StrucText-Eval: Evaluating Large Language Model's Reasoning Ability in Structure-Rich Text

Zhouhong Gu♡* , Haoning Ye♡*, Xingzhou Chen♡, Zeyang Zhou♡,
Hongwei Feng♡† , Yanghua Xiao♡♣†,
♡Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University
♣Fudan-Aishu Cognitive Intelligence Joint Research Center
{zhgu22, xzchen24}@m.fudan.edu.cn
{hnye19,zeyangzhou20,hwfeng,shawyh}@fudan.edu.cn

Abstract

The effective utilization of structured data, integral to corporate data strategies, has been challenged by the rise of large language models (LLMs) capable of processing unstructured information. This shift prompts the question: can LLMs interpret structured data directly in its unstructured form? We propose an automatic evaluation data generation method for assessing LLMs' reasoning capabilities on structure-rich text to explore this. Our approach supports 8 structured languages and 29 tasks, generating data with adjustable complexity through controllable nesting and structural width. We introduce StrucText-Eval, a benchmark containing 5,800 pre-generated and annotated samples designed to evaluate how well LLMs understand and reason through structured text. StrucText-Eval is divided into two suites: a regular Test suite (3,712 samples) and a Test-Hard suite (2,088 samples), the latter emphasizing the gap between human and model performance on more complex tasks. Experimental results show that while open-source LLMs achieve a maximum accuracy of 74.9% on the standard dataset, their performance drops significantly to 45.8% on the harder dataset. In contrast, human participants reach an accuracy of 92.6% on StrucText-Eval-Hard, highlighting LLMs' current limitations in handling intricate structural information. The benchmark and generation codes are open sourced in `https://github.com/MikeGu721/StrucText-Eval`

1 Introduction

Structured data, often represented by various structured languages such as JSON (Pezoa et al., 2016), YAML (Evans, 2001), ORG (org, 2023), or Markdown (Gruber, 2012), Latex (Lamport, 1985) etc., has consistently been central to corporate data strategies due to its ability to capture, store, and analyze essential information systematically. The inherent benefits of structured data lie in its standardized format and high degree of organization, which facilitates efficient data querying and machine processing, clearly surpassing the inherent chaos of unstructured data. However, with the advancement of large language models (LLMs) (Achiam et al., 2023; Touvron et al., 2023a,b; Sun et al., 2021), there has been a significant shift towards the effective utilization of unstructured data, attributed to the LLMs' capacity to comprehend and generate complex and nuanced semantics within such data (Brown et al., 2020). Considering that structured data can be directly presented in an unstructured format, it makes us wonder: ***whether it is possible to rely on LLMs to interpret structured data through unstructured format directly***.

Current LLM researchers have addressed their comprehension of the structure-rich text of limited categories: Graphs (Fatemi et al., 2023; Perozzi et al., 2024; Guo et al., 2023; Tang et al., 2023a; Chen et al., 2023), Tables (Sui et al., 2024; Campbell-Kelly, 2003; Pasupat and Liang, 2015) and JSON (Chen et al., 2024; Suzgun et al., 2022). However, these categories do not encompass all potential use cases of structure-rich text. For instance, scenarios requiring a direct understanding of articles in Latex or Markdown formats, data in YAML or ORG formats, or various custom-structured languages need to be adequately covered. Moreover, existing benchmarks often rely on manually annotated data for evaluation, which limits the development of robust evaluation frameworks and potentially facilitates model cheating (Zhou et al., 2023).

We propose a method for automatically generating evaluation data to assess models' capabilities in structure-rich text reasoning. This method is applied to 8 structured languages, as shown in Fig. 1, across 29 specific tasks, enabling data gen-

---

[*] Equal Contribution

[†] Corresponding authors.

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 223–244
July 27 - August 1, 2025 ©2025 Association for Computational Linguistics

Benchmark	**CSV/Table**	**Tree**	**JSON**	**YAML**	**XML**	**MD.**	**LaTeX**	**Org**	**Total**
BIG-Bench (Srivastava et al., 2022)	-	-	31,907	-	-	-	-	-	31,907
TEMPTABQA (Gupta et al., 2023)	1,208	-	-	-	-	-	-	-	1,208
Struc-Bench (Tang et al., 2023b)	4.1k	-	-	-	-	-	5.5k	-	9,600
DocCGen (Pimparkhede et al., 2024)	-	-	-	18k	-	-	-	-	18,000
SUC (Sui et al., 2024)	1,500	-	-	-	-	-	-	-	1,500
StrucText-Eval **(Ours)**	841	841	841	841	841	841	841	841	6,728

Table 1: Benchmark comparison across different structural language.

eration with controllable difficulty by adjusting the depth of structured nesting and the number of width and columns in the sample. Based on this method, we further introduce the ***Struc****ture*-Rich ***Text Eval****uation* Benchmark (**StrucText-Eval**), a comprehensive benchmark with 5,800 pre-generated and annotated samples designed to evaluate the proficiency of LLMs in deciphering embedded structures within input text. StrucText-Eval aims to evaluate whether LLMs understand raw structural tags, execute logical inferences based on the decoded semantics of these symbols, and organize their responses according to instruction requirements.

The StrucText-Eval dataset contains three test sets: Test (3,712 samples), Real-Test (928 samples), and Test-Hard (2,088 samples). Since the Test set questions were created using rule-based generation, we selected 928 questions from it and rewrote them using real-world examples to form the Real-Test set, making it closer to practical needs for LLMs to understand structured text. The Test-Hard set includes much longer questions, with an average length of 16,535 characters and the longest question containing 102,531 characters, greatly showing the gap between LLMs and humans in understanding structured data. The experimental results indicate that StrucText-Eval presents significant challenges in evaluating current LLMs' structured text processing capabilities. While various open-sourced models achieve maximum accuracy of 74.9% under different prompting methods, their performance declines markedly to 45.8% when tested on the more complex StrucText-Eval-Hard dataset. In contrast, human participants attain an accuracy of 92.6% on StrucText-Eval-Hard, highlighting the limitations of existing LLMs in comprehending and reasoning through complex structural information.

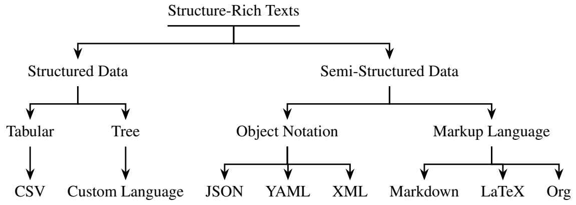


Figure 1: Taxonomy of Structure-Rich Texts covered in StrucText-Eval.

2 Related Work

2.1 Structural Text Understanding Enhancements

Recent efforts to enhance LLMs have focused on integrating external structures such as graphs, tool flows, and cross-domain representations to improve reasoning capabilities across various tasks. For instance, ControlLLM utilizes tool graphs to decompose complex multimodal tasks, resulting in enhanced performance on image and audio processing tasks by leveraging the topological dependencies of tools (Liu et al., 2023). Graph-based models like GraphGPT and BooG have shown promising results, with the former improving generalization across node classification and molecular tasks via graph instruction tuning (Zhao et al., 2023; Tang et al., 2024). At the same time, the latter employs virtual supernodes to unify graph structures across domains, fostering cross-domain task transferability (Cheng et al., 2024). Additionally, methods like RC2R demonstrate the effective combination of knowledge graphs and LLMs for domain-specific causal reasoning, particularly in financial risk propagation tasks (Yu et al., 2024). These advancements highlight the benefits of embedding structural elements, from graph architectures to domain-specific knowledge graphs, within LLM frameworks to improve task-specific inference and reasoning.

2.2 Structural Text Understanding Evaluation

Evaluating LLMs' understanding of structured data has become increasingly critical, though benchmarks remain limited. GraphQA and Struc-Bench are key datasets that assess LLMs' reasoning over graph-structured data and tabular text, respectively, illustrating the models' varying capabilities based on input encoding (Fatemi et al., 2023; Tang et al., 2023b). More specialized benchmarks, such as TEMPTABQA, evaluate temporal reasoning in tabular data, while TableLLM tests LLMs' proficiency in handling complex document-based table manipulation tasks (Gupta et al., 2023; Zhang et al., 2024). Other works, such as the evaluation of knowledge graph-based reasoning in complex time-series QA systems (JMFRN) (Huang et al., 2024), and privacy-oriented graph tasks in GHRatio (Yuan et al., 2024), further explore how LLMs handle intricate, structure-rich information, shedding light on their performance across different structured data formats.

Our work diverges from prior research by focusing exclusively on structure-based inference, deliberately removing semantic content to challenge LLMs to reason purely from structural patterns. Unlike previous approaches that use structural data as supplementary input for classification or semantic tasks (Pasupat and Liang, 2015; Sui et al., 2024), we design semantically agnostic tasks requiring models to infer meaning solely from symbolic structures. Moreover, while earlier benchmarks emphasize graph reasoning or tabular information retrieval, our work extends to a broader spectrum of structure-rich text types, encompassing various input formats and more complex dependency-based inference tasks.

3 StrucText-Eval Construction

3.1 Structure-Rich Texts Taxonomy

To explore structure-rich texts comprehensively, we propose a dataset for eight structured data types, each categorized within a taxonomy depicted in Fig. 1. This taxonomy encompasses both structured and semi-structured data formats. The structured data types include Tree ((Cormen et al., 2022)), Tabular ((Campbell-Kelly, 2003)), and Object Notation such as JSON ((Pezoa et al., 2016)), YAML ((Evans, 2001)), and XML ((Bray et al., 1998)). The semi-structured data types include Markup Languages like Markdown ((Gruber, 2012)), LaTeX ((Lamport, 1985)), and Org ((org, 2023)). Within StrucText-Eval, Tabular is stored in CSV format, whereas Tree is denoted by a custom format that nodes are represented as the string "xxx", connected with "->" and separated by "\n". For examples encompassing all languages and tasks, please refer to Sec. F in the Appendix.

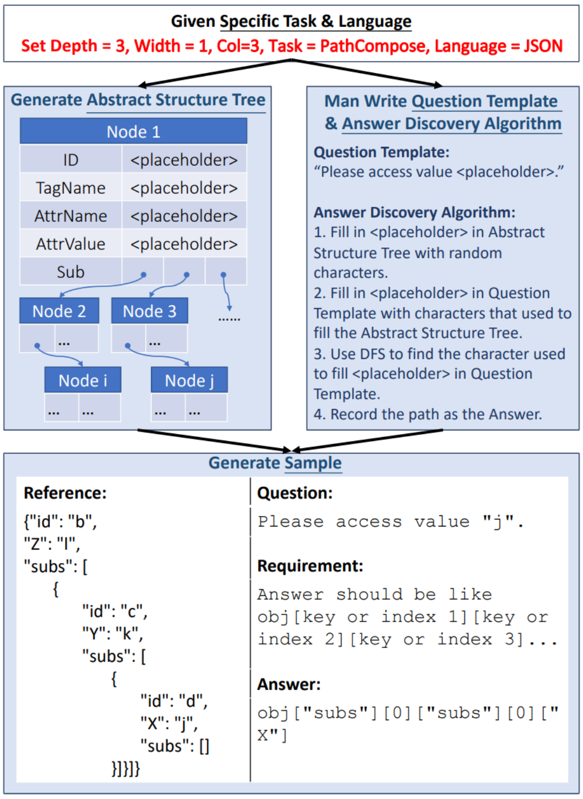
#Sample	#Reference	#GroundTruth	Depth	Width
StrucText-Eval-Test				
3,712	**804**	**47**	-	-
1,856	582	19	1	1
1,856	1,026	74	2	1
StrucText-Eval-Real-Test				
928	**562**	**74**	-	-
464	319	39	1	1
464	805	109	2	1
StrucText-Eval-Test-Hard				
2,088	**16,535**	**1,169**	-	-
232	573	22	1	1
232	614	26	1	2
232	663	25	1	3
232	992	80	2	1
232	2,108	136	2	2
232	3,866	283	2	3
232	5,036	312	3	1
232	32,428	2,229	3	2
232	102,531	7,411	3	3

Table 2: Statistics for StrucText-Eval test suite.

3.2 Generation of Test Suite

An example of JSON's PathCompose is shown in Fig. 2 to illustrate the dataset generation process. The generation process mainly entails constructing an abstract structure tree, manually drafting question templates, and developing corresponding answer discovery algorithms. The first step of the generation process is to define the complexity of the problem, characterized by depth, width, and column (Col), as well as its type, including task and language. During the construction of the abstract tree, depth represents the depth of the tree, width indicates the number of children for each non-leaf node, and Col specifies the number of fields associated with each node. When constructing the question template, predefined templates are retrieved based on the specified task. Finally, during sample generation, the selected task is used to identify the corresponding ground truth according to specific rules, and both the abstract tree and the ground truth are translated into the selected language.

