STRUCTURE-RICH TEXT BENCHMARK FOR KNOWLEDGE INFERENCE EVALUATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We construct a benchmark for LLMs (Large Language Models) composed of structure-rich and syntactically rigorous corpus with semantics-independent tasks, in purpose of evaluating the abilities of knowledge inference from small structured text and construction rules. The tasks also involve the capacity to generate strictly formatted response given the specification, i.e. to output the same structure-rich texts as the inputs. We also experimented on the popular LLMs with our benchmark to compare their competence to mine for information from syntax and condense information into structure.

1 INTRODUCTION

1.1 BACKGROUND

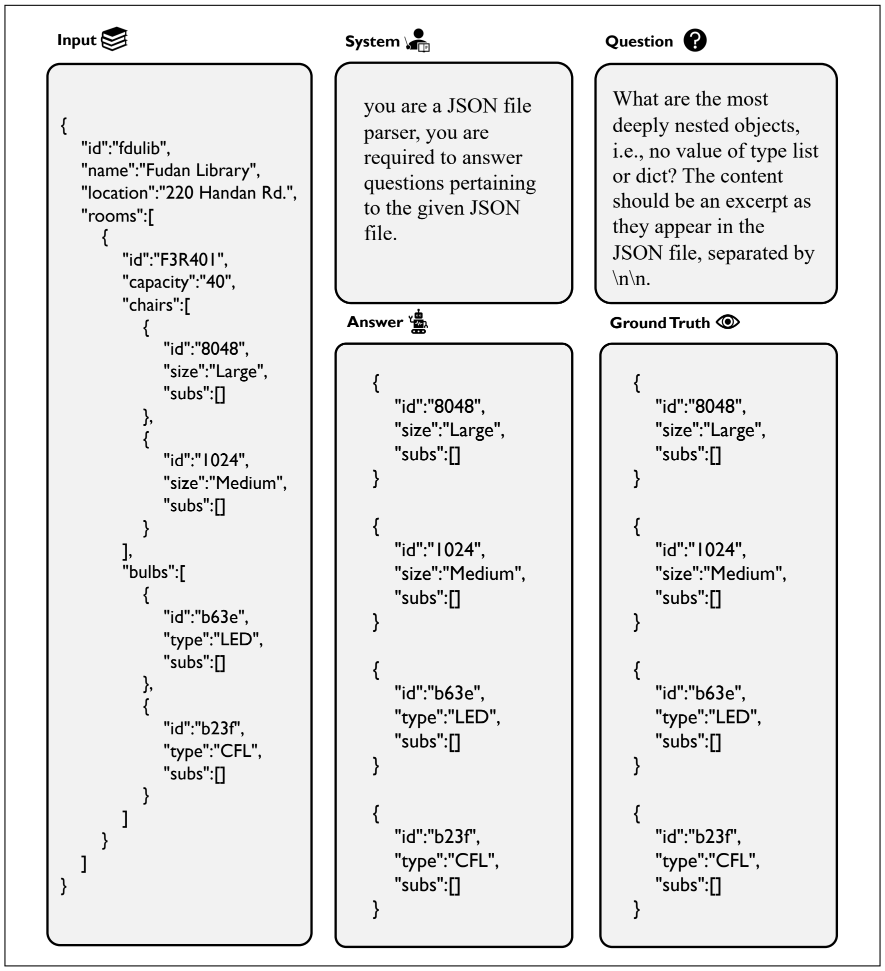
The explosive growth in the digitization of information in the world today has seen an evolving shift in the dynamics of textual data presentation. Textual communication, encapsulated traditionally in natural human languages, has diversified into more structured formats for seamless interactions among software programs. Notable examples include JavaScript Object Notation (JSON), Python, C++, Yet Another Markup Language (YAML), among others. These structured texts serve as a critical medium in various sectors like technology, health, engineering, finance, where they have found widespread application in data representation, configuration files, and codebases.

Language models, the brain behind most AI technologies, develop a comprehensive understanding of these structured formats. The ability for an artificial intelligence model to understand and generate these structured texts, therefore, presents a significant stride in contemporary global digitalization efforts. This understanding equates to unlocking a new level of competence and functionality in language models. It promises better system interoperability, more efficient parsing and interpreting of structured data, and an eventual enhancement in machine-learning-based applications. It also has far-reaching implications on the development of more adaptive and complex AI systems. Therefore, bringing language models to understand structured texts is not just a necessity, but a fundamental requirement in fast-tracking our collective global digitalization strides.

1.2 MOTIVATION

Current attempts to bridge the gap between language models and understanding of these structured texts have, however, met several limitations. Conventionally, language models have predominantly focused on understanding natural human languages, with limited attention to structured texts like JSON, YAML, and codebases. This insufficient attention can be attributed to the growing complexity, diversity, and inherent characteristics of those structured languages, such as abstract data types like trees and graphs. In recent years, significant efforts have centered on broadening the understanding capabilities of language models, with the existing body of work demonstrating considerable advancements in this regard.

However, these studies often operate within a narrow focus, primarily exploring the capacity of these models in generating and interpreting codes—a crucial aspect of structured text understanding. Although these approaches have generated promising results, their limited scope has left a critical part of the solution space unexplored. They largely concentrate on analyzing language models' abilities to decipher programming languages like Python and C++, leaving out other forms of structured



```
{
    "id":"fdulib",
    "name":"Fudan Library",
    "location":"220 Handan Rd.",
    "rooms":[
        {
            "id":"F3R401",
            "capacity":"40",
            "chairs":[
                {
                    "id":"8048",
                    "size":"Large",
                    "subs":[]
                },
                {
                    "id":"1024",
                    "size":"Medium",
                    "subs":[]
                }
            ],
            "bulbs":[
                {
                    "id":"b63e",
                    "type":"LED",
                    "subs":[]
                },
                {
                    "id":"b23f",
                    "type":"CFL",
                    "subs":[]
                }
            ]
        }
    ]
}
```

you are a JSON file parser, you are required to answer questions pertaining to the given JSON file.

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.

Answer:

```
{
    "id":"8048",
    "size":"Large",
    "subs":[]
}

{
    "id":"1024",
    "size":"Medium",
    "subs":[]
}

{
    "id":"b63e",
    "type":"LED",
    "subs":[]
}

{
    "id":"b23f",
    "type":"CFL",
    "subs":[]
}
```

Ground Truth:

```
{
    "id":"8048",
    "size":"Large",
    "subs":[]
}

{
    "id":"1024",
    "size":"Medium",
    "subs":[]
}

{
    "id":"b63e",
    "type":"LED",
    "subs":[]
}

{
    "id":"b23f",
    "type":"CFL",
    "subs":[]
}
```

Figure 1: Sample input and tasks of tree.

texts, most notably JSON and YAML. Moreover, these current works do not sufficiently account for the understanding of intricate data structures (like trees and graphs) inherent in structured texts—an essential competency for comprehensive textual understanding. This insufficiency in existing studies signals a compelling need to embark on further explorations. It exposes the urgency to develop a more holistic approach predicated not just on understanding programming codes but on a broader spectrum of structured texts.

1.3 Our Proposal

In this paper, we propose a novel benchmark for testing language models' ability to understand and generate different types of structured texts, like JSON, YAML, and programming languages like Python.

1.4 Our Method

We searched vastly structured texts that have well-documented syntax and are widely used in industry and academies. We include both fully-structured and half-structured data to enrich the diversity of our dataset. For each class of texts, we designed 3-5 tasks that are structure-related, some of which are specific to their classification.

Based on the syntax and rules for construction, we randomly generate the input texts, and procedurally generate the corresponding tasks data in the meantime. For some classes whose texts are easily accessible, we collect the texts from Internet at first and make use of the available parsers to analyze and extract the structure information that constitute the tasks to be constructed.

1.5 CONTRIBUTION

- We first thoroughly evaluated the LLMs' ability to understand and manipulate structure-rich texts.
- We present a taxonomy for structured texts and designed structure-related tasks for each class.
- With a total of 9 text classes and several tasks for each class, we present 32 structure-specific tasks and 2512 samples for our benchmark dataset.
- We also tested 4 LLMs and present the results under various metrics for assessment.

The remainder of this paper is arranged into 6 section. section 2 covers attempts in literature to build better models for structural texts centered tasks and construct benchmarks and datasets for such tasks, which have both focused on codebases and fell short of the diversity of structure-rich texts. In section 3 we propose a taxonomy for structure-rich texts that are popular in LLMs' use scenarios and define their generation rules we used, common applications and tasks we designed for each class. section 4 covers our method and framework for collecting input texts and QAs for out dataset. section 5 shows our evaluation on the benchmark for 4 LLMs, namely, GPT-4, Spark, Minimax and Ernie. In section 6 we discussed the performance (scores) distributed among the LLMs and different input types. We wrap up our work and experiment results in section 7.

2 RELATED WORKS

Previous explorations over structural information in text have mainly focused on programming language. Due to the difference between normal natural language texts and rich-structure texts, NLP models trained on natural language perform worse than models curated for structural tasksMou et al. (2016). Mou et al. (2016) built a tree-based convolutional neural network that outperformed NLP models on functionality classification tasks and pattern detection tasks over programming languages. Wang et al. (2020) achieved SOTA on code clone detection using graph neural networks and AST representation augmented by control flow of programming language inputs, which is different in architecture from common language models for NLP tasks. Zhou et al. (2019) built graph neural network to perform classification over learned rich node representations gained considerably increased accuracy and F1 scores. The weakness on structural text processing and achievements from alternative architecture shows insufficiency of attention on the realm of rich-structure text from the NLP models.

As for benchmark and datasets, previous works focus on codebases and scenarios of clone detection, vulnerability identification and so on. Svajlenko et al. (2014) proposed a benchmark over Java codes targeting on code clone detection tasks, leaving the inner structure of codes texts untouched. Zhou et al. (2019) also built a dataset from diversified large-scale open-source C projects (Linus, QEMU, Wireshark, FFmpeg) with a total of 48687 commits, manually labeled as vulnerable or non-vulnerable used for vulnerable function identification. The CODESEARCHNETHusain et al. (2019) for code query tasks consist of 6 million functions from open-source code, covering six programming language. It is noteworthy that the construction of the dataset leveraged the information in documentation associated with functions during data filtering. Our work also leveraged the information beyond code texts of which documentation and comments are most typical, when generating the ground truth for one of the python code tasks. Despite that Husain et al. (2019) used it to collect model input instead of ground truth, the similarity of the intention lies in the utilization of the semantic information as much as possible. However, rather than inspecting syntactic and structural information, the tasks in Husain et al. (2019) are semantic code search, which is more semantic-dominated. Other code related datasets include Raychev et al. (2016b) which contains 150000 JavaScript files with parsed ASTs, and Raychev et al. (2016a) which is the Python counterparts to the former one.

