HySem: A context length optimized LLM pipeline for unstructured tabular extraction

Narayanan PP
Indian Institute of Science, Bangalore
narayananp@iisc.ac.in

Anantharaman Palacode Narayana Iyer
JNResearch Labs LLP
ananth@jnresearch.com

Abstract

Regulatory compliance reporting in the pharmaceutical industry relies on detailed tables, but these are often under-utilized beyond compliance due to their unstructured format and arbitrary content. Extracting and semantically representing tabular data is challenging due to diverse table presentations. Large Language Models (LLMs) demonstrate substantial potential for semantic representation, yet they encounter challenges related to accuracy and context size limitations, which are crucial considerations for the industry applications. We introduce HySem, a pipeline that employs a novel context length optimization technique to generate accurate semantic JSON representations from HTML tables. This approach utilizes a custom fine-tuned model specifically designed for cost- and privacy-sensitive small and medium pharmaceutical enterprises. Running on commodity hardware and leveraging open-source models, HySem surpasses its peer open-source models in accuracy and provides competitive performance when benchmarked against OpenAI GPT-4o and effectively addresses context length limitations, which is a crucial factor for supporting larger tables.

1 Introduction

Data tables are essential for facilitating regulatory compliance and financial reporting across industries such as pharmaceuticals and finance. In the pharmaceutical sector, documents like Annual Product Quality Reviews (APQRs) often contain complex, ad-hoc structured tables that present significant challenges for integration into standard databases. While these tables hold valuable information, their unstructured format renders them non-queryable, hindering automated processing. Our objective is to convert these table-based assets, often stored in HTML format, into semantic JSON, which enables direct mapping of the generated JSON keys to the field names of the database schema, facilitating efficient integration and use in business analytics applications.

Processing real-world tables, particularly those prevalent in industries like pharmaceuticals, is a highly challenging task. Unlike structured databases with predictable schemas, pharmaceutical tables often have arbitrary placements of headers and data elements. Headers might appear mid-table, or there may be multiple, inconsistent levels of headers across different documents. **Appendix A.2** Figure 3 illustrates a complex pharmaceutical data table.

The arbitrariness in table presentation renders rule-based approaches ineffective for transforming HTML tables into semantic JSON. Furthermore, developing and maintaining such algorithms is costly and does not scale efficiently to handle frequent changes in table formats. In contrast, large language models (LLMs) can be trained to recognize patterns and relationships within the data, offering a more robust, scalable and adaptable solution. However, LLMs often encounter performance challenges when handling complex tabular structures, particularly very long tables that contain multiple instances of pharma-specific terminology.

Table Representation Learning Workshop at NeurIPS 2024

In this paper, we propose HySem, a pipeline designed to accurately transform HTML tables into semantic JSON representations (refer **Appendix A.3** for detailed illustrations of semantic JSON). Central to our approach is a novel Context Optimiser, which employs a dynamic token pruning technique to rewrite the input HTML tables. This process reduces the token count while maintaining the semantic integrity of the data. As a result, HySem can efficiently manage large and complex tables, including those featuring specialized pharmaceutical terminology, without sacrificing performance.

Our research was motivated by the business needs of small and medium pharmaceutical enterprises, where cost and data privacy considerations are crucial. Specifically, our key goals are:

- **On-Premise Model Deployment:** Ensuring all models operate on-premise to maintain stringent data security standards.
- **Compatibility with Commodity Hardware:** Enabling models to run efficiently on commodity hardware with cost-effective, readily available GPUs.
- **Model Size Constraints:** Limiting model size to under 10 billion parameters, making it suitable for GPUs with up to 16 GB of memory.
- **Open-Source Software Stack:** Leveraging open-source tools and models to minimize costs and foster accessibility.

Building on these foundational goals, we present the following contributions:

- **Context Optimizer:** A novel component that efficiently rewrites input data, significantly reducing token counts and enhancing processing speed.
- **Semantic Synthesizer:** A custom fine-tuned model designed to produce precise semantic representations from complex HTML tables.
- **Syntax Corrector:** An agentic system that identifies and corrects syntax errors in the output JSON with minimal human oversight.
- **Evaluation Methodology:** A new framework that evaluates the performance of our approach using a combination of intrinsic and extrinsic metrics.

2 Methodology

Our proposed LLM Pipeline, HySem, aims to address the challenge of converting unprocessed raw HTML tables into a semantic JSON data structure while addressing the limitations of LLMs concerning context length and inference time. By adopting a novel strategy for optimizing the number of tokens, we aim to meet the industry standards for high accuracy and reduced inference time, particularly for regulatory compliance reporting.

HySem achieves these requirements through a structured pipeline composed of three components: Context Optimizer ($\mathcal{A}_{\text{CO}}$), Semantic Synthesizer ($\mathcal{A}_{\text{SS}}$), Syntax Corrector ($\mathcal{A}_{\text{SC}}$). $\mathcal{A}_{\text{CO}}$ employs a novel methodology for optimizing the context window utilized by HTML tables, enabling processing of large tables. $\mathcal{A}_{\text{SS}}$ transforms this optimized HTML table into semantic JSON. $\mathcal{A}_{\text{SC}}$ reviews the generated JSON for any syntax errors and outputs a syntactically-valid JSON. Figure 1 illustrates the Hysem architecture in detail.

2.1 Context Optimizer Subsystem

The limited context size of large language models significantly constrains their ability to effectively process large tables [17, 36]. Although models with extended context lengths can alleviate this issue, they may sacrifice accuracy and increase processing times due to the quadratic complexity of self-attention mechanisms [28, 30, 33]. Recent advancements in Transformer architectures, such as Linformer [34] and Flash Attention techniques [7], have shown improvements in memory and time complexity. Nevertheless, the critical need to reduce token count persists, as it is essential for enabling faster inference times and optimizing the utilization of limited GPU memory, all without sacrificing model performance.

Our approach to optimizing context length utilization stems from the observation that significant token inefficiency arises when there is a mismatch between the tokenizer's vocabulary and domain-specific

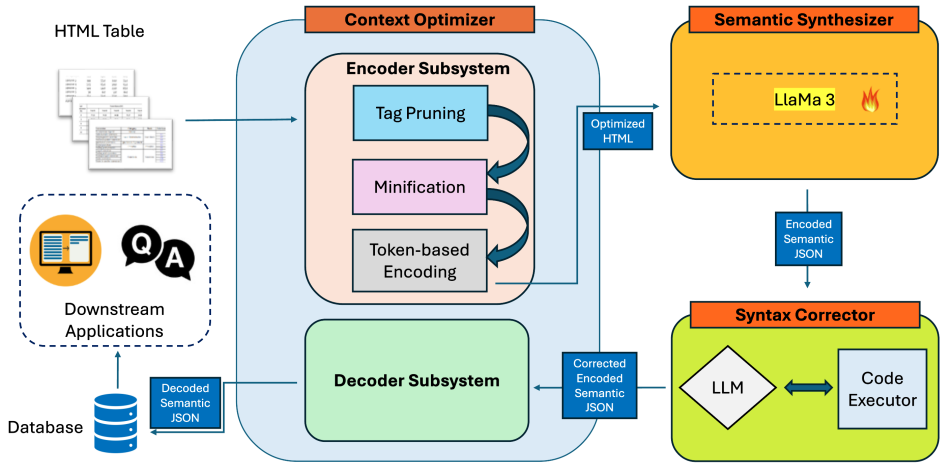

Figure 1: Hysem Architecture diagram

terminology in the input text. For example, "Amoxycillin," a widely recognized pharmaceutical medication, is not present in the Llama 3 tokenizer's vocabulary, leading to its representation by multiple tokens. In addition to domain-specific terminology, other data types commonly encountered in real-world tables, such as dates, string identifiers, and proper nouns, often fail to align with the tokenizer's vocabulary, further increasing token count and exacerbating inefficiency.

