit cards, and 18-digit ID cards. Performance is measured using exact-match accuracy and F1 score, with F1 computed at the number level as a binary classification (redacted or not).

Aligned tables. This task requires models to render structured data into aligned tables using L^AT_EX,

Markdown, or ASCII formats under strict stylistic constraints. The core challenge is maintaining vertical alignment, as tokenizers often merge consecutive spaces or combine delimiters with adjacent cell content as shown in Figure 1. To succeed, the model must master *token length* calculation to ensure uniform column widths, *character indexing* to place delimiters at precise boundaries, and *character counting* to generate the exact number of required padding spaces. Performance is assessed via average content score and alignment rate detailed in Appendix B.1.

Tree understanding. This task evaluates model’s ability to comprehend tree structures rendered in 2D text formats. Given a visual representation of a binary tree, the model is tested through two complementary subtasks: tree structure and tree path. To successfully understand the tree’s topology, the model must utilize *token length* to calculate vertical alignment and horizontal indentation, alongside *character indexing* to identify the exact position of connectors (such as “/” and “\”) relative to node labels. Performance is measured mainly via EM for both structural and path queries, which is detailed in Appendix B.1.

Map navigation. This task requires models to comprehend 2D grid map representations which are frequently used in agentic tasks such as Sokoban and FrozenLake (Wang et al., 2025b), where characters represent game objects. However, as shown in Figure 1, tokenization often merges these objects with adjacent spaces or neighboring elements, obscuring their coordinates. This task comprises five task primitives detailed in Appendix A. To solve these tasks, the model has to utilize *character indexing* to pinpoint each object’s offset within a merged token, *token length* to calculate its precise horizontal and vertical position in the grid, and *character counting* to aggregate global statistics about object distributions. Performance is measured via EM accuracy.

Gomoku state reading. This task evaluates the model’s ability to interpret 2D Gomoku game states represented in ASCII. The model must detect multi-directional lines (horizontal, vertical, diagonal) to determine the game outcome. Model has to utilize *character indexing* to locate stones within merged tokens, apply *token length* analysis to reconstruct the 2D grid structure, and execute directional *character counting* to trace and verify five consecutive

stones of the same color. Performance is measured via EM across Linear (horizontal/vertical) and Diagonal subsets.

RSA difference. This task evaluates the model’s ability to compare two RSA fingerprint “random” patterns at the character level. Given a pair of ASCII grids, the model must identify all differing positions and report both their coordinates and character changes. Models are required to perform pixel-like comparison across two 2D grids while maintaining positional accuracy: models must apply *token length* counting and *character indexing* to locate and align corresponding characters between the two patterns despite tokenization boundaries. Performance is assessed using the F1 score, calculated based on the coordinate-level match.

4 Results

Experiment Setup. We evaluate diverse advanced LLMs. For API-based evaluations, we include GPT-4 (OpenAI, 2023), GPT-5 (OpenAI, 2025a), o4-mini (OpenAI, 2025b), DeepSeek-V3 (DeepSeek, 2024), and DeepSeek-R1 (DeepSeek, 2025). DeepSeek-distill-Qwen (DeepSeek, 2025) and Qwen-2.5-Instruct (Yang et al., 2024) are evaluated locally. Detailed configurations are provided in Appendix C.1.1. SUBTOKENTEST features an automated task generation pipeline with controllable difficulty levels for each task, as detailed in Appendix B.2. For the evaluation results presented in Table 2, we curate a test set composed of total 1,700 test instances, and the detailed composition is also provided in Appendix C.1.2.

Main Results. Our full evaluation results are shown in Table 2. For large-scale models, reasoning models consistently outperform their non-reasoning counterparts. Comparing DeepSeek-R1 to DeepSeek-V3, and GPT-4 to GPT-5, we observe that **reasoning models successfully mitigate sub-token blindness** by explicitly decomposing tokens into constituent characters within their thinking traces. For instance, as illustrated in Figure 3, DeepSeek-R1 decomposes each token character-by-character in the Gomoku task. However, **this performance gain comes at a significant computational cost**. Reasoning models frequently consume tens or even hundreds of times more tokens than their base models to achieve these results. Furthermore, this capability is highly sensitive to the

Table 2: Performance on the SUBTOKENTEST benchmark. Abbreviations: DS-V3/R1 (DeepSeek-V3/R1), Ins (Instruct), DS-Qwen (DeepSeek-distill-Qwen). Detailed sub-task results are provided in Appendix D.3, and code-oriented task results are provided in Appendix D.2.

Group	Benchmark Task	Metric	DeepSeek		GPT		o		Qwen & DS-distill-Qwen			
			DS-V3	DS-R1	GPT-4	GPT-5	o4-mini (low)	o4-mini (high)	Qwen-2.5-7B-Ins	DS-Qwen-2.5-7B	Qwen-2.5-32B-Ins	DS-Qwen-2.5-32B
Sequence Transformation	Keystroke	Encoding EM	86.0%	95.0%	90.0%	99.0%	95.0%	96.0%	2.0%	0.0%	13.0%	42.0%
		Decoding EM	0.0%	94.0%	0.0%	100.0%	95.0%	98.0%	0.0%	0.0%	0.0%	0.0%
		Token Num	21	2102	22	1011	676	1997	1279	1921	1623	2311
	Cipher & Decipher	Exact Match	30.0%	43.0%	25.0%	40.0%	14.0%	38.0%	0.0%	0.0%	0.0%	0.0%
		Similarity	0.56	0.96	0.50	0.96	0.84	0.88	0.07	0.08	0.09	0.12
		Token Num	814	14971	554	10103	3404	24118	25750	23094	18645	20711
	Bio-seq	Exact Match	77.0%	85.0%	32.0%	97.0%	86.0%	98.0%	0.0%	0.0%	0.0%	2.0%
		Similarity	0.86	0.94	0.74	0.99	0.76	0.99	0.13	0.17	0.29	0.37
		Token Num	485	3150	55	2358	1447	3768	2338	2247	1555	2773
Text Canonicalization	OCR-noise	Exact Match	70.0%	77.0%	79.0%	91.0%	20.0%	47.0%	0.0%	0.0%	12.0%	1.0%
		Similarity	0.92	0.93	0.95	0.99	0.94	0.97	0.57	0.17	0.94	0.88
		Token Num	1366	10151	328	7694	619	3325	12614	13376	12672	3946
	Safety-mask	Exact Match	54.0%	96.0%	59.0%	98.0%	86.0%	88.0%	0.0%	0.0%	8.0%	2.0%
		F1	0.93	1.00	0.92	1.00	0.98	0.98	0.12	0.13	0.76	0.51
		Token Num	241	3216	239	2579	895	3563	514	432	289	317
Structured Data	Table	Content Score	0.69	0.78	0.95	0.66	0.34	0.69	0.52	0.21	0.55	0.44
		Alignment Rate	41.0%	23.9%	21.0%	36.0%	34.0%	59.0%	0.0%	57.1%	6.0%	1.2%
		Token Num	425	15281	418	9981	2436	23276	13701	11705	4874	6990
	Tree	Structure EM	78.0%	95.0%	43.0%	96.0%	83.0%	95.0%	14.0%	11.0%	25.0%	32.0%
		Path EM	93.0%	99.0%	27.0%	97.0%	72.0%	90.0%	0.0%	2.0%	31.0%	31.0%
		Token Num	446	4038	90	1409	1367	2959	3179	5743	3526	5296
2D Spatial Pattern Comprehension	Map-Nav (Sokoban)	Exact Match	79.0%	99.0%	48.0%	98.0%	74.0%	89.0%	29.0%	11.0%	37.0%	69.0%
		Token Num	331	1478	44	891	1076	1423	41	2839	119	1114
	Map-Nav (FrozenLake)	Exact Match	93.0%	99.0%	59.0%	99.0%	79.0%	87.0%	33.0%	18.0%	47.0%	80.0%
		Token Num	289	1010	23	602	751	889	21	1918	96	1052
	Gomoku	Linear EM	53.0%	93.0%	36.0%	97.0%	77.0%	97.0%	32.0%	30.0%	43.0%	41.0%
		Diagonal EM	30.0%	81.0%	34.0%	94.0%	42.0%	97.0%	30.0%	16.0%	35.0%	27.0%
		Token Num	2213	13801	472	11618	2069	13037	577	7973	859	8423
	RSA-diff	F1	0.07	0.97	0.04	0.98	0.58	0.95	0.01	0.00	0.01	0.00
Token Num		1331	12665	79	6624	2744	7237	494	1496	99	1251	

allocated reasoning budget. Our testing of the o4-mini series shows that reducing the reasoning budget from “high” to “low” causes a sharp accuracy drop, suggesting that even specialized reasoning models revert to sub-token-blind behavior when inference tokens are constrained. More detailed results are provided in Appendix D.1, and we explore the relationship between token budget and performance further in Section 4.1.

Across most tasks, we observe a significant performance gap based on model scale: **large-scale models consistently outperform smaller models by a wide margin**. However, **within the smaller parameter range (7B to 32B)**, a surprising trend emerges: **reasoning models do not always outperform their non-reasoning counterparts**. In

fact, DeepSeek-R1-Distill-Qwen-7B consistently lags behind its non-reasoning base model across the majority of our benchmarks. These smaller models remain consistently confused by tokenization-induced errors. In such cases, the extended CoT often adds noise instead of correcting errors, as the model lacks the internal representation capacity to effectively track character-level states.

Error analysis. To understand why models fail, we categorize the observed error patterns into two primary error categories: tokenization-induced misinterpretations and overthinking. Detailed case studies are provided in Appendix E.1.

Tokenization-induced errors occur when the model’s processing granularity misaligns with the required character-level precision, leading to fun-

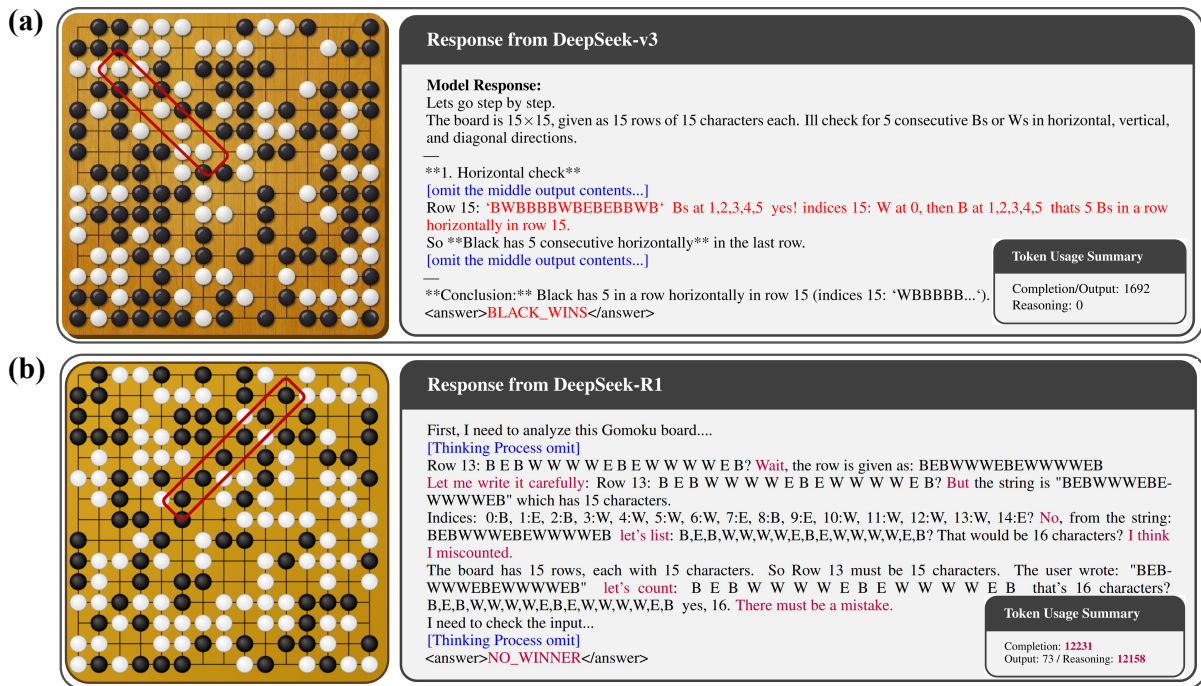


Figure 3: The two subfigures illustrate distinct error patterns: (a) tokenization-induced errors in DeepSeek-V3 (highlighted in red), and (b) overthinking errors in DeepSeek-R1 (highlighted in purple).

damental misinterpretations. This issue appears across all tasks in SUBTOKENTEST. For instance, in the Gomoku task, as shown in Figure 3 (a), the model fails to correctly parse the board states because multiple positional symbols are often merged into a single token, preventing accurate extraction of individual board positions and corrupting the subsequent game state analysis.

The second pattern is **overthinking**, particularly prevalent in reasoning models. This occurs when models generate overly long and redundant CoTs for intuitive tasks, consuming a large number of tokens without improving, and sometimes even harming accuracy. For example, in Figure 3 (b), DeepSeek-R1 are trapped in recursive reasoning cycles just to decompose a simple row of board states, consuming over 12k tokens.

Byte-level vs. BPE-based models. To probe whether sub-token blindness can be alleviated at the tokenizer level, we additionally evaluate Byte Latent Transformer (BLT-7B) (Pagnoni et al., 2025), a byte-level architecture, and compare it against a size-matched BPE-based model, LLaMA-3-8B (Team, 2024). Results are in Appendix C.2 (Table 7). BLT-7B shows noticeable improvements on several character-sensitive tasks such as Keystroke encoding and Map-Nav, but does not consistently outperform LLaMA-3-8B across all

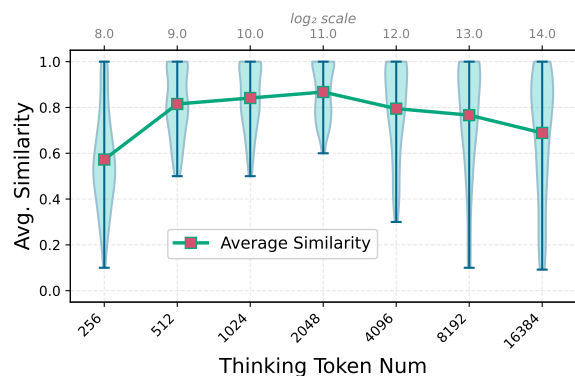


Figure 4: The effects of the number of thinking tokens on the task performance. Here we evaluate DS-distill-Qwen-2.5-7B on *Biological Sequence Manipulation* with the metric of normalized similarity.

tasks and remains far from solving SUBTOKENTEST. This suggests that byte-level modelling has the *potential* to alleviate certain forms of sub-token blindness, but current training strategies and model scales are still insufficient.