3 Taxonomy

3.1 Information Behind Structure

Previous benchmarks on LLMs have largely relied on the tasks where semantic information is critical and central to the correct response, e.g., question answering, summarization and information retrieval. Such tasks cover the majority of usage scenarios for language models, in which inputs are mainly sequential. However, they fail to demonstrate the ability of understanding and manipulating the structure of texts, which is too ubiquitous to be noticed in our daily use of texts.

The commonly used structure-rich texts are normally concise in volume. But through construction rules, they convey ample information that is not explicitly present in the input texts, demonstrating their expressiveness. Those implicit knowledges are inferred on-the-fly according to the rules and structure, starting from the source texts. The rules or syntaxes serve as a function that procedurally leads to different answers under different inputs(Such expressiveness is also characteristic of functional programming where lazy-evaluation is the main trait).

The implicit information is dark, in the sense that it is not readily present or detectable in the given input texts, but it dominates the construction and formation of texts (it is also massive in volume, if explicitly elicited).

Based on the expressiveness and information obscurity of structure-rich texts. We conclude that the abilities to understand these texts and tackle relevant tasks are critical to LLMs that excel in inference and generalization.

3.2 Taxonomy: Structured-Rich Text and Task

We searched present literature broadly and propose a overarching taxonomy that covers widely used structure-rich texts, including structured texts (table in relational database), semi-structured texts (XML, JSON, YAML, etc.), abstract data structure in ascii (Tree), and programming language (Python). For every class of input texts, we designed 3-5 tasks regarding the structure and semantics of the input texts. Most tasks are strongly correlated with the structure of input and their ground truths can be derived using program. On the other hand, a few tasks are semantics dependent, mostly when understanding the task instruction. Amongst all 32 tasks, only one task has answer that can not be procedurally obtained from input text. The examples of the input texts and tasks for each category are documented in A.4. For each task, we generated 20 sample input and constructed a dataset of 2512 QAs in total. In this subsection we will introduce each format class as perspective to their definition, application and tasks designed specifically. For a more detailed constructing rules we used to generate input data, see A.1. For example data for each category, see A.4.

3.2.1 Abstract Data Structure

In the realm of digital information, data is organized in a structure that could easily be stored, parsed, analyzed by machines. The theoretical model underlying these structures are abstract data structures, e.g., tree, list, graph, queue, stack, etc. Abstract data structure decides how data is queried, inserted, modified and deleted. These structures encode the relations and properties of data. The ability to understand abstract data structure will enable LLMs to rigidly follow the relation of semantic segments imposed from syntax, which is vital to data exchange between different format. The ability of LLMs solving graph problems has been explored in previous work (Wang et al. (2023)).

Tree, from the view of graph, is acyclic connected undirected graph. It is defined in the same way as a graph, i.e., a set of vertices and a set of edges connecting vertices thereof. Given the pervasive influence of tree on other recursive structured texts such as JSON, YAML and XML, we focused on tree structure and designed tree-specific tasks.

Regarding each tree, we designed 3 tasks pertaining to their structures, all of which are independent of the content in the nodes. A sample of input texts of tree category along with its associated tasks are shown in Figure 5. These tasks target on the ability to compose a path to a specified node and decide the height or depth of a node.

3.2.2 STRUCTURED

Generally, structured data refer to those well-structured data that obey certain elaborate data model which defines the relation of data elements and abstract models. Tabular data exported from data tables in relational databases are typical of such structured data. Data that are structured according to non-tabular data models are also deemed as structured data.

In this work, we focus on specific row-column structured text inputs. Such input denotes a set of objects with the same set of fields defined by the first row. The subsequent rows denotes the values of fields per object. The format is derived from tables in relational database, which has seen overwhelming domination in the realm of database.

Due to the simplicity of the input, apart from a value lookup task, we built statistical tasks as well as multiple-table tasks that are similar to inner join query, to better capture the features and practical use scenarios of such input data.

3.2.3 SEMI-STRUCTURED

As opposed to structured texts, semi-structured data are structured data that do not fit in the conventional data model (Buneman (1997)). Semi-structured data are structured and contain rich structural marks or tags that are different from semantic elements and compose the structure they are defined with. In this work, we focus specifically on texts with some extent of hierarchies designated by marks and reserved tokens. They are neither sequential nor tabular, contrary to row-column structured table as in subsubsection 3.2.2. Internally they are n-ary trees, which are dominated by recursive construction rules. Semi-structured data are employed in information exchange between application programming interfaces (JSON, YAML), storing well organized web pages on world wide webs (XML) and typesetting journals or scientific articles (Markdown, Org, LaTeX). All of these medium have been backbone of our digitalized world. In our effort to define subsets of these formats (see A.1) and generate some sample input texts from the rules we found that a simple pattern kept recurring in their construction. The pattern that intersects with every semi-structured text type we covered here is recursion. Each type have some syntax that is simple and atomic, which contribute little to the hierarchy or structural complexity, along with optional recursion, which is the essence of hierarchy construction and non-sequential structure.

OBJECT NOTATION

Object Notation is a format that could be used in modeling the in-memory objects data conceptualized in object-oriented programming language.

JSON

JSON is acronym for JavaScript Object Notation. It has been widely used in data storage, exchange, transmission and inter-operation between different API. Main stream programming languages offer complete support for exporting and importing JSON files, making it popular on occasions of complicate structured data transmission. Therefore, LLMs having a better understanding and manipulating ability with respect to JSON texts input will expedite the progress of LLMs bridging the gap between different participants and procedures in digital data flow happened in real world.

Due to the inherit hierarchy structure of Object Notations, we adopted a recursive scheme to define our input texts (see A.1.3). For JSON text, we built five tasks, inspecting abilities covering information retrieval, structure traversal, path construction, syntax correction, and depth calculation.

It is noteworthy that the depth calculation task is in essential a task of tree structure, which is in alignment with tree category in subsubsection 3.2.1. Another insights rooted in the fact that we used the same tasks interchangeably with little modification to the generating code for JSON and YAML, demonstrating their similarity internally.

YAML

YAML (Yet Another Markup Language or YAML Ain't Markup Language) is a data-oriented markup language that models object data. It supports a compact in-line style (although not recommended) that is equivalent in semantics to JSON. The difference lies in the additional syntax that

JSON does not support, e.g., comments. Besides, YAML is widely used in configuration files other than object serialization.

XML

XML(extensible markup language) is a markup language, but it is also widely used in configuration files and storing object and metadata. Here we focus mainly on its usage in object notation, instead of marking up documents.

Its syntax defines the specifications for markup (tags are typical markups in XML) and contents and recursive construction of elements constituted by tags, attributes and contents which could contain nested elements. We designed tasks regarding syntax correction, tag-based text retrieval and attribute-based text retrieval.

In terms of the influence in digitalization and importance of LLMs mastering YAML format, the same trend applies to YAML. In addition, XML as the foundation of world wide web data, mastering XML well will benefit LLMs in taking advantages of resources on the Internet better and promote the level of intelligence of language model.

MARKUP LANGUAGE

The syntax rules in markup languages are used mainly for markup excerpts of texts rather than modeling entities as others we mentioned above. Despite that the syntax of ORG, Markdown and LaTeX are no less complicated than Object Notation languages as of Chomsky hierarchy (Chomsky & Schützenberger (1959); Schützenberger (1963)), their common use scenarios are less dominated by recursive structure objects. So our tasks focus more on text retrieval and shallow-nested structure traversal. Nevertheless, we built longer input text to diversify our dataset.

Our construction of these formats used a small subset of legal commands with limited depth in nested (sub)sections.

Markdown

Markdown is a lightweight markup language for formatting text using reserved characters scattered among the content. The main goal of Markdown is to format text in a source-readable way. It has gain great popularity in technical writing, blogging, and collaborative documentations despite the syntax standardization is underway still and implementation is diverged.

Org

Org files are plain text files that include simple plain-text marks to express hierarchy. It is easy to read (for most of the time, but it could be intricate too) in source code form, which is similar to markdown. the syntax is intended to be read and modified by editors in a automated manner, which is richer and more customizable than markdown.

It is designed specifically for Org-mode of Emacs editor. The editor can read the markups in Org files and manipulate the hierarchy and other markups as a result of toggling outline objects or element status, e.g., check off to-do list items with a simple keystroke under Org-mode in Emacs. Note-taking and list management is Org's forte.

LaTeX

LaTeX is a set of TeX macros that has gain great popularity. It has simple syntax and rich command set, being highly extensible. It has been heavily used in academia for typesetting scientific documents.

Contrary to markdown and org, LaTeX file is hard to read and much more verbose. It has the most features in all markup languages we covered as a full-fledged typesetting tool.

3.2.4 PROGRAMMING LANGUAGE

Python is a dynamically typed language that has seen exclusive application in mathematics and data science. It has rich features that make it procedural, object-oriented and functional at the same time. The main trait of Python is usability (garbage collection provided by language), and expressiveness (it has syntax that captures most frequently recurring programming patterns, e.g., generators).

Due to its complexity in structure and syntax, we collected from Internet [1] 1932 files instead of procedural generation, and used ast module [2] and ast_scope module [3] to parse the source codes into abstract syntax tree. We designed problems regarding identifier scope and return type (when it is annotated explicitly). The answers are derived from parser, i.e., static analysis. We also have a task querying the purpose of the code, i.e., algorithm or model the code implements. For answering such tasks, we used the information in the python file names (out of input texts) to get the answer. This is the only task that require the understanding of input file semantics and can not be solved in a systematic AI way.