We introduce **Context Optimizer**, a novel token alignment technique designed to minimize the impact of "token-vocabulary" misalignment. This approach optimizes token representation by aligning cell contents in a table with the tokenizer's vocabulary, thereby reducing the number of tokens required to represent each cell.

The Context Optimizer operates in 2 phases: *Encoding phase* where we rewrite the cell contents into a more compact form and *Decoding phase* which restores the original contents.

2.1.1 Encoding Phase

In the encoding phase, our process is divided into two primary stages: standard pre-processing and a specialized token-based encoding method.

Pre-processing Steps

We begin with a set of conventional pre-processing steps. First, we remove tags and attributes from the HTML tables that do not contribute to semantic understanding, such as those meant for styling. After this, a minification step is applied, which strips away unnecessary white spaces, further optimizing the HTML table for encoding.

Token-Based Encoding

Next, we apply our custom token-based encoding technique. The goal here is to represent the content of each cell with the minimum number of tokens while ensuring that each cell has a unique representation. The algorithm for this process is detailed in Algorithm 1.

Prior to encoding, we first sort the cells in ascending order based on the number of tokens they contain. This strategy allows us to resolve *potential collisions* more easily, as cells with fewer tokens are processed first. A collision occurs when two distinct cell contents map to overlapping token sequences.

Our encoding process incorporates several high-level heuristics to enhance efficiency and accuracy:

a. Single Token Preservation: If a cell's content consists of a single token, it remains unchanged.

Theme	Subtheme including paper where data were retrieved
Theme 1: Women's knowledge and understanding of preeclampsia	Knowledge (1, 3–8)
	Absence or not recognising signs and symptoms (2, 5, 8–10)
	Range of information needs (1–3, 5, 8, 9)
Theme 2: Factors affecting help seeking behaviour from perspectives of women and their families'	Emotions affecting help-seeking (2, 4, 5, 7, 10)
	Social, cultural and economic disparities (6, 9)
	Social networks influencing help-seeking (5, 6, 10)
Theme 3: Factors affecting staff response	Practitioner-client communications and relationship(1, 4, 5)
	Not being taken seriously (4, 8, 10)

(a) HTML Table before optimization

Theme	Subtheme
Theme 1	Knowledge (1, 3–8)
	Absence
	Range of
Theme 2	Emotions
	Social, cultural
	Social networks
Theme 3	Pract
	Not being

(b) HTML Table after optimization

Figure 2: An illustration of a table optimized by the Context Optimizer is shown in 2a, presenting the original HTML table without any optimization. 2b displays the same table after optimization. In these figures, tokens within each cell are highlighted with distinct colors to facilitate easy observation of the token count per cell. The tokenization is performed using the LLaMa 3 tokenizer.

b. Multi-Token Optimization: For cells with multiple tokens, we aim to represent the content using only two tokens whenever possible. This approach addresses the initial attempt of encoding with a single token, which often compromised semantic richness and resulted in inaccurate JSON predictions.

c. Bracket Handling: We handle incomplete bracket sequences by checking if a token starts with an opening bracket (e.g., [, {) and lacks a matching closing bracket. In such cases, we concatenate subsequent tokens until the bracket is closed. This approach is crucial to prevent syntax errors in the generated JSON.

Figure 2 offers a detailed illustration of the HTML table before and after optimization. For example, in Figure 2a the cell content *"Theme 1: Women's knowledge and understanding of preeclampsia"* is tokenized into 15 distinct tokens, represented by various colors. After optimization, the same cell content transforms into *"Theme 1"*, consuming only 3 tokens as illustrated in Figure 2b.

The overall objective is to use the fewest tokens possible while maintaining uniqueness across all cell contents. By treating each tokenized cell as a unit, we can reduce the total number of tokens significantly without losing semantic integrity. A *mapping table* is maintained to track the original cell content and its optimized counterpart, facilitating later restoration. The encoded HTML table is processed by the downstream modules in the HySem pipeline, including the Semantic Synthesizer and the Syntax Corrector.

2.1.2 Decoding Phase

In the decoding phase, the output generated by the Semantic Synthesizer is decoded to restore the original lexicon used in the table. Given that the input HTML table is encoded in the Encoding Phase, the output JSON produced by the Semantic Synthesizer retains these encoded abbreviations within each of its nodes. Each node of the JSON is restored to its original lexicon by referencing the mapping table. The resulting JSON is both semantically accurate and more efficiently processed.

A key feature of our Context Optimizer is its **dynamic nature**, where the mapping of input words to optimized token sequences, as well as the corresponding decoding process, are entirely driven by the current input. This process operates without reliance on any pre-defined static mappings.

Algorithm 1 Token-Based Encoding Algorithm

Input: Table T with n cells $\{C_1, C_2, \ldots, C_n\}$, each containing text.
Output: Encoded table $T' = \{E_1, E_2, \ldots, E_n\}$ with unique token representations.

```
Sort cells {C_1, C_2, ..., C_n} in ascending order based on the number of tokens in each cell.
for each cell C_i ∈ T (after sorting) do
    Tokenize C_i: T_i = {t_1, t_2, ..., t_k} where t_j are tokens of C_i.
    Apply heuristics to determine the initial encoded form E_i.        ▷ Use predefined strategies

    if ∃j < i such that E_i = E_j then        ▷ Check if a previous encoding E_j is the same as E_i
        for t_k ∈ {t_2, t_3, ..., t_k} do
            E_i = E_i ∪ t_k.                                          ▷ Concatenate additional tokens
            if E_i ≠ E_j then
                break.                                                ▷ Stop once the encoding is unique
            end if
        end for
    end if
    Store the mapping (C_i, E_i).
end for
return encoded table T' = {E_1, E_2, ..., E_n}.
```

2.2 Semantic Synthesizer

Given the LLM's capability to comprehend deep semantic relationships, HySem adopts the open-sourced Meta-Llama-3-8B-Base model and fine-tunes it with a manually labeled dataset for transforming HTML tables into semantic JSON.