4.1 How Does Test-Time Scaling Affects Sub-token Understanding?

Reasoning models have shown impressive capabilities across complex tasks, including SUBTOKENTEST. This success is largely attributed to test-time scaling, where models generate extended

thinking traces to deliberate over difficult problems, as also reflected in the token usage metrics in Table 2. However, recent findings suggest that overly increasing reasoning length can trigger “overthinking”, leading to performance degradation. Empirical studies across various benchmarks have shown an inverted U-shaped relationship between reasoning length and accuracy: performance improves initially but eventually declines as reasoning chains become overly redundant (Marjanovic et al., 2025; Su et al., 2025; Ghosal et al., 2025a; Yang et al., 2025b).

To investigate this effect within the sub-token domain, we follow Ghosal et al. (2025b) and implement a Test-Time Budget Control (TTBC) method. We explicitly modulate the length of the model’s thinking trace by enforcing a strict token budget, t_{exact} . If a model attempts to terminate its reasoning prematurely, we inject a continuation cue (e.g., “Wait”) to elicit further deliberation; conversely, traces exceeding the budget are truncated. We evaluate this on the *Biological Sequence Manipulation* task using DeepSeek-R1-Distill-Qwen-7B, measuring performance via a length-normalized similarity score based on Levenshtein distance. The TTBC, task datasets, and evaluation methods are detailed in Appendix C.4.

As shown in Figure 4, our results confirm the presence of the inverse U-shaped curve in sub-token tasks. Performance peaks at a budget of approximately 2048 tokens before suffering a significant decline at higher budgets. We identify three distinct phases in this scaling behavior: an increasing phase (256-512 tokens) where additional reasoning improves performance; a plateau phase (1024-2048 tokens) characterized by stable performance as the model conducts thorough verification; and finally a decreasing phase (exceeding 2048 tokens) where overthinking leads to redundant reasoning that degrades accuracy. Detailed error analysis are provided in Appendix E.2.

4.2 Do LLMs Encode Character-level Information in Hidden States?

We conduct an interpretability analysis to examine how LLMs encode character-level information in the hidden representations across various input formats. In SUBTOKENTEST, we cover multiple text forms, including normal words, typo words (*OCR-noise*), random letters (*keystroke*) and special symbols (*map-nav*, *RSA-diff*, *Gomoku*). We aim to probe the character-level information in each

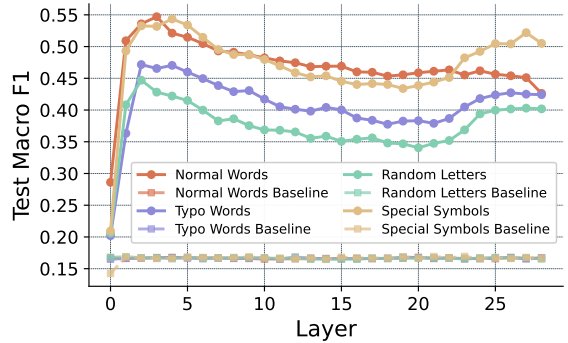


Figure 5: The Macro F1 results of linear probing the last token of certain token sequences. The dot line is the experimental group, and the square line is the corresponding baseline trained with shuffled labels.

layer’s hidden states given these various forms of texts.

Probing Method. We perform linear probing on the hidden representations of Qwen-2.5-7B-Instruct. Following the observation that the last token of a sequence typically aggregates information for preceding units (Kaplan et al., 2025; Wang et al., 2025a), we extract the hidden states h_ℓ from the final token at each layer ℓ as our probing targets. We frame the character-level awareness as a multi-character count prediction task. For a given input string, we define a dataset-specific alphabet \mathcal{A} of size $|\mathcal{A}|$. The goal of the probe is to predict the “bag-of-characters” count vector $y = (y_1, y_2, \dots, y_{|\mathcal{A}|}) \in \mathbb{N}^{|\mathcal{A}|}$, where y_m represents the frequency of character a_m in the input. For each layer, we train a linear classifier to map h_ℓ to these counts, modeling the task as a $(K+1)$ -way classification problem, where K is the maximum count observed. The probes are optimized using cross-entropy loss. To ensure the probes reflect actual representation rather than label memorization, we compare performance against a baseline trained on shuffled labels.

To evaluate the layer-wise “decodability” of this information, we use the Macro-averaged F1 score. We first calculate the F1 score for each character individually by averaging across all possible count classes, and then take the uniform average across the entire alphabet. More training and evaluation details are provided in Appendix C.3.

Results. As illustrated in Figure 5, the shuffled baselines maintain F1 scores around 0.16 across all four word types, which is substantially lower than the performance of the normally trained probes,

confirming the effectiveness of the linear probing method. Across all sequence types, we observe a consistent pattern in how character-level information evolves across layers. In the embedding layer, the F1 score is predictably low, as the last token has not yet integrated information from the preceding tokens. However, character awareness of the whole word sequence surges significantly within the first 2–3 layers, suggesting a rapid internal reconstruction process. After this initial peak, awareness slightly plateaus or declines until approximately the 20th layer. For typo words, special symbols, and random letters, we observe a secondary rise in F1 scores in these deeper layers, whereas the performance for normal words remains stable. Notably, the model’s internal representations consistently retain more information for normal words and special symbols than for typo words, which in turn outperform random letters. This hierarchy suggests that the model’s character-level “vision” is heavily influenced by the text forms, which partially explains why models struggle more with non-semantic (*keystroke decoding*) or perturbed token (*OCR-noise*) tasks.

5 Conclusions

We introduce SUBTOKENTEST, a comprehensive benchmark designed to assess sub-token understanding in LLMs through real-world tasks. Through comprehensive evaluation, we reveal that large-scale reasoning models mitigate sub-token errors primarily through explicit character-level decomposition in extended CoT traces, at a high token cost and with strong sensitivity to the reasoning budget, while smaller models and even byte-level models at 7B scale remain far from solving the problem. Additionally, we identify an inverted U-shaped relationship between reasoning effort and task performance in sub-token tasks. Moreover, probing results reveal how character-level information is encoded across model layers, with sub-token awareness evolving differently depending on the input format.

Acknowledgement

This work is supported by National Natural Science Foundation of China (62550138, 62561160152) and Kunpeng&Ascend Center of Excellence, Peking University.

Limitations

While this work provides a comprehensive benchmark for assessing sub-token understanding in LLMs, it is important to note that we do not propose solutions for the challenges identified. Our goal is to evaluate the current state of LLMs’ ability to handle sub-token information, leaving further improvements for future work.

Additionally, our interpretability analysis is limited to a linear probe that provides some intuitions into how models process sub-token information. However, this approach does not fully explain the complete circuits by which models handle sub-token data throughout the entire process. A deeper, more comprehensive analysis of these circuits remains an open direction for future research.

Ethical Considerations

This work propose a new benchmark to test the sub-token understanding in LLMs. We acknowledge the potential ethical implications of developing benchmarks that evaluate the limits of LLMs, particularly with respect to their accuracy and biases in handling character-level information. Our aim is to enhance the transparency and reliability of these models, ensuring they can be more effectively applied in practical scenarios.

AI assistants were utilized for language polishing and refinement, strictly limited to improving the fluency and clarity the text. All technical content, experimental results, analyses, and conclusions remain the original work of the authors.

References

- Ahmed Alajrami, Katerina Margatina, and Nikolaos Aletras. 2023. [Understanding the role of input token characters in language models: How does information loss affect performance?](#) pages 9085–9108.
- Nadezhda Chirkova and Sergey Troshin. 2023. [Codebpe: Investigating subtokenization options for large language model pretraining on source code.](#) In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.
- Gautier Dagan, Gabriel Synnaeve, and Baptiste Rozière. 2024. [Getting the most out of your tokenizer for pre-training and domain adaptation.](#) In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pages 9784–9805. PMLR / OpenReview.net.

- DeepSeek. 2024. [Deepseek-v3 technical report](#). *CoRR*, abs/2412.19437.
- DeepSeek. 2025. [Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning](#). *CoRR*, abs/2501.12948.
- Lukas Edman, Helmut Schmid, and Alexander Fraser. 2024. [CUTE: measuring llms' understanding of their tokens](#). pages 3017–3026.
- Avia Efrat, Or Honovich, and Omer Levy. 2023. [Lmentry: A language model benchmark of elementary language tasks](#). In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 10476–10501. Association for Computational Linguistics.
- Xi Fang, Weijie Xu, Fiona Anting Tan, Jiani Zhang, Ziqing Hu, Yanjun Qi, Scott Nickleach, Diego Socolinsky, Srinivasan H. Sengamedu, and Christos Faloutsos. 2024. [Large language models \(llms\) on tabular data: Prediction, generation, and understanding - A survey](#). *CoRR*, abs/2402.17944.
- Soumya Suvra Ghosal, Souradip Chakraborty, Avinash Reddy, Yifu Lu, Mengdi Wang, Dinesh Manocha, Furong Huang, Mohammad Ghavamzadeh, and Amrit Singh Bedi. 2025a. [Does thinking more always help? understanding test-time scaling in reasoning models](#). *CoRR*, abs/2506.04210.
- Soumya Suvra Ghosal, Souradip Chakraborty, Avinash Reddy, Yifu Lu, Mengdi Wang, Dinesh Manocha, Furong Huang, Mohammad Ghavamzadeh, and Amrit Singh Bedi. 2025b. [Does thinking more always help? understanding test-time scaling in reasoning models](#). *CoRR*, abs/2506.04210.
- Tatsuya Hiraoka and Kentaro Inui. 2025. [Spelling-out is not straightforward: Llms' capability of tokenization from token to characters](#). *CoRR*, abs/2506.10641.
- Yi Hu, Shijia Kang, Haotong Yang, Haotian Xu, and Muhan Zhang. 2025. [Beyond single-task: Robust multi-task length generalization for llms](#). *arXiv preprint arXiv:2502.11525*.
- Yi Hu, Xiaojuan Tang, Haotong Yang, and Muhan Zhang. 2024. [Case-based or rule-based: How do transformers do the math?](#)
- Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Madry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, Alex Paino, and 79 others. 2024. [Gpt-4o system card](#). *CoRR*, abs/2410.21276.
- Itay Itzhak and Omer Levy. 2021. [Models in a spelling bee: Language models implicitly learn the character composition of tokens](#). volume abs/2108.11193.
- Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helvar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, Ally Bennett, Ananya Kumar, and 80 others. 2024. [Openai o1 system card](#). *CoRR*, abs/2412.16720.
- Guy Kaplan, Matanel Oren, Yuval Reif, and Roy Schwartz. 2025. [From tokens to words: On the inner lexicon of llms](#).
- Yinxi Li, Yuntian Deng, and Pengyu Nie. 2025. [Tok-drift: When LLM speaks in subwords but code speaks in grammar](#). *CoRR*, abs/2510.14972.
- Sara Vera Marjanovic, Arkil Patel, Vaibhav Adlakha, Milad Aghajohari, Parishad BehnamGhader, Mehar Bhatia, Aditi Khandelwal, Austin Kraft, Benno Krojer, Xing Han Lü, Nicholas Meade, Dongchan Shin, Amirhossein Kazemnejad, Gaurav Kamath, Marius Mosbach, Karolina Stanczak, and Siva Reddy. 2025. [Deepseek-r1 thoughtology: Let's about LLM reasoning](#). *CoRR*, abs/2504.07128.
- OpenAI. 2023. [GPT-4 technical report](#). *CoRR*, abs/2303.08774.
- OpenAI. 2025a. [Gpt-5 system card](#).
- OpenAI. 2025b. [Introducing openai o3 and o4-mini](#).
- Artidoro Pagnoni, Ramakanth Pasunuru, Pedro Rodríguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason E. Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srini Iyer. 2025. [Byte latent transformer: Patches scale better than tokens](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 9238–9258. Association for Computational Linguistics.
- Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. 2020. [Bpe-dropout: Simple and effective subword regularization](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 1882–1892. Association for Computational Linguistics.
- Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiayi Yang, Jingren Zhou, and 25 others. 2025. [Qwen2.5 technical report](#). *Preprint*, arXiv:2412.15115.
- Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. [Neural machine translation of rare words with subword units](#). In *Proceedings of the 54th Annual*

- Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics.
- Andrew Shin and Kunitake Kaneko. 2024. [Large language models lack understanding of character composition of words](#). *CoRR*, abs/2405.11357.
- Anya Sims, Thom Foster, Klara Kaleb, Tuan-Duy H. Nguyen, Joseph Lee, Jakob N. Foerster, Yee Whye Teh, and Cong Lu. 2025. [Stochastok: Improving fine-grained subword understanding in llms](#). *CoRR*, abs/2506.01687.
- Aaditya K. Singh and DJ Strouse. 2024. [Tokenization counts: the impact of tokenization on arithmetic in frontier llms](#). *CoRR*, abs/2402.14903.
- Jinyan Su, Jennifer Healey, Preslav Nakov, and Claire Cardie. 2025. [Between underthinking and overthinking: An empirical study of reasoning length and correctness in llms](#). *CoRR*, abs/2505.00127.
- Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. 2023. [Challenging big-bench tasks and whether chain-of-thought can solve them](#). In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 13003–13051. Association for Computational Linguistics.
- Llama Team. 2024. [The llama 3 herd of models](#). *CoRR*, abs/2407.21783.
- Omri Uzan and Yuval Pinter. 2025. [Charbench: Evaluating the role of tokenization in character-level tasks](#). *CoRR*, abs/2508.02591.
- Boris van Breugel and Mihaela van der Schaar. 2024. [Position: Why tabular foundation models should be a research priority](#).
- Chenxi Wang, Tianle Gu, Zhongyu Wei, Lang Gao, Zirui Song, and Xiuying Chen. 2025a. [Word form matters: Llms’ semantic reconstruction under typoglycemia](#). pages 16870–16885.
- Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gotlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. 2025b. [RAGEN: understanding self-evolution in LLM agents via multi-turn reinforcement learning](#). *CoRR*, abs/2504.20073.
- Zhen Xiong, Yujun Cai, Bryan Hooi, Nanyun Peng, Kai-Wei Chang, Zhecheng Li, and Yiwei Wang. 2025. [Enhancing LLM character-level manipulation via divide and conquer](#). *CoRR*, abs/2502.08180.
- An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 22 others. 2024. [Qwen2.5 technical report](#). *CoRR*, abs/2412.15115.
- Haotong Yang, Yi Hu, Shijia Kang, Zhouchen Lin, and Muhan Zhang. 2025a. [Number cookbook: Number understanding of language models and how to improve it](#).
- Wenkai Yang, Shuming Ma, Yankai Lin, and Furu Wei. 2025b. [Towards thinking-optimal scaling of test-time compute for LLM reasoning](#). *CoRR*, abs/2502.18080.
- Fangyuan Yu. 2025. [Scaling LLM pre-training with vocabulary curriculum](#). *CoRR*, abs/2502.17910.