4 Data Construction

Based on the construction rules in A.1, we procedurally generated input texts with randomly generated placeholder content. And during the procedure of generating input texts, we have collected informations regarding outline hierarchy, identifiers used in markup or syntax marks, statistical properties and mapping from identifier to contents. With these information, we can find the ground truth to our designed tasks systematically. For the syntax correcting tasks, we will incorporate intentional syntax error with a probability, and the input texts for such tasks are constructed individually.

Among all the file format we covered, python input texts are not procedurally generated, but rather collected from Internet, which are rich in semantics with real-world application. The ground truth tasks asking the algorithms or models the codes intend to implement is taken from the filenames of python files which is an indicator of the models or algorithms under the hood.

Randomness is the core feature we tried to maintain during the procedural generation, since we want to mangle the semantics of out input texts and focus solely on structural information. The contents and identifiers used in syntax marks are randomly generated from a finite set of character elements.

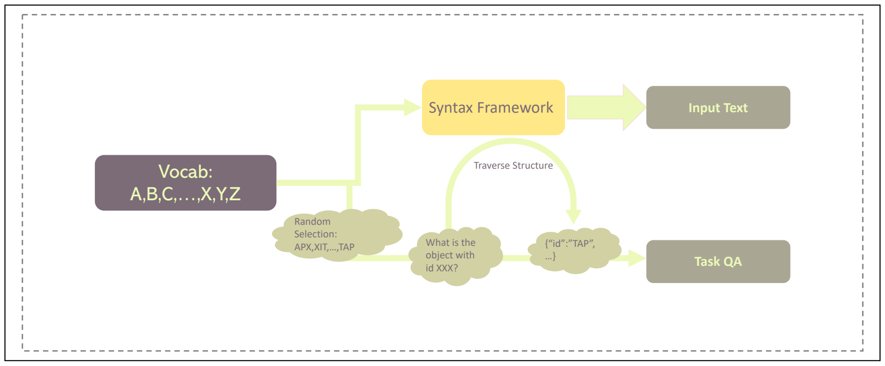


Figure 2: Process of input texts generation

5 Evaluation

With a dataset composed of 2512 QAs, we evaluated on 4 LLMs through network API, and adopted a heuristic elicitation method to offer more hints. To clarify, we queried each LLM with input text concatenated with question, and the response is evaluated against ground truth by exact match, rouge-1 score and T/F result from another judging LLM.

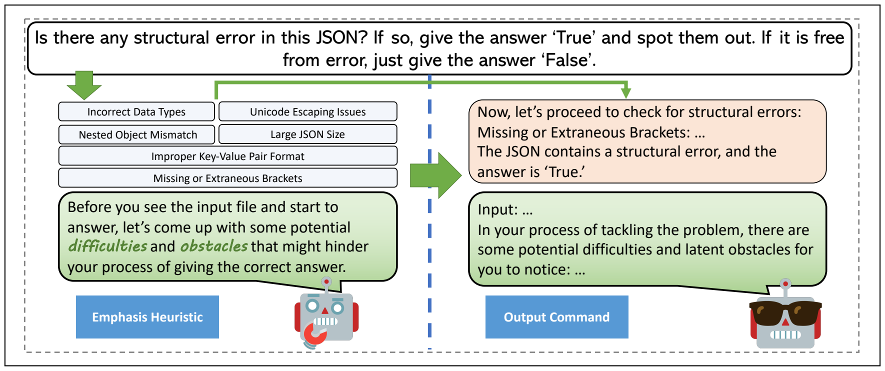
5.1 Baseline

First we used GPT-4 as baseline (due to inaccessibility of GPT-4, the experiment evaluated PYTHON and Org input on GPT3.5), i.e., only provide question with input text, without prompt

[1] `https://github.com/TheAlgorithms/Python`
[2] `https://pypi.org/project/AST/`
[3] `https://pypi.org/project/ast-scope/`

engineering and techniques alike. Evaluation of other LLMs are conducted under the same baseline setting, including Minimax [4], Spark from Xunfei [5] and Ernie from Baidu [6].

5.2 Prompt Engineering: Hint Elicitation



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'.

Before you see the input file and start to answer, let's come up with some potential *difficulties* and *obstacles* that might hinder your process of giving the correct answer.

Now, let's proceed to check for structural errors: Missing or Extraneous Brackets: ... The JSON contains a structural error, and the answer is 'True.'

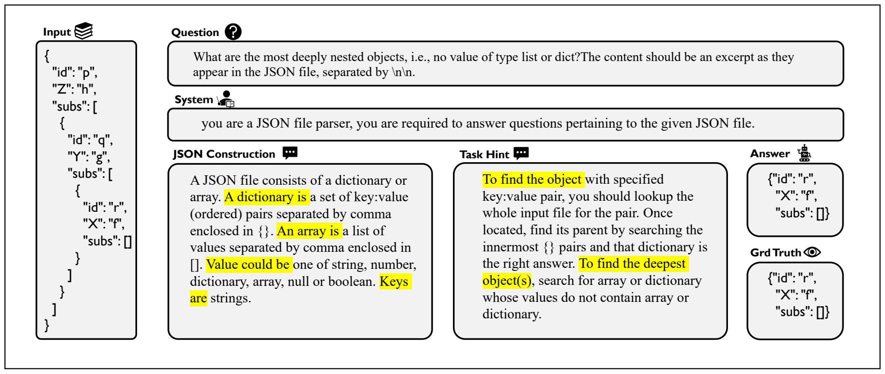
Input: ... In your process of tackling the problem, there are some potential difficulties and latent obstacles for you to notice: ...

Figure 3: Process of eliciting hint from LLMs to improve prompt

Apart from baseline experiment, we also explored the ability of LLMs to think of hints for task solution. The detailed framework is demonstrated in Figure 3. We use an eliciting query regarding the task's question to have LLMs generate some key points or obstacles before seeing the input texts. After the hints collection step, we will repeat baseline experiment with a difference in attached hints elicited before head, in hope that LLMs tackling these difficulties will help them think more verbosely to avoid errors.

5.3 Background Knowledge Enhancement

The structured text is pretty concise and expressive, most of its information is not readily explicit. There are a large volume of data that need to be inferred combining the raw input and structure information (or syntax rules). That leads to a intuitive conjecture that LLMs will likely to perform better on these structure dominant tasks provided that they have enough background knowledge about the syntax and construction rules for the rich-structure text in concern. To check the correlation of background knowledge on structure and the ability to answer the questions in our dataset, we conducted experiment with background knowledge included in the prompt when querying LLMs.



```
{
 "id": "p",
 "Z": "h",
 "subs": [
  {
   "id": "q",
   "Y": "g",
   "subs": [
    {
     "id": "r",
     "X": "f",
     "subs": []
    }
   ]
  }
 ]
}
```

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.

you are a JSON file parser, you are required to answer questions pertaining to the given JSON file.

A JSON file consists of a dictionary or array. A dictionary is a set of key:value (ordered) pairs separated by comma enclosed in {}. An array is a list of values separated by comma enclosed in []. Value could be one of string, number, dictionary, array, null or boolean. Keys are strings.

To find the object with specified key:value pair, you should lookup the whole input file for the pair. Once located, find its parent by searching the innermost {} pairs and that dictionary is the right answer. To find the deepest object(s), search for array or dictionary whose values do not contain array or dictionary.

{"id": "r", "X": "f", "subs": []}

{"id": "r", "X": "f", "subs": []}

Figure 4: Adding background knowledge for JSON tasks

[4]https://api.minimax.chat/v1/text/chatcompletion_pro
[5]ws://spark-api.xf-yun.com/v1.1/chat
[6]https://aip.baidubce.com/rpc/2.0/ai_custom/v1/wenxinworkshop/chat/eb-instant

6 Analysis

6.1 Baseline

Under the baseline experiment setting, GPT-4 outperforms all other LLMs with a significant margin and obtained accuracy over 0.96 in JSON, YAML, XML data, showing its proficiency in text retrieval, structure traversal, path construction, syntax correction, and depth calculation regarding object notation input texts. It answers only around half of the questions about Tree and Org. Tree input texts have the simplest syntax but require considerable calculation to rebuild the tree structure, which could make them difficult tasks for generative architecture. The Org type has similar syntax Markdown but with a relatively narrower niche in application and limited popularity. Such under-exploration means the input texts and its syntax might be entirely strange for LLMs, leading to the lack of necessary knowledge to do the inference towards solution. In terms of Tabular, Markdown, LaTeX and PYTHON, GPT-4 has gain similar scores around 0.7, which reflect the less familiarity over these inputs, given that the tasks are much easier as opposed to object notation input.

The performance of Minimax distribute over 8 types (other than Tree) similarly to GPT-4, except that the respective score is lower by 0.1-0.2. It barely answers correctly on tasks regarding Tree input.

As for Spark, its highest scores are earned in JSON and PYTHON tasks. YAML, XML and Markdown tasks shares scores around 0.5. Ratio of correct answers in Tabular, Latex and Org fell into around 30 percent. The score for Tree is 0.089 and lowest, which is the same as other LLMs.

The Ernie model gains its highest score around 0.6-0.7 in object notation types, with sub-optimum performance on Markdown and PYTHON, around accuracy of 50 percent. Tabular and LaTeX input texts have seen a score around 0.36. On Tree input, it earned its lowest score of 0.133, which is a marginal promotion over Minimax and Spark.

6.2 Hint Elicitation

For all 9 classes, only Tree, tabular and PYTHON has seen enhancement in three LLMs from hint eliciting prompt engineering. For object notation texts, hint has seen lower accuracy, with one exception on Ernie for JSON tasks. For markup languages, hint improved accuracy on Ernie but worsen the outcome on Minimax and Spark. In each cases, the baseline GPT4 possessed the highest scores. A detailed demonstration is listed in A.3

6.3 Background Knowledge

For background knowledge experiment, the result (see 8) from testing JSON input texts with background knowledge appended to query shows that such alternation is insufficient to enhance the performance of LLMs. The result could also be attributed to diverse factors.

- The limited understanding on these background knowledge
- The additional background knowledge has been included in the training set of LLMs, making the information superfluous
- The LLMs are currently unable to do complicate structul or logical inference, i.e., they failed to apply given rules on the raw input texts

7 Conclusion

We presented a taxonomy involving 9 popular structure-rich texts, with diversity in structural complexity and domain applied. We classified these inputs into 4 first level category by use scenarios and demonstrated their definition, construction rules we used in generation and common applications. By procedural construction and parsing, we built a benchmark with 2512 QAs and evaluated GPT-4, Minimax, Spark and Ernie. The experiment uncovered the under-exploration of structure-rich input texts for LLMs other than GPT-4, among which Tree is the most typical and where poor performance occurs. Prompt engineering of hint elicitation and additional background knowledge have both failed to address such inefficiency well.