Eight task categories have been delineated for eight languages, as detailed in Fig. 3b. Twenty-nine rules and question templates have been formulated for these tasks, with the specific rule templates detailed in Sec. G in the Appendix. Each sample in the dataset comprises four main fields: "Reference", "Question", "Requirement" and "An-


Given Specific Task & Language
Set Depth = 3, Width = 1, Col=3, Task = PathCompose, Language = JSON
Generate Abstract Structure Tree
Node 1
ID
<placeholder>
TagName
<placeholder>
AttrName
<placeholder>
AttrValue
<placeholder>
Sub
Node 2
Node 3
Node i
Node j
Man Write Question Template & Answer Discovery Algorithm
Generate Sample

Question Template:
"Please access value <placeholder>."

Answer Discovery Algorithm:
1. Fill in <placeholder> in Abstract Structure Tree with random characters.
2. Fill in <placeholder> in Question Template with characters that used to fill the Abstract Structure Tree.
3. Use DFS to find the character used to fill <placeholder> in Question Template.
4. Record the path as the Answer.

Reference:

```
{"id": "b",
"Z": "l",
"subs": [
    {
        "id": "c",
        "Y": "k",
        "subs": [
            {
                "id": "d",
                "X": "j",
                "subs": []
            }]}]}
```

Question:

```
Please access value "j".
```

Requirement:

```
Answer should be like
obj[key or index 1][key or
index 2][key or index 3]...
```

Answer:

```
obj["subs"][0]["subs"][0]["
X"]
```

Figure 2: The illustration of the dataset generation process, the Json PathCompose task, is an example.

swer". We give examples for each language and task in Sec. F in the Appendix.

3.3 Generation of Real-Test Suite

To enhance the alignment between StrucText-Eval-Test and real-world data, we selected a representative subset of samples for manual modification. We maintained consistency with StrucText-Eval-Test by extracting equal proportions of data across tasks, languages, and complexity levels. Five graduate students from computer science backgrounds were invited to modify the "Reference" and "Answer" parts of raw data by replacing abstract node values with meaningful real-world information. In the annotation process, each question is assigned a unique scenario (e.g., athletic activities, glassware specifications), and annotation needs to ensure the modified content is aligned with these scenarios and thereby facilitating diverse, non-repetitive datasets that closely approximate real-world applications. For instance, to annotate in an athletic scenario, an abstract JSON structure "a": "b", "c": "ddd" is transformed into "Name": "James", "Speciality": "Running". The comprehensive guidelines for manual rewriting are detailed in Appendix C.

3.4 Statistic Information

StrucText-Eval has assembled two datasets. StrucText-Eval-Test comprises 3,712 samples, StrucText-Eval-Real-Test comprises 928 samples, and StrucText-Eval-Test-Hard comprises 2,088 samples, each of the 29 specific tasks for eight languages as depicted in Fig. 3a. Detailed statistics regarding the number of samples, lengths, and complexity levels across all tasks, languages, and difficulties are detailed in Tab. 2.

4 Experiment Setup

To evaluate LLMs' current capability of processing structure-rich text and executing dependent inference, we conducted a series of experiments using StrucText-Eval in various settings. Our study utilizes both prompt-based and finetuning methods to analyze the performance variations.

4.1 Models

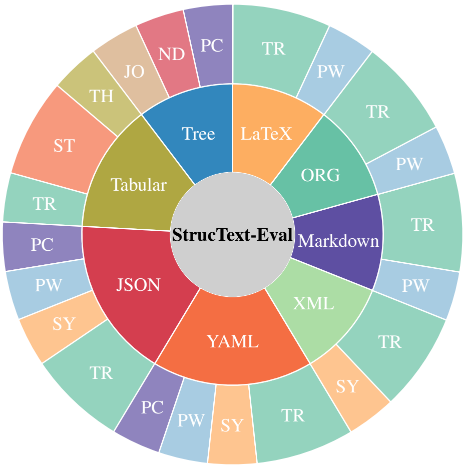
We tested six Open-Source LLMs in both StrucText-Eval Test and Test-Hard Suite, and we used the short name (in the bracket) of these LLMs in the experiments: **Qwen/Qwen2-7B-Instruct** (Qwen-2-7B), **Qwen/Qwen2-72B-Instruct** (Qwen-2-72B), **meta-llama/Meta-Llama-3.1-8B-Instruct** (Llama-3.1-8B), **meta-llama/Meta-Llama-3.1-72B-Instruct** (Llama-3.1-70B), **meta-llama/Meta-Llama-3.1-405B-Instruct** (Llama-3.1-405B), **mistralai/Mistral-7B-Instruct-v0.2** (Mistral-0.2-7B)

Considering the huge expense of using an API-based model, we only tested six Close-Source LLMs in StrucText-Eval-Hard: **gpt-4o-2024-08-06** (gpt-4o), **gpt-4o-mini-2024-07-18** (gpt-4o-mini), **gemini-1.5-pro**(gemini-1.5-pro), **gemini-1.5-flash**(gemini-1.5-flash), **GLM-4-Plus** (glm-4-plus), **GLM-4-Flash** (glm-4-flash).

4.2 Prompt-based Method

We also evaluated the impact of different prompt designs on the performance of LLMs by utilizing six distinct prompt configurations in the main experiments. Detailed implementation of these prompts can be found in the Appendix in Sec. E. The six primary prompt settings are as follows:

`Naive` involves a straightforward input of "Context", "Question", and "Options" into the LLMs to generate responses. **`Self-Chain-of-Thought`**



(a) Benchmark Decomposition

Task Name	Abbr.	Task Description
Syntax	SY	Focuses on detecting structural errors in data formats such as JSON, XML, and YAML.
PathWalk	PW	Focuses on extracting specific sections or subsections from structured documents such as org, LaTeX, or markdown files.
TextRetrieval	TR	Assesses the ability to extract specific information from various document formats, including text content and image filenames.
Statistic	ST	Concentrates on statistical queries to calculate the number of employees meeting specific salary conditions.
Join	JO	Assesses the ability to filter data sets that meet specific criteria by combining multiple tables in a database through SQL queries.
Tree.Height	TH	Evaluates calculating the height of the longest path from the root node to any leaf node in a tree structure.
Node.Depth	ND	Assesses the depth of any node in a tree structure relative to the root node.
PathCompose	PC	Evaluates reasoning of paths and multi-level data indexing within hierarchical or tree-like structures.

(b) Descriptions of tasks for evaluating structured data understanding in large language models

Figure 3: The tasks within StrucText-Eval and their description.

(Self-CoT) (Kojima et al., 2022) incorporates a step-by-step reasoning prompt to guide the model through logical reasoning. **Plan-and-Solve CoT (PS-CoT)** (Wang et al., 2023) emphasizes problem decomposition before solving, encouraging the model to first break down the problem before generating a solution. **With Hint (w/ hint)** provide manually curated hints to the model to observe its performance when additional information is injected. Since this approach introduces supplementary data, it is delineated by a dashed line from other methods in Table 3. **Few-Shot Demonstration** involves appending a few training data directly to the prompt. The **Simple Few-Shot Demonstration** uses only the shortest examples from the training set as few-shot demonstrations.

4.3 Evaluation Method

We use the RougeL metric (Lin, 2004) to assess the degree of character-level similarity between model outputs in the main content of this paper. Sometimes, the task requires the LLM to generate the entire reasoning path leading to the answer, which results in high RougeL scores. So, we assign a score of 0 if the RougeL score falls below 0.75.

Additionally, we present the results of other evaluation metrics, including LLM-as-Judge-Score (Zheng et al., 2023), BLEU (Papineni et al., 2002), and Exact Match, in Tab. 6 in the Appendix. Furthermore, we conduct a consistency analysis across these metrics compared to human judgments, as shown in Fig. 5.

5 Analysis

5.1 Overall Performance in StrucText-Eval

The overall performance in StrucText-Eval is presented in Table 3, revealing significant variations in the performance of different models across various languages and tasks. For instance, the Qwen2-72B-Instruct model demonstrates optimal performance on JSON-formatted tasks with an 85.8% accuracy under the “Naive” prompt. It also achieves notable results in YAML and CSV tasks, with accuracies of 82.7% and 86.4%, respectively. In contrast, the Meta-Llama-3.1-8B-Instruct-Turbo model performs poorly under the same settings, achieving only 64.6% accuracy on LaTeX tasks. Manually injected hints (w/ hint) generally improve model performance, particularly in tasks requiring deep reasoning, such as those involving YAML and JSON. For example, the Meta-Llama-3.1-70B-Instruct-Turbo model’s accuracy improves from 75.4% under the “Naive” prompt to 84.9% with the “w/ Hint” strategy. However, with “Self-CoT” and “PS-CoT” prompts, specific models like Qwen2-7B-Instruct exhibit lower accuracy across multiple tasks, especially when handling complex structures such as XML and Tree data, performing significantly worse than other prompting methods.

These performance disparities can be primarily attributed to training sample biases and the influence of different prompting strategies. JSON, being a widely used format in internet data, is frequently encountered by many large models during training, leading to a pronounced advantage in handling JSON-formatted tasks—a clear manifestation of training sample bias. Moreover, the choice of prompting strategy directly affects a model’s infer-

Model	**Prompt**	**Languages**								**Tasks**								**all**
		JSON	**LaTeX**	**Md.**	**ORG**	**CSV**	**Tree**	**XML**	**YAML**	**PC**	**PW**	**SY**	**TR**	**JO**	**ST**	**ND**	**TH**	
Qwen2-7B	Base	70.4	68.8	68.0	54.5	83.5	68.9	57.6	68.5	48.5	74.2	49.2	72.4	79.5	78.4	47.7	93.2	30.0
	Self-CoT	12.8	1.5	1.5	9.1	29.0	4.5	3.6	3.5	4.5	6.4	6.1	8.1	27.3	26.1	2.3	6.8	17.2
	PS-CoT	31.7	31.7	19.4	20.1	67.0	36.4	25.8	24.9	9.8	19.8	32.6	34.1	63.6	60.2	25.0	72.7	29.1
	w/ Hint	70.8	66.1	66.5	58.1	85.2	56.8	55.2	70.2	43.9	72.3	43.2	75.3	86.4	77.3	45.5	65.9	44.0
Qwen2-72B	Base	85.8	73.7	75.1	67.1	92.6	86.4	71.2	82.7	80.3	81.5	62.9	80.8	90.9	90.9	77.3	95.5	42.6
	Self-CoT	85.4	69.9	70.8	65.2	95.5	**90.2**	79.5	89.7	78.8	77.1	81.1	81.7	90.9	95.5	84.1	95.5	51.0
	PS-CoT	89.5	70.1	68.9	61.7	92.0	84.8	81.1	93.4	76.5	77.6	87.9	80.8	81.8	93.2	86.4	**97.7**	65.3
	w/ Hint	90.0	72.5	**79.1**	68.6	94.9	81.1	72.7	90.8	81.1	84.0	77.3	82.4	95.5	92.0	72.7	86.4	49.4
Llama-3.1-8B	Base	43.9	64.6	49.3	48.3	42.6	50.0	26.5	46.9	30.3	49.4	1.5	61.0	11.4	45.5	22.7	79.5	21.3
	Self-CoT	52.2	40.6	49.2	39.0	66.5	43.2	36.6	55.2	40.9	40.2	39.7	53.0	77.3	65.9	52.3	36.4	48.5
	PS-CoT	45.8	18.7	34.0	32.8	64.0	63.1	44.6	41.3	48.8	50.5	44.7	32.8	69.8	56.8	64.3	62.8	55.9
	w/ Hint	44.9	62.2	55.9	48.1	29.0	54.5	30.5	51.4	31.8	45.4	9.1	63.4	2.3	22.7	38.6	90.9	26.9
Llama-3.1-70B	Base	93.8	70.9	69.8	62.8	72.7	51.5	78.7	88.8	81.8	75.4	82.6	81.0	72.7	59.1	47.7	43.2	50.8
	Self-CoT	93.6	71.4	69.7	54.8	96.0	84.1	87.1	95.9	88.6	67.9	86.4	**85.7**	97.7	93.2	77.3	**97.7**	**76.7**
	PS-CoT	**94.5**	68.7	72.7	61.7	93.7	83.2	93.9	**98.5**	**90.8**	77.0	**93.9**	84.2	93.2	90.9	81.8	90.9	72.9
	w/ Hint	93.6	**73.9**	77.4	**71.6**	72.7	74.2	80.4	93.6	88.6	**84.9**	84.1	83.5	70.5	60.2	65.9	75.0	58.4
Llama-3.1-405B	Base	82.0	62.9	70.0	60.9	96.6	65.9	61.5	78.1	74.2	69.4	32.6	82.4	97.7	94.3	45.5	79.5	38.3
	Self-CoT	87.7	62.2	74.2	62.2	95.5	75.8	78.5	90.8	87.9	73.2	63.6	83.4	**100.0**	90.9	59.1	88.6	67.1
	PS-CoT	84.5	67.4	76.0	66.7	92.0	86.7	**94.7**	94.7	88.3	79.1	93.2	81.1	97.7	85.2	**90.9**	88.6	74.9
	w/ Hint	85.4	68.3	75.1	66.7	**98.3**	70.5	74.5	87.2	74.2	78.0	59.1	84.9	97.7	**97.7**	50.0	84.1	46.5
Mistral-7B	Base	32.5	42.1	44.9	40.2	9.1	4.5	14.8	33.5	6.1	30.8	0.0	47.7	0.0	6.8	0.0	0.0	11.3
	Self-CoT	56.5	35.1	40.3	36.6	34.7	15.9	33.7	54.3	28.8	49.2	64.4	43.9	6.8	23.9	13.6	13.6	8.1
	PS-CoT	43.9	19.7	22.9	15.6	14.8	18.2	34.6	44.1	18.9	30.6	56.8	29.1	22.7	6.8	13.6	22.7	19.5
	w/ Hint	34.6	39.4	52.7	40.5	10.2	6.8	12.7	36.5	9.8	34.3	0.0	48.9	0.0	8.0	0.0	0.0	10.6

Table 3: RougeL score for open sourced LLMs' performance. **Bolded** text represent the best performance in the column. Underlined text represent the second best performance in the column.