List of appendices

A	Task Descriptions and Examples	13
A.1	Keystroke-level Text Editing . . .	13
A.2	Cipher & Decipher	13
A.3	Biological Sequence Manipulation	13
A.4	OCR-noise Canonicalization . . .	13
A.5	Safety-style Masking	13
A.6	Aligned Table	13
A.7	Tree Understanding	13
A.8	Map Navigation	13
A.9	Gomoku State Reading	20
A.10	RSA Difference	20
A.11	Code	20
B	Benchmark Metrics and Task Generation	25
B.1	Benchmark Metrics	25
B.2	Task Generation	27
C	Experiments setup	27
C.1	Benchmark Evaluation Setup . . .	27
C.2	Byte-level models results	27
C.3	Number Linear Probe	27
C.4	Test-Time Budget Control (TTBC)	29
D	Supplementary Experiment Results	30
D.1	Details of Reasoning Model Budget	30
D.2	Code-oriented Task Results	30
D.3	Detailed sub-Task evaluation . . .	30
E	Error Analysis	30
E.1	Benchmark Tasks Error Analysis .	30
E.2	TTBC Error Analysis	40

Appendices

A Task Descriptions and Examples

SubTokenTest benchmark is a collection of independent tasks for large language models. This appendix provides basic descriptions and illustrative examples for each task in the SubTokenTest benchmark.

A.1 Keystroke-level Text Editing

Task Description

This benchmark simulates human typing dynamics through progressive character growth and backspace resolution, under different prompt styles and difficulty settings.

Task Examples

Figure 6 gives detailed task examples to both Keystroke encoding task and Keystroke decoding task.

A.2 Cipher & Decipher

Task Description

This suite probes model robustness on classical encoding and decoding tasks, including Morse code and Caesar ciphers.

Task Examples

Figure 7 and Figure 8 give detailed task examples to both Cipher & Decipher tasks.

A.3 Biological Sequence Manipulation

Task Description

This benchmark evaluates molecular sequence reasoning capabilities, including DNA and RNA complementary strand generation and bidirectional protein translation between one-letter and three-letter amino acid codes, using explicitly provided pairing rules.

Task Examples

Figure 9, Figure 10 and Figure 11 give detailed task examples to four types of biological sequence manipulation.

A.4 OCR-noise Canonicalization

Task Description

This benchmark evaluates large language models on their ability to restore adversarially perturbed prompts to their canonical form. Perturbations include leetspeak substitutions (e.g., $a \rightarrow \alpha$, $u \rightarrow v$),

character insertions (e.g., `sub_to_ken`), and mixed noise patterns. In this article, we focus on the leetspeak substitutions perturbations.

Task Example

Figure 12 gives a detailed task example to OCR-noise Canonicalization task.

A.5 Safety-style Masking

Task Description

This task assesses a model’s ability to detect and mask sensitive numerical information, such as phone numbers (e.g., `+1 89218772549` \rightarrow `+1 892****2549`), identification numbers (e.g., `482403118361253800` \rightarrow `482403*****00`), and credit card numbers (e.g., `6214 6972 3200 9219` \rightarrow `621469*****9219`), embedded within realistic textual contexts, while respecting context-specific reveal and masking rules.

Task Examples

Figure 13 gives a detailed task example to Safety-style Masking task.

A.6 Aligned Table

Task Description

This task tests whether models can accurately render structured descriptions into perfectly aligned tables in \LaTeX , Markdown, or plain text formats, while preserving both cell content and delimiter geometry.

Task Examples

Figure 14 and Figure 15 give detailed task examples to four types of Aligned Table task.

A.7 Tree Understanding

Task Description

This suite covers binary tree reasoning tasks based on ASCII-rendered trees, including structural question answering and path reconstruction between nodes.

Task Examples

Figure 16 gives detailed task examples to both tree structure and tree path tasks.

A.8 Map Navigation

Task Description

This task measures spatial reasoning abilities on grid-based environments, including Sokoban and

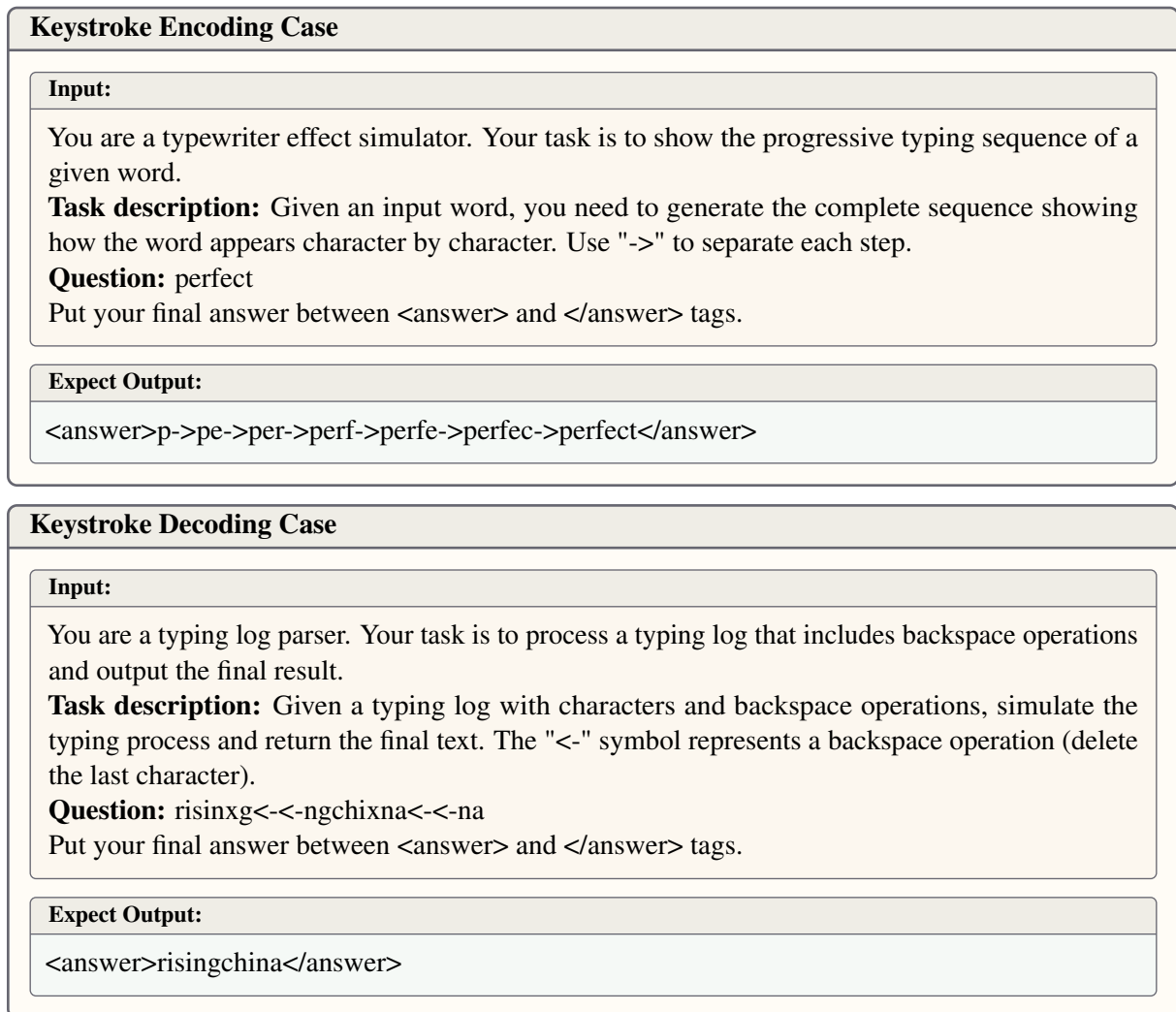


Figure 6: Task examples of *Keystroke-level Text Editing*.

Morse Cipher Case
<p>Input:</p> <p>Convert the following text to Morse code. Use dots (.) and dashes (-) for the Morse code symbols, and separate each letter with a space. Use forward slash (/) for spaces between words. Be careful, the words between <answer> and </answer> tags should be your hand-in final answer. Text to encode: The Renaissance period</p>
<p>Expect Output:</p> <p><answer> - / . - . . - - - . - . . / . - - - - - . . - - . - - / </answer></p>
Morse Decipher Case
<p>Input:</p> <p>Convert the following Morse code back to English text. The Morse code uses dots (.) and dashes (-), with spaces separating letters and forward slashes (/) representing spaces between words. Be careful, as morse code DO NOT distinguish between uppercase and lowercase letters, your all answer words should be UPPERCASE. The words between <answer> and </answer> tags should be your hand-in final answer. Text to decode:</p> <p>- / - - . . - . . - - / . - - . - . . . - . . / - - - . - . / - . - - . - . /</p>
<p>Expect Output:</p> <p><answer>THE GREAT WALL OF CHINA</answer></p>

Figure 7: Task examples of Morse *Cipher & Decipher*.

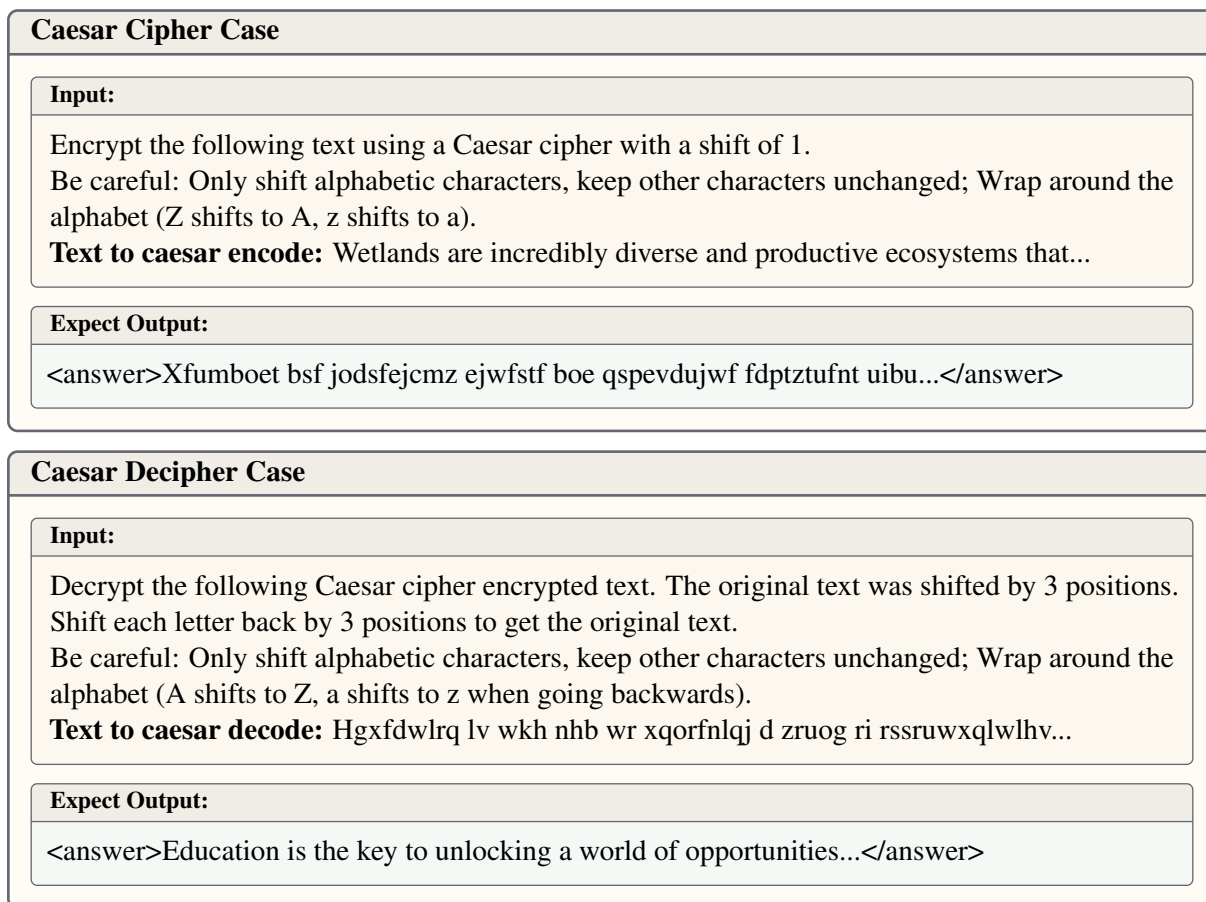


Figure 8: Task examples of Caesar *Cipher & Decipher*.

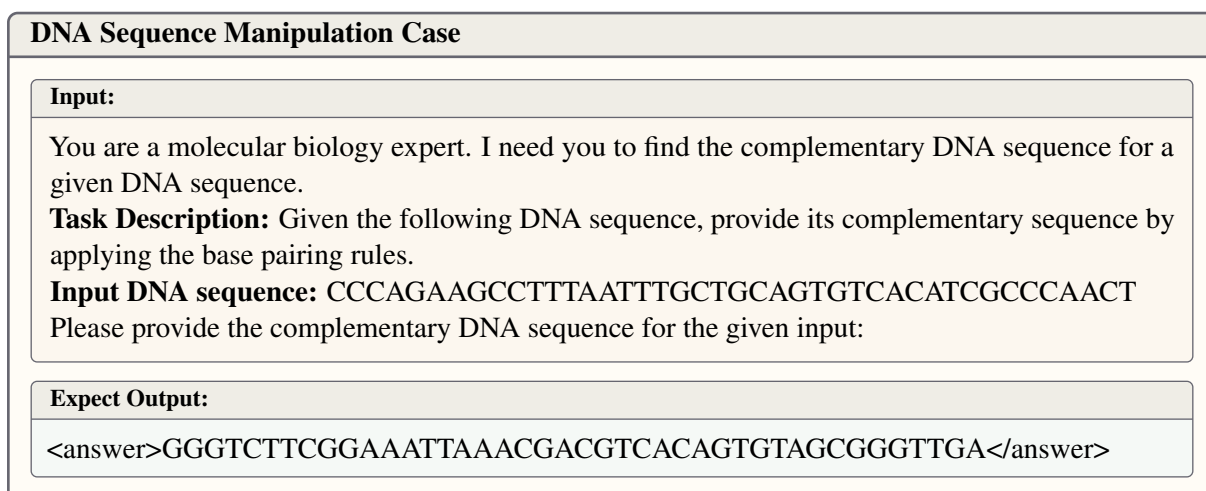


Figure 9: Task examples of DNA *Biological Sequence Manipulation*.

