SLOT: Structuring the Output of Large Language Models

Darren Yow-Bang Wang*, Zhengyuan Shen*, Soumya Smruti Mishra
Zhichao Xu, Yifei Teng, Haibo Ding
Amazon Web Services
{ybwang, donshen, soumish, xzhichao, yifeit, hbding}@amazon.com

Abstract

Structured outputs are essential for large language models (LLMs) in critical applications like agents and information extraction. Despite their capabilities, LLMs often generate outputs that deviate from predefined schemas, significantly hampering reliable application development. We present SLOT (**S**tructured **L**LM **O**utput **T**ransformer), a model-agnostic approach that transforms unstructured LLM outputs into precise structured formats. While existing solutions predominantly rely on constrained decoding techniques or are tightly coupled with specific models, SLOT employs a fine-tuned lightweight language model as a post-processing layer, achieving flexibility across various LLMs and schema specifications. We introduce SLOTBENCH, curated by a data synthesis pipeline alongside a formal evaluation methodology that quantifies both schema accuracy and content fidelity. Our results demonstrate that fine-tuned Mistral-7B model with constrained decoding achieves near-perfect schema accuracy (99.5%) and content similarity (94.0%), outperforming Claude-3.5-Sonnet by substantial margins (+25 and +20 percentage points, respectively). Notably, even compact models like Llama-3.2-1B can match or exceed the structured output capabilities of much larger proprietary models when equipped with SLOT, enabling reliable structured generation in resource-constrained environments.

1 Introduction

The emergence of large language models (LLMs) (OpenAI, 2023; Gemini, 2023) has led to a wide range of applications that leverage their natural language understanding capabilities. In such applications, developers are often required to carefully craft prompts to elicit specific and reliable responses from LLMs, followed by post-processing of the generated outputs to derive

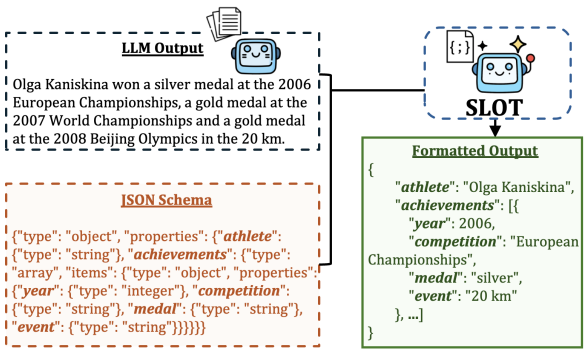

Figure 1: SLOT converts a textual LLM response into structured JSON with a pre-defined schema.

structured and precise results (Liu et al., 2023). This process becomes particularly critical when LLMs are integrated as components of within more complex systems, such as for function calling or multi-agent collaboration, where inaccuracies in earlier stages can propagate through the workflow, adversely affecting overall system performance.

An emerging requirement for LLM-based applications is to support structured output (Lu et al., 2025; Tam et al., 2024a; Liu et al., 2025; Kulkarni and Srikumar, 2025, *inter alia*). Although certain proprietary models (e.g., GPT-4o, Hurst et al., 2024) inherently support structured output through specialized training and constrained decoding (CD) mechanisms, replicating this capability for other LLMs presents significant challenges. In particular, for platforms that serve a diverse set LLMs and required to provide day-1 support to new model releases, including open-weight and proprietary models, post-training each model for structured output without undermining general purpose performance is neither feasible nor scalable.

To tackle these challenges, we introduce SLOT for converting unstructured text into structured output. Different from existing approaches that relies on LLM post-training and constrained decoding, SLOT post-processes LLM's unstructured output by leveraging a lightweight language model,

* Contributed equally.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track, pages 472–491
November 4-9, 2025

therefore being task and model-agnostic. Specifically, we fine-tune a lightweight language model to map unstructured text to the target schema, without modifying the underlying LLM. SLOT ensures broad compatibility with both current and emerging models, regardless of their task specialization or output constraints, effectively bridging the gap between the generative flexibility of LLMs and the rigorous structural requirement for downstream integration in modern software systems. Our main contributions are as follows:

- We introduce SLOT for structured output conversion applicable to any textual LLM outputs.
- We introduce SLOTBENCH, curated by a synthetic data pipeline to ensure data diversity.
- We present evaluation metrics for SLOT covering schema accuracy and content similarity.
- We demonstrate lightweight models including Llama-3.2 (1B/3B) and Mistral-7B outperform existing solutions across evaluation dimensions through supervised fine-tuning.