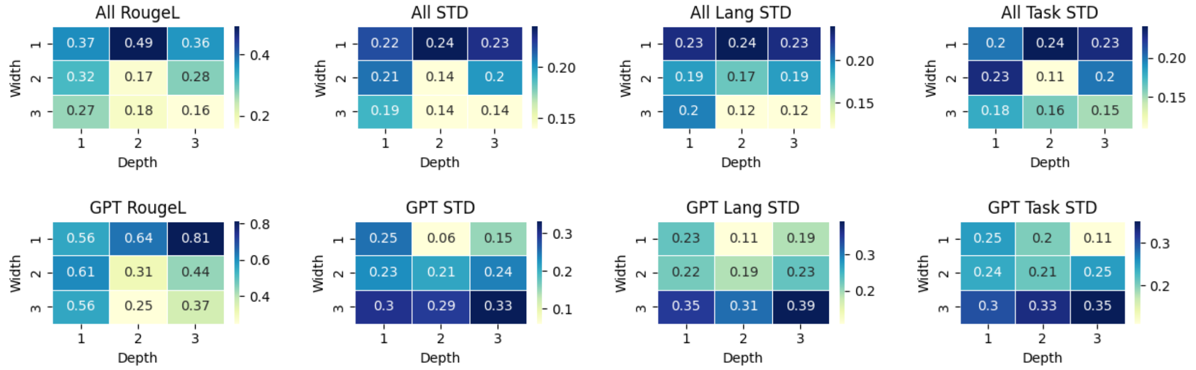


Figure 4: Heatmaps illustrating the correlation of RougeL scores and standard deviations (STD) across different models and evaluation criteria. The rows represent different levels of depth, and the columns represent varying levels of width, indicating increasing task complexity. "All" refers to combined results across languages and tasks, while "GPT" shows results specific to GPT-based models. "Lang STD" and "Task STD" indicate the variability in performance across different languages and tasks, respectively.

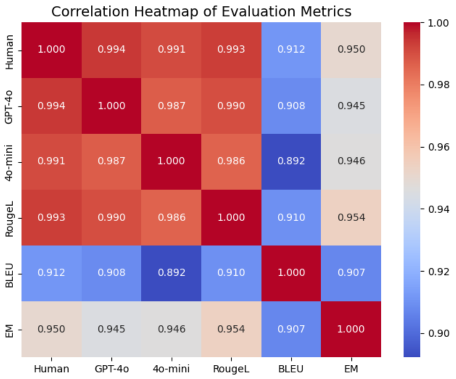
ence capabilities. The "w/ Hint" method, which introduces human reasoning rules, compensates for the model's limitations in reasoning through complex structures. Conversely, while the "Self-CoT" and "PS-CoT" approaches encourage step-by-step reasoning, they often result in logical inconsistencies and reasoning errors in complex tasks due to the requirement for autonomous generation of reasoning paths.

5.2 Performance Comparison on StrucText-Eval Test and Real-Test

Fig. 6 shows that most LLMs demonstrate comparable performance across both test sets, with variations typically within three percentage points. This consistency validates the effectiveness of our synthetic data design in simulating real-world scenarios. Moreover, introducing rule hints makes the performance disparity between the two test sets more pronounced. Llama3.1-405B's advantage in Real-Test further amplifies, exceeding its Test set performance by over six percentage points. Sim-

Model	Prompt			
	Base	w/ Hint	3-Shot	Simple 3-Shot
Human	92.6	-	-	-
GPT-4o-Turbo	**51.1**	**54.2**	**69.5**	<u>49.7</u>
GPT-4o-Mini	39.3	47.7	<u>65.6</u>	39.9
Gemini1.5-Pro	11.2	15.7	53.0	12.5
Gemini1.5-Pro-Flash	12.9	12.9	38.3	11.9
GLM-4-Plus	<u>47.3</u>	<u>50.9</u>	65.8	**51.7**
GLM-4-Flash	40.9	47.8	55.2	41.7
QWen-2-7B	29.6	35.0	51.9	30.0
QWen-2-72B	<u>42.5</u>	<u>45.3</u>	**61.4**	36.2
Llama-3.1-8B	22.3	26.7	33.7	34.2
Llama-3.1-70B	**45.8**	**56.0**	<u>58.4</u>	**50.1**
Llama-3.1-405B	34.4	41.7	48.7	<u>40.6</u>
Mistral-0.2-7B	7.0	9.5	21.0	6.9

Table 4: Performance of all LLMs and Humans on StrucText-Eval-Hard. **Bolded** text represent the best performance in the column. <u>Underlined</u> text represent the second best performance in the column.



	Human	GPT-4o	4o-mini	RougeL	BLEU	EM
Human	1.000	0.994	0.991	0.993	0.912	0.950
GPT-4o	0.994	1.000	0.987	0.990	0.908	0.945
4o-mini	0.991	0.987	1.000	0.986	0.892	0.946
RougeL	0.993	0.990	0.986	1.000	0.910	0.954
BLEU	0.912	0.908	0.892	0.910	1.000	0.907
EM	0.950	0.945	0.946	0.954	0.907	1.000

Figure 5: Correlation between different evaluation metrics.

ilarly, Llama3.1-8B demonstrates enhanced performance on Real-Test, achieving results approximately 3.5 percentage points higher. However, Qwen2-7B exhibits a contrasting trend, with its Real-Test performance falling approximately six percentage points below its Test set results. These divergent patterns suggest that rule hints influence models' capacity to generalize to authentic data.

5.3 Overall Performance on StrucText-Eval Hard

Table 4 presents the performance of various models on the StrucText-Eval Hard dataset, characterized by more complex tasks with longer sequences and deeper structures. This complexity results in a significant performance decline across all models. For instance, the accuracy of the Qwen2-72B-Instruct model decreases from 78.4% to 65.0%, while the Meta-Llama-3.1-70B-Instruct-Turbo model's accuracy drops sharply from 75.4% to 43.2%. Unlike the standard dataset, the Hard dataset demands more advanced reasoning skills, and even with the "w/ Hint" strategy, models achieve only limited improvements, in contrast to the substantial gains observed in more straightforward contexts. Notably, human accuracy on StrucText-Eval-Hard reaches 95.7%, significantly surpassing that of the best-performing large language models (LLMs), highlighting a considerable gap in models' capabilities for structured reasoning.

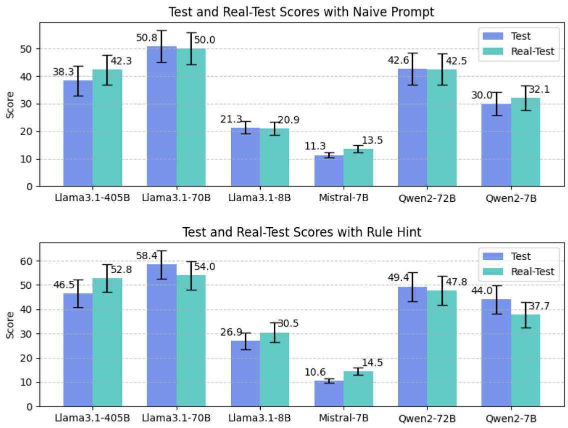


Figure 6: Performance comparison among open-source models on StrucText-Eval Test and Real-Test.

This performance gap can be primarily attributed to biases in training data and the limitations of current prompting methods. The StrucText-Eval Hard dataset, with increased question complexity and depth, requires models to possess enhanced abstraction abilities and a deeper understanding of complex structures. However, most models are trained on relatively more straightforward structured text, which makes them less effective when tackling deeply nested reasoning tasks. Additionally, prompting methods like "w/ Hint" fail to achieve human-level understanding in multilayered scenarios. The differences in prompting methods become more pronounced with increased complexity; more straightforward methods, such as Self-CoT, need to be revised for guiding models through multi-step reasoning in these challenging contexts. While the "3-shot demonstration" approach significantly improves model performance, the simpler "simple 3-shot" method, despite following similar reasoning rules, fails to match the former due to its insufficient complexity.

5.4 Performance Gap on Human & LLMs with Different Ability

Fig. 7 reveal significant performance variations among GPT-4, Qwen-2.7B, and human participants in structured data processing tasks. GPT-4 demonstrates superior performance in compu-

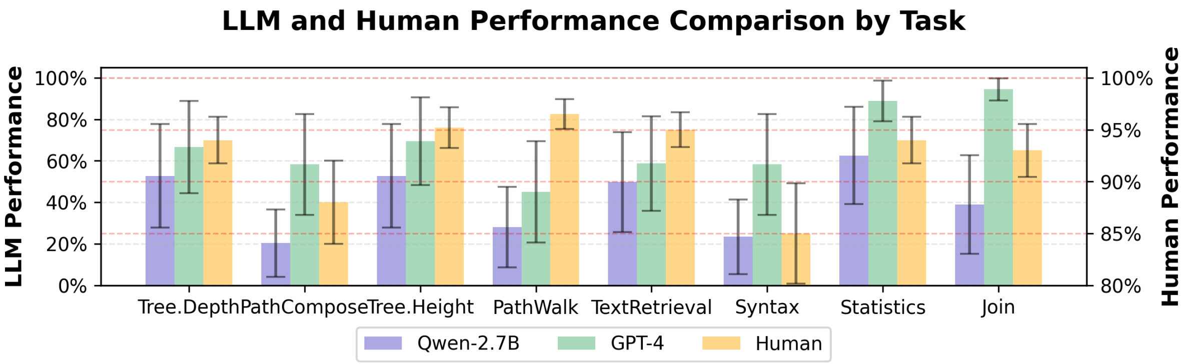


Figure 7: Performance on StrucText-Eval Hard on best LLM, famous small llm, and human in each task. LLM Performance is plotted against the left y-axis, while Human Performance is plotted against the right y-axis

tational tasks, achieving over 88% accuracy in Join and Statistics operations, substantially outperforming Qwen2-7B's modest results, which are 38.89% and 62.50%, respectively. Moreover, GPT-4 exhibits enhanced stability across tasks, particularly in computational operations, with standard deviations below 0.35, whereas Qwen-2.7B shows higher variability with standard deviations exceeding 0.4.

Human participants excel in copy-intensive tasks such as PathWalk with 96.50% and TextRetrieval with 95.00%, significantly surpassing both models' performance in these areas. However, in computational tasks, human performance aligns closely with GPT-4, suggesting that advanced language models have achieved near-human capability in specific structured computational operations. These findings underscore the complementary strengths of human cognition and artificial intelligence in processing structured data while highlighting the impact of model scale on performance stability and complex reasoning capabilities.

5.5 Model Performance Across Different Difficulty Levels, Languages, and Tasks

Figure 4 illustrates the performance variations of models across different languages and tasks. The two figures on the left reveal that, while numerical differences exist among models, including GPT models, they exhibit a consistent trend: Increasing the reference's depth and width results in a significant decline in performance. Notably, all models show a high variance in performance when the depth and width are high, suggesting that the StrucText-Eval effectively distinguishes the capabilities of most models under these conditions.

However, for GPT models, substantial variance in performance is observed only when the depth and width increase significantly, indicating that the StrucText-Eval-Hard Test suite is necessary to differentiate the performance of more advanced models better. Additionally, there is considerable variance in model performance across different languages and tasks, suggesting substantial differences in models' proficiency in handling various linguistic and task-specific challenges. This discrepancy is likely due to biases in training samples and the varying difficulty levels of those samples, as suggested by earlier analyses.

5.6 Correlation Between Different Metrics

Figure 5 presents the correlations between various evaluation metrics. The high correlation between Human Judge and GPT-4o Judge (0.9937) indicates a strong alignment between GPT-4o's automated assessments and human evaluations. Although Exact Match exhibits a notable correlation with Human Judge (0.9501), its stringent criteria often result in scores significantly lower than those of human evaluators, making it less suitable for capturing the diversity and naturalness of model outputs. Among the metrics, RougeL stands out with a correlation of 0.9932 with Human Judge, demonstrating its effectiveness in capturing surface-level textual similarity while maintaining high consistency with human judgments. Compared to the more rigid Exact Match and the relatively lower correlation of BLEU, RougeL offers a better balance between textual similarity and evaluation accuracy.