Concretely, the Semantic Synthesizer $\mathcal{A}_{\text{SS}}$ accepts a HTML table $\mathcal{H}_{\text{i}}$ optimized by the Context Optimizer, as input and produces JSON $\mathcal{J}_{\text{i}}$ as the output in the same encoded semantic space as the optimized input HTML table. Our initial experiments indicate that adopting Prompt Engineering to achieve this transformation results in the generation of less accurate JSON representations, as evidenced by our Intrinsic and Extrinsic evaluations. We found that the LLM is highly sensitive to prompts and struggles to effectively capture the wide variety present in these tables. We identified several common failure patterns present in the JSONs generated by the LLM, which are cataloged as *Semantic Failures* in Table 1.

To address these failure modes, we fine-tune the LLM to generate JSON representations that accurately reflect the input HTML table.

Dataset

For our dataset, we require inputs as HTML tables and labels as semantic JSON. We utilize the following two open-sourced datasets for tabular HTML sources:

PubTabNet: PubTabNet [27] is a large-scale dataset for image-based table recognition, containing over 568,000 images of tables from scientific papers along with their corresponding HTML annotations.

FinTabNet: FinTabNet [44] is a dataset specifically designed for recognizing tables in financial documents, containing over 112,000 tables. Each table instance annotation includes fields such as the HTML structure and bounding box coordinates, similar to PubTabNet.

For our purposes, we extract the HTML from these sources and hand-label the corresponding JSON annotations for these tables. We filter tables that fit within the LLM's context length (8k context window for Llama3) and select 1,364 HTML tables from both PubTabNet and FinTabNet. We split the data into 756 training samples and 608 testing samples. We chose these datasets as they are public and are reasonable representations of the Pharmaceutical and Finance verticals respectively.

We also created additional hand-labeled datasets to address the custom needs of the industry using the proprietary customer supplied data. These proprietary datasets are used to fine-tune models that use Semantic Synthesizer LLM as the base model.

Failure Mode	Example
Lexical errors	"Characteristics" in place of "Characteristic"
Missing words	"123" instead of "123 (n=25)"
Missing entire rows	An entire row under a subheading missed
Misaligned keys/values	A cell value placed under a different parent key
Merged Cells	"95% CI": ["- 211.5", "69.0"] ⇒ "95% CI": "- 211.5\n69.0"
Unicode errors	"0.88 (0.53 \u2013 1.33)" ⇒ "0.88 (0.53 - 1.33)"

Table 1: Semantic Failures

Failure Mode	Description
Missing List Enclosure	The output is expected to be a list of dictionaries, but the LLM generated the dictionaries without enclosing them in a list.
Unmatched Curly Braces	The JSON output is missing one or more curly braces, resulting in an incomplete or unbalanced JSON structure.
Missing Commas	The JSON output is missing one or more commas, leading to improperly formatted JSON.
Incorrect Placement of Quotes	For long numeric strings that include commas, such as 123,456,789, the expected output should enclose the entire string in quotes. Instead, the output incorrectly inserts quotes within the numeric string, producing a format like 123,"456,789".

Table 2: Structural Failures

2.3 Syntax Corrector

Syntax errors in the LLM-generated JSON output render the table unusable for further processing, such as ingestion into databases. Consequently, correcting these syntax errors is a critical functionality, especially for enabling automated workflows in industrial settings. To address these challenges, we developed a Syntax Corrector, based on *reflective agentic framework*.

Specifically, the Syntax Corrector $\mathcal{A}_{\text{sc}}$ accepts a syntactically invalid JSON $\mathcal{J}_{\text{i}}$ as input and produces a syntactically valid JSON $\mathcal{J}_{\text{v}}$ through *iterative refinement*. Through self-reflection [13, 23, 20, 3, 26], $\mathcal{A}_{\text{sc}}$ iteratively refines the JSON output until a syntactically valid result is achieved or the maximum number of iterations is reached. The algorithm for this process is detailed in Algorithm 2 of **Appendix A.5**. Table 2 displays the common syntax error patterns observed in the LLM output.

3 Evaluation Methodology

For the accurate transformation of HTML tables into semantic JSON, two key objectives must be satisfied. First, ***content preservation:*** all strings from the HTML table cells must exactly occur in the JSON output, ensuring no information is lost during the conversion. Second, ***semantic accuracy:*** the generated JSON must accurately represent the hierarchical and relational structure of the original HTML table.

Building on this foundation, we develop intrinsic and extrinsic evaluation methods to measure the content and semantic accuracy of the JSON produced by HySem. **Appendix A.4** provides detailed illustrations for intrinsic and extrinsic evaluation methods.

3.1 Intrinsic Evaluation

In intrinsic evaluation, we assess the representation of content from HTML cells in the generated JSON. We parse the HTML using Beautiful Soup to extract the *set of all the cell contents* present in the HTML input, denoted as $\mathcal{H}_{\text{set}} = \{c_1, c_2, \ldots, c_m\}$. Similarly, we take the *set of all elements* in the JSON, denoted as $\mathcal{J}_{\text{set}} = \{e_1, e_2, \ldots, e_n\}$

To evaluate if each content item c_i from $\mathcal{H}_{\text{set}}$ is present in $\mathcal{J}_{\text{set}}$, we define an indicator function $I(c_i \in \mathcal{J}_{\text{set}})$:

$$I(c_i \in \mathcal{J}_{\text{set}}) = \begin{cases} 1 & \text{if } c_i \in \mathcal{J}_{\text{set}}, \\ 0 & \text{otherwise.} \end{cases}$$

The Intrinsic Score (ISC) is then computed as:

$$\text{ISC} = \frac{\sum_{i=1}^{m} I(c_i \in \mathcal{J}_{\text{set}})}{m}.$$

Here, m denotes the cardinality of $\mathcal{H}_{\text{set}}$.

3.2 Extrinsic Evaluation

Extrinsic evaluation assesses the semantic structure of the JSON by evaluating its ability to answer targeted questions. This method avoids direct comparison to the ground truth JSON ($\mathcal{G}_{\text{t}}$), which can vary in representation. Instead, we systematically validate the structure by formulating questions that probe specific semantic elements: In order to form a minimalistic set of questions that validate the structure, we leverage the paths from the root node to each leaf node in the JSON. The number of paths corresponds to the number of leaf nodes. Let $\mathcal{P}$ denote the set of these paths. Formally, we define $\mathcal{P}$ as:

$$\mathcal{P} = \{p_i \mid p_i = (n_1, n_2, \ldots, n_{k-1})\}$$

Here, $(n_1, n_2, \ldots, n_k)$ represents a sequence of nodes starting from the root node n_1 till a leaf node n_k in the JSON structure. For each path $p_i \in \mathcal{P}$, we prompt an LLM ($\mathcal{M}_{\text{q}}$) with $\mathcal{G}_{\text{t}}$ and p_i as inputs, instructing it to generate a single question $\mathcal{Q}_{\text{i}}$. This question is targeted so that the value at the leaf node of the path is the expected answer $\mathcal{K}_{\text{e}}$.