2 Background and Problem Formulation

Structured Output and JSON Schema. In the context of LLM, the term *structured output* means that model-generated content conforms to a predefined, machine readable format rather than free-form natural language (Dong et al., 2024; Jiang et al., 2023b; OpenAI, 2024). Generating structured data from unstructured inputs enables the ability of LLM-based applications to answer questions via function calling, extract structured data and build multi-step agent workflows that allow LLMs to take actions. One of the most widely-used structure format is *JSON Schema*, a declarative language to define structure and constraints for JSON data. JSON schema maintains a consistent pattern, making it easier to ensure data validity and exchange structured data between applications. In this paper, we focus specifically on JSON schema due to its wide adoption.

Problem Formulation. We focus on the post-processing setting (Fig. 1), where we assume access to an LLM's unstructured output. We define our task as a text-to-structure problem, i.e., *to transform an existing LLM's free-form text output into a structured format according to a specified JSON schema*. Given x as an input text and f as the formatting specification, let M_θ be a generative model parameterized by θ: $M_\theta(x, f) \rightarrow y'$, and we seek to optimize the probability distribution $P(y|x, f; \theta)$ where y is the groundtruth structured output and y' the model's structured output. Note that the input text x typically represents a response from an LLM rather than a user query. Our post-processing based method is flexible and does not require access to model weights of the LLM used to generate unstructured response, making it suitable for the case of serving a diverse array of LLMs in a platform.

3 Evaluation Framework

The evaluation of structured output is inherently multi-faceted, as structure-wise, the output needs to adhere to the target JSON schema; while also not derailing from the unstructured input's semantic meaning. Therefore, we design our evaluation framework with the focus on (1) *schema accuracy* and (2) *content similarity*.

Schema Accuracy. We define schema accuracy as $A_s(y' \mid f)$: whether the response JSON string y' exactly matches the user-demanded schema f in terms of key strings and value types. To accurately assess the LLM's formatting capability, we directly evaluate response y'. The response y' must be a valid JSON string by itself to be considered correct.

Content Similarity. The desired structured output should not derail from the unstructured input in terms of semantic meaning. Assume that the groundtruth JSON structured output y is provided, we can compute the content similarity between groundtruth output y and model's output y' as $sim_C(y, y')$. In practice, we evaluate the semantic similarity between y and y'. For each value in the prediction y', we compute its semantic similarity using a pre-trained Sentence-BERT model (Reimers and Gurevych, 2019; Zhang et al., 2020) against the corresponding value (with matching key; including all ancestor keys for nesting structures) in the groundtruth JSON y. Missing keys result in a score of 0. The average of these scores represents the "soft-precision", denoted $sim_P(y, y')$. Similarly, we calculate "soft-recall", denoted $sim_R(y, y')$, by averaging SBERT scores for gold JSON values against their counterparts in the prediction. The content similarity score is therefore the harmonic mean of soft-precision and soft-recall (details in Appx. C.2): $sim_C(y, y') = 2 \times \frac{sim_P(y,y') \times sim_R(y,y')}{sim_P(y,y') + sim_R(y,y')}$.

Additional evaluation dimensions are considered in related work (see Appx. C). For instance, task

performance metrics (Tam et al., 2024b; Beurer-Kellner et al., 2024; Jiang et al., 2024; Shorten et al., 2024; Geng et al., 2025) assess impact on original tasks but may lack generalizability. Computational efficiency metrics, such as latency or speed-up (Willard and Louf, 2023; Geng et al., 2025), prioritize algorithmic performance over output quality. Furthermore, JSON validity checks (Zhou et al., 2023; Beurer-Kellner et al., 2024; Jiang et al., 2024; Agarwal et al., 2025; Geng et al., 2025; Li et al., 2024; Xia et al., 2024) ensure syntax but often do not guarantee semantic correctness. These metrics alone may not simultaneously address the dual requirements of structural correctness and semantic preservation to structured output evaluation, hence are not focused in this work.