6 Conclusion

The capability to directly interpret structural-rich text in a free-text format is an essential skill all LLMs require. In response, we have developed StrucText-Eval to evaluate this capability of LLMs. We find that the proficiency of current LLMs in training on these structural-rich texts varies depending on user frequency, leading to different performance when the same tasks are performed in various languages. LLMs' understanding of structural-rich texts remains superficially tied to the training data, and these models need a profound understanding of the structure itself. This deficiency becomes evident when LLMs encounter complex structures composed of common languages or need to parse structural-rich text by custom languages, resulting in significant performance degradation.

Limitation

This paper focuses on evaluating LLM's reasoning abilities on structure-rich text by designing a benchmark called StrucText-Eval. However, StrucText-Eval includes only eight types of structured languages and encompasses a total of 29 different tasks. Given the vast array of actual structured languages and the myriad methodologies employed beyond these 29 types, StrucText-Eval can only partially represent the LLMs' capacity to understand structure-rich text. Additionally, due to regional restrictions, we are unable to utilize some highly effective baseline LLMs, such as Gemini and Claude. Therefore, the conclusions drawn in this paper are based on the assumption that GPT-4 and GPT-4 Turbo represent the top-tier LLMs now.

Ethical Concern

We contend that this article is devoid of ethical concerns for several reasons:

1. **Nature of StrucText-Eval Content**: StrucText-Eval is primarily composed of structured language syntax and some nonsensical strings, which do not present potential ethical issues such as gender bias or racial discrimination.

2. **Objective Presentation of Experimental Results**: The experimental results pertaining to StrucText-Eval objectively demonstrate the comprehension abilities of various large models on structure-rich text included in the benchmark. We have thoroughly validated the outputs and assessment details of the models to ensure that the entire evaluation adheres to the experimental setup and maintains objectivity.

3. **Completion of Manual Tasks**: All manual tasks associated with this study were conducted by the authors themselves, thereby eliminating any issues of unfair labor practices or unethical cost imposition.

Acknowledge

This work was supported by the National Natural Science Foundation of China(62476145).

References

2023. *Org Mode Manual: History and Acknowledgments*. Free Software Foundation. Accessed: 2024-03-18.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Tim Bray, Jean Paoli, C Michael Sperberg-McQueen, Eve Maler, and François Yergeau. 1998. Extensible markup language (xml) 1.0.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Martin Campbell-Kelly. 2003. *The history of mathematical tables: from Sumer to spreadsheets*. Oxford University Press.

Weize Chen, Chenfei Yuan, Jiarui Yuan, Yusheng Su, Chen Qian, Cheng Yang, Ruobing Xie, Zhiyuan Liu, and Maosong Sun. 2024. Beyond natural language: Llms leveraging alternative formats for enhanced reasoning and communication. *arXiv preprint arXiv:2402.18439*.

Zhikai Chen, Haitao Mao, Hang Li, Wei Jin, Hongzhi Wen, Xiaochi Wei, Shuaiqiang Wang, Dawei Yin, Wenqi Fan, Hui Liu, et al. 2023. Exploring the potential of large language models (llms) in learning on graphs. *arXiv preprint arXiv:2307.03393*.

Yao Cheng, Yige Zhao, Jianxiang Yu, and Xiang Li. 2024. Boosting graph foundation model from structural perspective. *arXiv preprint arXiv:2407.19941*.

Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms*. MIT press.

Clark Evans. 2001. Yaml draft 0.1. Yahoo! Tech groups: sml-dev. Archived from the original on 2001-06-03.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. 2023. Talk like a graph: Encoding graphs for large language models. *arXiv preprint arXiv:2310.04560*.

John Gruber. 2012. Markdown: Syntax. *URL http://daringfireball. net/projects/markdown/syntax. Retrieved on June*, 24:640.

Jiayan Guo, Lun Du, Hengyu Liu, Mengyu Zhou, Xinyi He, and Shi Han. 2023. Gpt4graph: Can large language models understand graph structured data? an empirical evaluation and benchmarking. *arXiv preprint arXiv:2305.15066*.

Vivek Gupta, Pranshu Kandoi, Mahek Bhavesh Vora, Shuo Zhang, Yujie He, Ridho Reinanda, and Vivek Srikumar. 2023. Temptabqa: Temporal question answering for semi-structured tables. *arXiv preprint arXiv:2311.08002*.

Rikui Huang, Wei Wei, Xiaoye Qu, Wenfeng Xie, Xianling Mao, and Dangyang Chen. 2024. Joint multi-facts reasoning network for complex temporal question answering over knowledge graph. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 10331–10335. IEEE.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Leslie Lamport. 1985. Latex : A document preparation system.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Zhaoyang Liu, Zeqiang Lai, Zhangwei Gao, Erfei Cui, Zhiheng Li, Xizhou Zhu, Lewei Lu, Qifeng Chen, Yu Qiao, Jifeng Dai, et al. 2023. Controlllm: Augment language models with tools by searching on graphs. *arXiv preprint arXiv:2310.17796*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*.

Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Mehran Kazemi, Rami Al-Rfou, and Jonathan Halcrow. 2024. Let your graph do the talking: Encoding structured data for llms. *arXiv preprint arXiv:2402.05862*.

Felipe Pezoa, Juan L Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. 2016. Foundations of json schema. In *Proceedings of the 25th international conference on World Wide Web*, pages 263–273.

Sameer Pimparkhede, Mehant Kammakomati, Srikanth Tamilselvam, Prince Kumar, Ashok Pon Kumar, and Pushpak Bhattacharyya. 2024. Doccgen: Document-based controlled code generation. *arXiv preprint arXiv:2406.11925*.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. 2022. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, pages 645–654.

Yu Sun, Shuohuan Wang, Shikun Feng, Siyu Ding, Chao Pang, Junyuan Shang, Jiaxiang Liu, Xuyi Chen, Yanbin Zhao, Yuxiang Lu, et al. 2021. Ernie 3.0: Large-scale knowledge enhanced pre-training for language understanding and generation. *arXiv preprint arXiv:2107.02137*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. 2023a. Graphgpt: Graph instruction tuning for large language models. *arXiv preprint arXiv:2310.13023*.

Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. 2024. Graphgpt: Graph instruction tuning for large language models. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 491–500.

Xiangru Tang, Yiming Zong, Yilun Zhao, Arman Cohan, and Mark Gerstein. 2023b. Struc-bench: Are large language models really good at generating complex structured data? *arXiv preprint arXiv:2309.08963*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*.

Guanyuan Yu, Xv Wang, Qing Li, and Yu Zhao. 2024. Fusing llms and kgs for formal causal reasoning behind financial risk contagion. *arXiv preprint arXiv:2407.17190*.

Hanyang Yuan, Jiarong Xu, Cong Wang, Ziqi Yang, Chunping Wang, Keting Yin, and Yang Yang. 2024. Unveiling privacy vulnerabilities: Investigating the role of structure in graph data. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4059–4070.

Xiaokang Zhang, Jing Zhang, Zeyao Ma, Yang Li, Bohan Zhang, Guanlin Li, Zijun Yao, Kangli Xu, Jinchang Zhou, Daniel Zhang-Li, et al. 2024. Tablellm: Enabling tabular data manipulation by llms in real office usage scenarios. *arXiv preprint arXiv:2403.19318*.

Qifang Zhao, Weidong Ren, Tianyu Li, Xiaoxiao Xu, and Hong Liu. 2023. Graphgpt: Graph learning with generative pre-trained transformers. *arXiv preprint arXiv:2401.00529*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

Kun Zhou, Yutao Zhu, Zhipeng Chen, Wentong Chen, Wayne Xin Zhao, Xu Chen, Yankai Lin, Ji-Rong Wen, and Jiawei Han. 2023. Don't make your llm an evaluation benchmark cheater. *arXiv preprint arXiv:2311.01964*.

A Case Study

Two case studies illustrate the evaluation setup of StrucText-Eval (Figure 8). In the JSON-based Text Retrieval task, GPT4-Turbo accurately identified deeply nested objects and adhered to the free-text format for outputting dictionary types, reflecting its firm grasp of structured text. Minimax also produced a correct answer but deviated from the prescribed format, a common issue explored in existing research. In contrast, GPT4-Turbo initially failed to merge two tables and deduce the correct record count without fine-tuning in the SQL-based Join task. However, a finetuned model steadily improved, achieving the correct solution after 5100 training steps. This progression demonstrates the importance of task-specific fine-tuning in enhancing models' capabilities in handling complex SQL queries and database structures.

Aspect	Requirements
Structure	• Maintain the original data structure and format • Do not alter the nesting levels or relationships
Content	• Use real-world examples from assigned scenarios (e.g., e-commerce, finance, sports) • Ensure data values are realistic and scenario-appropriate • Maintain semantic relationships between related fields
Reference	• Base modifications on actual examples from the assigned scenario • Keep data consistency within each reference • Avoid sensitive or identifiable information

Table 5: Guidelines for Manual Data Annotation

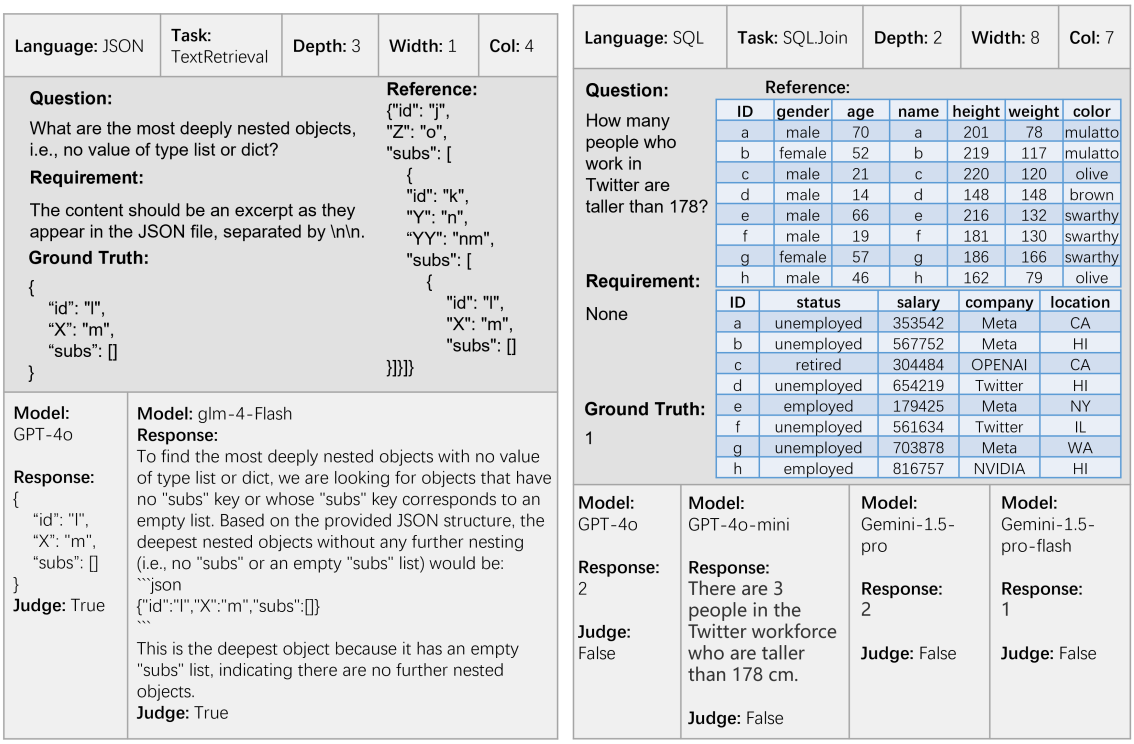
B Few-Shot Demonstration on Structural Text Inference

Figure 9 demonstrates that model performance improves with an increasing number of demonstrations under Few-Shot settings. In the 3-shot scenario, GPT-4 achieves an accuracy of 69.5%, significantly outperforming models like Gemini-Pro-Flash and Mistral, which remain around 21% or lower. The Qwen-2-72B-Instruct model shows steady improvement as more examples are provided, although it continues to trail behind GPT-4. Generally, performance increases from 1-shot to 3-shot, but the gains become less pronounced at 5-shot, with some models showing overfitting. In contrast, the performance of CoT and PS approaches remains less consistent as the number of demonstrations increases.

This trend suggests that a more significant number of examples helps models to understand problem structures and reasoning processes better, thereby enhancing their inference capabilities. However, providing too many examples can lead to models overfitting to specific patterns, which diminishes their ability to generalize to new tasks. The quality and diversity of examples are critical—high-quality examples can guide practical reasoning, while poor examples may mislead the models. While few-shot learning enhances model adaptability, those with limited pretraining data or lower parameter counts may struggle to capitalize on this approach entirely. For CoT and PS methods, the reasoning process requires additional steps, which means that simply increasing the number of few-shot demonstrations does not consistently yield performance improvements.