Author Contributions

XXX

Acknowledgments

XXX

References

Peter Buneman. Semistructured data. In *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pp. 117–121, 1997.

Noam Chomsky and Marcel P Schützenberger. The algebraic theory of context-free languages. In *Studies in Logic and the Foundations of Mathematics*, volume 26, pp. 118–161. Elsevier, 1959.

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019.

Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. Convolutional neural networks over tree structures for programming language processing. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Veselin Raychev, Pavol Bielik, and Martin Vechev. Probabilistic model for code with decision trees. *ACM SIGPLAN Notices*, 51(10):731–747, 2016a.

Veselin Raychev, Pavol Bielik, Martin Vechev, and Andreas Krause. Learning programs from noisy data. *ACM Sigplan Notices*, 51(1):761–774, 2016b.

Marcel Paul Schützenberger. On context-free languages and push-down automata. *Information and control*, 6(3):246–264, 1963.

Jeffrey Svajlenko, Judith F Islam, Iman Keivanloo, Chanchal K Roy, and Mohammad Mamun Mia. Towards a big data curated benchmark of inter-project code clones. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pp. 476–480. IEEE, 2014.

Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. Can language models solve graph problems in natural language?, 2023.

Wenhan Wang, Ge Li, Bo Ma, Xin Xia, and Zhi Jin. Detecting code clones with graph neural network and flow-augmented abstract syntax tree. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 261–271. IEEE, 2020.

Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. *Advances in neural information processing systems*, 32, 2019.

A Appendix

A.1 Generation of Input Texts

A.1.1 Tree

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

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

A.1.2 Tabular

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

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

A.1.3 JSON

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

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

A.1.4 YAML

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

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

A.1.5 XML

$$\begin{aligned} firstline &\coloneqq \texttt{<?xml version="1.0" encoding="UTF-8"?>} \\ XML &\coloneqq \\ &\quad firstline \\ &\quad XMLObject \\ tag &\coloneqq \texttt{[A-Z]+} \\ val &\coloneqq \texttt{[a-z]+} \\ attr &\coloneqq \texttt{[A-Z]+="}val\texttt{"} \\ content &\coloneqq \texttt{[a-z }\backslash\texttt{n}\backslash\texttt{t]*} \\ XMLObject &\coloneqq \\ &\quad <tag(\ \ attr)*> \\ &\quad ((\backslash\texttt{t})*XMLObject)* \\ &\quad content \\ &\quad </tag> \end{aligned}$$

A.1.6 LaTeX

In LaTeX input texts, we include `textbf` and `includegraphics` commands to accommodate for the text retrieval tasks. The headings serve as anchors for structure traversal.

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

A.1.7 Markdown

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

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

A.1.8 Org

In Org input texts, the syntax is obtained from JSON construction rules by replacing the markups for heading, including figures and bold font face.

$$\begin{aligned}
\textit{heading} &\coloneqq \texttt{[*]* [a-z]+}\\
\textit{inclg} &\coloneqq \textit{lb}\{2\}\texttt{[a-z]+[.](png|jpg|jpeg|gif)}\textit{rb}\{2\}\\
\textit{bf} &\coloneqq \texttt{[*][a-z]+[*]}\\
\textit{content} &\coloneqq (\texttt{[a-z]}|\textit{bf}|\textit{inclg})+\\
\textit{Org} &\coloneqq \textit{heading}\backslash\text{n}\textit{content}(\backslash\text{n}\textit{Org})*
\end{aligned}$$

A.1.9 Python

Python input texts is the only type of texts that is not randomly generated but rather collected from Internet, so the code text should conform with the python programming language syntax as documented in their websites `https://docs.python.org`.

A.2 Experimental Data

Table 1: GPT-4 w/o prompt engineering

Metric	Tree	Tabular	JSON	YAML	XML	Markdown	Org	LaTeX	PYTHON
Exact Match	0.52	0.78	0.64	0.93	0.9	0.36	0.25	0.38	0.259
Rouge-1	0.107	0.107	0.383	0.368	0.195	0.894	0.496	0.944	0.068
GPTJudge	0.556	0.7	0.983	0.967	0.972	0.736	0.5	0.819	0.787

Table 2: Miminax w/o prompt engineering

Metric	Tree	Tabular	JSON	YAML	XML	Markdown	Org	LaTeX	PYTHON
Exact Match	0.133	0.738	0.470	0.430	0.567	0.283	0.233	0.267	0.201
Rouge-1	0.101	0.105	0.501	0.490	0.206	0.690	0.506	0.742	0.091
GPTJudge	0.067	0.525	0.850	0.770	0.883	0.683	0.450	0.683	0.679

Table 3: Spark w/o prompt engineering

Metric	Tree	Tabular	JSON	YAML	XML	Markdown	Org	LaTeX	PYTHON
Exact Match	0.383	0.975	0.310	0.540	0.433	0.333	0.250	0.267	0.263
Rouge-1	0.007	0.001	0.023	0.011	0.018	0.026	0.032	0.035	0.000
GPTJudge	0.089	0.280	0.800	0.472	0.533	0.529	0.300	0.296	0.702

Table 4: Ernie w/o prompt engineering

Metric	Tree	Tabular	JSON	YAML	XML	Markdown	Org	LaTeX	PYTHON
Exact Match	0.167	0.637	0.200	0.240	0.200	0.217	0.133	0.117	0.295
Rouge-1	0.056	0.051	0.161	0.162	0.084	0.226	0.137	0.283	0.066
GPTJudge	0.133	0.375	0.700	0.660	0.617	0.433	0.200	0.350	0.550

Table 5: Miminax w/ prompt engineering

Metric	Tree	Tabular	JSON	YAML	XML	Markdown	Org	LaTeX	PYTHON
Exact Match	0.150	0.738	0.390	0.400	0.517	0.250	0.183	0.233	0.205
Rouge-1	0.027	0.065	0.241	0.290	0.181	0.429	0.360	0.571	0.093
GPTJudge	0.183	0.575	0.850	0.730	0.850	0.600	0.433	0.683	0.708

Table 6: Spark w/ prompt engineering

Metric	Tree	Tabular	JSON	YAML	XML	Markdown	Org	LaTeX	PYTHON
Exact Match	0.483	0.963	0.420	0.540	0.833	0.417	0.400	0.300	0.268
Rouge-1	0.002	0.001	0.003	0.004	0.006	0.029	0.015	0.027	0.000
GPTJudge	0.214	0.457	0.533	0.286	0.483	0.100	0.250	0.222	0.705

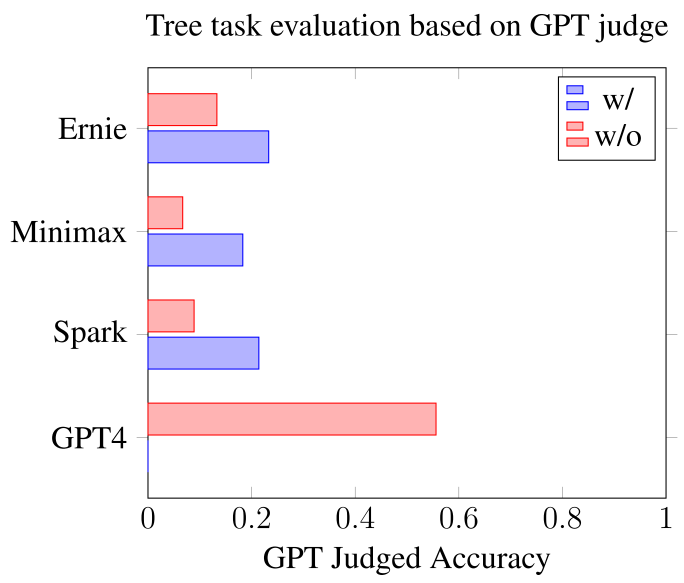
Table 7: Ernie w/ prompt engineering

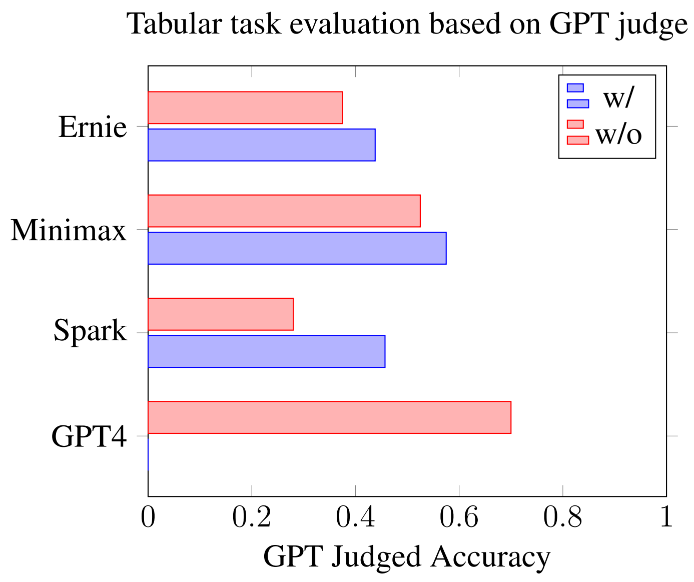
Metric	Tree	Tabular	JSON	YAML	XML	Markdown	Org	LaTeX	PYTHON
Exact Match	0.183	0.637	0.230	0.160	0.217	0.100	0.083	0.133	0.238
Rouge-1	0.042	0.041	0.167	0.115	0.101	0.143	0.104	0.115	0.044
GPTJudge	0.233	0.438	0.810	0.580	0.593	0.450	0.450	0.417	0.608