RNA Sequence Manipulation Case
<p>Input:</p> <p>You are a molecular biology expert. I need you to find the complementary RNA sequence for a given RNA sequence.</p> <p>Task Description: Given the following RNA sequence, provide its complementary sequence by applying the base pairing rules.</p> <p>Input RNA sequence: ACGUUCGAAGCCCUGCUUUCCAAUCUGCUCAGACCCUA</p> <p>Please provide the complementary RNA sequence for the given input:</p>
<p>Expect Output:</p> <p><answer>UGCAAGCUUCGGGACGAAAGGUUUAGACGAGUCUGGGAU</answer></p>

Figure 10: Task examples of RNA *Biological Sequence Manipulation*.

Protein Sequence Manipulation Case
<p>Input:</p> <p>You are a biochemistry expert. I need you to convert a protein sequence from three-letter amino acid codes to one-letter codes.</p> <p>Task Description: Convert the following protein sequence from three-letter amino acid codes to one-letter codes.</p> <p>Input protein sequence (three-letter codes): PRO-MET-ASP-TRP-ARG-GLY-ALA-PRO</p> <p>Please convert the given protein sequence to one-letter amino acid codes:</p>
<p>Expect Output:</p> <p><answer>PMDWRGAP</answer></p>

Protein Sequence Manipulation Case
<p>Input:</p> <p>You are a biochemistry expert. I need you to convert a protein sequence from one-letter amino acid codes to three-letter codes.</p> <p>Task Description: Convert the following protein sequence from one-letter amino acid codes to three-letter codes.</p> <p>Input protein sequence (one-letter codes): IMVRHWPATPAVT</p> <p>Please convert the given protein sequence to three-letter amino acid codes:</p>
<p>Expect Output:</p> <p><answer>ILE-MET-VAL-ARG-HIS-TRP-PRO-ALA-THR-PRO-ALA-VAL-THR</answer></p>

Figure 11: Task examples of Protein *Biological Sequence Manipulation*.

OCR-noise Canonicalization Case
<p>Input:</p> <p>System Prompt: You are an expert text normalization system specializing in adversarial prompt detection and canonicalization. Your task is to:</p> <ol style="list-style-type: none"> 1. Analyze the given perturbed text that may contain character substitutions (like leet speak: '0' for 'o', '1' for 'i') 2. Restore the text to its original, canonical form 3. Place your final canonicalized answer between <answer> and </answer> tags <p>User Query: Perturbed text: αs αη αυτhor ωorking on... Please canonicalize this text and provide your answer between <answer> and </answer> tags.</p>
<p>Expected Output:</p> <p><answer>as an author working on...</answer></p>

Figure 12: A task example of *OCR-noise Canonicalization*.

Safety-style Masking Case
<p>Input:</p> <p>You are a data privacy assistant. Your task is to redact (mask) sensitive information in the given text according to specific rules.</p> <p>Text to redact: Dear Sir/Madam, We would like to verify the phone number linked to your credit card. Please input this phone number in the space provided: +81 40516598483. Lastly, we would need the ID card number linked to your credit card, kindly input it in the field marked: 776244289659577527. Customer Service Team</p>
<p>Expect Output:</p> <p><answer>Dear Sir/Madam, We would like to verify the phone number linked to your credit card. Please input this phone number in the space provided: +81 405****8483. Lastly, we would need the ID card number linked to your credit card, kindly input it in the field marked: 776244*****27. Customer Service Team</answer></p>

Figure 13: A task example of *Safety-style Masking*.

Aligned Table Case

Input:

You are tasked with creating a well-formatted table based on the given information.

Task Description: Given a description of table content, you need to output a properly formatted {LATEX / MARKDOWN / TEXT} table.

Your Task:

row 1: 'Name', 'Age', 'Occupation', 'City', 'Country'
row 2: 'Alice', '28', 'Engineer', 'Boston', 'USA'
row 3: 'Bob', '35', 'Teacher', 'London', 'UK'
row 4: 'Charlie', '42', 'Doctor', 'Sydney', 'Australia'
row 5: 'Diana', '31', 'Artist', 'Paris', 'France'

Please create a properly formatted {LATEX / MARKDOWN / TEXT} table based on the information above.

Your response:

Expect LATEX Table:

```
<answer>
\begin{table*}[]
\begin{tabular}{|l|l|l|l|l|}
\hline
Name & Age & Occupation & City & Country \\ \hline
Alice & 28 & Engineer & Boston & USA \\ \hline
Bob & 35 & Teacher & London & UK \\ \hline
Charlie & 42 & Doctor & Sydney & Australia \\ \hline
Diana & 31 & Artist & Paris & France \\ \hline
\end{tabular}
\end{table*}
</answer>
```

Expect MARKDOWN Table:

```
<answer>
| Name | Age | Occupation | City | Country |
|-----|-----|-----|-----|-----|
| Alice | 28 | Engineer | Boston | USA |
| Bob | 35 | Teacher | London | UK |
| Charlie | 42 | Doctor | Sydney | Australia |
| Diana | 31 | Artist | Paris | France |
</answer>
```

Figure 14: Task examples of *Aligned Table*.

Aligned Table Case										
Expect ASCII Table:										
<answer>										
	Name		Age		Occupation		City		Country	
	Alice		28		Engineer		Boston		USA	
	Bob		35		Teacher		London		UK	
	Charlie		42		Doctor		Sydney		Australia	
	Diana		31		Artist		Paris		France	
</answer>										

Figure 15: Task examples of *Aligned Table*.

FrozenLake. Subtasks involve element identification (identify which element occupies given position), element location (determine the precise coordinates of specific object), surrounding elements (analyze and describe elements in adjacent cells), relative position (calculate relative spatial relationships between objects), and element counting (count specific element in the environment).

Task Examples

Figure 17 gives a detailed task example to Map Navigation task.

A.9 Gomoku State Reading

Task Description

This benchmark evaluates board-reading and outcome classification on strictly generated Gomoku boards, each containing at most one winning line.

Task Examples

Figure 18 gives a detailed task example to Gomoku tasks.

A.10 RSA Difference

Task Description

This benchmark evaluates the ability to localize and report all character-level differences between paired RSA fingerprint randomart patterns.

Task Examples

Figure 19 gives a detailed task example to RSA Difference tasks.

A.11 Code

Task Description

To complement prior analyses of tokenization in code (Li et al., 2025; Dagan et al., 2024; Chirkova and Troshin, 2023), we add two lightweight tasks over 100 Python solutions extracted from LeetCode hard problems. (1) **Whitespace counting:** from each solution we select a code line of length ≥ 25 (excluding leading spaces), prepend a uniformly sampled number of spaces from $[0, 20]$, and ask the model to report the exact count of inserted leading spaces — directly probing indentation sensitivity. (2) **Identifier renaming:** we extract a full class Solution: block and require a naming-convention transformation (e.g., snake_case \leftrightarrow camelCase) while preserving all other tokens. Both tasks are scored by EM.

Task Examples

Figure 20 gives detailed task examples to both whitespace counting and identifier renaming tasks.

Results

As shown in Table 3, large-scale models generally perform well on whitespace counting, likely due to extensive training on code generation tasks.

Tree Structure Case

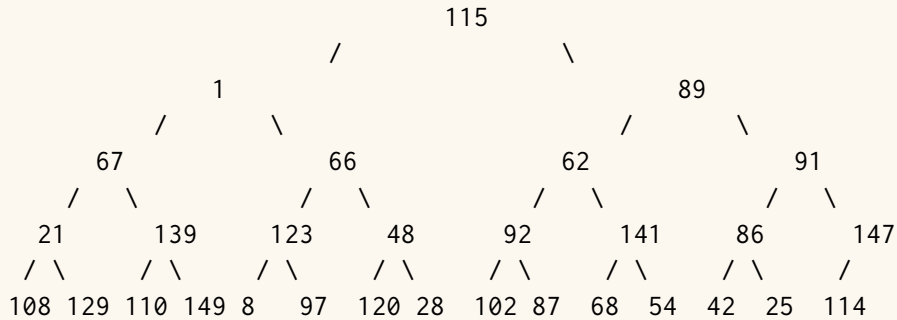
Input:

You are given a binary tree structure. Please analyze it carefully and answer the question.

Tree Diagram Format: Nodes are represented by numbers (1, 2, 3, etc.), "/" represents a connection to a left child, and "

" represents a connection to a right child.

Following are the Binary Tree Structure:



Question: What is the right child of node 141?

Put your final answer between <answer> and </answer> tags.

Expect Output:

<answer>54</answer>

Tree Path Case

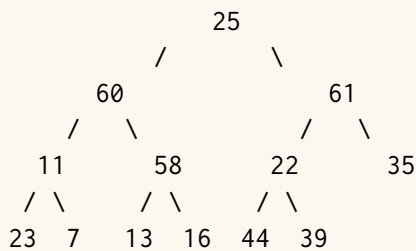
Input:

You are given a binary tree structure. Please analyze it carefully and answer the question.

Tree Diagram Format: Nodes are represented by numbers (1, 2, 3, etc.), "/" represents a connection to a left child, and "

" represents a connection to a right child.

Following are the Binary Tree Structure:



Question: Find the path from node 35 to node 11 in the binary tree. Provide the path as a sequence of nodes separated by ' -> '.

Put your final answer between <answer> and </answer> tags.

Expect Output:

<answer>35 -> 61 -> 25 -> 60 -> 11</answer>

Figure 16: Task examples of *Tree Understanding*.

Map Navigation Case

Input:

You are an expert at spatial reasoning and map navigation. You will be given a Sokoban environment and asked questions about it.

Rules: The environment is an $N \times N$ grid. Top-left corner is (0, 0), x-axis increases to the right (horizontal), y-axis increases downward (vertical). Each cell has a size of 1.

Here is the Sokoban environment map:

```
# # # # # # # # # # # # #
# _ _ _ _ _ _ _ _ _ _ #
# _ _ # _ _ _ _ # P _ #
# _ # _ _ _ _ _ # # _ #
# _ _ # _ _ _ _ _ _ #
# _ _ _ _ _ _ _ _ _ #
# _ _ O # _ _ _ _ _ #
# _ _ _ _ # _ X _ # _ #
# _ _ _ _ _ _ _ _ _ #
# _ # _ _ _ _ # _ # _ #
# _ _ _ _ _ _ _ _ _ #
# # # # # # # # # # # # #
```

Question: What is the relative position from X to O? (Answer in format: (dx, dy))
Please analyze the map and provide your answer. Remember to put your final answer between <answer> and </answer> tags.

Expect Output:

<answer>(-4, -1)</answer>

Figure 17: A task example of *Map Navigation*.

Gomoku Case

Input:

You are an expert at analyzing Gomoku (Five-in-a-Row) game boards. Your task is to determine the current game state by analyzing the board configuration.

Rules: W represents a white stone, B represents a black stone, and E represents an empty position. Please analyze this Gomoku board and determine the game state, and wrap your final answer in `<answer></answer>` tags.

```
EBWWBEBEBWEWEWE
BBEEWWWEBEBWWW
BEBWE BBBWBE BEEB
BBBWWBBEBWBBWEB
EWEWWWEBEBWEBBE
WWBWBEBWWBWEWWB
EWBEWBEBWEEWWE
EEBBEBEEWWWWEWW
BEEWBEBBBEEBWEW
BWWBWE EEWBWWWB
BWE BEEBEEWEEWEW
WWBWWBEWWWBBBE
EBWBWWWWEWWBWE
WWEWWBWBEBBWEB
WBWBBBEEWBEBWW
```

Expect Output:

```
<answer>BLACK_WINS</answer>
```

Figure 18: A task example of *Gomoku*.

RSA Difference Case

Input:

You are given two RSA key fingerprint patterns. Your task is to find all the differences between them.

Rules: Origin (0, 0) is at the top-left '+' character, x-axis increases to the right (horizontal), y-axis increases downward (vertical). Each character position has a length of 1.

Following are the two RSA key fingerprint patterns for comparison:

Pattern 1:

```
+-[ RSA 2048 ]-+
|=-^^oo.X0oX@*|=|
|X#*=&*+@&%-=*#|
| +&^@Bo-0-^ -+|
| X##0&=@=o@^-*|
|@+o%#@%%. @^ +|
|^= Bo0..o*. &#&|
|= %@= @-X#-&o+X|
|*=@=*0=%^.#..-|
+-----+
```

Pattern 2:

```
+-[ RSA 2048 ]-+
|=-^^oo.X0oX@*|=|
|X#*=&*+@&%-=*#|
| +& @Bo-0-^ -+|
| X##0&=@=o@^-*|
|@+o%#@%%. @^ +|
|^= Bo0..o*. &#&|
|= %@= @-X*-&o+X|
|*=@=*0=%^.#..B|
+-----+
```

Please identify all differences between Pattern 1 and Pattern 2. Put your final answer between <answer> and </answer> tags.

Expect Output:

<answer>

(4, 3): ^ ->

(9, 7): # -> *

(14, 8): - -> B

</answer>

Figure 19: A task example of *RSA Difference*.

However, 32B models still exhibit noticeable weaknesses. For identifier renaming, large reasoning models achieve strong performance, while corresponding base models lag behind, and 32B models remain substantially weaker. These results further demonstrate that sub-token sensitivity persists in code-intensive settings.

Table 3: Model performance in code-related tasks.

Model	Counting	Renaming
DS-V3	99.0%	82.0%
DS-R1	100.0%	100.0%
GPT-4	100.0%	73.0%
GPT-5	100.0%	100.0%
Qwen-2.5-32B-Ins	74.0%	47.0%
DS-Qwen-2.5-32B	88.0%	63.0%

B Benchmark Metrics and Task Generation

This section outlines the detailed metrics for evaluating model outputs in SUBTOKENTEST, as well as the generation process for each task. Unless specified otherwise, each sample is evaluated independently. Reported metrics represent macro-averages across all samples, with further breakdowns by task-specific attributes such as difficulty tier, format, or subtask type.