An Evaluator LLM ($\mathcal{M}_{\text{eval}}$) takes as input the HySem-generated JSON ($\mathcal{J}_{\text{p}}$), a question ($\mathcal{Q}_{\text{i}}$), and the expected answer ($\mathcal{K}_{\text{e}}$). The Evaluator LLM predicts an answer ($\mathcal{K}_{\text{p}}$) and compares its prediction with the expected answer ($\mathcal{K}_{\text{e}}$), all in a single pass through the LLM.

Hypothesis: *Verifying each leaf node in the JSON is sufficient to confirm the structure's accuracy. If the JSON is incorrect, it will fail to answer questions about these leaf nodes accurately.*

The Evaluator LLM computes the score for the i-th question-answer pair as follows:

$$\mathcal{M}_{\text{eval}}(p_i) = \begin{cases} 1 & \text{if } \mathcal{K}_{\text{p}} = \mathcal{K}_{\text{e}}, \\ 0 & \text{otherwise.} \end{cases}$$

The Extrinsic Score (ESC) is computed as:

$$ESC = \frac{1}{|\mathcal{P}|} \sum_{p_i \in \mathcal{P}} \text{Pred}(p_i).$$

4 Results

We present the results of HySem, measured on 608 testing samples manually annotated from the open-sourced FinTabNet and PubTabNet datasets. The performance of our pipeline was evaluated using both intrinsic and extrinsic metrics, as described in the previous section, ensuring a thorough and well-rounded assessment. The results demonstrate that HySem delivers competitive performance compared to industry-leading models and outperforms popular open-source LLMs while excelling significantly in terms of *token efficiency*.

4.1 Baselines and Benchmarking

Our method uses LLaMA-3-Base as the foundational LLM, which we fine-tune to enhance performance for our specific table transformation task. To assess the improvements gained, we benchmarked HySem against popular open-source and proprietary models, including Meta LLaMA-3-8B-Instruct,

Model	Intrinsic Score	Extrinsic Score
Meta-Llama-3-8B-Instruct	66.25	84.42
Phi-3-Medium-128K-Instruct	56.78	82.39
GPT-4o	93.43	90.15
HySem (Ours)	91.12	88.39

Table 3: Benchmarking Results

Description	Metric
Number of test samples	608
Number of tokens before Token-based Encoding (A)	366608
Number of tokens after Token-based Encoding (B)	224082
Token Efficiency(%)	38.87

Table 4: Token Efficiency Metrics

Microsoft Phi-3-Medium-128K-Instruct, and OpenAI GPT-4o. Table 3 shows our benchmarking results.

LLaMA-3-8B-Instruct: HySem surpasses LLaMA-3-8B-Instruct by over 25% in intrinsic accuracy and 3.97% in extrinsic scores, proving that our task-specific fine-tuning significantly enhances the model's capabilities.

Phi-3-Medium-128K-Instruct: The Phi-3 model displayed weak performance on intrinsic metrics but fared better in extrinsic evaluations, achieving 82.39%. HySem outperformed Phi-3 in both areas, demonstrating its superiority as a fine-tuned model specialized for this task.

GPT-4o: While GPT-4o demonstrates superior overall accuracy metrics, largely due to extensive training on diverse datasets, particularly involving HTML and XML table formats [28], HySem maintains a competitive edge. Although trailing by 2% in accuracy, HySem offers substantial advantages such as on-premise deployment on commodity hardware, cost-effectiveness, and enhanced data privacy—essential requirements for many enterprises.

4.2 Token Reduction Efficiency

We evaluated the token reduction efficiency of HySem's Context Optimizer, which enhances token usage while maintaining semantic accuracy. HySem improves model efficiency, resulting in faster inference times and reduced computational overhead.

Token Efficiency is defined as:

$$\text{Token Efficiency} = \left(1 - \frac{B}{A}\right) \times 100 \tag{1}$$

Table 4 shows that HySem achieves over 38% token efficiency, reflecting a substantial reduction in the token count required for processing, thereby enhancing the LLM inference throughput. The results demonstrate significant benefits gained with using our Token-based encoding method.

5 Conclusion and Future Work

We successfully implemented the HySem LLM Pipeline, which generates semantic JSON output from HTML tables. The Context Optimizer significantly improved context utilization, enabling us to process larger tables. Currently, we are piloting our product with a well-known pharmaceutical enterprise that operates multiple production plants and serves a global customer base. This model powers several downstream applications, including analytics from regulatory compliance documents and the automatic creation of these documents. Our future work includes supporting even larger tables that span multiple pages, developing techniques to improve processing speed, and building custom models for other verticals.

References

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

[2] Armen Aghajanyan, Dmytro Okhonko, Mike Lewis, Mandar Joshi, Hu Xu, Gargi Ghosh, and Luke Zettlemoyer. HTLM: Hyper-text pre-training and prompting of language models. In *International Conference on Learning Representations*, 2022.

[3] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *arXiv preprint arXiv:2310.11511*, 2023.

[4] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *IEEE transactions on neural networks and learning systems*, 2022.

[5] Vadim Borisov, Kathrin Sessler, Tobias Leemann, Martin Pawelczyk, and Gjergji Kasneci. Language models are realistic tabular data generators. In *The Eleventh International Conference on Learning Representations*, 2023.

[6] Xinyue Chen, Pengyu Gao, Jiangjiang Song, and Xiaoyang Tan. Hiqa: A hierarchical contextual augmentation rag for massive documents qa. *arXiv preprint arXiv:2402.01767*, 2024.

[7] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[8] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[9] Xi Fang, Weijie Xu, Fiona Anting Tan, Ziqing Hu, Jiani Zhang, Yanjun Qi, Srinivasan H. Sengamedu, and Christos Faloutsos. Large language models (LLMs) on tabular data: Prediction, generation, and understanding - a survey. *Transactions on Machine Learning Research*, 2024.

[10] Zafeirios Fountas, Martin A Benfeghoul, Adnan Oomerjee, Fenia Christopoulou, Gerasimos Lampouras, Haitham Bou-Ammar, and Jun Wang. Human-like episodic memory for infinite context llms. *arXiv preprint arXiv:2407.09450*, 2024.

[11] Izzeddin Gur, Ofir Nachum, Yingjie Miao, Mustafa Safdari, Austin Huang, Aakanksha Chowdhery, Sharan Narang, Noah Fiedel, and Aleksandra Faust. Understanding HTML with large language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2803–2821, Singapore, December 2023. Association for Computational Linguistics.

[12] Chi Han, Qifan Wang, Hao Peng, Wenhan Xiong, Yu Chen, Heng Ji, and Sinong Wang. LM-infinite: Zero-shot extreme length generalization for large language models. In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3991–4008, Mexico City, Mexico, June 2024. Association for Computational Linguistics.

[13] Ziwei Ji, Tiezheng Yu, Yan Xu, Nayeon Lee, Etsuko Ishii, and Pascale Fung. Towards mitigating LLM hallucination via self reflection. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 1827–1843, Singapore, December 2023. Association for Computational Linguistics.

[14] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[15] Keti Korini and Christian Bizer. Column type annotation using chatgpt. *arXiv preprint arXiv:2306.00745*, 2023.