4 SLOTBENCH

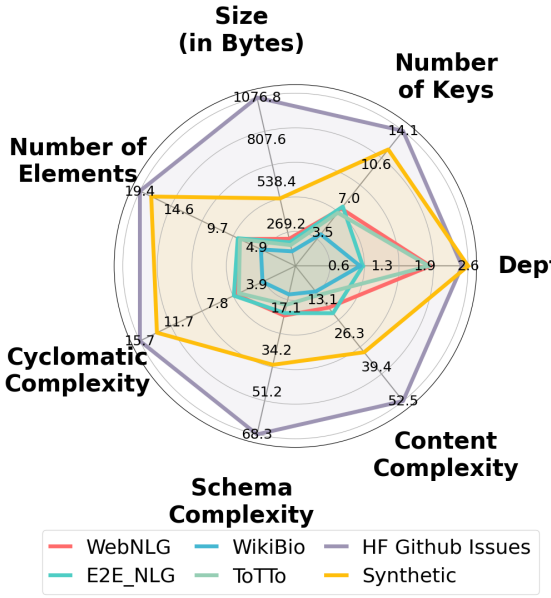

Figure 2: JSON complexity in different dimensions for SLOTBENCH.

To enable training and evaluation of SLOT, we designed a data pipeline (Appx Fig. 5) to repurpose existing public datasets for structured output, and synthesize challenging datasets. The main challenge in creating datasets for text-to-structure tasks lies in obtaining high-quality (x, f, y) triples that: (1) cover diverse domains and text styles, (2) contain valid and well-formed structured outputs, and (3) ensure the structured outputs faithfully represent information present in the input text without hallucination. Therefore, we created SLOTBENCH with a combination of *synthetic data generation* and *careful curation of public datasets*. Our final dataset consists of a synthetic training set mixture of 126K examples and multiple test sets totaling over 9K examples across diverse domains and formats. To analyze the quality of the synthetic data versus existing public datasets, we define seven dimensions to characterize the relative JSON complexity of different datasets, including depths, number of keys, number of elements, size (bytes), cyclomatic complexity, schema complexity and content complexity (details in Appx. D.3), and the breakdowns can be found from Fig. 2.

SLOTBENCH Splits	Train	Val	Test
WebNLG	13,211	-	2,779
E2E NLG	12,568	-	2,347
WikiBio	25,000	650	2,500
ToTTo ◇	-	-	500
HF GitHub Issues ◇	-	-	1,000
Synthetic ⋆	126,000	-	-
Total	176,779	650	9,126

Table 1: Statistics of the SLOTBENCH. ◇ refers to partially synthesized for repurposing and ⋆ refers to fully synthesized.

Test Case Curation. To evaluate LLM Formatter comprehensively, we curated test sets from 5 public datasets spanning different domains and complexity levels (details in Appx. D.1):

- **WebNLG** (Zhou and Lampouras, 2020): Originally designed for structured data-to-text generation, the dataset contains factual descriptions about entities and their relationships.
- **E2E NLG** (Puzikov and Gurevych, 2018): Contains restaurant domain descriptions paired with attribute-value structures.
- **WikiBio** (Lebret et al., 2016): A dataset of biographical sentences paired with infobox-style structured data from Wikipedia.
- **ToTTo** (Parikh et al., 2020): A table-to-text dataset containing highlighted tables cells paired with corresponding descriptive sentences.
- **HF GitHub Issues**: To evaluate model performance on complex, real-world scenarios, we curated a challenging benchmark from Hugging Face Transformers GitHub issues[1]. This dataset captures diverse software issue elements (e.g., system/error information, reproduction steps, code snippets, expected behaviors). It exhibits significantly higher complexity in structure and technical content compared to existing benchmarks, providing a rigorous test of the model's

[1] https://api.github.com/repos/huggingface/transformers/issues

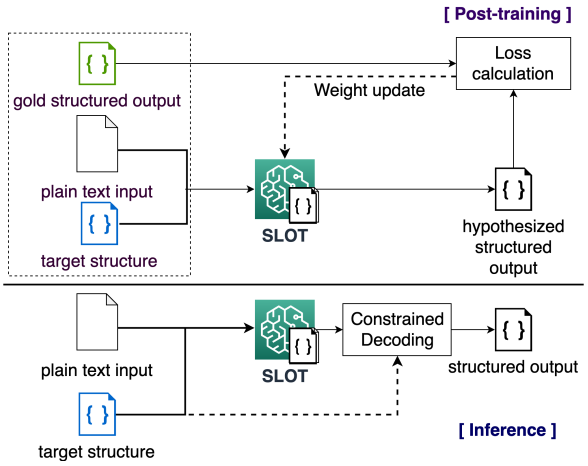

Figure 3: Our proposed framework for post-training and hosting an LLM for structured output generation.

ability to handle intricate nested structures and technical inputs.