C Detail about Manual Works

This paper involves the manual works in writing Question Templates, acquisition of human performance on StrucText-Eval-Hard-Test and construct-



Language: JSON | **Task:** TextRetrieval | **Depth:** 3 | **Width:** 1 | **Col:** 4

Question:

What are the most deeply nested objects, i.e., no value of type list or dict?

Requirement:

The content should be an excerpt as they appear in the JSON file, separated by \n\n.

Ground Truth:

```
{
   “id”: "l",
   “X”: "m",
   “subs”: []
}
```

Reference:

```
{"id": "j",
"Z": "o",
"subs": [
   {
   "id": "k",
   "Y": "n",
   “YY": "nm",
   "subs": [
      {
         "id": "l",
         "X": "m",
         "subs": []
}]}]}
```

Model: GPT-4o

Response:

```
{
   “id”: "l",
   “X”: "m",
   “subs”: []
}
```

Judge: True

Model: glm-4-Flash

Response:

To find the most deeply nested objects with no value of type list or dict, we are looking for objects that have no "subs" key or whose "subs" key corresponds to an empty list. Based on the provided JSON structure, the deepest nested objects without any further nesting (i.e., no "subs" or an empty "subs" list) would be:

~~~
```json
{"id":"l","X":"m","subs":[]}
```
~~~

This is the deepest object because it has an empty "subs" list, indicating there are no further nested objects.

Judge: True

Language: SQL | **Task:** SQL.Join | **Depth:** 2 | **Width:** 8 | **Col:** 7

Question:

How many people who work in Twitter are taller than 178?

Requirement:

None

Ground Truth:

1

Reference:

ID	gender	age	name	height	weight	color
a	male	70	a	201	78	mulatto
b	female	52	b	219	117	mulatto
c	male	21	c	220	120	olive
d	male	14	d	148	148	brown
e	male	66	e	216	132	swarthy
f	male	19	f	181	130	swarthy
g	female	57	g	186	166	swarthy
h	male	46	h	162	79	olive

ID	status	salary	company	location
a	unemployed	353542	Meta	CA
b	unemployed	567752	Meta	HI
c	retired	304484	OPENAI	CA
d	unemployed	654219	Twitter	HI
e	employed	179425	Meta	NY
f	unemployed	561634	Twitter	IL
g	unemployed	703878	Meta	WA
h	employed	816757	NVIDIA	HI

Model: GPT-4o

Response: 2

Judge: False

Model: GPT-4o-mini

Response: There are 3 people in the Twitter workforce who are taller than 178 cm.

Judge: False

Model: Gemini-1.5-pro

Response: 2

Judge: False

Model: Gemini-1.5-pro-flash

Response: 1

Judge: False

Figure 8: Cases for performance of different LLMs and finetuned stages on Structured Text.

ing Real-Test Suite. All annotation works are carried out by the authors of this paper, so there is no payment for manual annotation.

C.1 Development of Question Templates

The development and validation of Question Templates constituted a significant component of our methodological framework. Three researchers collaboratively formulated and verified 29 distinct Question Templates. To ensure transparency and reproducibility, we have made these templates accessible to the academic community through our public repository.

C.2 Human Performance Evaluation

To establish a robust human baseline for the StrucText-Eval-Hard-Test, we conducted a comprehensive evaluation process. Three researchers independently responded to an identical set of 500 questions, with each researcher dedicating approximately 17 hours to this task. The human performance metrics presented in Table 4 represent the mean scores calculated from this substantial dataset of 1,500 responses.

C.3 Construction of Real-Test Suite

The development of the StrucText-Eval-Real-Test Suite involved five researchers in a systematic annotation process. Initially, the first author generated 928 diverse scenario categories, encompassing domains such as athletics, financial services, glassware specifications, academic writing etc. Subsequently, these scenarios were systematically assigned to individual questions. The annotators were tasked with modifying samples according to their assigned scenarios, adhering to specific annotation guidelines as detailed in Table 5. This process resulted in a comprehensive test suite of 928 questions.

D Other Metrics

Given the substantial expense in evaluating all results using multiple metrics, we selected a subset of 300 test results for each model on the StrucText-Hard dataset, using a naive prompting method for assessment. The complete evaluation results are presented in Table 6.

E Detail Prompt

The prompts used in the experiment can be categorized into three types: Example of Base Prompts are shown in Tab. 7. Example of CoT Prompts are shown in Tab. 8. Example of Few-Shot Prompts are shown in Tab. 9. Example of Rule Hints are shown in Tab. 10.

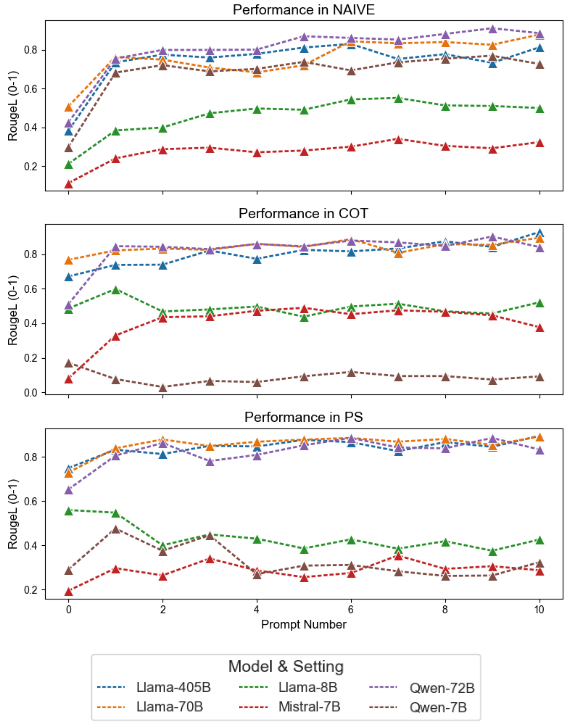


Figure 9: The model's performance on StrucText-Eval-Test under different Few-Shot Demonstration settings.

Model	Human	GPT-4o	4o-Mini	RougeL	BLEU	EM
GPT-4o-Turbo	**56.13**	**55.75**	**51.00**	**51.1**	<u>45.94</u>	**40.31**
GPT-4o-Mini	36.15	36.02	40.73	39.3	**46.08**	33.93
Gemini1.5-Pro	12.39	12.80	10.62	11.2	12.60	8.75
Gemini1.5-Pro-Flash	13.83	13.19	12.96	12.9	14.01	9.67
GLM-4-Plus	<u>52.90</u>	<u>52.62</u>	<u>46.02</u>	<u>47.3</u>	32.75	<u>38.27</u>
GLM-4-Flash	41.50	41.34	38.99	40.9	37.43	34.80
QWen-2-7B	32.95	31.99	30.10	29.6	27.98	18.70
QWen-2-72B	<u>40.87</u>	<u>38.66</u>	31.24	<u>42.5</u>	<u>37.76</u>	**35.67**
Llama-3.1-8B	21.78	21.98	22.36	22.3	20.88	14.75
Llama-3.1-70B	**46.64**	**41.38**	**40.83**	**45.8**	**41.50**	<u>27.46</u>
Llama-3.1-405B	35.01	35.97	<u>35.88</u>	34.4	28.00	21.29
Mistral-0.2-7B	7.85	7.33	7.32	7.0	5.09	4.47

Table 6: Performance of all LLMs and Humans on StrucText-Eval-Hard based on different metrics (1,000 samples for each metrics).

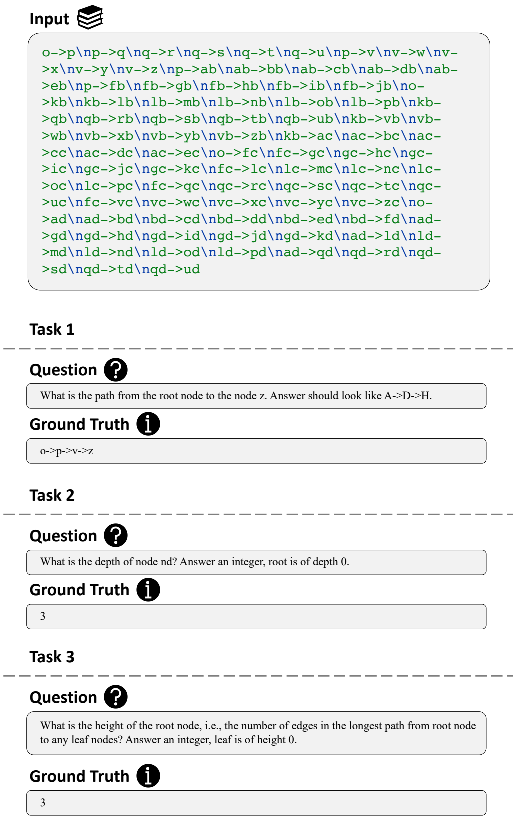
F Examples for All Languages & Tasks

In this section, we provide detailed examples for each language we discuss, illustrating how specific tasks are executed within those languages. These examples are meant to offer clear insights into the application and utility of each language in various contexts. Through these demonstrations, readers can better understand the unique features and capabilities of each language when applied to different tasks.

F.1 Tree

See Figure 10.

Figure 10: Sample input and tasks of Tree.



Input

```
o->p\np->q\nq->r\nq->s\nq->t\nq->u\np->v\nv->w\nv-
>x\nv->y\nv->z\np->ab\nab->bb\nab->cb\nab->db\nab-
>eb\np->fb\nfb->gb\nfb->hb\nfb->ib\nfb->jb\no-
>kb\nkb->lb\nlb->mb\nlb->nb\nlb->ob\nlb->pb\nkb-
>qb\nqb->rb\nqb->sb\nqb->tb\nqb->ub\nkb->vb\nvb-
>wb\nvb->xb\nvb->yb\nvb->zb\nkb->ac\nac->bc\nac-
>cc\nac->dc\nac->ec\no->fc\nfc->gc\ngc->hc\ngc-
>ic\ngc->jc\ngc->kc\nfc->lc\nlc->mc\nlc->nc\nlc-
>oc\nlc->pc\nfc->qc\nqc->rc\nqc->sc\nqc->tc\nqc-
>uc\nfc->vc\nvc->wc\nvc->xc\nvc->yc\nvc->zc\no-
>ad\nad->bd\nbd->cd\nbd->dd\nbd->ed\nbd->fd\nad-
>gd\ngd->hd\ngd->id\ngd->jd\ngd->kd\nad->ld\nld-
>md\nld->nd\nld->od\nld->pd\nad->qd\nqd->rd\nqd-
>sd\nqd->td\nqd->ud
```

Task 1

Question ?

What is the path from the root node to the node z. Answer should look like A->D->H.

Ground Truth ℹ

o->p->v->z

Task 2

Question ?

What is the depth of node nd? Answer an integer, root is of depth 0.

Ground Truth ℹ

3

Task 3

Question ?

What is the height of the root node, i.e., the number of edges in the longest path from root node to any leaf nodes? Answer an integer, leaf is of height 0.

Ground Truth ℹ

3

F.2 Tabular

See Figure 11.

F.3 JSON

See Figure 12.

F.4 YAML

See Figure 13.

F.5 XML

See Figure 14.

F.6 LaTeX

See Figure 15.

F.7 Markdown

See Figure 16.

F.8 Org

See Figure 17.