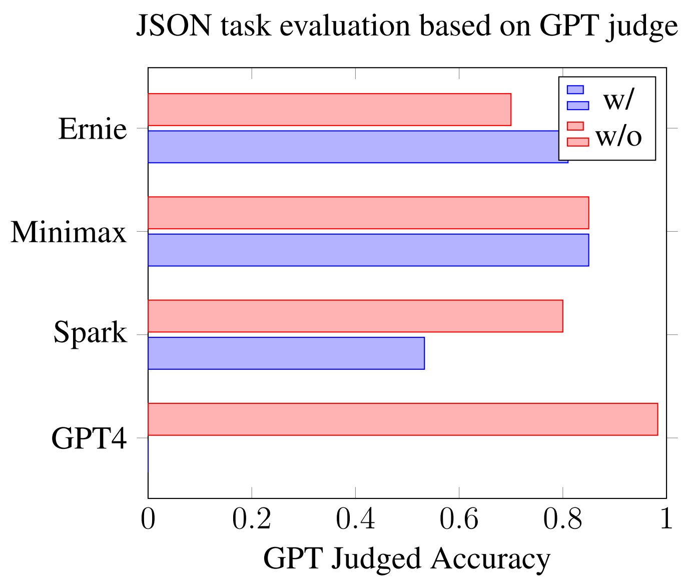
Table 8: JSON input with background knowledge

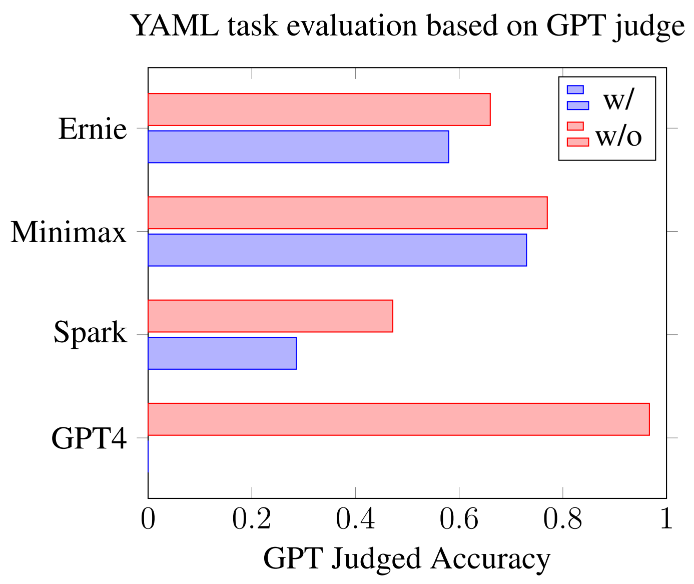
	Background Knowledge	Minimax:Base	Minimax:Hint	Ernie:Base	Ernie:Hint
GPTJudege	w/o	0.857	0.859	0.7	0.81
	w/	0.82	0.85	0.75	0.8
Exact Match	w/o	0.469	0.394	0.2	0.23
	w/	0.42	0.33	0.25	0.19
Rouge-1	w/o	0.504	0.243	0.161	0.167
	w/	0.524	0.223	0.146	0.173

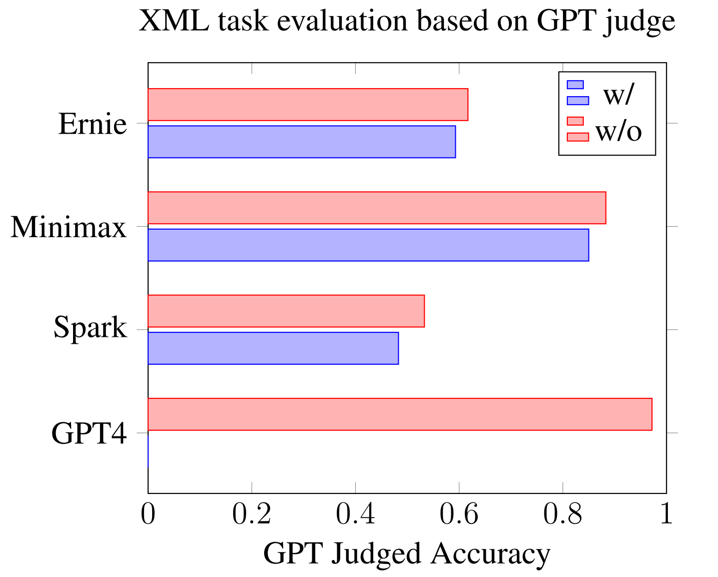
A.3 Comparation on GPT Judged Accuracy of LLMs

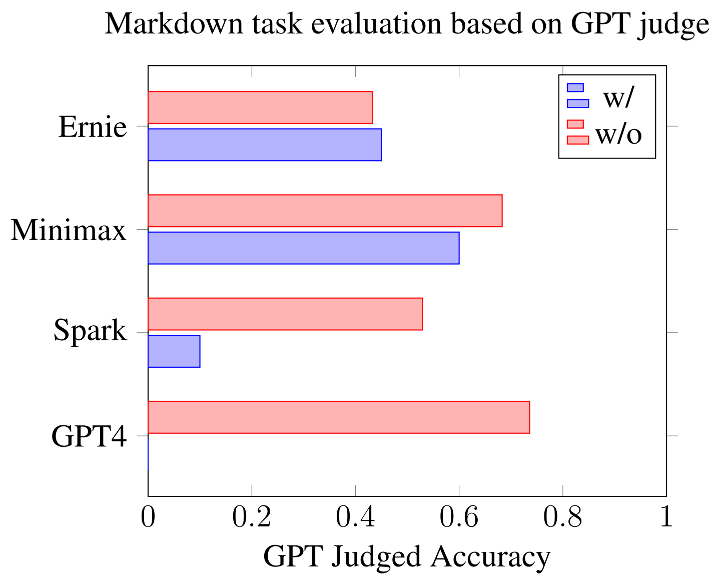


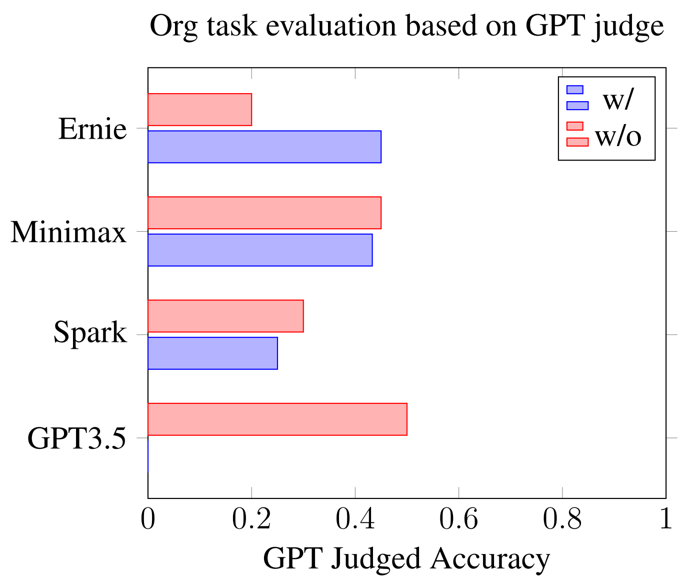


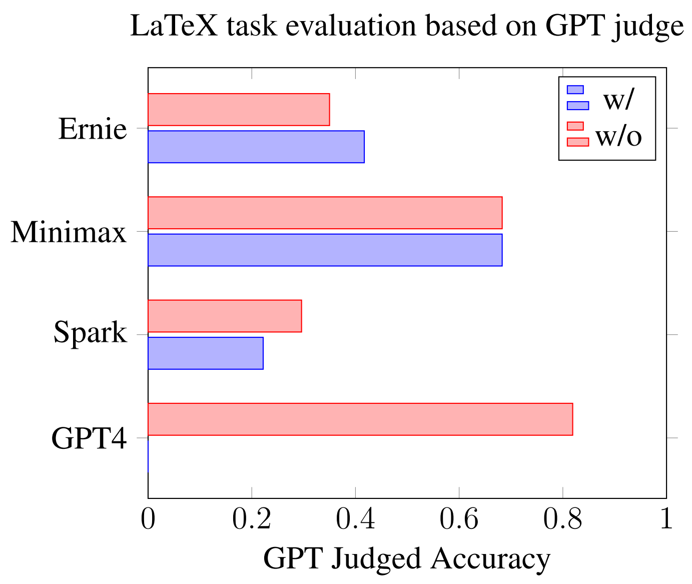

JSON task evaluation based on GPT judge
w/
w/o
Ernie
Minimax
Spark
GPT4
0
0.2
0.4
0.6
0.8
1
GPT Judged Accuracy

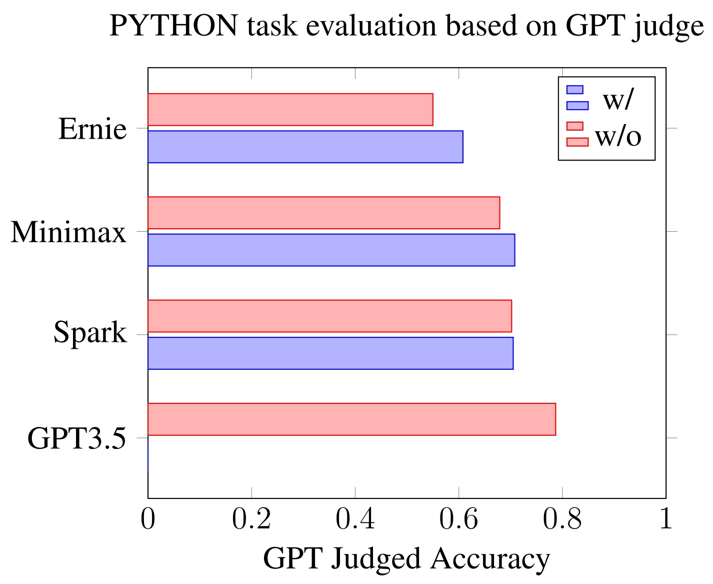

YAML task evaluation based on GPT judge
w/
w/o
Ernie
Minimax
Spark
GPT4
0
0.2
0.4
0.6
0.8
1
GPT Judged Accuracy


XML task evaluation based on GPT judge
Ernie
Minimax
Spark
GPT4
w/
w/o
0
0.2
0.4
0.6
0.8
1
GPT Judged Accuracy


Markdown task evaluation based on GPT judge
Ernie
Minimax
Spark
GPT4
w/
w/o
0
0.2
0.4
0.6
0.8
1
GPT Judged Accuracy


Org task evaluation based on GPT judge
Ernie
Minimax
Spark
GPT3.5
w/
w/o
0
0.2
0.4
0.6
0.8
1
GPT Judged Accuracy


LaTeX task evaluation based on GPT judge
Ernie
Minimax
Spark
GPT4
w/
w/o
0
0.2
0.4
0.6
0.8
1
GPT Judged Accuracy



A.4 SAMPLE OF INPUT AND TASKS

A.4.1 TREE

See Figure 5.