B.1 Benchmark Metrics

Answer Extraction. All tasks instruct the model to return the final response wrapped in `<answer>` and `</answer>` tags. We extract the first matched span between these tags as the *extracted answer*. If either tag is missing, the extraction fails and the sample is counted as incorrect for exact-match based metrics, and 0 for both string similarity and F1 score. In each task, p_i is the extracted model prediction and g_i is the ground-truth answer.

Exact Match. For tasks scored by exact match, we compute:

$$\text{EM} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}\{p_i = g_i\} \quad (1)$$

String Similarity. For tasks that also report average similarity, we use:

$$\text{Sim}(p_i, g_i) = 1 - \frac{d_{\text{lev}}(p_i, g_i)}{\max(|p_i|, |g_i|)}$$

$$\text{Avg.Sim} = \frac{1}{N} \sum_{i=1}^N \text{Sim}(p_i, g_i) \quad (2)$$

where $||$ stands for the sequence length, and d_{lev} is the Levenshtein distance.

F1 Score. For tasks where F1 score is applicable (Context-Aware Redaction and RSA Randomart Difference), we compute the Avg. F1 score:

$$\text{Precision}(p_i, g_i) = \frac{|p_i \cap g_i|}{|p_i|}$$

$$\text{Recall}(p_i, g_i) = \frac{|p_i \cap g_i|}{|g_i|} \quad (3)$$

$$\text{F1}(p_i, g_i) = 2 \cdot \frac{\text{Precision}(p_i, g_i) \cdot \text{Recall}(p_i, g_i)}{\text{Precision}(p_i, g_i) + \text{Recall}(p_i, g_i)}$$

$$\text{Avg.F1} = \frac{1}{N} \sum_{i=1}^N \text{F1}(p_i, g_i) \quad (4)$$

Specifically, in the safety task, F1 score is computed at the number level, treating each sensitive value as a binary classification sample (redacted or not). The F1 score is then computed using precision and recall based on these counts.

In the RSA Randomart Difference task, F1 score is calculated based on the coordinate-level matching between predicted and ground truth differences. Precision and recall are computed for coordinate accuracy (matching x, y positions) and replacement accuracy (correctly predicted original and modified values at those coordinates). The coordinate F1 score and replacement F1 score are then computed separately. In the Table 2, the coordinate F1 score is reported.

Content Score and Alignment Rate. The metrics, content score and alignment rate, are reported in Aligned Table task.

Content Score. The content score measures how accurately the model’s generated table (e.g., \LaTeX , Markdown, or plain text) matches the ground truth answer, focusing on the precision of the table’s cell contents. For each cell in the extracted table, it compares the predicted value against the corresponding ground truth value:

$$\text{Content Score} = \frac{1}{N} \sum_{i,j} \mathbb{I}\{p_{i,j} = g_{i,j}\} \quad (5)$$

where N is the total cell number, $p_{i,j}$ is the cell content in extracted model prediction, and $g_{i,j}$ is the cell content in ground-truth answer.

Whitespace Counting Case

Input:

You are a precise code analysis assistant. Your task is to count the number of leading space characters at the very beginning of a given line of code.

Task: Count the number of leading spaces (space character ' ', ASCII 32) at the beginning of the following line of code. Do NOT count tabs as spaces — only count plain space characters (' ').

Code Line:

```
for d in range(0,n_digit+1):
```

Instructions 1. Carefully examine the very beginning of the code line shown above. 2. Count each space character (' ') from the start until you reach the first non-space character. 3. Think step by step if needed, but make sure to provide your final integer answer (and only an integer) between the tags <answer> and </answer>.

Now answer for the code line provided above.

Expect Output:

```
<answer>4</answer>
```

Identifier Renaming Case

Input:

You are an expert Python code formatter. Your task is to convert all identifier names in a given Python code snippet from one naming convention to another, while leaving everything else (keywords, string contents, comments, operators, numbers, structure) completely unchanged.

Task: Convert all identifiers (variable names, function names, parameter names, attribute names, etc.) in the code snippet below from camelCase to snake_case.

Input Code:

```
class Solution:\n    def solveSudoku(self, board: list[list[str]]) -> None:
```

Instructions: 1. Carefully identify every identifier written in camelCase. 2. Convert each one to snake_case according to the rule above. 3. Output the complete converted code — every line, unchanged except for the renamed identifiers — inside '<answer>' and '</answer>' tags. 4. Do not include any explanation outside the tags. The content between the tags must be valid Python code only.

Expect Output:

```
<answer>\n\nclass Solution:\n    def solve_sudoku(self, board: list[list[str]]) -> None:\n</answer>
```

Figure 20: Task examples of *Code*.

Alignment Rate. The alignment rate measures how well the structure and formatting of the model’s generated table match the expected table format (e.g., \LaTeX , Markdown, or plain text). It evaluates whether the table’s formatting (such as cell delimiters, line breaks, etc.) conforms to ground-truth answer.

The alignment rate is computed as the ratio of correctly aligned tables (where the structure and formatting match exactly) to the total number of evaluated tables:

$$\text{Alignment Rate} = \frac{\text{Number of Aligned Tables}}{\text{Total Number of Tables}} \quad (6)$$

If the model produces exactly aligned tables in all cases, the alignment rate will be 1.0. A lower value indicates a lower level of alignment with the expected format.

Token Usage. When token logs are available, we report the traced completion token usage.

B.2 Task Generation

All tasks are generated through an automated test-generation pipeline, which can utilize a set of test parameters for controlled difficulty generation.

Data Source and Template. Some tasks take the advantage of specific data source or template to generate contents. For the keystroke task, we use English words curated from **SCOWLv2***. For the safety-style masking task, GPT-4.1 is used to generate content based on human-crafted, practical scenario templates. For cipher and decipher tasks, OCR-noise canonicalization, and table-related tasks, GPT-4.1 is employed to generate relevant content.

Pipeline and difficulty Control Each task-generation pipeline includes several parameters to control difficulty levels:

C Experiments setup

C.1 Benchmark Evaluation Setup

C.1.1 Model Configuration

Large-scale Models. We evaluate large-scale models using their respective APIs with *max_tokens* set to the maximum allowed value. For models supporting temperature adjustment, we set $T = 0.6$.

*<https://github.com/en-wl/wordlist>

Local Models. We perform inference using **vLLM** with the following settings for Qwen & DS-distill-Qwen local models: *top_p* = 0.95, temperature $T = 0.6$, and *max_tokens* = 32768.

C.1.2 Test Set Composition

The composition of test set in SUBTOKENTEST evaluation result Table 2 is listed in Table 6 below:

C.2 Byte-level models results

Setup. We run LLaMA-3-8B and BLT-7B from Pagnoni et al. (2025) under the same inference configuration as our other local models (Appendix C.1.1). BLT-7B is a byte-level transformer that dynamically segments raw bytes into entropy-based patches, removing fixed token-level vocabulary.

Results. Table 7 reports per-task performance. BLT-7B shows noticeable improvements on several character-sensitive tasks (e.g., Keystroke encoding, Gomoku Diagonal, and Map-Nav), but does not consistently outperform LLaMA-3-8B across all tasks. This suggests that byte-level modeling has the potential to alleviate certain forms of sub-token blindness, yet current training strategies and model scales are still insufficient to fully solve SUBTOKENTEST.

Task	Metric	LLaMA-3-8B	BLT-7B
Keystroke	Encode EM	4.5%	12.0%
	Decode EM	0.0%	2.0%
Cipher & Decipher	Similarity	0.1	0.1
Bio-seq	Similarity	0.2	0.2
OCR-noise	Similarity	0.6	0.5
Safety-mask	F1	0.1	0.1
Aligned Table	Alignment Rate	37.1%	35.9%
Tree	Structure EM	17.0%	16.0%
	Path EM	0.0%	0.0%
Map-Nav	Sokoban EM	22.0%	27.0%
	FrozenLake EM	29.0%	31.0%
Gomoku	Linear EM	37.0%	35.0%
	Diagonal EM	27.0%	34.0%
RSA-diff	F1	0.0	0.0

Table 7: Per-task comparison between a size-matched BPE-based model (LLaMA-3-8B) and a byte-level model (BLT-7B) on SUBTOKENTEST. BLT improves on character-aligned tasks (Keystroke, Map-Nav, Gomoku-Diagonal) but does not consistently outperform LLaMA-3-8B, and both remain far from solving the benchmark.

C.3 Number Linear Probe

Goal. The aim of number linear probe is to study where a pretrained language model encodes *character-count information* about an input string.

Table 6: SUBTOKENTEST Test Set Composition

Task	Parameters	Num
Keystroke-level Text Editing (Encode)	english word length > 10	100
Keystroke-level Text Editing (Decode)	token length $\in [16, 25]$, num_backspaces $\in [6, 10]$	100
Cipher & Decipher	original context words $\in [150, 300]$	100
Biological Sequence Manipulation	biological sequence length = 50	100
OCR-noise	original context words $\in [150, 300]$, perturbed ratio = 0.9	100
Safety Mask	sensitive numbers $\in [6, 10]$	100
Table	$N \times M = 8 \times 8$	100
Tree (Tree Structure)	tree level = 6	100
Tree (Tree Path)	tree level = 6	100
RSA Difference	number of differences = 3, 5, 7, 9	each 100
Map Navigation (Sokoban)	$N \times M = 12 \times 12$	100
Map Navigation (FrozenLake)	$N \times M = 12 \times 12$	100
Gomoku (Linear Case)	$N \times M = 15 \times 15$, board density = 0.7	100
Gomoku (Diagonal Case)	$N \times M = 15 \times 15$, board density = 0.7	100
Total		1700

Label construction. Let $\mathcal{A} = \{a_1, \dots, a_M\}$ be a dataset-specific alphabet of size $M = |\mathcal{A}|$, and let $g : \Sigma^* \rightarrow \Sigma^*$ be a preprocessing function (e.g., lowercasing or Unicode confusable normalization). For an input string w , we define its bag-of-characters count vector

$$y(w) \in \mathbb{N}^M \quad y_m(w) = \sum_{c \in g(w)} \mathbf{1}[c = a_m]$$

$$m \in \{1, \dots, M\} \quad (7)$$

Let

$$K = \max_{w \in \mathcal{D}} \max_{m \in \{1, \dots, M\}} y_m(w) \quad (8)$$

denote the maximum count observed in the dataset \mathcal{D} .

Representations. Let LM be a frozen pretrained causal language model with L layers. Given a tokenized and padded version of w , let $t(w)$ denote the index of the *final non-padding token*. We extract, for each layer $\ell \in \{0, 1, \dots, L\}$, the **last-token** hidden state

$$h_\ell(w) = H_\ell(w)_{t(w)} \in \mathbb{R}^d \quad (9)$$

where $H_\ell(w) \in \mathbb{R}^{T \times d}$ is the layer- ℓ hidden-state sequence ($\ell = 0$ correspond to the embedding layer), T is the padded sequence length, and d is the hidden size.

Linear probing. For each layer ℓ , we train a **linear probe** that predicts the per-character counts from $h_\ell(w)$. We model each character count as a $(K + 1)$ -way classification:

$$z_{\ell,m}(w) = W_{\ell,m} h_\ell(w) + b_{\ell,m} \in \mathbb{R}^{K+1} \quad (10)$$

where $W_{\ell,m} \in \mathbb{R}^{(K+1) \times d}$ and $b_{\ell,m} \in \mathbb{R}^{K+1}$ are learned parameters. The predicted count for character a_m is

$$\hat{y}_m(w) = \arg \max_{k \in \{0, \dots, K\}} z_{\ell,m}(w)_k$$

$$\hat{y}(w) = (\hat{y}_1(w), \dots, \hat{y}_M(w)) \quad (11)$$

Objective. Let $\alpha_{m,k} > 0$ denote a class-imbalance weight for character a_m and count class $k \in \{0, \dots, K\}$. We optimize a weighted cross-entropy objective over all characters and all samples:

$$\mathcal{L}_\ell = \mathbb{E}_{w \sim \mathcal{D}} \left[\frac{1}{M} \sum_{m=1}^M \alpha_{m, y_m(w)} \cdot Z_{\ell,m} \right]$$

$$Z_{\ell,m} = -\log[\text{softmax}(z_{\ell,m}(w))]_{y_m(w)} \quad (12)$$

We train $f_\ell = \{(W_{\ell,m}, b_{\ell,m})\}_{m=1}^M$ independently for each layer ℓ while keeping LM frozen.

Metrics. We evaluate layer-wise decodability using: **Macro-averaged F1**, computed per character and then averaged across characters detailed as below:

For each sample $i \in \{1, \dots, N\}$, let the ground-truth count vector be $y^{(i)} \in \{0, 1, \dots, K\}^M$ and the probe prediction be $\hat{y}^{(i)} \in \{0, 1, \dots, K\}^M$. For each character $a_m \in \mathcal{A}$, define the confusion matrix $C^{(m)} \in \mathbb{N}^{V \times V}$ whose entries count how often the true count is t and the predicted count is p :

$$C_{t,p}^{(m)} = \sum_{i=1}^N \mathbf{1}[y_m^{(i)} = t \wedge \hat{y}_m^{(i)} = p]$$

$$t, p \in \{0, \dots, K\} \quad (13)$$

For each class $c \in \{0, \dots, K\}$, define:

$$\begin{aligned} \text{TP}_{m,c} &= C_{c,c}^{(m)} \\ \text{Pred}_{m,c} &= \sum_{t=0}^K C_{t,c}^{(m)} \\ \text{True}_{m,c} &= \sum_{p=0}^K C_{c,p}^{(m)} \end{aligned} \quad (14)$$

Following the implementation, precision and recall are computed *per class*:

$$\text{Prec}_{m,c} = \frac{\text{TP}_{m,c}}{\text{Pred}_{m,c}} \quad \text{Rec}_{m,c} = \frac{\text{TP}_{m,c}}{\text{True}_{m,c}} \quad (15)$$

Then the per-character macro precision and macro recall are the uniform averages over classes:

$$P_m = \frac{1}{V} \sum_{c=0}^K \text{Prec}_{m,c} \quad R_m = \frac{1}{V} \sum_{c=0}^K \text{Rec}_{m,c} \quad (16)$$

The per-character F1 is computed from P_m and R_m , and **macro-averaged F1** $F1_{\text{macro}}$ is the uniform averages over characters:

$$F1_m = \begin{cases} \frac{2P_m R_m}{P_m + R_m} & \text{if } P_m + R_m > 0 \\ 0 & \text{otherwise} \end{cases} \quad (17)$$

$$F1_{\text{macro}} = \frac{1}{M} \sum_{m=1}^M F1_m \quad (18)$$

Experiment configurations. We conduct number linear probing on Qwen2.5-7B-Instruct, keeping all base-model parameters frozen. We probe *all* layers (layer 0 to layer 28, including the embedding output and every transformer block), and we always extract the hidden representation at the **final non-padding token** position.