Training Data Creation. We use Claude-3.5-Sonnet to systematically generate synthetic training data, ensuring broad coverage by sampling across variables related to diversity and quality (Appx. D.2). For each generation batch, we first sample 1-3 demonstrations from the WikiBio training set. We then randomly sample values for each control dimension to define target characteristics, incorporating these into our prompt (Appx. F.2) to guide generation. The prompt encourages format adherence, content diversity, and semantic coherence between input text and structured output. Temperature sampling ($T \in [0, 0.5]$) is applied to balance creativity and consistency.

Data Filtering. To ensure data quality, we use a two-stage validation process to filter generated triples that are unfactual or contain hallucinations. Stage 1 checks format validity, ensuring f (schema) and y (output) are parsable, structurally consistent, and adhere to type agreements and required structural relationships. Because content validity is challenging for semantic text-to-structure tasks where rule-based/fuzzy matching fails, Stage 2 employs an LLM validator. This validator checks if each structured output field's content is reasonably inferable from the source text (x). This data synthesis pipeline yielded 126K high-quality training examples for training SLOT (§ 6).

5 Training SLOT

Our primiary goal is to train SLOT to accurately transform unstructured text input into a structured representation conforming to a specified schema, optimizing for the evaluation metrics detailed in § 3, which include structural validity, schema compliance, and semantic accuracy of extracted values. To achieve this, we employ Supervised Fine-tuning (SFT) as our core training strategy.

We build SLOT upon a pre-trained decoder language model (e.g., Llama-3 and Mistral), which provides a strong foundation in natural language understanding and generation, significantly reducing the need for training from scratch and allowing us to focus computational resources on adapting the model to the specific nuances of the text-to-structure task. The autoregressive nature is well-suited for generating sequential structured outputs like JSON strings token by token.

We adopt SFT due to its effectiveness in teaching pre-trained models to follow instructions and generate outputs in specific formats when provided with high-quality input-output examples (Ouyang et al., 2022). Our task perfectly fits this paradigm: we have source text (x_i), a clear instruction embedded within the schema specification (f_i), and a desired target output (y_i). The training dataset $\mathcal{D} = \{(x_i, f_i, y_i)\}$, synthesized as described in § 4, forms the basis for SFT. During fine-tuning, each triple (x_i, f_i, y_i) from dataset $\mathcal{D}$ is formatted into an instruction-following prompt. A typical format concatenates the task instruction, the target schema, and the input text into a single input sequence for the model. We then apply the standard causal language modeling loss on the target structured output y_i only, masking out the input prompt tokens from loss calculation.

Unlike prior works (Jiang et al., 2024; Agarwal et al., 2025, *inter alia*), our approach is task-agnostic – SLOT decouples formatting from task-specific generation, which ensures compatibility with any LLM output for any task without modification, while avoiding generation quality degradation from constrained decoding. This positions SLOT as a universal adapter between general-purpose LLMs and structured data applications.

6 Experiments

6.1 Experiment Setting

Methods Compared. We evaluate SLOT against state-of-the-art methods for structured output generation, which fall into two categories. First, we use proprietary LLMs, specifically Claude-3.5-Haiku and Claude-3.5-Sonnet (Anthropic, 2024a,b) with