G Rules & Rule Hints

We list all the rules in Regular Express in this section, and list all the hints for these rules in Lis. 1.

```
# -*- coding: utf-8 -*-
Variables:
!<INPUT 0>! – Language
!<INPUT 1>! – Question
!<INPUT 2>! – Reference
!<INPUT 3>! – Requirement
<commentblockmarker>###</commentblockmarker>
you are a !<INPUT 0>! file parser, you are required to answer questions pertaining to the given !<INPUT
0>! file.

### Question:
!<INPUT 1>!

### Reference:
!<INPUT 2>!

### Requirement:
!<INPUT 3>!

Please follow the format below for your output:

### Answer:
xxxxx
```

Table 7: Prompt of `Naive Prompt` method

G.1 Tree

We build tree-structured input as a list of edges in a tree, in a format of "`father->child`", separated by newline.

$$\begin{aligned} identifier &\coloneqq \texttt{[a-z]+} \\ Edge &\coloneqq identifier\texttt{->}identifier \\ Tree &\coloneqq Edge(\backslash\texttt{n}Edge)\star \\ InputFile &\coloneqq Tree \end{aligned}$$

G.2 Tabular

Formally, input texts are classified as tabular data given that they are composed of a list of newline separated lines, each of which is a list of text cells delimited by comma.

$$\begin{aligned} head &\coloneqq \texttt{[A-Z][a-z]*} \\ cell &\coloneqq \texttt{[A-Za-z0-9]+} \\ headline &\coloneqq identifier(,identifier)\star \\ subline &\coloneqq cell(,cell)\star \\ Tabular &\coloneqq headline(\backslash\texttt{n}subline)+ \\ InputFile &\coloneqq Tabular \end{aligned}$$

G.3 JSON

Due to the inherit hierarchy structure of Object Notations, we adopted a recursive scheme to define our input texts.

$$\begin{aligned} lb_{(left\ bracket)} &\coloneqq \texttt{[[]} \\ rb &\coloneqq \texttt{[]]} \\ val &\coloneqq \texttt{[a-z]+} \\ key &\coloneqq \texttt{[A-Z]+} \\ JSON &\coloneqq \{ \\ &\qquad \texttt{"id":"}val\texttt{"} \\ &\qquad \texttt{"subs":}lbrb | lbJSON(,\backslash\texttt{n}JSON \\ &\qquad)\star rb \\ &\qquad \texttt{("}key\texttt{":"}val\texttt{"}\backslash\texttt{n)+} \\ &\qquad \} \\ InputFile &\coloneqq JSON \end{aligned}$$

G.4 YAML

The rules for constructing YAML and XML input are similarly recursive.

```
# -*- coding: utf-8 -*-
Variables:
!<INPUT 0>! – Language
!<INPUT 1>! – Question
!<INPUT 2>! – Reference
!<INPUT 3>! – Requirement
<commentblockmarker>###</commentblockmarker>
you are a !<INPUT 0>! file parser, you are required to answer questions pertaining to the given !<INPUT
0>! file.

### Question:
!<INPUT 1>!

### Reference:
!<INPUT 2>!

### Requirement:
!<INPUT 3>!

Please follow the format below for your output:

### Reasoning Prcess:
xxxx

### Answer:
xxxxx
```

Table 8: Prompt of `CoT` method

$$\begin{aligned} YAML :=& \\ & \texttt{id} : val \\ & \texttt{subs} : lbrb | (\texttt{\textbackslash n}(\texttt{\textbackslash t}) * \texttt{-}\ YAML) \\ & + (key : val\texttt{\textbackslash n}) + \\ InputFile :=& \ YAML \end{aligned}$$

G.5 XML

$$\begin{aligned} firstline :=& \texttt{<?xml version="1.0"} \\ & textttencoding = \text{“UTF-8”}\texttt{?>} \\ XML :=& \\ & firstline \\ & XMLObject \\ tag :=& \texttt{[A-Z]+} \\ val :=& \texttt{[a-z]+} \\ attr :=& \texttt{[A-Z]+="}val\texttt{"} \\ content :=& \texttt{[a-z \textbackslash n\textbackslash t]*} \\ XMLObject :=& \\ & <tag(\ \ attr) * > \\ & ((\texttt{\textbackslash t}) * XMLObject) * \\ & content \\ & </tag> \\ InputFile :=& \ XML \end{aligned}$$

G.6 LaTeX

In LaTeX input texts, we include `textbf` and `includegraphics` commands to accommo-

```
# -*- coding: utf-8 -*-
Variables:
!<INPUT 0>! – Language
!<INPUT 1>! – Demonstration
!<INPUT 2>! – Question
!<INPUT 3>! – Reference
!<INPUT 4>! – Requirement
<commentblockmarker>###</commentblockmarker>
you are a !<INPUT 0>! file parser, you are required to answer questions pertaining to the given !<INPUT
0>! file.

### Demonstration:
!<INPUT 1>!

### Question:
!<INPUT 2>!

### Reference:
!<INPUT 3>!

### Requirement:
!<INPUT 4>!

Please follow the format below for your output:

### Answer:
xxxxx
```

Table 9: Prompt of `Few Shot` method

date for the text retrieval tasks. The headings serve as anchors for structure traversal.

$$\begin{aligned}
command &:= \backslash(\texttt{section}|\texttt{subsection}| \\
&\qquad \texttt{subsubsection}) \\
heading &:= command\{\texttt{[a-z]+}\}|\texttt{[a-z]+} \\
inclg &:= \\
&\qquad \texttt{\textbackslash includegraphics[width=} \\
&\qquad \texttt{0.5\textbackslash textwidth]}\{\texttt{[a-z]+[.]} \\
&\qquad \texttt{(png|jpg|jpeg|gif)}\} \\
bf &:= \backslash\texttt{textbf}\{\texttt{[a-z]+}\} \\
content &:= (\texttt{[a-z]}|bf|inclg)+ \\
LaTeX &:= heading\backslash\texttt{n}content(\backslash\texttt{n}LaTeX)* \\
InputFile &:= LaTeX
\end{aligned}$$

G.7 Markdown

In markdown input texts, the syntax counterparts for heading, text face and including figure are employed in our dataset.

$$\begin{aligned}
heading &:= \texttt{[\#]* [a-z]+} \\
inclg &:= !lb\texttt{alt}rb\backslash\texttt{([a-z]+[.](png} \\
&\qquad \texttt{|jpg|jpeg|gif)} \\
&\qquad \texttt{"hover text"\textbackslash)} \\
bf &:= \texttt{[*]\{2\}[a-z]+[*]\{2\}} \\
content &:= (\texttt{[a-z]}|bf|inclg)+ \\
Markdown &:= heading\backslash\texttt{n} \\
&\qquad content(\backslash\texttt{n}Markdown)* \\
InputFile &:= Markdown
\end{aligned}$$

G.8 Org

In Org input texts, the syntax is obtained from JSON construction rules by replacing the markups

```
# -*- coding: utf-8 -*-
Variables:
!<INPUT 0>! – Language
!<INPUT 1>! – Question
!<INPUT 2>! – Reference
!<INPUT 3>! – Requirement
!<INPUT 4>! – Rule Hint
<commentblockmarker>###</commentblockmarker>
you are a !<INPUT 0>! file parser, you are required to answer questions pertaining to the given !<INPUT 0>! file.

### Question:
!<INPUT 1>!

### Reference:
!<INPUT 2>!

### Requirement:
!<INPUT 3>!

### Rule Hint:
!<INPUT 4>!

Please follow the format below for your output:

### Answer:
xxxxx
```

Table 10: Prompt of `\w Hint` method

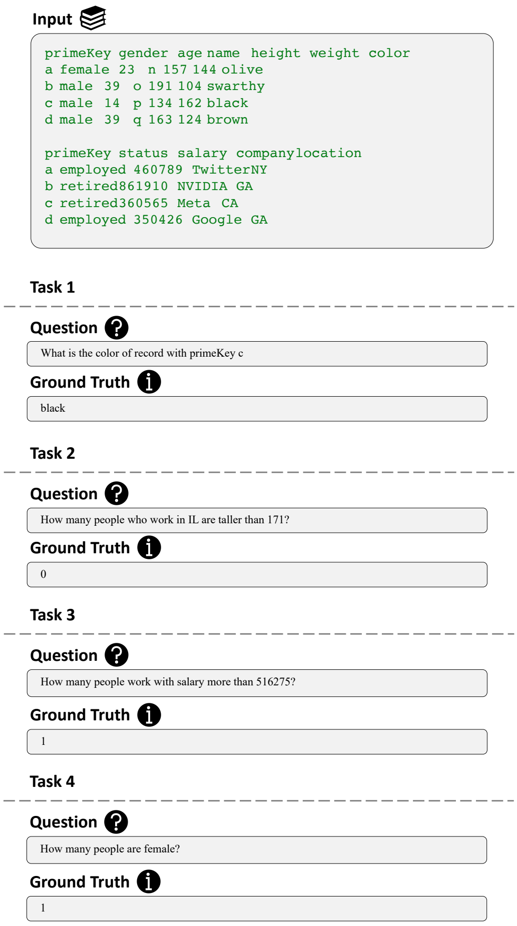
for heading, including figures and bold font face.

$$\begin{aligned} heading &:= \texttt{[*]* [a-z]+} \\ inclg &:= lb\{2\}\texttt{[a-z]+[.](png|jpg|} \\ &\quad \texttt{jpeg|gif)}rb\{2\} \\ bf &:= \texttt{[*][a-z]+[*]} \\ content &:= (\texttt{[a-z]}|bf|inclg)+ \\ Org &:= heading\backslash ncontent(\backslash nOrg)\ast \\ InputFile &:= Org \end{aligned}$$

Listing 1: All rule hints in StrucText-Eval

```
SQL,Tree,JSON,YAML,XML,Markdown,LaTeX,
    ↪ ORG
To find the value of specific field of
    ↪ record with specified primeKey.
    ↪ You have to first, locate the line
    ↪  with the specific primeKey. Then
    ↪ find the required value under the
    ↪ desired column in that line.
To get the number of people with salary
    ↪ above a threshold, you need to
    ↪ find the table with salary
    ↪ information. Then you go over each
    ↪  line and check the salary field.
    ↪ During the process count only
    ↪ those lines with value of salary
    ↪ strictly greater than the
    ↪ specified threshold towards your
    ↪ final sum. The sum after checking
    ↪ each line is the right answer.
To get the number of female, first find
    ↪ the table with column name ''.
    ↪ Then check each line for field
    ↪ gender, and count these lines with
    ↪  value 'female' towards your final
    ↪  sum. The process applies to
    ↪ finding number of male too.
To get the number of people living in
    ↪ specified city who are also taller
    ↪  than threshold, you need to first
    ↪  join the two table on primeKey,
    ↪ and check each row of joined table
    ↪  for lines that satisfies both
    ↪ condition, i.e., lines with city
    ↪ specified in query and height
    ↪ strictly greater than threshold.
    ↪ The total number of such rows is
    ↪ the right answer.
To answer the height of tree, you need
    ↪ to take a recursive strategy. For
    ↪ each node, you will find its
    ↪ height by first finding its
    ↪ children's heights. Then, the
```

Figure 11: Sample input and tasks of tabular data.



Input

```
primeKey gender age name height weight color
a female 23  n 157 144 olive
b male  39  o 191 104 swarthy
c male  14  p 134 162 black
d male  39  q 163 124 brown

primeKey status salary companylocation
a employed 460789 TwitterNY
b retired861910 NVIDIA GA
c retired360565 Meta CA
d employed 350426 Google GA
```

Task 1

Question

What is the color of record with primeKey c

Ground Truth

black

Task 2

Question

How many people who work in IL are taller than 171?

Ground Truth

0

Task 3

Question

How many people work with salary more than 516275?

Ground Truth

1

Task 4

Question

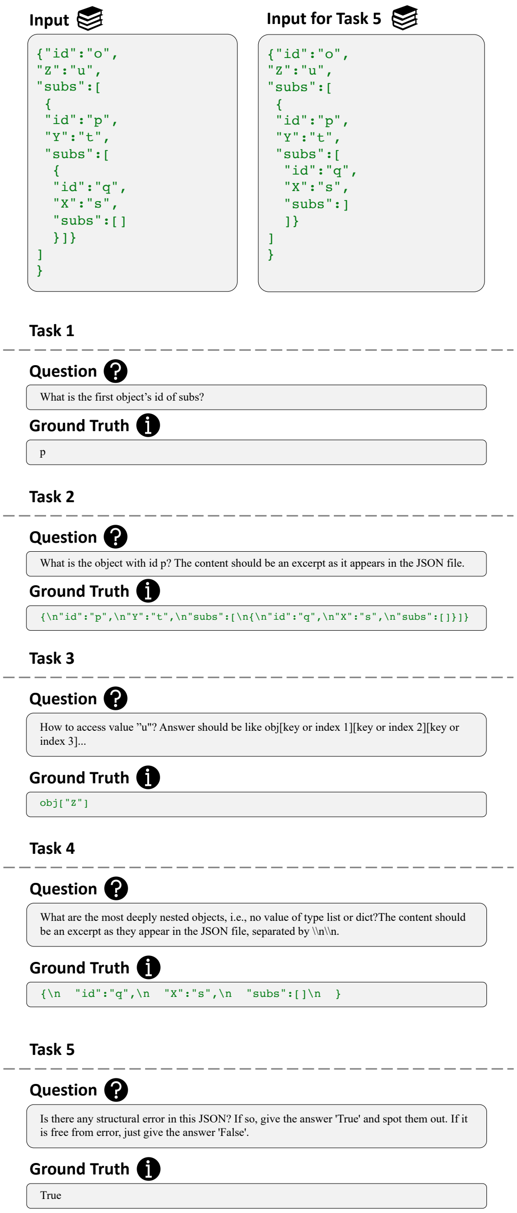
How many people are female?

Ground Truth

1

```
    ↪ height of node is the maximum
    ↪ subtree heights plus 1. The base
    ↪ case occurs when a node has no
    ↪ children, i.e., it's a leaf node.
    ↪ Leaf's height is defined to be 0,
    ↪ without the need of further
    ↪ queries. Then the height the tree
    ↪ is the height of its root node.
To find the depth of a node, you need to
    ↪  find the number of edges from
    ↪ root to node. You have to start
    ↪ from the root with depth 0 and
    ↪ assign the depth for each node
    ↪ recursively. For any given node,
    ↪ it gets depth of current depth.
    ↪ Increment the depth by 1 before go
    ↪  to its subtree and repeat the
    ↪ process until every node gets a
    ↪ depth.
To get the path from root to a node, you
    ↪  need to find recursively. For any
    ↪  node, you can find the path to
    ↪ the target node by find path from
    ↪ its children to target. Then check
    ↪  each child's output, if any child
    ↪  returns with valid path instead
    ↪ of an empty path indicating target
    ↪ -not-found, the path from node to
```

Figure 12: Sample input and tasks of JSON.