A.4.2 TABULAR

See Figure 6.

A.4.3 JSON

See Figure 7.

A.4.4 YAML

See Figure 8.

A.4.5 XML

See Figure 9.

A.4.6 LATEX

See Figure 10.

A.4.7 MARKDOWN

See Figure 11.

A.4.8 ORG

See Figure 12.

A.4.9 PYTHON CODE

See Figure 13.

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

o->p->v->z

Task 2

Question

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

Ground Truth

3

Task 3

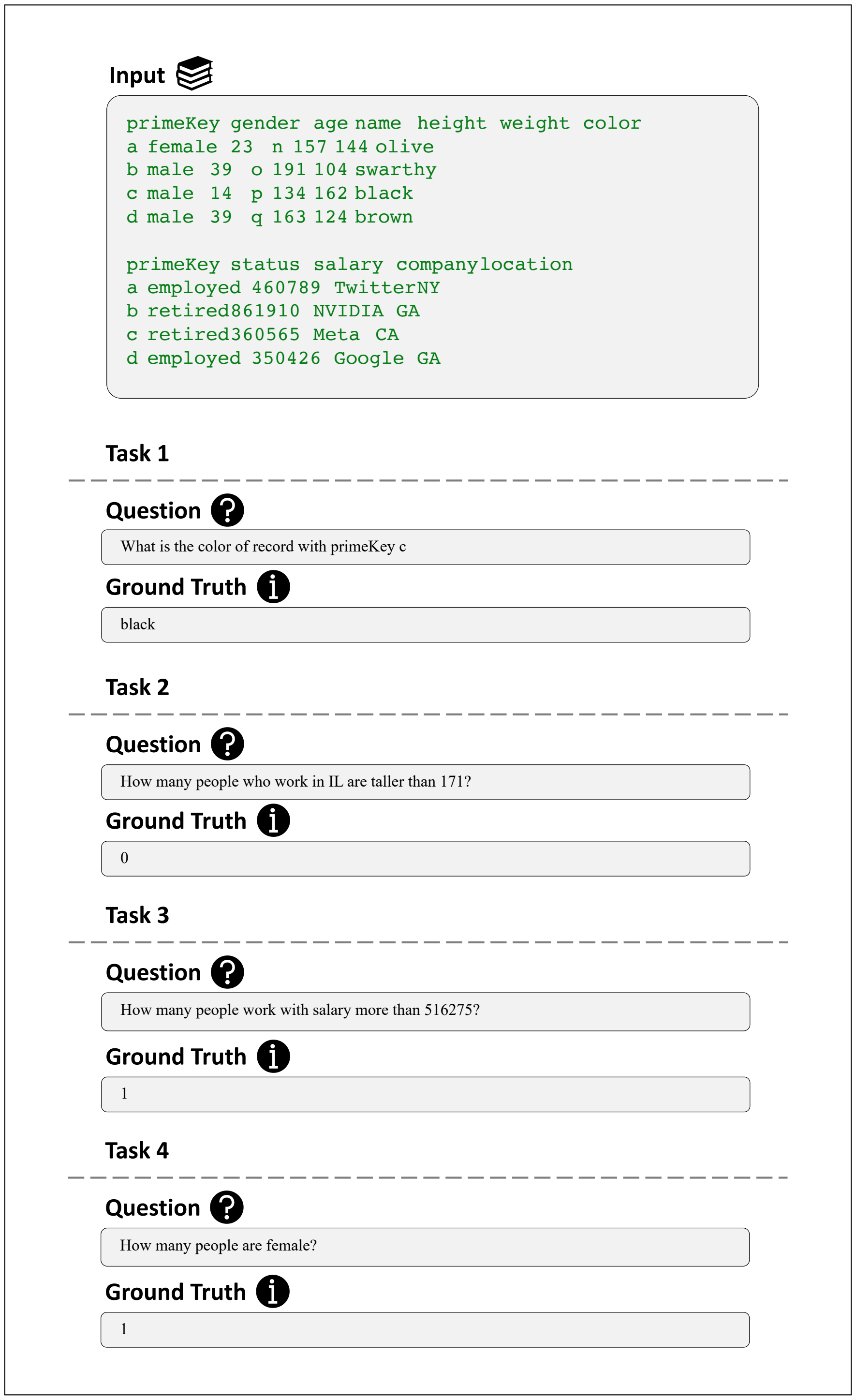
Question

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

3

Figure 5: Sample input and tasks of Tree.



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

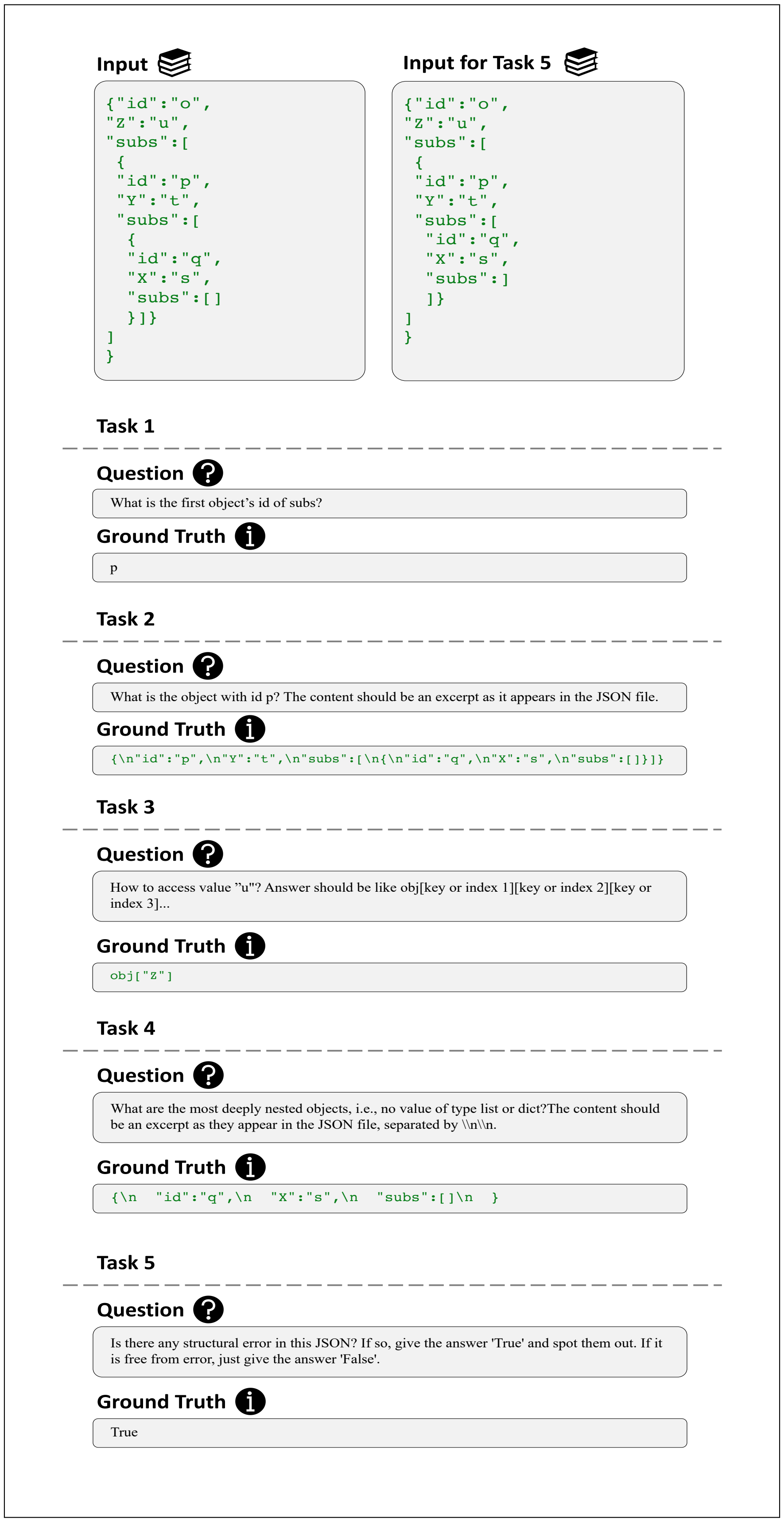
Question

How many people are female?

Ground Truth

1

Figure 6: Sample input and tasks of tabular data.