SLOTBENCH Test Splits	Schema Accuracy (%)						Content Similarity (%)					
	Web-NLG	E2E-NLG	Wiki-Bio	ToTTo	GitHub Issues	**Avg**	Web-NLG	E2E-NLG	Wiki-Bio	ToTTo	GitHub Issues	**Avg**
LLM Baselines												
Claude-3.5-Haiku												
+Prompting	50.3	98.9	99.3	99.0	97.6	89.0	40.5	90.4	86.8	96.4	89.4	80.7
Claude-3.5-Sonnet												
+Prompting	11.8	77.0	97.8	91.7	95.3	74.7	9.4	92.8	85.4	91.1	90.9	73.9
Qwen-2.5-14B												
+Prompting	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.1	0.0
+OT	100	100	100	99.8	76.7	95.3	79.3	89.8	86.7	96.8	70.4	84.6
+XG	100	100	100	99.4	97.1	99.4	53.5	86.6	86.7	96.8	89.4	82.6
Qwen-2.5-32B												
+Prompting	0.0	0.0	0.0	0.0	0.5	0.1	0.0	0.0	0.0	0.0	0.5	0.1
+OT	100	100	100	99.6	76.4	95.2	80.7	93.9	91.1	97.6	70.2	86.7
+XG	100	100	100	99.4	97.1	99.3	80.6	93.8	91.0	97.5	89.6	90.5
Llama-3.3-70B												
+Prompting	3.2	54.1	32.6	28.8	0.9	23.9	2.6	48.5	30.7	28.4	0.9	22.2
+OT	100	100	100	99.0	78.0	95.4	80.2	85.3	90.6	97.4	70.3	84.8
+XG	100	100	100	98.8	99.1	**99.6**	80.3	85.7	91.0	97.4	91.0	89.1
SLOT												
Llama-3.2-1B												
+Prompting	0.0	0.0	42.2	2.1	3.5	9.6	0.1	0	32.0	1.5	1.7	7.1
+SFT	97.7	100	99.4	95.4	52.2	88.9	79.8	98.2	94.3	90.0	46.2	81.7
+OT	100	100	99.9	97.9	56.9	90.9	72.2	76.1	85.6	93.7	51.3	75.8
+SFT & OT	100	99.5	100	100	15.4	83.0	83.3	94.6	93.9	95.7	13.4	76.2
+XG	100	100	99.8	91.7	78.1	93.9	68.6	76.3	83.7	90.3	73.4	78.4
+SFT & XG	100	100	100	100	81.2	96.2	82.3	98.2	94.3	95.3	78.1	89.6
Llama-3.2-3B												
+Prompting	0.0	0.0	1.5	0.0	0.1	0.3	0.0	0.0	0.0	0.0	0.0	0.0
+SFT	99.8	99.7	100	98.6	75.6	94.7	85.3	98.1	95.9	94.7	69.4	88.7
+OT	100	100	100	96.5	56.9	90.7	74.3	76.1	87.4	95.3	56.3	77.9
+SFT & OT	99.3	99.8	99.4	99.0	36.9	86.9	84.9	96.0	94.4	95.5	34.7	81.1
+XG	100	100	99.6	98.3	84.0	96.4	67.0	77.5	84.0	94.4	77.7	80.1
+SFT & XG	100	100	100	99.8	91.5	98.2	85.5	98.1	95.6	96.7	86.1	92.4
Mistral-7B-v0.2												
+Prompting	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
+SFT	99.5	100	100	98.3	93.1	98.2	87.0	98.7	96.2	96.7	85.9	92.9
+OT	100	100	99.9	88.6	60.7	89.8	73.8	83.0	86.3	87.6	58.4	77.8
+SFT & OT	100	100	99.9	97.9	70.9	93.7	86.6	94.9	94.7	96.0	64.7	87.4
+XG	100	100	99.8	88.2	73.1	92.2	74.0	82.9	85.9	87.9	70.2	80.2
+SFT & XG	100	100	100	99.8	97.5	**99.5**	87.4	98.7	96.3	98.0	89.7	**94.0**

Table 2: Average percentage schema accuracy and content similarity of different base and fine-tuned LLMs. OT denotes Outlines, XG denotes XGrammar.

standard prompting to establish strong baselines. Second, we experiment with open-weight LLMs with constrained decoding, including Qwen-2.5 (14B, 32B) (Qwen et al., 2025) and Llama-3.3-70B (Grattafiori et al., 2024) using two leading constrained decoding frameworks: Outlines (Willard and Louf, 2023) and XGrammar (Dong et al., 2024). We also include direct prompting results for reference in Appx. F.1.

We employ three lightweight language models in their `Instruct` versions: Llama-3.2 (1B / 3B) (Grattafiori et al., 2024), and Mistral-7B-v0.2 (Jiang et al., 2023a). We also evaluate SLOT combined with constrained decoding methods in the inference time, which provides an opportunity to understand how model-based and rule-based approaches can complement each other.