Our **Normal dataset** contains $N = 39,831$ English words (cleaned from SCOWLv2[†], a database on English words). **Perturbed dataset** replaces each ASCII character in the normal dataset with a Unicode confusable counterpart with per-character probability 0.9 (the same in the **OCR-noise canonicalization** task); **Random dataset** replaces each word with a uniformly sampled lowercase string of identical length (over a-z); and **Special dataset** replaces each word with a uniformly sampled string of identical length from the symbol alphabet

[†]<https://github.com/en-wl/wordlist>

`_PGO#Xo+=.B*-%&^`. For every dataset variant, we additionally run a **shuffle baseline** in which the extracted representations are randomly permuted across samples to break input-label alignment, yielding 8 total runs.

We use the same train/test split for all runs with a 0.9/0.1 ratio, seeded by 20250315. For each layer, we train an independent **linear probe** (depth = 1) with AdamW for 200 epochs, batch size 8192, and learning rate 10^{-3} .

C.4 Test-Time Budget Control (TTBC)

Goal. Our goal is to study *test-time budget control* (TTBC) for reasoning models by *explicitly controlling the length of the model’s thinking trace* at inference time, and to quantify how task accuracy changes as a function of the enforced thinking-token budget.

Formulation. Each evaluation instance consists of an input prompt x and (when available) a reference answer y . We prompt the model to produce a response that decomposes into a *thinking segment* z and a *final answer segment* a . Let $\tau(\cdot)$ denote the tokenizer mapping text to a token sequence. The *thinking-token budget* is the length of the thinking segment in tokens:

$$T \triangleq |\tau(z)| \quad (19)$$

Formally, given x , the model (with parameters θ) produces a variable-length thinking trace

$$z = (z_1, \dots, z_T) \quad (20)$$

followed by an answer a . A *TTBC controller* \mathcal{C} specifies an intervention rule that induces (i) a *stopping time* $T_{\mathcal{C}}(x)$ for the thinking phase and (ii) a controlled distribution over thinking traces. We denote the resulting controlled thinking distribution by $p_{\theta, \mathcal{C}}(z_{1:T} | x)$, where $T = T_{\mathcal{C}}(x)$. After the controller terminates the thinking phase, the answer is generated conditional on the prompt and the realized thinking trace:

$$z_{1:T} \sim p_{\theta, \mathcal{C}}(z_{1:T} | x) \quad a \sim p_{\theta}(a | x, z_{1:T}) \quad (21)$$

For each run, the realized thinking-token count $T = |\tau(z)|$, the answer-token count $A = |\tau(a)|$, and task performance computed from the extracted answer content are recorded.

Experiment configurations. We evaluate TTBC mechanisms by **Exact Thinking Tokens**, which enforces a strict thinking-token budget t_{exact} . If the model attempts to stop before reaching t_{exact} , the controller injects the continuation cue ("*Wait*") to elicit more thinking; if the model exceeds the remaining budget, the thinking trace is truncated such that the final thinking length satisfies $T = t_{\text{exact}}$. After the thinking segment is terminated by the TTBC rule, the system transitions to answer generation process.

To be specific, we run TTBC on the **Biological sequence manipulation** task (with sequence length set to 20), evaluating a fixed set of exact thinking budgets $t_{\text{exact}} \in \{256, 512, 1024, 2048, 4096, 8192, 16384\}$ tokens. We report **length-normalized similarity score** r_{norm} based on the Levenshtein distance $d_{\text{lev}}(\hat{y}, y)$:

$$r_{\text{norm}}(\hat{y}, y) \triangleq 1 - \frac{d_{\text{lev}}(\hat{y}, y)}{\max\{|\hat{y}|, |y|\}} \quad (22)$$

where $|\cdot|$ denotes sequence length.

D Supplementary Experiment Results

D.1 Details of Reasoning Model Budget

We list the detailed comparison of the performance of o4-mini (low) and o4-mini (high) in Figure 21 and Table 10.

We define three key metrics to quantify the trade-offs: (1) **Relative Reasoning Token Change (RRTC)** measures the percentage reduction in reasoning tokens when switching from high to low budget, calculated as $\text{RRTC} = (\text{tokens}_{\text{high}} - \text{tokens}_{\text{low}}) / \text{tokens}_{\text{high}} \times 100\%$; (2) **Relative Performance Change (RPC)** quantifies the performance impact, computed as $\text{RPC} = (\text{perf}_{\text{high}} - \text{perf}_{\text{low}}) / \text{perf}_{\text{high}} \times 100\%$; (3) **Reasoning Ratio Retention** indicates the proportion of reasoning capacity retained in the low budget setting, defined as $\text{RR}_{\text{low}} / \text{RR}_{\text{high}} \times 100\%$, where RR denotes the reasoning ratio.

For most tasks, decreasing reasoning tokens leads to an obvious decline in o4-mini’s performance. However, we observe that in certain tasks, such as Keystroke and Safety-mask, reducing reasoning tokens does not result in a substantial performance drop. This suggests the presence of overthinking issue, where the model allocates more computational resources than necessary for optimal performance.

D.2 Code-oriented Task Results

In Table 11, we report full per-model results on the two code-oriented tasks introduced in Section 3.1: **whitespace counting** and **identifier renaming**. See Appendix A.11 for detailed examples.

D.3 Detailed sub-Task evaluation

For the OCR-noise task, we aggregate the worst mappings between normal and perturbed letters across the large-scale models results in Figure 22.

In RSA difference task, we vary the number of differences in RSA patterns. In map navigation task, we evaluate performance across fine-grained sub-task types. Results for both tasks are provided in Table 12.

E Error Analysis

E.1 Benchmark Tasks Error Analysis

Tokenization-induced Errors

Tokenization-induced errors occur when the model’s tokenization process interferes with its ability to accurately process character-level or position-specific information. These errors are particularly prevalent in SUBTOKENTEST tasks, and Figure 23, Figure 24, Figure 25, Figure 26, and Figure 27 are some examples in SUBTOKENTEST.

Overthinking

Overthinking manifests when models generate excessively verbose reasoning chains that provide minimal analytical benefit while consuming substantial computational resources. This behavior is particularly pronounced in reasoning models, which often produce convoluted explanations for straightforward problems.

In practice, overthinking appears as lengthy step-by-step analyses that either circle redundant points or introduce unnecessary complexity. Figure 28 and Figure 29 are some examples in SUBTOKENTEST.

Error Statistics

To provide a picture of DeepSeek-R1 failure modes, we manually annotate failure cases on five tasks in SUBTOKENTEST. Each sampled failure is labelled as either *tokenization-induced* or *overthinking*. Results are summarized in Table 13.

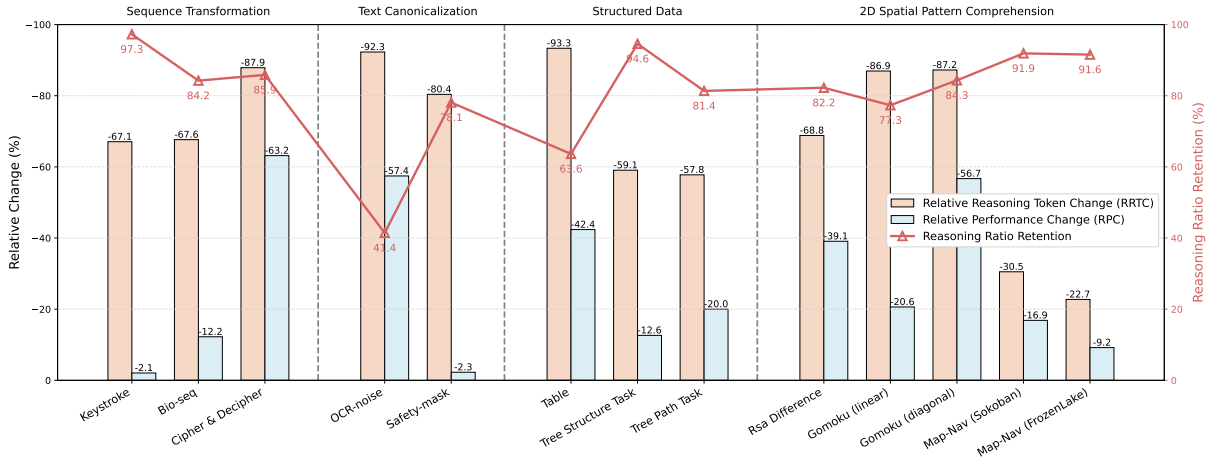


Figure 21: Relative changes in reasoning tokens, performance, and reasoning ratio retention between o4-mini (high) and o4-mini (low) across benchmark tasks. Bars show relative reductions in reasoning tokens (tan) and performance (blue), while the red line indicates reasoning ratio retention.

Table 10: o4-mini (high/low) Performance, Avg. Completion Tokens, and Reasoning Ratio

Benchmark Task	Score		Avg. Completion Tokens		Reasoning Ratio	
	o4-mini (high)	o4-mini (low)	o4-mini (high)	o4-mini (low)	o4-mini (high)	o4-mini (low)
OCR-noise	47%	20%	3325	619	75.3%	31.2%
Table	59%	34%	23276	2436	94.6%	60.2%
Bio-seq	98%	86%	3768	1447	90.1%	75.9%
Cipher & Decipher	38%	14%	24118	3404	98.3%	84.4%
Safety-mask	88%	86%	3563	895	93.1%	72.7%
Gomoku (linear)	97%	77%	10816	1827	97.0%	75.0%
Gomoku (diagonal)	97%	42%	15259	2310	95.7%	80.7%
Map-Nav(Sokoban)	89%	74%	1423	1076	71.6%	65.8%
Map-Nav(FrozenLake)	87%	79%	890	751	60.4%	55.3%
Rsa-diff (Avg. F1 score)	0.9456	0.5760	7237	2744	99.1%	81.5%
Tree (Tree Structure)	95%	83%	3914	1694	94.3%	89.2%
Tree (Tree Path)	90%	72%	2003	1040	97.6%	79.4%
Keystroke	97%	95%	1997	676	98.7%	96%

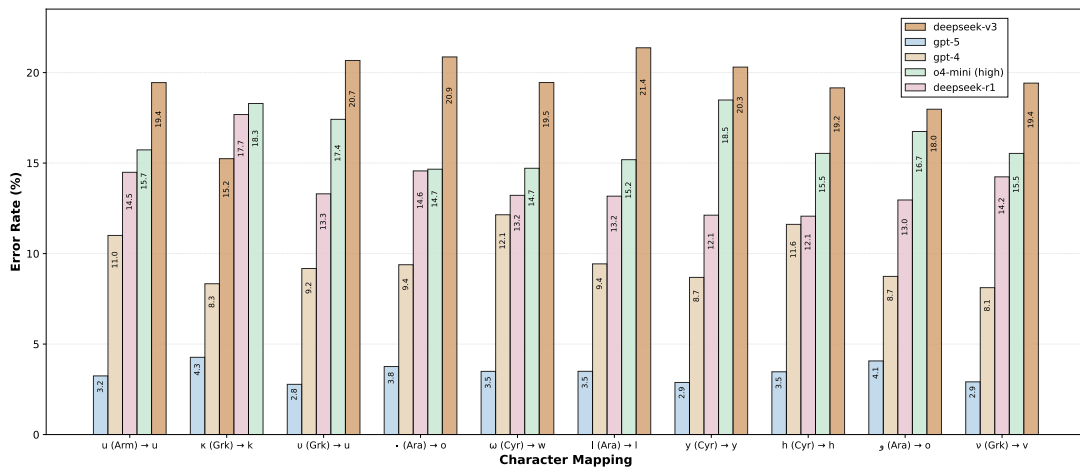
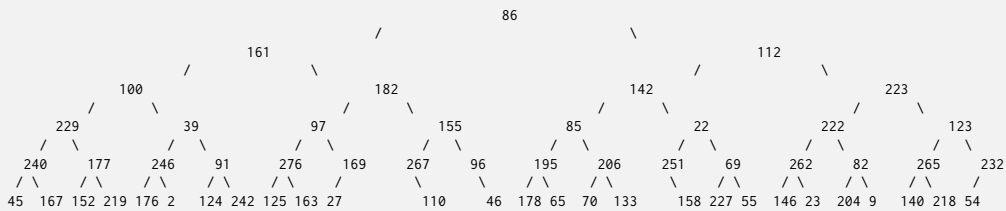


Figure 22: Worst mappings between normal and perturbed letters across the large-scale models

Response from o4-mini (high)

Tree Structure Task



Question: What is the right child of node 262?

Model Response:

Here's how the bottom-level children line up under each node at level 4 (240, 177, 246, 91, 276, 169, 267, 96, 195, 206, 251, 69, 262, 82, 265, 232):

- 240→45,167
- 177→152,219
- 246→176,2
- 91→124,242
- 276→125,163
- 169→27
- 267→110
- 96→46
- 195→178,65
- 206→70,133
- 251→158,227
- 69→55,146
- 262→23,204
- 82→9,140
- 265→218,54
- 232→None

Thus, the right child of 262 is 204.

<answer>204</answer>

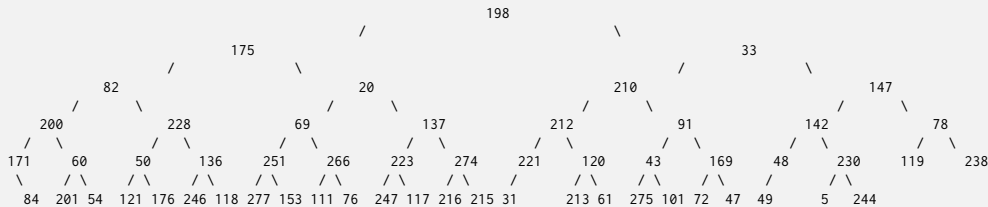
Ground Truth:

<answer>23</answer>

Figure 23: Tokenization-induced error example in Tree Understanding task.