Input

```
{"id":"o",
"Z":"u",
"subs":[
 {
 "id":"p",
 "Y":"t",
 "subs":[
  {
  "id":"q",
  "X":"s",
  "subs":[]
  }]}
]
}
```

Input for Task 5

```
{"id":"o",
"Z":"u",
"subs":[
 {
 "id":"p",
 "Y":"t",
 "subs":[
  "id":"q",
  "X":"s",
  "subs":]
  ]}
]
}
```

Task 1

Question

What is the first object's id of subs?

Ground Truth

p

Task 2

Question

What is the object with id p? The content should be an excerpt as it appears in the JSON file.

Ground Truth

`{\n"id":"p",\n"Y":"t",\n"subs":[\n{\n"id":"q",\n"X":"s",\n"subs":[]}]}`

Task 3

Question

How to access value ”u"? Answer should be like obj[key or index 1][key or index 2][key or index 3]...

Ground Truth

`obj["Z"]`

Task 4

Question

What are the most deeply nested objects, i.e., no value of type list or dict?The content should be an excerpt as they appear in the JSON file, separated by \\n\\n.

Ground Truth

`{\n  "id":"q",\n  "X":"s",\n  "subs":[]\n  }`

Task 5

Question

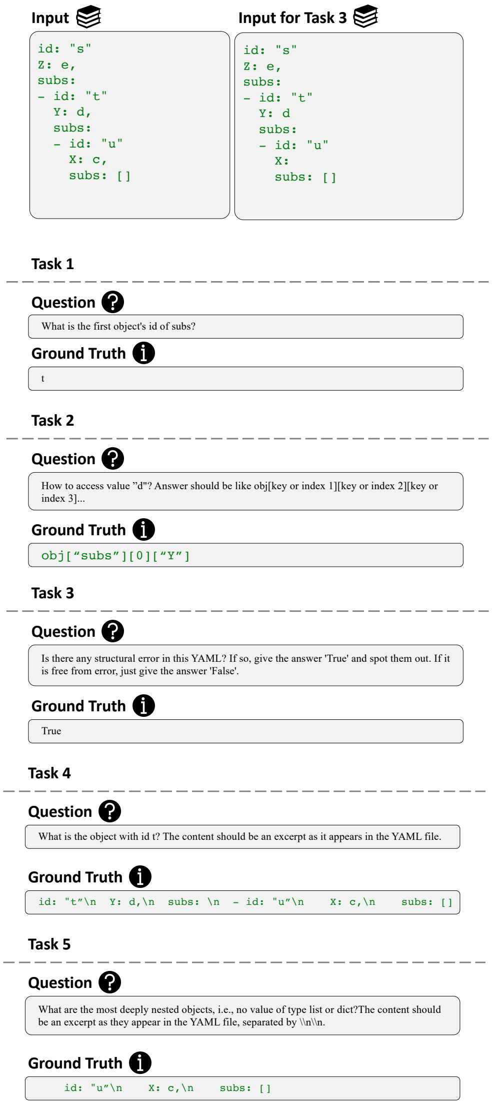
Is there any structural error in this JSON? If so, give the answer 'True' and spot them out. If it is free from error, just give the answer 'False'.

Ground Truth

True

```
    ↪ target is that path from its child
    ↪  to target prepended with itself.
    ↪ The answer can be found by
    ↪ searching with root as starting
    ↪ point.
To find the object with specified id,
    ↪ you need to first parse the json
    ↪ file and get the outermost object,
    ↪  starting from which search the
    ↪ subs field recursively and looking
    ↪  for the desired value in id field
    ↪  for each visited object. Retrieve
    ↪  the content of that object once
    ↪ found.
To find the first object's id of subs,
    ↪ first parse the json file and get
```

Figure 13: Sample input and tasks of YAML.



Input

```
id: "s"
Z: e,
subs:
- id: "t"
  Y: d,
  subs:
  - id: "u"
    X: c,
    subs: []
```

Input for Task 3

```
id: "s"
Z: e,
subs:
- id: "t"
  Y: d
  subs:
  - id: "u"
    X:
    subs: []
```

Task 1

Question

What is the first object's id of subs?

Ground Truth

t

Task 2

Question

How to access value ”d"? Answer should be like obj[key or index 1][key or index 2][key or index 3]...

Ground Truth

`obj[“subs”][0][“Y”]`

Task 3

Question

Is there any structural error in this YAML? If so, give the answer 'True' and spot them out. If it is free from error, just give the answer 'False'.

Ground Truth

True

Task 4

Question

What is the object with id t? The content should be an excerpt as it appears in the YAML file.

Ground Truth

```
id: "t”\n  Y: d,\n  subs: \n  - id: "u”\n    X: c,\n    subs: []
```

Task 5

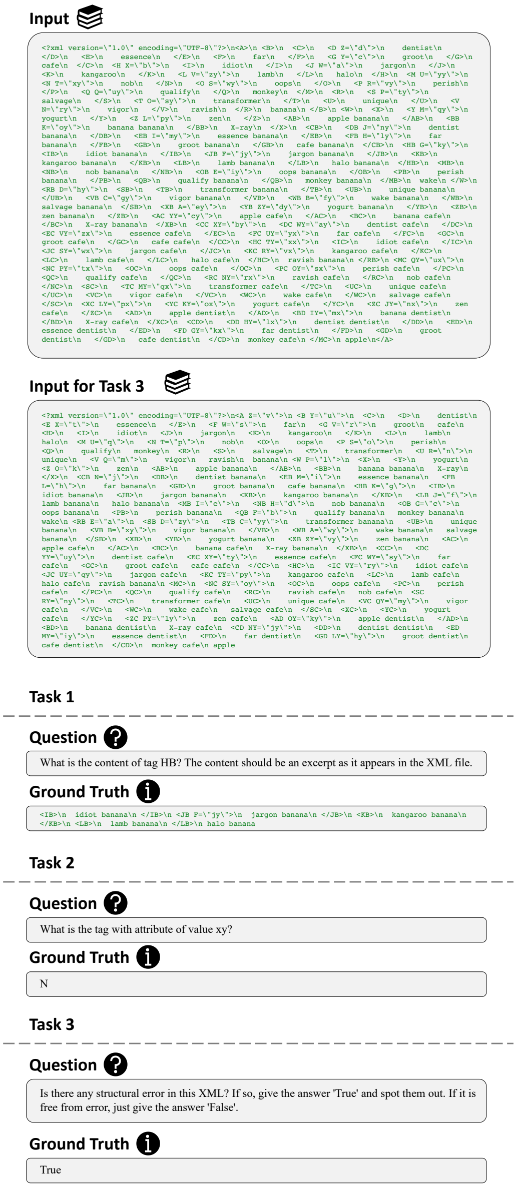
Question

What are the most deeply nested objects, i.e., no value of type list or dict?The content should be an excerpt as they appear in the YAML file, separated by \\n\\n.

Ground Truth

```
    id: "u”\n    X: c,\n    subs: []
```

Figure 14: Sample input and tasks of XML.



Input

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<A>\n <B>\n  <C>\n   <D Z=\"d\">\n    dentist\n
</D>\n   <E>\n    essence\n   </E>\n   <F>\n    far\n   </F>\n   <G Y=\"c\">\n    groot\n   </G>\n
cafe\n  </C>\n  <H X=\"b\">\n   <I>\n    idiot\n   </I>\n   <J W=\"a\">\n    jargon\n   </J>\n
<K>\n    kangaroo\n   </K>\n   <L V=\"zy\">\n    lamb\n   </L>\n   halo\n  </H>\n  <M U=\"yy\">\n
<N T=\"xy\">\n    nob\n   </N>\n   <O S=\"wy\">\n    oops\n   </O>\n   <P R=\"vy\">\n    perish\n
</P>\n   <Q Q=\"uy\">\n    qualify\n   </Q>\n   monkey\n  </M>\n  <R>\n   <S P=\"ty\">\n
salvage\n   </S>\n   <T O=\"sy\">\n    transformer\n   </T>\n   <U>\n    unique\n   </U>\n   <V
N=\"ry\">\n    vigor\n   </V>\n   ravish\n  </R>\n  banana\n </B>\n <W>\n  <X>\n   <Y M=\"qy\">\n
yogurt\n   </Y>\n   <Z L=\"py\">\n    zen\n   </Z>\n   <AB>\n    apple banana\n   </AB>\n   <BB
K=\"oy\">\n    banana banana\n   </BB>\n   X-ray\n  </X>\n  <CB>\n   <DB J=\"ny\">\n    dentist
banana\n   </DB>\n   <EB I=\"my\">\n    essence banana\n   </EB>\n   <FB H=\"ly\">\n    far
banana\n   </FB>\n   <GB>\n    groot banana\n   </GB>\n   cafe banana\n  </CB>\n  <HB G=\"ky\">\n
<IB>\n    idiot banana\n   </IB>\n   <JB F=\"jy\">\n    jargon banana\n   </JB>\n   <KB>\n
kangaroo banana\n   </KB>\n   <LB>\n    lamb banana\n   </LB>\n   halo banana\n  </HB>\n  <MB>\n
<NB>\n    nob banana\n   </NB>\n   <OB E=\"iy\">\n    oops banana\n   </OB>\n   <PB>\n    perish
banana\n   </PB>\n   <QB>\n    qualify banana\n   </QB>\n   monkey banana\n  </MB>\n  wake\n </W>\n
<RB D=\"hy\">\n  <SB>\n   <TB>\n    transformer banana\n   </TB>\n   <UB>\n    unique banana\n
</UB>\n   <VB C=\"gy\">\n    vigor banana\n   </VB>\n   <WB B=\"fy\">\n    wake banana\n   </WB>\n
salvage banana\n  </SB>\n  <XB A=\"ey\">\n   <YB ZY=\"dy\">\n    yogurt banana\n   </YB>\n   <ZB>\n
zen banana\n   </ZB>\n   <AC YY=\"cy\">\n    apple cafe\n   </AC>\n   <BC>\n    banana cafe\n
</BC>\n   X-ray banana\n  </XB>\n  <CC XY=\"by\">\n   <DC WY=\"ay\">\n    dentist cafe\n   </DC>\n
<EC VY=\"zx\">\n    essence cafe\n   </EC>\n   <FC UY=\"yx\">\n    far cafe\n   </FC>\n   <GC>\n
groot cafe\n   </GC>\n   cafe cafe\n  </CC>\n  <HC TY=\"xx\">\n   <IC>\n    idiot cafe\n   </IC>\n
<JC SY=\"wx\">\n    jargon cafe\n   </JC>\n   <KC RY=\"vx\">\n    kangaroo cafe\n   </KC>\n
<LC>\n    lamb cafe\n   </LC>\n   halo cafe\n  </HC>\n  ravish banana\n </RB>\n <MC QY=\"ux\">\n
<NC PY=\"tx\">\n  <OC>\n    oops cafe\n   </OC>\n   <PC OY=\"sx\">\n    perish cafe\n   </PC>\n
<QC>\n    qualify cafe\n   </QC>\n   <RC NY=\"rx\">\n    ravish cafe\n   </RC>\n   nob cafe\n
</NC>\n  <SC>\n   <TC MY=\"qx\">\n    transformer cafe\n   </TC>\n   <UC>\n    unique cafe\n
</UC>\n   <VC>\n    vigor cafe\n   </VC>\n   <WC>\n    wake cafe\n   </WC>\n   salvage cafe\n
</SC>\n  <XC LY=\"px\">\n   <YC KY=\"ox\">\n    yogurt cafe\n   </YC>\n   <ZC JY=\"nx\">\n    zen
cafe\n   </ZC>\n   <AD>\n    apple dentist\n   </AD>\n   <BD IY=\"mx\">\n    banana dentist\n
</BD>\n   X-ray cafe\n  </XC>\n  <CD>\n   <DD HY=\"lx\">\n    dentist dentist\n   </DD>\n   <ED>\n
essence dentist\n   </ED>\n   <FD GY=\"kx\">\n    far dentist\n   </FD>\n   <GD>\n    groot
dentist\n   </GD>\n   cafe dentist\n  </CD>\n  monkey cafe\n </MC>\n apple\n</A>
```

Input for Task 3

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<A Z=\"v\">\n <B Y=\"u\">\n  <C>\n   <D>\n    dentist\n
<E X=\"t\">\n    essence\n   </E>\n   <F W=\"s\">\n    far\n   <G V=\"r\">\n    groot\n   cafe\n
<H>\n   <I>\n    idiot\n   <J>\n    jargon\n   <K>\n    kangaroo\n   </K>\n   <L>\n    lamb\n
halo\n  <M U=\"q\">\n   <N T=\"p\">\n    nob\n   <O>\n    oops\n   <P S=\"o\">\n    perish\n
<Q>\n    qualify\n   monkey\n  <R>\n   <S>\n    salvage\n   <T>\n    transformer\n   <U R=\"n\">\n
unique\n   <V Q=\"m\">\n    vigor\n   ravish\n  banana\n <W P=\"l\">\n  <X>\n   <Y>\n    yogurt\n
<Z O=\"k\">\n    zen\n   <AB>\n    apple banana\n   </AB>\n   <BB>\n    banana banana\n   X-ray\n
</X>\n  <CB N=\"j\">\n   <DB>\n    dentist banana\n   <EB M=\"i\">\n    essence banana\n   <FB
L=\"h\">\n    far banana\n   <GB>\n    groot banana\n   cafe banana\n  <HB K=\"g\">\n   <IB>\n
idiot banana\n   <JB>\n    jargon banana\n   <KB>\n    kangaroo banana\n   </KB>\n   <LB J=\"f\">\n
lamb banana\n   halo banana\n  <MB I=\"e\">\n   <NB H=\"d\">\n    nob banana\n   <OB G=\"c\">\n
oops banana\n   <PB>\n    perish banana\n   <QB F=\"b\">\n    qualify banana\n   monkey banana\n
wake\n <RB E=\"a\">\n  <SB D=\"zy\">\n   <TB C=\"yy\">\n    transformer banana\n   <UB>\n    unique
banana\n   <VB B=\"xy\">\n    vigor banana\n   </VB>\n   <WB A=\"wy\">\n    wake banana\n   salvage
banana\n  </SB>\n  <XB>\n   <YB>\n    yogurt banana\n   <ZB ZY=\"vy\">\n    zen banana\n   <AC>\n
apple cafe\n   </AC>\n   <BC>\n    banana cafe\n   X-ray banana\n  </XB>\n  <CC>\n   <DC
YY=\"uy\">\n    dentist cafe\n   <EC XY=\"ty\">\n    essence cafe\n   <FC WY=\"sy\">\n    far
cafe\n   <GC>\n    groot cafe\n   cafe cafe\n  </CC>\n  <HC>\n   <IC VY=\"ry\">\n    idiot cafe\n
<JC UY=\"qy\">\n    jargon cafe\n   <KC TY=\"py\">\n    kangaroo cafe\n   <LC>\n    lamb cafe\n
halo cafe\n  ravish banana\n <MC>\n  <NC SY=\"oy\">\n   <OC>\n    oops cafe\n   <PC>\n    perish
cafe\n   </PC>\n   <QC>\n    qualify cafe\n   <RC>\n    ravish cafe\n   nob cafe\n  <SC
RY=\"ny\">\n   <TC>\n    transformer cafe\n   <UC>\n    unique cafe\n   <VC QY=\"my\">\n    vigor
cafe\n   </VC>\n   <WC>\n    wake cafe\n   salvage cafe\n  </SC>\n  <XC>\n   <YC>\n    yogurt
cafe\n   </YC>\n   <ZC PY=\"ly\">\n    zen cafe\n   <AD OY=\"ky\">\n    apple dentist\n   </AD>\n
<BD>\n    banana dentist\n   X-ray cafe\n  <CD NY=\"jy\">\n   <DD>\n    dentist dentist\n   <ED
MY=\"iy\">\n    essence dentist\n   <FD>\n    far dentist\n   <GD LY=\"hy\">\n    groot dentist\n
cafe dentist\n  </CD>\n  monkey cafe\n apple
```