[16] Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. Table-gpt: Table fine-tuned gpt for diverse table tasks. *Proc. ACM Manag. Data*, 2(3), may 2024.

[17] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

[18] Tennison Liu, Zhaozhi Qian, Jeroen Berrevoets, and Mihaela van der Schaar. GOGGLE: Generative modelling for tabular data by learning relational structure. In *The Eleventh International Conference on Learning Representations*, 2023.

[19] Elita Lobo, Oktie Hassanzadeh, Nhan Pham, Nandana Mihindukulasooriya, Dharmashankar Subramanian, and Horst Samulowitz. Matching table metadata with business glossaries using large language models. *arXiv preprint arXiv:2309.11506*, 2023.

[20] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

[21] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

[22] Avanika Narayan, Ines Chami, Laurel Orr, and Christopher Ré. Can foundation models wrangle your data? *Proc. VLDB Endow.*, 16(4):738–746, dec 2022.

[23] Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. Automatically correcting large language models: Surveying the landscape of diverse automated correction strategies. *Transactions of the Association for Computational Linguistics*, 12:484–506, 2024.

[24] Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics ACL 2024*, pages 963–981, Bangkok, Thailand and virtual meeting, August 2024. Association for Computational Linguistics.

[25] Sohan Patnaik, Heril Changwal, Milan Aggarwal, Sumit Bhatia, Yaman Kumar, and Balaji Krishnamurthy. CABINET: Content relevance-based noise reduction for table question answering. In *The Twelfth International Conference on Learning Representations*, 2024.

[26] Matthew Renze and Erhan Guven. Self-reflection in llm agents: Effects on problem-solving performance. *arXiv preprint arXiv:2405.06682*, 2024.

[27] Brandon Smock, Rohith Pesala, and Robin Abraham. Pubtables-1m: Towards comprehensive table extraction from unstructured documents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4634–4642, 2022.

[28] Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 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, 2024.

[29] Anirudh S. Sundar and Larry Heck. cTBLS: Augmenting large language models with conversational tables. In Yun-Nung Chen and Abhinav Rastogi, editors, *Proceedings of the 5th Workshop on NLP for Conversational AI (NLP4ConvAI 2023)*, pages 59–70, Toronto, Canada, July 2023. Association for Computational Linguistics.

[30] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 55:1 – 28, 2020.

[31] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

[32] Yuzhang Tian, Jianbo Zhao, Haoyu Dong, Junyu Xiong, Shiyu Xia, Mengyu Zhou, Yun Lin, José Cambronero, Yeye He, Shi Han, et al. Spreadsheetllm: Encoding spreadsheets for large language models. *arXiv preprint arXiv:2407.09025*, 2024.

[33] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[34] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[35] Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. Chain-of-table: Evolving tables in the reasoning chain for table understanding. In *The Twelfth International Conference on Learning Representations*, 2024.

[36] Peng Xu, Wei Ping, Xianchao Wu, Lawrence McAfee, Chen Zhu, Zihan Liu, Sandeep Subramanian, Evelina Bakhturina, Mohammad Shoeybi, and Bryan Catanzaro. Retrieval meets long context large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

[37] Jiahuan Yan, Bo Zheng, Hongxia Xu, Yiheng Zhu, Danny Chen, Jimeng Sun, Jian Wu, and Jintai Chen. Making pre-trained language models great on tabular prediction. In *The Twelfth International Conference on Learning Representations*, 2024.

[38] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

[39] Pengcheng Yin, Graham Neubig, Wen tau Yih, and Sebastian Riedel. TaBERT: Pretraining for joint understanding of textual and tabular data. In *Annual Conference of the Association for Computational Linguistics (ACL)*, July 2020.

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

[41] Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M Patel. Reactable: Enhancing react for table question answering. *arXiv preprint arXiv:2310.00815*, 2023.

[42] Bowen Zhao, Changkai Ji, Yuejie Zhang, Wen He, Yingwen Wang, Qing Wang, Rui Feng, and Xiaobo Zhang. Large language models are complex table parsers. *arXiv preprint arXiv:2312.11521*, 2023.

[43] Mingyu Zheng, Hao Yang, Wenbin Jiang, Zheng Lin, Yajuan Lyu, Qiaoqiao She, and Weiping Wang. Chain-of-thought reasoning in tabular language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[44] Xinyi Zheng, Douglas Burdick, Lucian Popa, Xu Zhong, and Nancy Xin Ru Wang. Global table extractor (gte): A framework for joint table identification and cell structure recognition using visual context. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 697–706, 2021.

[45] Jin Ziqi and Wei Lu. Tab-CoT: Zero-shot tabular chain of thought. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 10259–10277, Toronto, Canada, July 2023. Association for Computational Linguistics.

A Appendix

A.1 Related Works

LLMs for Tabular Data

Tables are a fundamental method for presenting structured information across various industrial applications, including regulatory information in the pharmaceutical industry and financial reports for businesses. Some shared characteristics and inherent challenges in tables as mentioned by [9] include Heterogeneity [5], Sparsity, Context-based interconnection [18], Lack of prior knowledge [4, 5], etc. Recent advancements in Large Language Models [1, 14, 31, 8] have shown promise in addressing these challenges for various table related tasks. Some table related tasks and techniques include entity matching and data-imputation [16, 15], tabular Q&A [6, 35, 41, 25], schema augmentation [22], Serialization [39, 37], Table Manipulation [40], Table Understanding [19], Prompt Engineering [43, 45], Table RAG [29]. Role-play [42].

LLMs and HTML

Recent research has explored various uses of HTML to enhance NLP tasks. [21, 38] utilize LLMs for Web Navigation tasks involving HTML input formats. [2] developed the **HyperText Language Model (HTLM)**, which leverages HTML's structural elements to improve tasks like zero-shot summarization and question answering. Their approach utilizes HTML for structured prompting and template-based guidance but is largely confined to standard NLP tasks and assumes predefined HTML structures, such as *<title>* elements. [11] focused on HTML understanding through tasks like semantic classification, description generation, and autonomous web navigation. They identified the **context window length** as a significant bottleneck, noting that even models supporting longer token sequences struggle with performance degradation when processing larger snippet sizes.

In contrast to the aforementioned works, our approach focuses not merely on HTML, but specifically on the more complex domain of **HTML tables** particularly Pharmaceutical tables, which present significant challenges due to their hierarchical and relational structure. HTML tables often contain multi-level headers, rowspan and colspan attributes, and intricate relationships between cell elements, making their transformation into a semantic format far more demanding. Our work uniquely tackles this complexity by performing a **table transformation task**, converting HTML tables into semantic JSON. This transformation is further enhanced by our **Context Optimizer**, which efficiently reduces token usage while preserving the intricate relationships between table elements. Unlike previous approaches, which focus on predefined HTML structures, our method handles the complexities of tables dynamically, ensuring both structural accuracy and computational efficiency. **To the best of our knowledge**, this is the first approach to transform complex HTML tables into semantic JSON in a context-length optimized manner.