Response from deepseek-v3

Tree Structure Task



Question: What is the parent node of node 275?

Model Response:

Let's trace the path to node 275 in the given tree.

Starting from root 198:

- 198 → left child 175 → right child 20 → left child 69 → right child 266 → left child 251 → right child 101 → left child 275?

Wait, check carefully:

Actually, 69 has left child 251 and right child 266.

266 has left child 277 and right child 153? No, the diagram shows:

```
69
 / \
251 266
 / \ / \
... / \ ...
```

Wait, let's read the tree carefully:

From 69: Left child 251 → children: 277, 153

Right child 266 → children: 111, 76

So 275 is not under 69.

Check 147 subtree:

147 → left child 142 → right child 230 → left child 72 → right child 47? No, 230 has children: 119, 238.

So not there.

Check 33 subtree:

33 → left child 210 → left child 221 → left child 221 → right child 31? No.

33 → right child 147 → left child 142 → left child 48 → right child 49? No.

Check 175 subtree:

175 → left child 82 → right child 228 → left child 50 → left child 121 → right child 176?

Check 33 → left child 210 → right child 91 → left child 43 → right child 169 → left child 275? Let's verify:

```
91
 / \
43 169
 / \ / \
213 61 275 101
```

Yes! 169 has left child 275 and right child 101.

So parent of 275 is 169.

<answer>169</answer>

Ground Truth:

<answer>43</answer>

Figure 24: Tokenization-induced error example in Tree Understanding task.

Response from gpt-4

Safety-mask Task

You are a data privacy assistant. Your task is to redact (mask) sensitive information in the given text according to specific rules.

****Redaction Rules:****

- **18-digit ID Card Number**:** Keep the first 6 digits and last 2 digits, mask the middle 10 digits with asterisks (*)
- Example: 123456789012345678 123456*****78
- **Phone Number**** (format: +[area code] [11-digit number]): Keep '+' sign and area code, keep first 3 digits and last 4 digits of phone number, mask the middle 4 digits with asterisks (*)
- Example: +12 12345678901 +12 123****8901
- **Credit Card Number**:** Keep the first 6 digits and last 4 digits, mask all middle digits with asterisks (*)
- Example: 1234 5678 9012 3456 123456*****3456

****Text to redact:****

In a recent customer record audit, we identified discrepancies in the contact details provided. The primary contact number listed for Mr. 703324375459458270 does not match our records. We attempted to reach out using the alternative +82 87901400470 provided but were unable to establish contact. It is crucial to update this information promptly to ensure seamless communication going forward. Additionally, we noticed that the credit card on file expired last month. We kindly request that Mr. 620140087033621286 updates the payment method to avoid any disruptions to their service. During the investigation into the discrepancies in the records, we discovered that the 963871589151698749 was not updated in our system following the recent renewal. This oversight may have contributed to the inconsistencies in the provided contact information. To rectify this, we urge Mr. 727095999340105436 to provide us with the updated details at their earliest convenience. Moreover, we noticed that the credit card ending in 7194 2835 7809 3796 is nearing its expiration date. We advise Mr. 842012233125116498 to update the payment information to prevent any delays in processing upcoming payments. Thank you for your prompt attention to these matters.

****Your redacted text (place between <answer></answer> tags):****

Model Response:

<answer>In a recent customer record audit, we identified discrepancies in the contact details provided. The primary contact number listed for Mr. 703324*****59270 does not match our records. We attempted to reach out using the alternative +82 879****0470 provided but were unable to establish contact. It is crucial to update this information promptly to ensure seamless communication going forward. Additionally, we noticed that the credit card on file expired last month. We kindly request that Mr. 620140*****1286 updates the payment method to avoid any disruptions to their service. During the investigation into the discrepancies in the records, we discovered that the 963871*****9749 was not updated in our system following the recent renewal. This oversight may have contributed to the inconsistencies in the provided contact information. To rectify this, we urge Mr. 727095*****5436 to provide us with the updated details at their earliest convenience. Moreover, we noticed that the credit card ending in 719428*****3796 is nearing its expiration date. We advise Mr. 842012*****6498 to update the payment information to prevent any delays in processing upcoming payments. Thank you for your prompt attention to these matters.</answer>

Evaluation Result:

Incorrectly Redacted/Missing (5):

id_card: 703324375459458270	703324*****70	[incorrect_or_missing]
id_card: 620140087033621286	620140*****86	[incorrect_or_missing]
id_card: 963871589151698749	963871*****49	[incorrect_or_missing]
id_card: 727095999340105436	727095*****36	[incorrect_or_missing]
id_card: 842012233125116498	842012*****98	[incorrect_or_missing]

Figure 25: Tokenization-induced error example in Safety-mask task.

Response from deepseek-r1

Safety-mask Task

You are a data privacy assistant. Your task is to redact (mask) sensitive information in the given text according to specific rules.

****Redaction Rules:****

1. ****18-digit ID Card Number****: Keep the first 6 digits and last 2 digits, mask the middle 10 digits with asterisks (*)

- Example: 123456789012345678 123456*****78

2. ****Phone Number**** (format: +[area code] [11-digit number]): Keep '+' sign and area code, keep first 3 digits and last 4 digits of phone number, mask the middle 4 digits with asterisks (*)

- Example: +12 12345678901 +12 123****8901

3. ****Credit Card Number****: Keep the first 6 digits and last 4 digits, mask all middle digits with asterisks (*)

- Example: 1234 5678 9012 3456 123456*****3456

****Text to redact:****

In the latest customer records review, we identified a discrepancy in +81 65885951847 associated with 791203559727501899 for account number 626578496720551032. It seems like there was a mix-up during the data entry process, leading to incorrect information being linked to the account. This issue needs urgent attention to ensure accurate customer details, especially considering the upcoming audit next month. The team is working on rectifying the error and updating the phone number linked to 342091894702652726 to reflect the correct information. As part of the security protocol, please ensure that any changes made to customer records, such as updating +86 41971093480 or 588853786584622252, are documented properly and verified by a supervisor to avoid any data integrity issues. For any further inquiries regarding this matter or assistance needed in updating the information, please reach out to the respective department heads. We value data accuracy and privacy highly, and it is essential to maintain strict confidentiality when handling sensitive information like 829629097998353799 and +86 65088382784. Your cooperation in ensuring proper data management practices is greatly appreciated.

****Your redacted text (place between <answer></answer> tags):****

Model Response:

<answer>In the latest customer records review, we identified a discrepancy in +81 658****1847 associated with 791203*****99 for account number 626578*****32. It seems like there was a mix-up during the data entry process, leading to incorrect information being linked to the account. This issue needs urgent attention to ensure accurate customer details, especially considering the upcoming audit next month. The team is working on rectifying the error and updating the phone number linked to 342091*****26 to reflect the correct information. As part of the security protocol, please ensure that any changes made to customer records, such as updating +86 419****3480 or 588853*****22, are documented properly and verified by a supervisor to avoid any data integrity issues. For any further inquiries regarding this matter or assistance needed in updating the information, please reach out to the respective department heads. We value data accuracy and privacy highly, and it is essential to maintain strict confidentiality when handling sensitive information like 829629*****99 and +86 650****2784. Your cooperation in ensuring proper data management practices is greatly appreciated.</answer>

Evaluation Result:

Incorrectly Redacted/Missing (1):

id_card: 588853786584622252 588853*****52 [incorrect_or_missing]

Figure 26: Tokenization-induced error example in Safety-mask task.

Task	Metric	DS-V3	DS-R1	GPT-4	GPT-5	Qwen-32B-Ins	DS-Qwen-32B
Whitespace counting	EM	99.0%	100.0%	100.0%	100.0%	74.0%	88.0%
Identifier renaming	EM	82.0%	100.0%	73.0%	100.0%	47.0%	63.0%

Table 11: Exact-match accuracy on the two code-oriented tasks. Bold marks the best score per row. Large-scale reasoning models (DS-R1, GPT-5) saturate both tasks, while 32B-scale open models lag considerably on identifier renaming.

Table 12: Detailed performance on RSA Differences and Map Navigation Tasks. Abbreviations: DS-V3/R1 (DeepSeek-V3/R1), Ins (Instruct), DS-Qwen (DeepSeek-distill-Qwen).

Group	Benchmark Task	Metric	DeepSeek		GPT		o		Qwen & DS-distill-Qwen			
			DS-V3	DS-R1	GPT-4	GPT-5	o4-mini (low)	o4-mini (high)	Qwen-2.5-7B-Ins	DS-Qwen-2.5-7B	Qwen-2.5-32B-Ins	DS-Qwen-2.5-32B
2D Spatial Pattern Comprehension	RSA-diff	F1 (3 diff.)	0.1075	0.9633	0.0687	0.9833	0.7799	0.9627	0.0169	0.0020	0.0198	0.0040
		F1 (5 diff.)	0.0549	0.9669	0.0378	0.9649	0.5699	0.9343	0.0047	0.0063	0.0073	0.0067
		F1 (7 diff.)	0.0697	0.9610	0.0314	0.9986	0.5013	0.9377	0.0065	0.0016	0.0156	0.0038
		F1 (9 diff.)	0.0641	0.9730	0.0385	0.9792	0.4528	0.9477	0.0082	0.0062	0.0079	0.0051
		Avg. F1	0.0741	0.9661	0.0441	0.9815	0.576	0.9456	0.0091	0.0040	0.0127	0.0049
		Token Num	1331	12665	79	6624	2744	7237	494	1496	99	1251
	Map-Nav (Sokoban)	Element ID Acc.	100.0%	84.0%	92.0%	100.0%	80.0%	76.0%	96.0%	0.0%	80.0%	40.0%
		Element Loc Acc.	88.0%	100.0%	40.0%	100.0%	72.0%	96.0%	12.0%	36.0%	40.0%	96.0%
		Surrounding Elem Acc.	56.0%	96.0%	8.0%	92.0%	84.0%	96.0%	0.0%	0.0%	12.0%	72.0%
		Relative Pos Acc.	76.0%	100.0%	52.0%	100.0%	60.0%	88.0%	8.0%	8.0%	16.0%	68.0%
		Avg. Acc.	79.0%	99.0%	48.0%	98.0%	74.0%	89.0%	29.0%	11.0%	37.0%	69.0%
		Token Num	331	1478	44	891	1076	1423	41	2839	119	1114
	Map-Nav (FrozenLake)	Element ID Acc.	100.0%	100.0%	80.0%	100.0%	100.0%	100.0%	90.0%	25.0%	85.0%	75.0%
		Element Loc Acc.	90.0%	100.0%	55.0%	95.0%	85.0%	90.0%	5.0%	35.0%	35.0%	90.0%
Surrounding Elem Acc.		85.0%	95.0%	60.0%	100.0%	85.0%	95.0%	5.0%	15.0%	55.0%	75.0%	
Relative Pos Acc.		90.0%	100.0%	25.0%	100.0%	40.0%	50.0%	0.0%	10.0%	20.0%	75.0%	
Count Elem Acc.		100.0%	100.0%	75.0%	100.0%	85.0%	100.0%	65.0%	5.0%	40.0%	85.0%	
Avg. Acc.		93.0%	99.0%	59.0%	99.0%	79.0%	87.0%	33.0%	18.0%	47.0%	80.0%	
Token Num	289	1010	23	602	751	890	21	1918	96	1052		

Response from deepseek-r1

Keystroke Task

You are a typewriter effect simulator. Your task is to show the progressive typing sequence of a given word.

TASK DESCRIPTION: Given an input word, you need to generate the complete sequence showing how the word appears character by character as if someone is typing it.

RULES: 1. Start with the first character 2. Add one character at a time until the complete word is formed 3. Use "←" (arrow) to separate each step 4. Do not include any extra characters, spaces, or explanations 5. Only output the progressive sequence

IMPORTANT: - Your response should ONLY contain the progressive sequence - No explanations, no additional text - Follow the exact format shown in examples

INPUT: maintainability

Model Response:

```
<answer>m←ma←mai←main←maint←mainta←maintain←maintaina←maintainab←maintainabi←maintainabil←maintainabili←maintainabilit←maintainability</answer>
```

Ground Truth:

```
</answer>m←ma←mai←main←maint←mainta←maintai←maintain←maintaina←maintainab←maintainabi←maintainabil←maintainabili←maintainabilit←maintainability</answer>
```

Figure 27: Tokenization-induced error example in Keystroke task.

Task	Overthinking	Tokenization	Total
Gomoku-Linear	2	5	7
Gomoku-Diagonal	10	9	19
Keystroke-Encode	1	4	5
Keystroke-Decode	4	2	6
Bio-seq	9	6	15

Table 13: Proportion of error types in DeepSeek-R1

Response from deepseek-r1

RSA Difference Task

You are given two RSA key fingerprint patterns. Your task is to find all the differences between them.