Task 1

Question

What is the content of tag HB? The content should be an excerpt as it appears in the XML file.

Ground Truth

```
<IB>\n  idiot banana\n </IB>\n <JB F=\"jy\">\n  jargon banana\n </JB>\n <KB>\n  kangaroo banana\n
</KB>\n <LB>\n  lamb banana\n </LB>\n halo banana
```

Task 2

Question

What is the tag with attribute of value xy?

Ground Truth

N

Task 3

Question

Is there any structural error in this XML? If so, give the answer 'True' and spot them out. If it is free from error, just give the answer 'False'.

Ground Truth

True

```
    ↪ the outermost object, in the
    ↪ outermost object’s subs list, get
    ↪ the first element. That element is
    ↪  another object, and its id is the
    ↪  answer.
To find the error in the json file, you
    ↪ need to parse the json file and
    ↪ report any syntax error if
    ↪ encountered any. Potential errors
    ↪ include missing ending curly
    ↪ braces.
To get the path to access specified
    ↪ value. You have to do a recursive
    ↪ search along the subs fields,
    ↪ starting from the outermost parsed
    ↪  object. For each visited object,
    ↪ check each fields except for subs,
    ↪  and record the path along the way
    ↪ , i.e., subs inside brackets and
```

```
    ↪ index into subs inside brackets,
    ↪ and at which field you find the
    ↪ value.
To get the most deeply nested objects,
    ↪ start from the outermost object,
    ↪ recursively search along the subs
    ↪ fields. For each object, check its
    ↪  subs field, any object with an
    ↪ empty subs is one most deeply
    ↪ nested object.
To find the object with specified id,
    ↪ you need to first parse the yaml
    ↪ file and get the outermost object,
    ↪  starting from which search the
    ↪ subs field recursively and looking
    ↪  for the desired value in id field
    ↪  for each visited object. Retrieve
    ↪  the content of that object once
```

Figure 15: Sample input and tasks of LaTeX.

Input

```
o
monkey \textbf{banana}nob wake yogurt groot wake
jargon ravish
\section{p}
nob nob wake
\textbf{cafe}yogur\includegraphics[width=0.5\textwid
th]{mh.jpeg}t groot wake jargon ravish
\subsection{q}
oops nob wake yogurt groot wake
jargon\textbf{dentist} ravish
```

Task 1

Question

Extract all bold texts. Print those raw texts separated by \\n.

Ground Truth

```
banana\ncafe\ndentist
```

Task 2

Question

Extract all included graph files. Print those file names separated by \\n.

Ground Truth

```
mh.jpeg
```

Task 3

Question

What is the content of 1th section? The content should be an excerpt as it appears in the LaTeX file, including the heading line and any sub-section.

Ground Truth

```
\section{p}
nob nob wake
\textbf{cafe}yogur\includegraphics[width=0.5\textwidth]{mh.jpeg
}t groot wake jargon ravish
\subsection{q}
oops nob wake yogurt groot wake jargon\textbf{dentist} ravish
```

```
    ↪ found.
To find the first object's id of subs,
    ↪ first parse the yaml file and get
    ↪ the outermost object, in the
    ↪ outermost object's subs list, get
    ↪ the first element. That element is
    ↪  another object, and its id is the
    ↪  answer.
To find the error in the yaml file, you
    ↪ need to parse the yaml file and
    ↪ report any syntax error if
    ↪ encountered any. Potential errors
    ↪ include missing key before colon.
To get the path to access specified
    ↪ value. You have to do a recursive
    ↪ search along the subs fields,
    ↪ starting from the outermost parsed
    ↪  object. For each visited object,
    ↪ check each fields except for subs,
    ↪  and record the path along the way
    ↪ , i.e., subs inside brackets and
    ↪ index into subs inside brackets,
    ↪ and at which field you find the
    ↪ value.
To get the most deeply nested objects,
    ↪ start from the outermost object,
    ↪ recursively search along the subs
    ↪ fields. For each object, check its
    ↪  subs field, any object with an
    ↪ empty subs is one most deeply
    ↪ nested object.
To find the content of a specific tag,
```

Figure 16: Sample input and tasks of Markdown.

Input

```
w
banana cafe vigor cafe peris![alt](mj.gif "hover
text")h perish monkey wake
# x
cafe cafe vigor cafe perish peris**banana**h monkey
wake
## y
dentist cafe vigor c**cafe**![alt](nj.jpg "hover
text")afe perish perish monkey wake
```

Task 1

Question

Extract all bold texts. Print those raw texts separated by \\n.

Ground Truth

```
cafe\nbanana
```

Task 2

Question

Extract all included image files. Print those file names separated by \\n.

Ground Truth

```
mj.gif\nnj.jpg
```

Task 3

Question

What is the content of 1th section? The content should be an excerpt as it appears in the markdown file, including the heading line and any sub-section.

Ground Truth

```
# x
cafe cafe vigor cafe perish peris**banana**h monkey wake
## y
dentist cafe vigor c**cafe**![alt](nj.jpg "hover text")afe
perish perish monkey wake
```

Figure 17: Sample input and tasks of Org.

Input

```
p
kanga*lamb*roo zen yogurt X-ray halo zen nob qualify
* q
lamb zen yogurt X-ray halo zen nob qu[[ei.jpg]]alify
** r
monkey zen yogurt X-ray halo zen nob qualify
```

Task 1

Question

Extract all bold texts. Print those raw texts separated by \\n.

Ground Truth

```
lamb
```

Task 2

Question

Extract all included image files. Print those file names separated by \\n.

Ground Truth

```
ei.jpg
```

Task 3

Question

What is the content of 1th subsection under 1th section? The content should be an excerpt as it appears in the org file, including the heading line and any sub-section.

Ground Truth

```
** r\nmonkey zen yogurt X-ray halo zen nob qualify
```

```
    ↪ you need to search for desired tag
    ↪  throughout the xml file. Once
    ↪ located, find the surrounding left
    ↪  and right angle, these area is
    ↪ tha starting tag. Then find the
    ↪ ending tag, which is the tag
    ↪ surrounded by angle with exception
    ↪  that right angle is preceded by a
    ↪  slash. The content between
    ↪ starting and ending tags is the
    ↪ answer.
To find the tag name of particular
    ↪ attribute value, just search the
    ↪ file for that value and find the
    ↪ surrounding left and right angles,
    ↪  i.e., boundary of tag. The word
    ↪ next to left angle is tag name.
To find the error in the xml file, you
    ↪ need to parse the xml file and
    ↪ report any syntax error if
    ↪ encountered any. Potential errors
    ↪ include missing ending tags.
To find the bold texts, search for
    ↪ double stars, i.e., **, the
    ↪ content between two occurences of
    ↪  double stars is the bold texts.
    ↪ Note that the bold range should
    ↪ start from the double stars
    ↪ occurring at i-th spot throughout
    ↪ the whole input file, where i is
    ↪ odd, and end with double stars
    ↪ occurring at jth spot where j is
    ↪ even. For example, text between
    ↪ double stars appearing first and
    ↪ second time.
To find the content of certain section,
    ↪ starting from the headings start
    ↪ with one hashtag, and go to the
    ↪ ith heading as specified in number
    ↪  of sections. Then start from that
    ↪  line, look for j-th heading with
    ↪ 2 hashtags as specified in
    ↪ subsection number. For kth
    ↪ subsubsection, look for kth
    ↪ heading with 3 hashtags starting
    ↪ from the located subsubsection.
    ↪ Stop searching early if the
    ↪ subsection or subsubsection is not
    ↪  queried.
To find the image files, look for texts
    ↪ matching ![*](TARGET "*"), the
    ↪ TARGET part is filename. Star
    ↪ means any text is possible.
To find the bold texts, search for macro
    ↪  textbf, and everything after \\
    ↪ textbf{ and before the first }
    ↪ encountered is bold text.
Note that section title is enclosed by
    ↪ \\section{}, and \\subsection for
    ↪ subsection, \\subsubsection for
    ↪ subsubsection. To find the content
    ↪  of certain section, look for ith
    ↪ section as specified, and start
    ↪ from there look for jth subsection
    ↪ . And from located subsection,
    ↪ look for kth subsubsection as
    ↪ queried. Search may stop early if
    ↪ subsection or subsubsection is not
    ↪  queried.
To find the image files imported, search
    ↪  for pattern \\includegraphics[*]{
    ↪ TARGET}, the TARGET part is the
    ↪ filename. Star means any text is
    ↪ possible.
To find the bold texts, search for
    ↪ single star, i.e., *, the content
    ↪ between two occurrences of single
    ↪ star is the bold texts. Note that
    ↪ the bold range should start from
    ↪ the single star occurring at i-th
    ↪ spot throughout the whole input
    ↪ file, where i is odd, and end with
    ↪  single star occurring at jth spot
    ↪  where j is even. For example,
    ↪ text between single star appearing
    ↪  first and second time.
Note that section, subsection,
    ↪ subsubsection titles are preceded
    ↪ by *, **, *** respectively, with
    ↪ one or more whitespaces in between
    ↪ . To find the content of certain
    ↪ section, look for ith section as
    ↪ specified, and start from there
    ↪ look for jth subsection. And from
    ↪ located subsection, look for kth
    ↪ subsubsection as queried. Search
    ↪ may stop early if subsection or
    ↪ subsubsection is not queried.
To find the image files, look for texts
    ↪ matching [[TARGET]], the TARGET
    ↪ part is filename
```

H Detail Setting

All experiments and training process is carried out on a three 3090 GPUs service. The setting of API calling is illustrated in Tab. 11

Random Seed				
torch.manual_seed	torch.cuda.manual_seed_all	numpy.random.seed	random.seed	torch.backends.cudnn.deterministirc
42	42	42	42	True
AutoCausalLM				
temperature	top_p	top_k	num_beams	max_new_token
0.95	0.95	5	2	1

Table 11: All the parameter setting in our experiments.