Context Optimization for Tables in LLMs

Recent advancements in context-length optimization have primarily focused on improving LLM performance for text-based tasks [24, 10, 12], but limited work has been done to address the unique challenges presented by tables. [28] tackled context length limitations by serializing tables into text formats. While these methods help fit table data within the model's context window, serialization can lead to the loss of structural nuances and detailed information. In contrast, [32] introduces a novel encoding method tailored for spreadsheet data. While effective in reducing token usage and computational costs, it addresses a different use-case (spreadsheets) compared to our work (HTML tables). Additionally, the encoding method primarily leverages format and structure based optimisations, whereas our approach rewrites the input to ensure alignment with the tokenizer.

Our approach specifically targets HTML tables of arbitrary structure, ensuring that hierarchical and relational elements are preserved during HTML to JSON transformation task. By utilizing our Context Optimizer, we ensure semantic integrity is maintained even during token pruning, unlike text-based methods where reducing detail (e.g., abbreviating "potassium clavulanate" to "potassium") would alter the meaning. This is a significant advantage in transforming HTML tables into semantic JSON without compromising accuracy.

A.2 Complex Pharmaceutical Data Table

Table 7: Summary of Mean Changes from Baseline* in HbA_{1c}, Fasting Plasma Glucose, and Body Weight at Week 12 and at Final Visit (24-week study)

	GLUCOPHAGE 500 mg Twice Daily	GLUCOPHAGE XR 1000 mg Once Daily	GLUCOPHAGE XR 1500 mg Once Daily
Hemoglobin A_{1c} (%)	**(n=67)**	**(n=72)**	**(n=66)**
Baseline	7.06	6.99	7.02
Change at 12 Weeks	0.14	0.23	0.04
(95% CI)	(-0.03, 0.31)	(0.10, 0.36)	(-0.08, 0.15)
Change at FINAL VISIT	0.14[a]	0.27	0.13
(95% CI)	(-0.04, 0.31)	(0.11, 0.43)	(-0.02, 0.28)
FPG (mg/dL)	**(n=69)**	**(n=72)**	**(n=70)**
Baseline	127.2	131.0	131.4
Change at 12 Weeks	12.9	9.5	3.7
(95% CI)	(6.5, 19.4)	(4.4, 14.6)	(-0.4, 7.8)
Change at FINAL VISIT	14.0	11.5	7.6
(95% CI)	(7.0, 21.0)	(4.4, 18.6)	(1.0, 14.2)
Body Weight (lbs)	**(n=71)**	**(n=74)**	**(n=71)**
Baseline	210.3	202.8	192.7
Change at 12 Weeks	0.4	0.9	0.7
(95% CI)	(-0.4, 1.5)	(0.0, 2.0)	(-0.4, 1.8)
Change at FINAL VISIT	0.9	1.1	0.9
(95% CI)	(-0.4, 2.2)	(-0.2, 2.4)	(-0.4, 2.0)

Figure 3: This table compares various dosage regimens of Glucophage and Glucophage XR. The challenge in converting it to semantic JSON lies in its nested categories, including dosage, measurement types, and time points. Each combination of dose, metric, time point, and statistical details (mean change, confidence intervals) must be accurately mapped to database schema keys. Ensuring data integrity while managing variability in column structures (e.g., different dosage forms, units, and confidence interval ranges) is essential for creating a usable semantic representation.

A.3 Semantic JSON

In our paper, the term **"Semantic JSON"** refers to a JSON representation where the keys are designed to accurately mirror the hierarchical structure of the JSON tree and align directly with the database schema. This ensures that the JSON output is not only well-structured but also semantically meaningful. Figure 5 provides an example of such a Semantic JSON output, as generated by Hysem.

To highlight the effectiveness of our approach, we compare the outputs generated by other models, namely Meta-Llama-3-8B-Instruct and Microsoft-Phi-3-128K-Instruct, as illustrated in Figures 6 & 7 respectively. These illustrations show how the JSON representations from these models differ from ours, particularly in terms of hierarchical accuracy and schema alignment.

Direction	Slipped side		Non-slipped side		
	Median	Range	Median	Range	*p* -Value
Medial/lateral	1.52 medial	1.33 lateral to 4.28 medial	1.74 medial	0.16–3.34 medial	0.717
Cranial/caudal	0.16 caudal	2.80 caudal to 3.58 cranial	2.28 cranial	0.02 caudal to 4.25 cranial	0.003
Anterior/posterior	2.28 posterior	7.25 posterior to 1.33 anterior	0.91 posterior	2.95 posterior to 1.02 anterior	0.03
Two-plane (frontal)	3.13	0.4–4.8	3.11	0.51–4.52	0.379
Three-plane (total)	3.92	0.52–8.54	3.3	0.52–5.21	0.148

Figure 4: Sample Table to illustrate Semantic JSON

HySem output versus Meta LLama 3 8B Instruct and Microsoft Phi 3 128K Instruct

The sample illustrates that the Llama 3 output has missed certain fields, such as 'Slipped Side' and 'Non-slipped Side.' In contrast, Phi 3 has introduced several errors, including misplacing values under the incorrect hierarchy. For instance, the p-value of 0.717 is erroneously placed under the 'range_values' subtree for the 'Non-slipped Side.'


```
{"Direction": {
  "Medial/lateral": {
   "Slipped side": {
    "Median": "1.52 medial",
    "Range": "1.33 lateral to 4.28 medial"
   },
   "Non-slipped side": {
    "Median": "1.74 medial",
    "Range": "0.16 to 3.34 medial"
   },
   "p-Value": "0.717"
  },
  "Cranial/caudal": {
   "Slipped side": {
    "Median": "0.16 caudal",
    "Range": "2.80 caudal to 3.58 cranial"
   },
   "Non-slipped side": {
    "Median": "2.28 cranial",
    "Range": "0.02 caudal to 4.25 cranial"
   },
   "p-Value": "0.003"
  },
  "Anterior/posterior": {
   "Slipped side": {
    "Median": "2.28 posterior",
    "Range": "7.25 posterior to 1.33 anterior"
   },
   "Non-slipped side": {
    "Median": "0.91 posterior",
    "Range": "2.95 posterior to 1.02 anterior"
   },
   "p-Value": "0.030"
  },
  "Two-plane (frontal)": {
   "Slipped side": {
    "Median": "3.13",
    "Range": "0.4 to 4.8"
   },
   "Non-slipped side": {
    "Median": "3.11",
    "Range": "0.51 to 4.52"
   },
   "p-Value": "0.379"
  },
  "Three-plane (total)": {
   "Slipped side": {
    "Median": "3.92",
    "Range": "0.52 to 8.54"
   },
   "Non-slipped side": {
    "Median": "3.30",
    "Range": "0.52 to 5.21"
   },
   "p-Value": "0.148"
  }}}
```