Training Setup. For SLOT training, we employ LoRA (Hu et al., 2022) fine-tuning with a standard language modeling objective on completions. We selected checkpoints based on a balanced metric combining schema accuracy and content similarity, evaluated on a validation subset of 650 instances from WikiBio. For inference, We used vLLM (Kwon et al., 2023) to optimize generation throughput. The complete hyperparameters for both training and inference are detailed in Appx Table 5.

6.2 Results and Analysis

Schema Accuracy and Content Similarity. Shown in Table 2, proprietary models establish competitive baseline performance, with Claude-3.5-Sonnet achieving 74.7% schema accuracy and 73.9% content similarity on average. However, even these frontier models fall short of the reliabil-

ity required for applications like function calling, suggesting potential limitations in pure prompting approaches. In contrast, small open-weight models perform extremely poorly via direct prompting – Llama-3.2-1B, 3B, and Mistral-7B-v0.2 achieve near-zero schema accuracy and content similarity across most datasets.

SLOT significantly improves performance via supervised fine-tuning. The 1B parameter model achieves 88.9% overall schema accuracy (comparable to Claude-3.5-Haiku) and 81.7% content similarity (exceeding Claude-3.5-Sonnet's 73.9%). More impressively, Mistral-7B-v0.2 reaches 98.2% schema accuracy and 92.9% content similarity, surpassing all proprietary baselines on both metrics. These results demonstrate that specialized training can outperform general capabilities of much larger models for structured generation tasks. Dataset complexity significantly impacts performance across all methods. The GitHub Issues dataset is the most challenging across all configurations, given the prevalence of deeply nested structures and technical content (Fig. 2). Yet here SLOT shows remarkable gains – Mistral-7B improves from 0% to 93.1% schema accuracy. Conversely, E2E NLG consistenly yields the highest content similarity across models, likely due to its simpler descriptions with well-defined attributes.

Synergy Between Fine-Tuning and Constrained Decoding. We observe that constrained decoding methods reveal interesting algorithmic tradeoffs between structure and semantics. XGrammar performs markedly better on complex nested structures (achieving 97.1% schema accuracy on GitHub issues with Qwen-2.5-32B compared to Outlines' 76.4%). This advantage stems from XGrammar's adaptive token caching strategy that efficiently handles context-independent tokens and context expansion. The byte-level pushdown automaton approach with persistent execution stack enables efficient validation of complex structures, while still allowing for semantic coherence during generation. Outlines offers efficiency advantage through finite-state indexing for simpler structures, but its traditional pushdown automaton faces scalability challenges with deeply nested structures – resulting timeout errors with complex schemas.

Most significantly, combining SLOT with constrained decoding creates a powerful synergy. Mistral-7B-v0.2 with SFT + XGrammar achieves near-perfect results: 99.5% schema accuracy and 94.0% content similarity. Even the 1B parameter model reaches impressive performance (96.2% schema accuracy, 89.6% content similarity) with this combined approach, demonstrating that structured output generation does not necessarily require massive model sizes, but rather targeted training and appropriate constraints. This synergy occurs because SFT teaches the model to inherently produce well-structured and semantically appropriate outputs, while constrained decoding provides a complementary guarantee of structural validity, which is particularly pronounced in smaller models. Detailed error analysis is in Appx. E.2.

Impact of Training Data. Our detailed analysis of training data contributions (Table 3, Table 4) reveals that the diversity and complexity of synthetic data plays a crucial role in SLOT's success. For instance, with Llama-3.2-1B, public data alone achieves 63.1% schema accuracy and 68.3% content similarity. Synthetic data alone performs substantially better for schema accuracy, reaching 89.6%, while achieving 74.4% content similarity. The combined dataset demonstrates complementary benefits, achieving 89.0% schema accuracy and 81.7% content similarity, with particular gains in content preservation. This pattern holds across model sizes, with the strongest results typically coming from the combined dataset.

7 Related Works

To enable the model generation to conform to a target JSON schema, existing works mainly opted for three approaches, which have their strengths and weaknesses. Direct prompting, although straightforward, often yields lower performance (Tam et al., 2024b). Constrained decoding methods, exemplified by works like (Dong et al., 2024; OpenAI, 2024), does not rely on dedicated post-training, but only supports limited types of JSON schemas and may underperform training-based methods. Post-training generally improves performance compared to constrained decoding, but also requires access to model weights. The training itself adds complexity, and is not scalable for adaptation to new LLMs. More details can be found in Appx. A.