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

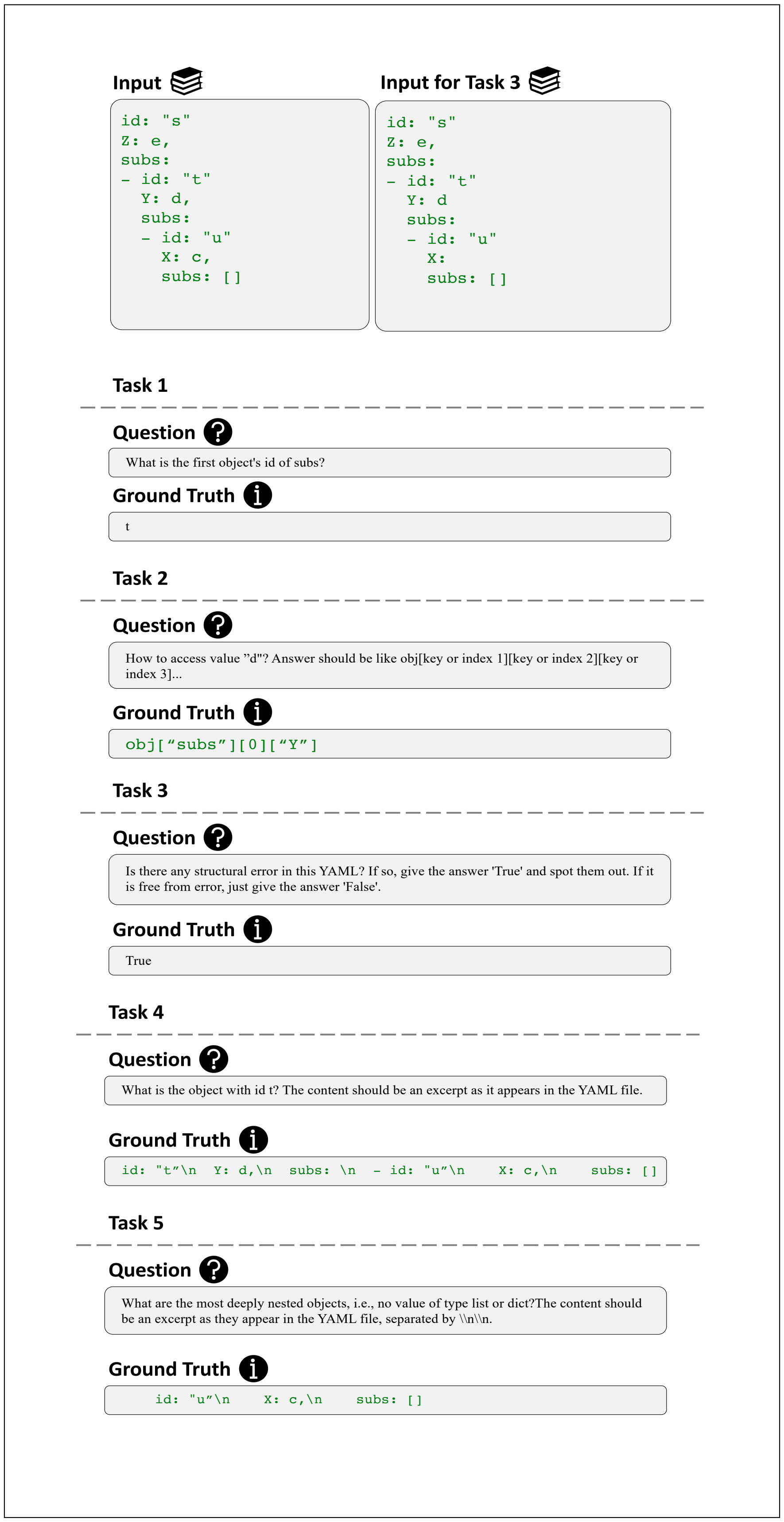
Question

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

Figure 7: Sample input and tasks of JSON.



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

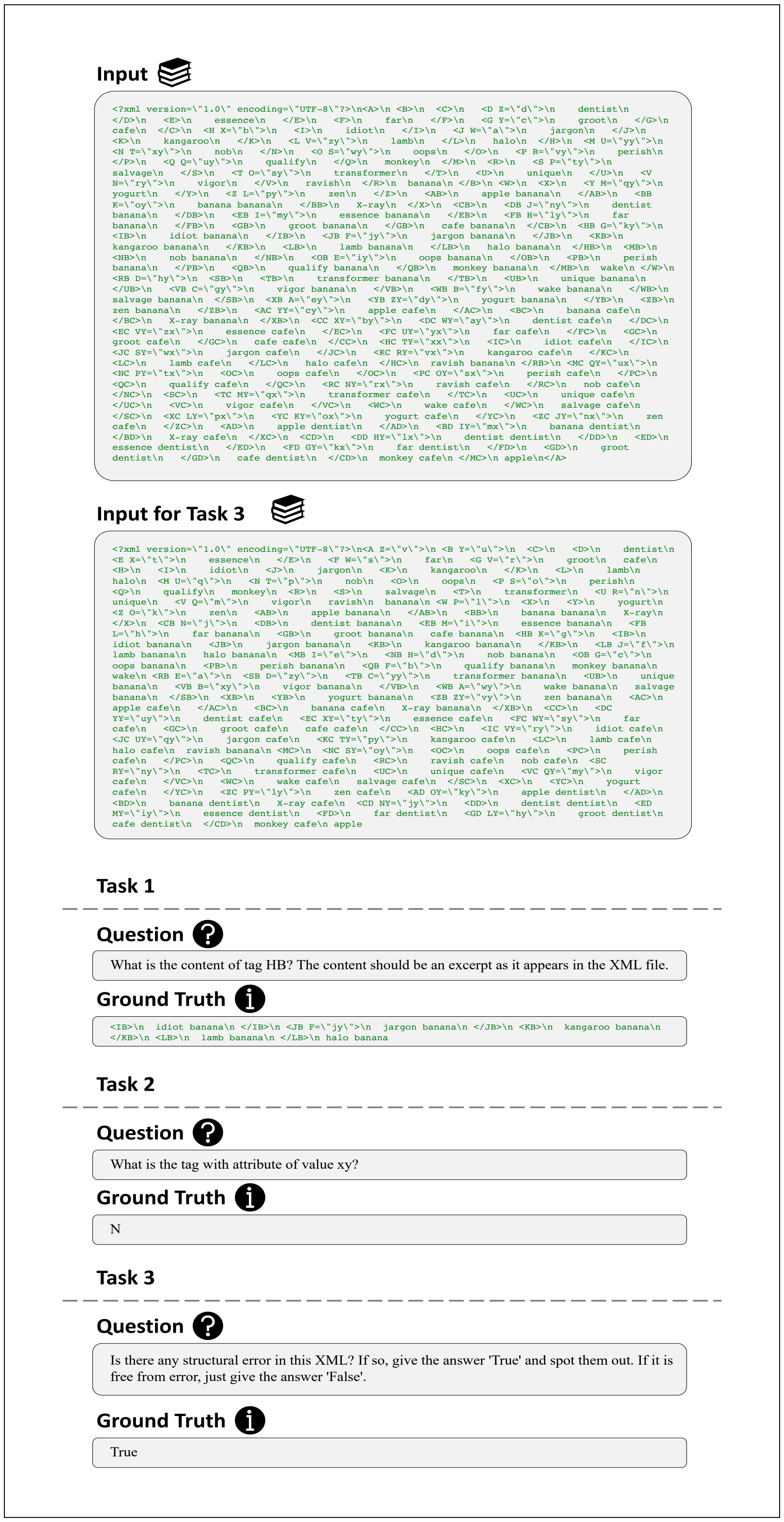
Question

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 8: Sample input and tasks of YAML.



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

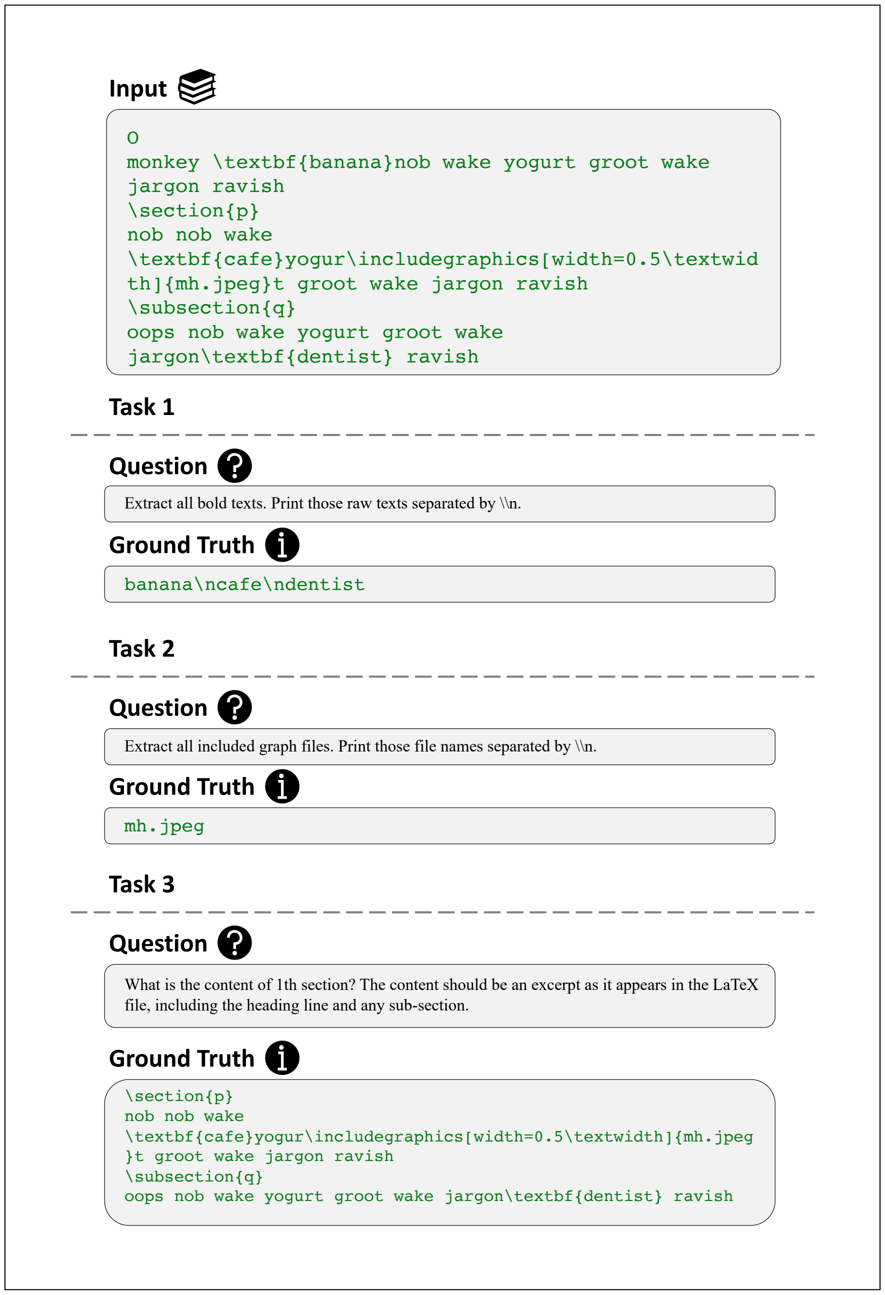
Question

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

Figure 9: Sample input and tasks of XML.