Figure 5: Hysem JSON


```
{
 "Direction": [
  {
   "value":
"Medial/lateral",
   "p-value": 0.717,
   "Median": "1.52
medial",
   "Range": "1.33
lateral to 4.28 medial"
  },
  {
   "value":
"Cranial/caudal",
   "p-value": 0.003,
   "Median": "0.16
caudal",
   "Range": "2.80
caudal to 3.58 cranial"
  },
  {
   "value":
"Anterior/posterior",
   "p-value": 0.03,
   "Median": "2.28
posterior",
   "Range": "7.25
posterior to 1.33
anterior"
  },
  {
   "value": "Two-plane
(frontal)",
   "p-value": 0.379,
   "Median": "3.13",
   "Range": "0.4 to 4.8"
  },
  {
   "value": "Three-
plane (total)",
   "p-value": 0.148,
   "Median": "3.92",
   "Range": "0.52 to
8.54"
  }
 ]
}
```


Figure 6: LlaMa3 JSON


```
{ "directional_comparisons": [
  {
   "direction": "Medial/lateral",
   "median_values": {
    "slipped_side": 1.52,
    "non_slipped_side": 1.74 },
   "range_values":
  {
    "slipped_side": [
     0.16,
     3.34],
    "non_slipped_side": [
     0.717]}},
  {
   "direction": "Cranial/caudal",
   "median_values": {
    "slipped_side": 2.8,
    "non_slipped_side": 2.28},
   "range_values": {
    "slipped_side": [
     0.02,
     4.25],
    "non_slipped_side": 0.379} },
  {
   "direction": "Anterior/posterior",
   "median_values": {
    "slipped_side": 2.28,
    "non_slipped_side": 0.91
   },
   "range_values":
  {
    "slipped_side": [
     2.95,
     1.02
    ],
    "non_slipped_side": 0.379}},
  {
   "direction": "Two-plane (frontal)",
   "median_values": {
    "slipped_side": 3.13,
    "non_slipped_side": 3.11
   },
   "range_values": [
    0.51,
    4.52]],
  {
   "direction": "Three-plane (total)",
   "median_values": {
    "slipped_side": 3.92,
    "non_slipped_side": 3.3
   },
   "range_values": [
    0.52,
    5.21
   ]}]}
```


Figure 7: Phi3 JSON

A.4 Evaluation Methodology Illustrations

The performance of HySem is assessed using both intrinsic and extrinsic evaluation methods. Intrinsic evaluation focuses on how accurately the input HTML strings are converted to JSON, while extrinsic evaluation assesses how effectively the semantic structure is preserved.

	Time	Polyol		Xylitol		*p* value one-way ANOVA
		Subjects (*n*)	**Mean ± SD**	**Subjects (*n*)**	**Mean ± SD**	
Gum use	Baseline	90	5.32 ± 0.43	89	5.41 ± 0.35	0.29
	6 months	79	5.22 ± 0.21	77	5.33 ± 0.46	0.31
	12 months	72	5.33 ± 0.42	71	5.16 ± 0.42	0.03
No-gum use	24 months	64	5.33 ± 0.46	66	5.15 ± 0.64	0.04
	p value one-way ANOVA		0.42		<0.01	

Figure 8: Sample of HTML Table for illustration

For the given HTML table, the HySem generated semantic JSON and the ground-truth (GT) JSON are as below.


```
{
 "Gum use": {
  "Time": {
   "Baseline": {
    "Polyol": {
     "Subjects (n)": "90",
     "Mean ± SD": "5.32 ± 0.43"
    },
    "Xylitol": {
     "Subjects (n)": "89",
     "Mean ± SD": "5.41 ± 0.35"
    },
    "p value one-way ANOVA": "0.29"
   }
  },
  "6 months": {
   "Polyol": {
    "Subjects (n)": "79",
    "Mean ± SD": "5.22 ± 0.21"
   },
   "Xylitol": {
    "Subjects (n)": "77",
    "Mean ± SD": "5.33 ± 0.46"
   },
   "p value one-way ANOVA": "0.31"
  },
  "12 months": {
   "Polyol": {
    "Subjects (n)": "72",
    "Mean ± SD": "5.33 ± 0.42"
   },
   "Xylitol": {
    "Subjects (n)": "71",
    "Mean ± SD": "5.16 ± 0.42"
   },
   "p value one-way ANOVA": "0.03"
  }
 },
 "No-gum use": {
  "24 months": {
   "Polyol": {
    "Subjects (n)": "64",
    "Mean ± SD": "5.33 ± 0.46"
   },
   "Xylitol": {
    "Subjects (n)": "66",
    "Mean ± SD": "5.15 ± 0.64"
   },
   "p value one-way ANOVA": "0.04"
  }
 }
}
```


Figure 9: Hysem JSON


```
{"Gum use": {
  "Time": {
   "Baseline": {
    "Polyol": {
     "Subjects (n)": "90",
     "Mean ± SD": "5.32 ± 0.43"
    },
    "Xylitol": {
     "Subjects (n)": "89",
     "Mean ± SD": "5.41 ± 0.35"
    },
    "p value one-way ANOVA": "0.29"
   }},
  "6 months": {
   "Polyol": {
    "Subjects (n)": "79",
    "Mean ± SD": "5.22 ± 0.21"
   },
   "Xylitol": {
    "Subjects (n)": "77",
    "Mean ± SD": "5.33 ± 0.46"
   },
   "p value one-way ANOVA": "0.31"
  },
  "12 months": {
   "Polyol": {
    "Subjects (n)": "72",
    "Mean ± SD": "5.33 ± 0.42"
   },
   "Xylitol": {
    "Subjects (n)": "71",
    "Mean ± SD": "5.16 ± 0.42"
   },
   "p value one-way ANOVA": "0.03"
  }},
 "No-gum use": {
  "24 months": {
   "Polyol": {
    "Subjects (n)": "64",
    "Mean ± SD": "5.33 ± 0.46"
   },
   "Xylitol": {
    "Subjects (n)": "66",
    "Mean ± SD": "5.15 ± 0.64"
   },
   "p value one-way ANOVA": {
    "Polyol": {
     "Mean ± SD": "0.42"
    },
    "Xylitol": {
     "Mean ± SD": "<0.01"
    }}}}}
```


Figure 10: GT JSON

The **Intrinsic Evaluator's** evaluation results are presented in Table 5. Here, each unique cell content of the original HTML table is compared directly to the nodes of the JSON.

We initially used metrics to measure how well the JSON preserved the *"frequency"* of each string from the HTML. However, this method presented problems. For e.g., a cell content could appear repeatedly in the JSON for each row entry, unlike its single occurrence in the HTML. This discrepancy can skew the accuracy of metrics based on string frequency.

To address this, we simplified the approach by ensuring each unique string from the HTML appears at least once in the JSON. While it's a reasonable estimate of performance, we recognise that this is an area for future improvement.