8 Conclusion

We introduced SLOT, a model-agnostic and task-agnostic framework for generating structured outputs from LLMs. While our solution can be theoretically applied to any specified schema, we focused

on JSON as our primary use case and developed two objective metrics – Schema Accuracy and Content Similarity – to evaluate the quality of generated JSON outputs. Through a comprehensive data pipeline, we synthesized and validated training data spanning diverse text lengths, styles, industry verticals, and JSON complexity levels. We demonstrate that with supervised fine-tuning, lightweight open-weight language models can outperform larger proprietary models and exhibit strong generalizability. This finding underscores the efficacy of training-based approaches for structured output generation, potentially improving the accessibility of structured data in real-world applications. Future work will focus on investigating more sophisticated data synthesis strategies and exploring advanced post-training methodologies, including preference tuning and reinforcement learning.

Limitations

Our data synthesis approach primarily focused on cases where the input schema and gold structured output capture the majority of entities and relationships present in the input text. We did not extensively explore scenarios where only a subset of the textual information needs to be structured, or where complex filtering or transformation of the input content is required. We conducted latency analysis but did not reported in detail since the results can be hardware-dependent and may not generalize across different computing environments. Finally, we note that an ideal evaluation of SLOT would involve end-to-end testing, where the input text is generated by a preceding LLM instead of using arbitrary texts that emulate LLM output. These limitations suggest potential areas for future work in handling more diverse LLM output and structuring requirements.

References

Bhavik Agarwal, Ishan Joshi, and Viktoria Rojkova. 2025. Think inside the json: Reinforcement strategy for strict llm schema adherence. *Preprint*, arXiv:2502.14905.

Anthropic. 2024a. Claude 3.5 haiku.

Anthropic. 2024b. Claude 3.5 sonnet.

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. 2024. Guiding llms the right way: Fast, non-invasive constrained generation. *Preprint*, arXiv:2403.06988.

Harrison Chase. 2022. LangChain.

Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. 2024. Xgrammar: Flexible and efficient structured generation engine for large language models. *Preprint*, arXiv:2411.15100.

Gemini. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Saibo Geng, Hudson Cooper, Michał Moskal, Samuel Jenkins, Julian Berman, Nathan Ranchin, Robert West, Eric Horvitz, and Harsha Nori. 2025. Jsonschemabench: A rigorous benchmark of structured outputs for language models. *Preprint*, arXiv:2501.10868.

Georgi Gerganov. 2023. llama.cpp.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

GuidanceAI. 2023. Guidance.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Albert Q Jiang, A Sablayrolles, A Mensch, C Bamford, D Singh Chaplot, Ddl Casas, F Bressand, G Lengyel, G Lample, L Saulnier, et al. 2023a. Mistral 7b. arxiv. *arXiv preprint arXiv:2310.06825*, 10.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji rong Wen. 2023b. Structgpt: A general framework for large language model to reason over structured data. In *Conference on Empirical Methods in Natural Language Processing*.

Xin Jiang, Xiang Li, Wenjia Ma, Xuezhi Fang, Yiqun Yao, Naitong Yu, Xuying Meng, Peng Han, Jing Li, Aixin Sun, and Yequan Wang. 2024. Sketch: A toolkit for streamlining llm operations. *Preprint*, arXiv:2409.03346.

Atharv Kulkarni and Vivek Srikumar. 2025. Reinforcing code generation: Improving text-to-sql with execution-based learning. *arXiv preprint arXiv:2506.06093*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Rémi Lebret, David Grangier, and Michael Auli. 2016. Neural text generation from structured data with application to the biography domain. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1203–1213, Austin, Texas. Association for Computational Linguistics.

Diya Li, Yue Zhao, Zhifang Wang, Calvin Jung, and Zhe Zhang. 2024. Large language model-driven structured output: A comprehensive benchmark and spatial data generation framework. *ISPRS International Journal of Geo-Information*, 13(11).

Jason Liu. 2022a. instructor.

Jerry Liu. 2022b. LlamaIndex.