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

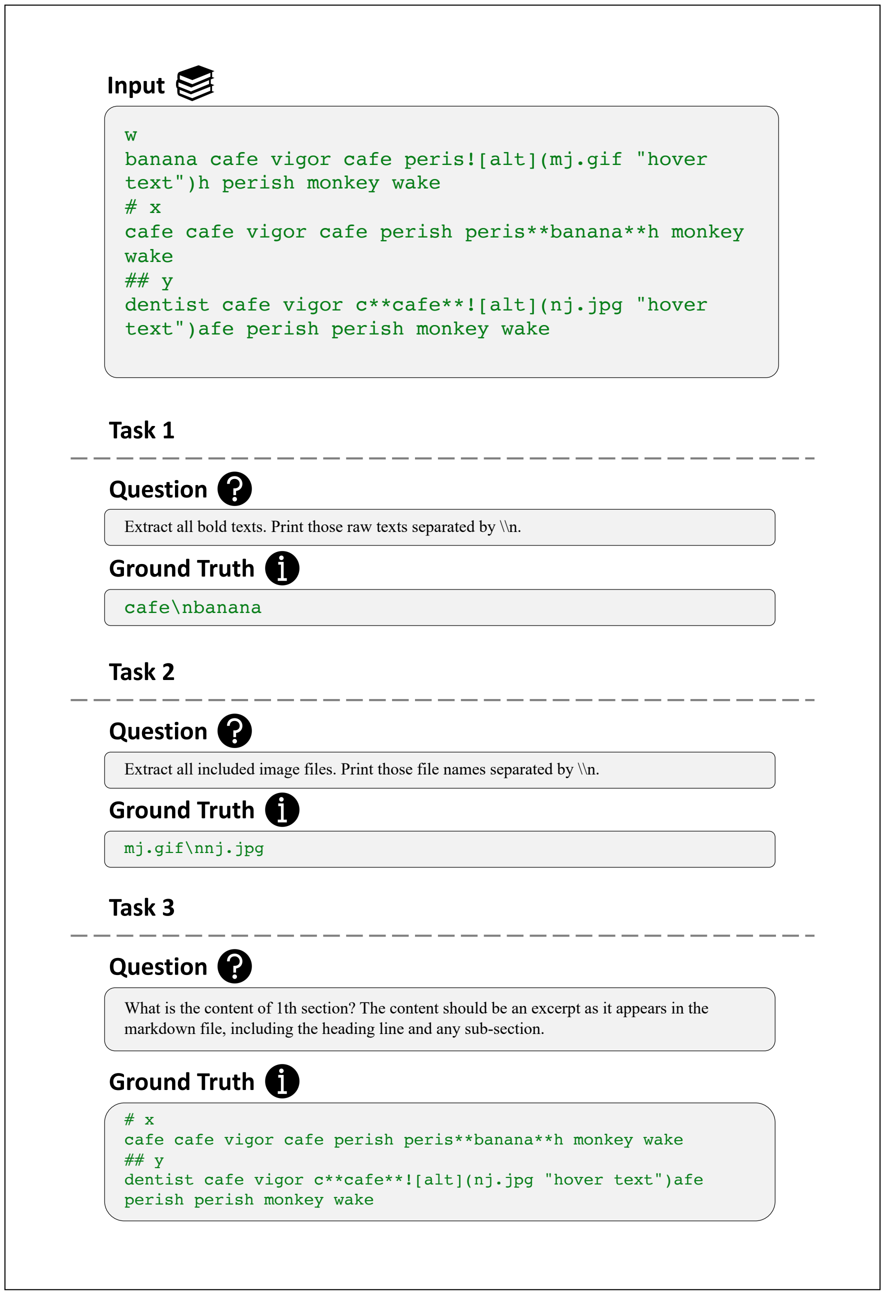
Question

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
```

Figure 10: Sample input and tasks of LaTeX.



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

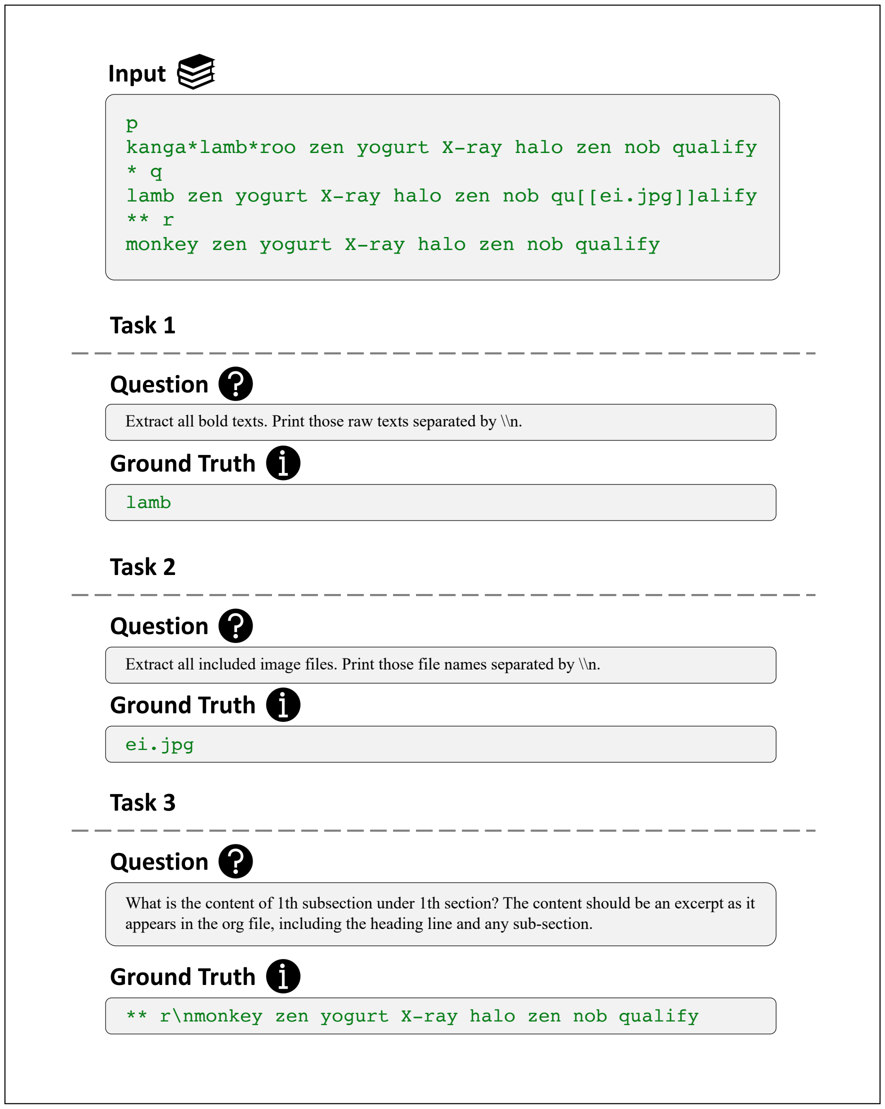
Question

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 11: Sample input and tasks of Markdown.



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

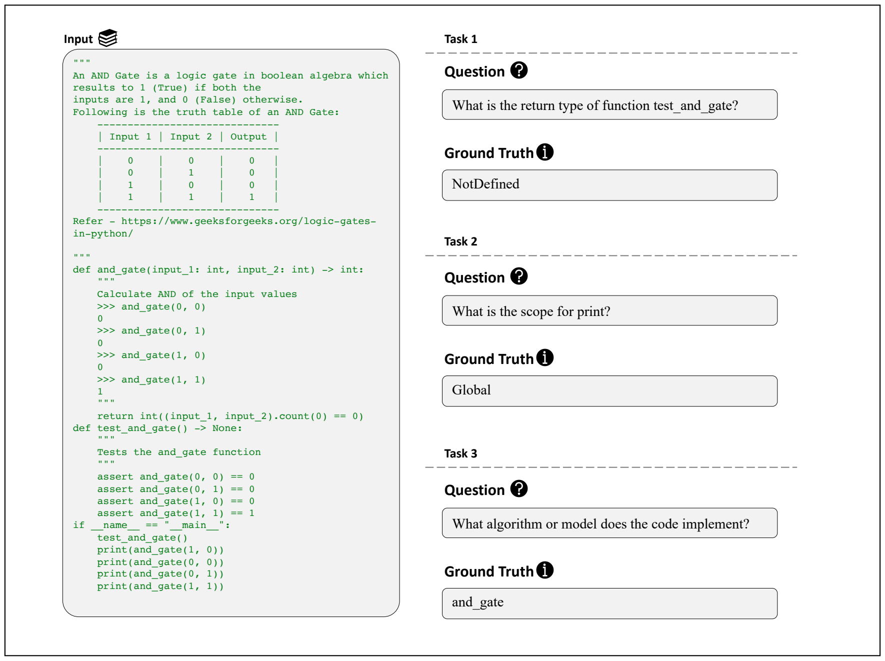
Question

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
```

Figure 12: Sample input and tasks of Org.



```
"""
An AND Gate is a logic gate in boolean algebra which
results to 1 (True) if both the
inputs are 1, and 0 (False) otherwise.
Following is the truth table of an AND Gate:
    ------------------------------
    | Input 1 | Input 2 | Output |
    ------------------------------
    |    0    |    0    |    0   |
    |    0    |    1    |    0   |
    |    1    |    0    |    0   |
    |    1    |    1    |    1   |
    ------------------------------
Refer - https://www.geeksforgeeks.org/logic-gates-
in-python/

"""
def and_gate(input_1: int, input_2: int) -> int:
    """
    Calculate AND of the input values
    >>> and_gate(0, 0)
    0
    >>> and_gate(0, 1)
    0
    >>> and_gate(1, 0)
    0
    >>> and_gate(1, 1)
    1
    """
    return int((input_1, input_2).count(0) == 0)
def test_and_gate() -> None:
    """
    Tests the and_gate function
    """
    assert and_gate(0, 0) == 0
    assert and_gate(0, 1) == 0
    assert and_gate(1, 0) == 0
    assert and_gate(1, 1) == 1
if __name__ == "__main__":
    test_and_gate()
    print(and_gate(1, 0))
    print(and_gate(0, 0))
    print(and_gate(0, 1))
    print(and_gate(1, 1))
```

Task 1

Question

What is the return type of function test_and_gate?

Ground Truth

NotDefined

Task 2

Question

What is the scope for print?

Ground Truth

Global

Task 3

Question

What algorithm or model does the code implement?

Ground Truth

and_gate

Figure 13: Sample input and tasks of Python codes.