Cell Content	**Presence in Hysem JSON (Yes/No)**
"Gum use"	✓
"Time"	✓
"Baseline"	✓
"Polyol"	✓
"Subjects (n)"	✓
"90"	✓
"Mean ± SD"	✓
"5.32 ± 0.43"	✓
"Xylitol"	✓
"89"	✓
"5.41 ± 0.35"	✓
"p value one-way ANOVA"	✓
"0.29"	✓
"6 months"	✓
"79"	✓
"5.22 ± 0.21"	✓
"77"	✓
"5.33 ± 0.46"	✓
"0.31"	✓
"12 months"	✓
"72"	✓
"5.33 ± 0.42"	✓
"71"	✓
"5.16 ± 0.42"	✓
"0.03"	✓
"No-gum use"	✓
"24 months"	✓
"64"	✓
"5.33 ± 0.46"	✓
"66"	✓
"5.15 ± 0.64"	✓
"0.04"	✓
"<0.01"	x
"0.42"	x
Intrinsic Score:	**94.11%**

Table 5: Intrinsic Evaluation results

The **Extrinsic evaluation** measures the semantic accuracy of the LLM generated JSON. In the example table, there are 21 unique paths from the root node to each leaf node of the GT JSON. As mentioned, the number of paths is equal to the number of leaf nodes. LLM ($\mathcal{M}_q$) accepts GT JSON and a path and outputs a single question such that the expected answer is the leaf node of the path. Table 6 illustrates the paths, corresponding LLM generated questions and the GT answer.

Path	Question	GT Answer
Gum use > Time > Baseline > Polyol > Subjects (n)	What is the number of subjects in the baseline group that used gum containing polyol?	90
Gum use > Time > Baseline > Polyol > Mean ± SD	What is the average amount of gum used by subjects at baseline, with a standard deviation?	5.32 ± 0.43
Gum use > Time > Baseline > Xylitol > Subjects (n)	What is the number of subjects in the baseline group that used Xylitol?	89
Gum use > Time > Baseline > Xylitol > Mean ± SD	What is the average amount of Xylitol used by subjects at baseline, with a standard deviation?	5.41 ± 0.35
Gum use > Time > Baseline > p value one-way ANOVA	What is the p-value of the one-way ANOVA test for the baseline data?	0.29
Gum use > 6 months > Polyol > Subjects (n)	What is the number of subjects in the group that used gum for 6 months and was given polyol?	90
Gum use > 6 months > Polyol > Mean ± SD	What is the mean value of gum usage after 6 months for polyol?	5.22 ± 0.21
Gum use > 6 months > Xylitol > Subjects (n)	What is the number of subjects in the group that used Xylitol for 6 months?	77
Gum use > 6 months > Xylitol > Mean ± SD	What is the mean value of Xylitol at 6 months, along with its standard deviation?	5.33 ± 0.46
Gum use > 6 months > p value one-way ANOVA	What is the p-value of the one-way ANOVA test for the comparison between Polyol and Xylitol at 6 months?	0.31
Gum use > 12 months > Polyol > Subjects (n)	How many subjects were in the group that used gum for 12 months and had a polyol treatment?	72
Gum use > 12 months > Polyol > Mean ± SD	What is the average value of gum use for 12 months with polyol, along with its standard deviation?	5.33 ± 0.42
Gum use > 12 months > Xylitol > Subjects (n)	What is the number of subjects in the group that used Xylitol for 12 months?	71
Gum use > 12 months > Xylitol > Mean ± SD	What is the mean value of Xylitol at 12 months, along with its standard deviation?	5.16 ± 0.42
Gum use > 12 months > p value one-way ANOVA	What is the p-value of the one-way ANOVA test for the comparison at 12 months?	0.03
No-gum use > 24 months > Polyol > Subjects (n)	How many subjects were in the group that did not use gum and had a follow-up at 24 months, with a focus on polyol?	64
No-gum use > 24 months > Polyol > Mean ± SD	What is the mean value of gum usage after 24 months for subjects using polyol, along with its standard deviation?	5.33 ± 0.46
No-gum use > 24 months > Xylitol > Subjects (n)	How many subjects were in the group that did not use gum and had data collected at 24 months, with a focus on xylitol?	66
No-gum use > 24 months > Xylitol > Mean ± SD	What is the mean value of Xylitol at 24 months for subjects with no gum use, along with its standard deviation?	5.15 ± 0.64
No-gum use > 24 months > p value one-way ANOVA > Polyol > Mean ± SD	What is the mean difference in standard deviation of 'p value one-way ANOVA' for 'Xylitol' for 'Polyol' at 24 months in a group with no gum use?	0.42
No-gum use > 24 months > p value one-way ANOVA > Xylitol > Mean ± SD	What is the mean difference in standard deviation of 'p value one-way ANOVA' for 'Xylitol' at '24 months' under 'No-gum use'?	< 0.01

Table 6: Paths, Questions and GT Answers of Extrinsic Evaluation

The evaluator LLM ($\mathcal{M}_{\text{eval}}$), predicts an answer, given the LLM-generated JSON, a single question, and the expected answer as inputs. It then compares its prediction with the expected answer to generate a score, all in a single pass through the LLM. Table 7 shows the predicted answers and the extrinsic scores computed for each question, along with the overall extrinsic score.

Predicted Answer	**Score**
90	1
5.32 ± 0.43	1
89	1
5.41 ± 0.35	1
0.29	1
90	1
5.22 ± 0.21	1
77	1
5.33 ± 0.46	1
0.31	1
72	1
5.33 ± 0.42	1
71	1
5.16 ± 0.42	1
0.03	1
64	1
5.33 ± 0.46	1
66	1
5.15 ± 0.64	1
0.04	0
0.03	0
Extrinsic Score	90.47%

Table 7: Extrinsic Evaluation results

A.5 Syntax Corrector: Algorithm

Our Semantic Synthesizer incorporates a custom fine-tuned model specialized in generating JSON outputs. However, occasional syntax errors may occur in the generated output. The Syntax Corrector, part of the HySem pipeline, handles these erroneous JSONs by performing LLM-assisted auto-correction via self-reflection. We instruct the LLM with the following prompt:

```
"The JSON given below contains syntax errors.  Your task is to correct
them and provide the corrected output.  Provide ONLY the corrected output,
without any additional explanation.  'input_string':  {json_input}"
```

Algorithm 2 presents the syntax correction algorithm.

Algorithm 2 Syntax Corrector

Require: Syntactically invalid JSON $\mathcal{J}_{\text{invalid}}$ from LLM
Ensure: Syntactically valid JSON $\mathcal{J}_{\text{valid}}$

$\mathcal{J}_{\text{current}} \leftarrow \mathcal{J}_{\text{invalid}}$
iterations $\leftarrow 0$
while syntax errors in $\mathcal{J}_{\text{current}}$ **and** iterations $<$ max_iterations **do**
    if $\mathcal{J}_{\text{current}}$ is syntactically invalid **then**
        $\mathcal{J}_{\text{current}} \leftarrow \mathcal{A}_{\text{sv}}(\mathcal{J}_{\text{current}})$
    end if
    iterations $\leftarrow$ iterations + 1
end while
$\mathcal{J}_{\text{valid}} \leftarrow \mathcal{J}_{\text{current}}$
return $\mathcal{J}_{\text{valid}}$