Following are the two RSA key fingerprint patterns for comparison:

```
+-----[ RSA 2048 ]-----+ +-----[ RSA 2048 ]-----+
|.00=. .XooX#^B&%-^*X+%o=.+X-^=&*+--^B#| |.00=. .XooX#^B&%-^*X+%o=.+X-^=&*+--^B#|
|+B=&B*^.^+o**+^*#XX+*X-#*+OoO.B%#. *XOX| |+B=&B*^.^+o**+^*#XX+*X-#*+OoO.B%#. *XOX|
|O*oo.+&. .#B=@O-@e@#=#. .X+-&O-o=. %@ &#| |O*oo.+&. .#B=@O-@e@#=#. .X+-&O-o=. %@ &#|
|B#&+X&+@^.@.O&OX^@oB@o%-B-- o&oX* #o&#| |B#&+X&+@^.@.O&OX^@oB@o%-B-- o&oX* #o&#|
|+&*XX+B=*%o.X-oB@o%*^*=-=OB%@^X@# @*.%| |+&*XX+B=*%o.X-oB@o%*^*=-=OB%@^X@# @*.%|
|+B-.B @+-#=@X+=%oB*#oBX%+X*XB* X+--X&| |+B-.B @+-#=@X+=%oB*#oBX%+X*XB* X+--X&|
|B%&o%+^ %% +O@^o@+O%*#=#@ &^*^.*^=^XBX| |B%&o%+^ %% +O@^o@+O%*#=#@ &^*^.*^=^XBX|
|.o. &B+% .X.*BB&#.X*O&@X.BX%&*%# *@#-| |.o. &B+% .X.*BB&#.X*O&@X.BX%&*%# *@#-|
|X^B -B.-=&OX==# #oo^-*#*#Oo#*#@^#oO| |X^B -B.-=&OX==# #oo^-*#*#Oo#*#@^#oO|
|+*+-. *B +.B%X %& o&o ^*^+-o*o-B^*OX@.&| |+*+-. *B +.B%X %& o&o ^*^+-o*o-B^*OX@.&|
|##X ^@#*O.OB X@o+.X +.^oB*oBX.&*XO@&| |##X ^@#*O.OB X@o+.X +.^oB*oBX.&*XO@&|
|*X&@^O#-@.^@o=*o.%+@BBOo+^+^O^B*%@-^ | |*X&@^O#-@.^@o=*o.%+@BBOo+^+^O^B*%@-^ |
|=^@oB -Xo @O. .*&-o-B@##o&=^o=^@ XB=-.@| |=^@oB -Xo @O. .*&-o-B@##o&=^o=^@ XB=-.@|
|=-^^-000&^O+^@X^X@##+@XO+oo@oXO=-%@B@| |=-^^-000&^O+^@X^X@##+@XO+oo@oXO=-%@B@|
|&oBB B.= +^@ ^ -@-X.*-*#B-+OB^oOo#o+| |&oBB B.= +^@ ^ -@-X.*-*#B-+OB^oOo#o+|
|#&O%+ ^*oOo#B*O-. *XB.^*@&=-.@OB*-X&B+| |#&O%+ ^*oOo#B*O-. *XB.^*@&=-.@OB*-X&B+|
|+%+=%O%&@%+. B+o @X*+*%oX@*#@o.=#O =| |+%+=%O%&@%+. B+o @X*+*%oX@*#@o.=#O =|
|X-B@--B=&@B.O. .+XX%@+@o+ .X +Oo.X=-O #| |X-B@--B=&@B.O. .+XX%@+@o+ .X +Oo.X=-O #|
+-----+ +-----+
```

Model Thinking: We are given two patterns, Pattern 1 and Pattern 2. We need to compare them character by character, noting any differences. The coordinate system has origin at the top-left '+' character of each pattern. The patterns are given as lines of text.

Wait, that's only 40 characters? Actually, I counted up to index 39. Let's count the string length properly.

Let's write it out:

[Thinking Process omit]

So indeed 40 characters. But the top border has 43. That means the box is not exactly aligned? Wait, the top border line is: "+-----[RSA 2048]-----+"

[Thinking Process omit]

So indeed, there are 40 characters: indices 0 to 39. That means the left bar is at x=0 and the right bar is at x=39. But the top border has a '+' at x=0 and x=42.

Wait, if we count the total length, it is 40 characters? Let's count manually: Write the string:

[Thinking Process omit]

Actually, Pattern1 x=14 is '.' and Pattern2 x=14 is also '.'? Wait, in Pattern2, after "O." there is "O", so the characters are: x=12 '.', x=13 'O', x=14 '.', x=15 'O'. So at x=14, both are '.'. The difference is at x=15: Pattern1 has '.', Pattern2 has 'O'. So difference at (15,18): . -> O.

[Thinking Process omit]

So it is 41 characters? Actually, from index 0 to 40: that's 41 characters. But I counted up to index 40: that gives 41 characters. Wait, but my list above has indices 1 to 40? Let's do 0-based indexing:

[Thinking Process omit]

That is 40 characters? Actually, indices 0 to 39 give 40 characters. But index 39 is '+'? In my list, index 39 is '+'? Wait, index 38 would be the 39th character. So total indices 0 to 38: 39 characters? I need to be precise.

[Thinking Process omit]

Wait, in Pattern2, after "O." there is "O", so the characters are: x=12 '.', x=13 'O', x=14 '.', x=15 'O'. So at x=15, both are '.'. Also, later in the line:

[Thinking Process omit]

But wait, are there any other lines? We checked y=1 to y=18. We should also check y=0 and y=19, but they are borders and identical.

[Thinking Process omit]

<answer> (19,5): % -> O (30,14): X -> (17,17): o -> (29,18): + -> % </answer>

But wait: in the example format, they had spaces around the arrow?

[Thinking Process omit]

So I'll write exactly as shown.

Thus, final answer.

Model Response:

<answer> (19, 5): % -> O (30, 14): X -> (17, 17): o -> (29, 18): + -> % </answer>

Ground Truth:

(29, 18): + -> % (15, 18): . -> O (30, 14): X -> (19, 5): % -> O (17, 17): o ->

Figure 28: Overthinking example in RSA Difference task.

Response from deepseek-r1

Gomoku Task

You are an expert at analyzing Gomoku (Five-in-a-Row) game boards.
Your task is to determine the current game state by analyzing the board configuration.

Rules:

- W represents a white stone
- B represents a black stone
- E represents an empty position
- Victory condition: 5 consecutive stones of the same color in a row (horizontal, vertical, or diagonal)
- Only one player can win, or no one wins

You should analyze the board carefully and provide your reasoning.

Then, wrap your final answer in <answer></answer> tags with exactly one of these three options:

- WHITE_WINS if white has 5 consecutive stones
- BLACK_WINS if black has 5 consecutive stones
- NO_WINNER if neither player has achieved victory

Please analyze this Gomoku board and determine the game state:

Board size: 15x15

```
EWWWEEWWBBWWBW
BEBBWEWEBEBBBW
WEBWWBBEBWWWBWE
EWBBEEEBBBBEWE
WWBEBWBWWEWEEB
BWBEBWBWBBWWEW
WBBBBWBWBWBWBE
EWWBEWBEWWWBWB
BEBWBBBBWWEBEEW
BEBWBEWEEEEEWBB
BWEBWBWBWWEWBB
EWWBEWBEBEWEBE
BEBWWWEBEWWWWEB
BWWBBEEBEWEEBEE
EBWBBBBEBWBBWWW
```

What is the current game state?

Model Thinking:

First, I need to analyze this Gomoku board....

[Thinking Process omit]

Row 13: B E B W W W W E B E W W W E B? Wait, the row is given as: BEBWWWEBEWWWWEB

Let me write it carefully: Row 13: B E B W W W W E B E W W W W E B? But the string is "BEBWWWEBEWWWWEB" which has 15 characters.

Indices: 0:B, 1:E, 2:B, 3:W, 4:W, 5:W, 6:W, 7:E, 8:B, 9:E, 10:W, 11:W, 12:W, 13:W, 14:E? No, from the string: BEBWWWEBEWWWWEB — let's list: B,E,B,W,W,W,W,E,B,E,W,W,W,W,E,B? That would be 16 characters? I think I miscounted.

The board has 15 rows, each with 15 characters. So Row 13 must be 15 characters. The user wrote: "BEBWWWEBEWWWWEB" — let's count: B E B W W W W E B E W W W W E B — that's 16 characters? B,E,B,W,W,W,W,E,B,E,W,W,W,W,E,B — yes, 16. There must be a mistake.

I need to check the input...

[Thinking Process omit]

Row 13: BEBWWWEBEWWWWEB -> let's count: B E B W W W W E B E W W W W E B -> that's 16 characters?

B, E, B, W, W, W, W, E, B, E, W, W, W, W, E, B -> 16. But it should be 15. Perhaps there's a typo. Let's see the user's input: "BEBWWWEBEWWWWEB" — if I take it as 15 characters, maybe it's "BEBWWWEBEWWWWE" or something? But from the context, I need to assume it's 15x15, so perhaps I misread.

[Thinking Process omit]

Line 13: "BEBWWWEBEWWWWEB" - let's count: B E B W W W W E B E W W W W E B -> if I count the letters: B=1, E=2, B=3, W=4, W=5, W=6, W=7, E=8, B=9, E=10, W=11, W=12, W=13, W=14, E=15, B=16? That seems like 16. But in the string, there are no spaces, so "BEBWWWEBEWWWWEB" has 15 characters? Let's write it out: B E B W W W W E B E W W W W E B — that's 16 characters. Perhaps it's "BEBWWWEBEWWWWE" with 15? But the user wrote "BEBWWWEBEWWWWEB" which ends with B, so maybe it's 15 and I miscounted.

Let's use a counter: B E B W W W W E B E W W W W E B

[Thinking Process omit]

Now, I'll count the characters in row13: "BEBWWWEBEWWWEB" - let's write each character: 1:B, 2:E, 3:B, 4:W, 5:W, 6:W, 7:W, 8:E, 9:B, 10:E, 11:W, 12:W, 13:W, 14:W, 15:E, 16:B?

[Thinking Process omit]

Now, let's take row13: "BEBWWWEBEWWWEB" - if I paste this into a text editor, it shows 15 characters? Let's count manually: B-1, E-2, B-3, W-4, W-5, W-6, W-7, E-8, B-9, E-10, W-11, W-12, W-13, W-14, E-15, B-16? No, after W-14, it is E and then B, but in the string, "EWWWEB" might be considered as E, W, W, W, W, E, B which is 7 characters. So from start: B E B W W W E B E W W W W E B -> that's 16. But perhaps "BEBWWWEB" is 8 characters? B,E,B,W,W,W,W,E,B? That's 9? I'm getting confused.

[Thinking Process omit]

Row13: "BEBWWWEBEWWWEB" - now, let's say this has 15 characters. So I need to interpret it as 15 characters. Let's list: B, E, B, W, W, W, W, E, B, E, W, W, W, W, E?

[Thinking Process omit]

I think I should proceed with row13 as "BEBWWWEBEWWWE" for analysis. Otherwise, I can't continue.

So for row13, I'll use: B E B W W W W E B E W W W W E

[Thinking Process omit]

So after thorough checking, I cannot find any five consecutive stones of the same color in any direction.

Therefore, the game state should be **NO_WINNER**.

Model Response:

After analyzing the board configuration for horizontal, vertical, and diagonal sequences of five consecutive stones, no such sequence was found for either white (W) or black (B). Therefore, neither player has achieved victory.

<answer>**NO_WINNER**</answer>

Ground Truth: BLACK_WINS

Figure 29: Overthinking example in Gomoku task.

E.2 TTBC Error Analysis

Increasing Phase

As the number of thinking tokens t_{exact} increases from 256 to 512, the model begins to engage in a more detailed thinking process before providing the final answer for biological sequence tasks. Figure 30 is an example from the DeepSeek-R1-Distill-Qwen-7B in the increasing phase.

Plateau Phase

When the thinking tokens, t_{exact} , range from 1024 to 2048, the model starts to perform double-checking after receiving the "**Wait**" continuation cue. Figure 31 is an example from the DeepSeek-R1-Distill-Qwen-7B in the plateau phase.

Decreasing Phase

Once the thinking tokens, t_{exact} , exceed 2048, reaching values such as 4096 or 8192, the model begins to exhibit mistakes due to overthinking. Figure 32 is an example from the DeepSeek-R1-Distill-Qwen-7B in the decreasing phase.

Response from DS-distill-Qwen-2.5-7B with t_{exact} 256

<|im_start|>system

You are a helpful reasoning assistant.

<|im_end|>

<|im_start|>user

[Instructions omitted]

****Input DNA sequence:**** GTAGTCGTGGTATTTTCG

Please provide the complementary DNA sequence for the given input.<|im_end|>

<|im_start|>assistant

<think>

Okay, so I need to find the complementary DNA sequence for the given input, which is GTAGTCGTGGTATTTTCG. Hmm, I remember that DNA is made up of four nucleotides...

...

Starting with the first base, G. Since G pairs with C, the first complement will be C

Next is T. T pairs with A, so the second complement is A.

Then comes A. A pairs with T, so the third

[Thinking process is truncated by TTBC rule.]

</think>

<answer>

CATCCGAGTTGTGCAACTGC

</answer>

Expected answer: CATCAGCACACCATAAAAGC

Figure 30: Increasing phase example in biological sequence task.

Response from DS-distill-Qwen-2.5-7B with t_{exact} 1024

```
<|im_start|>system
You are a helpful reasoning assistant.
<|im_end|>
<|im_start|>user
[Instructions omitted]
**Input protein sequence (three-letter codes):** GLU-TYR-TRP-PRO-GLN-VAL-....

Please provide the one-letter protein sequence for the given input.<|im_end|>
<|im_start|>assistant
<think>
Okay, so I need to convert this protein sequence from three-letter codes to one-
letter codes. Let me see, the input is GLU-TYR-TRP-PRO-GLN-VAL-...
[Thinking process omitted]
So, joining them without any separators, it should be EYWPQVSDFS.....

Wait, let me double-check each conversion to make sure I didn't make a mistake.
GLU to E, correct. TYR to Y, correct. TRP to W, correct. PRO to P, correct...
[Thinking process is truncated by TTBC rule.]
</think> <answer>
EYWPQVSDFSTAILSVMFMMC
</answer>
Expected answer: EYWPQVSDFSTAILSVPFMM
```

Figure 31: Plateau phase example in biological sequence task.

Response from DS-distill-Qwen-2.5-7B with t_{exact} 8192

<|im_start|>system

You are a helpful reasoning assistant.

<|im_end|>

<|im_start|>user

[Instructions omitted]GCGAUCAGUG

Input RNA sequence: AUAACCAGAUGCGAUCAGUG

Please provide the complementary RNA sequence for the given input.<|im_end|>

<|im_start|>assistant

<think>

Okay, so I need to find the complementary RNA sequence for the given RNA sequence. Hmm, I remember that in RNA, the bases pair up in a specific way. Let me think...

[Thinking process omitted]

Wait, let me make sure I'm doing this right. Each base in the original RNA is replaced by its complement. So, for each position, I replace A with U, U with A, C with G, and G with C.

Let me go through each position step by step:

1. The first base is A. Its complement is U.
2. The second base is U. Its complement is A.

.....

[Thinking process omitted]

Wait, let me double-check to make sure I didn't mix up any bases....

So the complementary sequence is U A U U G G U C U A C G C U A G U C A C

.....

[Thinking process omitted]

Wait, but I'm a bit confused because sometimes I think the complementary sequence is written in reverse complement form, but in this case, since we're just finding the complementary sequence without reversing.

Wait, but I just thought of something else. Let me make sure I didn't mix up any bases. Let me list them again:

Original: U G A U A A C C A G A U ...

Complement: A C U A U U G G U C U A ...

.....

[Thinking process is truncated by TTBC rule.]

</think>

<answer>

ACUAUUGGUCUACCCUAGUCACUU

</answer>

Expected answer: UAUUGGUCUACGCUAGUCAC

Figure 32: Decreasing phase example in biological sequence task.