Michael Xieyang Liu, Frederick Liu, Alexander J. Fiannaca, Terry Koo, Lucas Dixon, Michael Terry, and Carrie J. Cai. 2024. "we need structured output": Towards user-centered constraints on large language model output. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, CHI '24, page 1–9. ACM.

Shu Liu, Sumanth Hegde, Shiyi Cao, Alan Zhu, Dacheng Li, Tyler Griggs, Eric Tang, Akshay Malik, Kourosh Hakhamaneshi, Richard Liaw, Philipp Moritz, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. Skyrl-sql: Matching gpt-4o and o4-mini on text2sql with multi-turn rl.

Xiaoxia Liu, Jingyi Wang, Jun Sun, Xiaohan Yuan, Guoliang Dong, Peng Di, Wenhai Wang, and Dongxia Wang. 2023. Prompting frameworks for large language models: A survey. *ArXiv*, abs/2311.12785.

Ya-Ting Lu, Haolun Li, Xin Cong, Zhong Zhang, Yesai Wu, Yankai Lin, Zhiyuan Liu, Fangming Liu, and Maosong Sun. 2025. Learning to generate structured output with schema reinforcement learning. *ArXiv*, abs/2502.18878.

OpenAI. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

OpenAI. 2024. Introducing structured outputs in the api.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Ankur P Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. 2020. ToTTo: A controlled table-to-text generation dataset. In *Proceedings of EMNLP*.

Yevgeniy Puzikov and Iryna Gurevych. 2018. E2E NLG challenge: Neural models vs. templates. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 463–471, Tilburg University, The Netherlands. Association for Computational Linguistics.

Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *Preprint*, arXiv:1908.10084.

Connor Shorten, Charles Pierse, Thomas Benjamin Smith, Erika Cardenas, Akanksha Sharma, John Trengrove, and Bob van Luijt. 2024. Structuredrag: Json response formatting with large language models. *Preprint*, arXiv:2408.11061.

Zhi Rui Tam, Cheng-Kuang Wu, Yi-Lin Tsai, Chieh-Yen Lin, Hung-yi Lee, and Yun-Nung Chen. 2024a. Let me speak freely? a study on the impact of format restrictions on large language model performance. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1218–1236, Miami, Florida, US. Association for Computational Linguistics.

Zhi Rui Tam, Cheng-Kuang Wu, Yi-Lin Tsai, Chieh-Yen Lin, Hung yi Lee, and Yun-Nung Chen. 2024b. Let me speak freely? a study on the impact of format restrictions on performance of large language models. *Preprint*, arXiv:2408.02442.

Shuhe Wang, Xiaofei Sun, Xiaoya Li, Rongbin Ouyang, Fei Wu, Tianwei Zhang, Jiwei Li, and Guoyin Wang. 2023a. Gpt-ner: Named entity recognition via large language models. *Preprint*, arXiv:2304.10428.

Xiao Wang, Weikang Zhou, Can Zu, Han Xia, Tianze Chen, Yuansen Zhang, Rui Zheng, Junjie Ye, Qi Zhang, Tao Gui, Jihua Kang, Jingsheng Yang, Siyuan Li, and Chunsai Du. 2023b. Instructuie: Multi-task instruction tuning for unified information extraction. *Preprint*, arXiv:2304.08085.

Brandon T. Willard and Rémi Louf. 2023. Efficient guided generation for large language models. *Preprint*, arXiv:2307.09702.

Congying Xia, Chen Xing, Jiangshu Du, Xinyi Yang, Yihao Feng, Ran Xu, Wenpeng Yin, and Caiming Xiong. 2024. Fofo: A benchmark to evaluate llms' format-following capability. *Preprint*, arXiv:2402.18667.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. Bertscore: Evaluating text generation with bert. In *International Conference on Learning Representations*.

Giulio Zhou and Gerasimos Lampouras. 2020. WebNLG challenge 2020: Language agnostic delexicalisation for multilingual RDF-to-text generation. In *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*, pages 186–191, Dublin, Ireland (Virtual). Association for Computational Linguistics.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *Preprint*, arXiv:2311.07911.

A Related Work Details

Direct Prompting. The most straightforward approach involves explicitly instructing LLMs to generate structured outputs, supported by frameworks like LangChain (Chase, 2022), LlamaIndex (Liu, 2022b), and Instructor (Liu, 2022a). However, Tam et al. (2024b) demonstrated that this approach often produces invalid structures and can lower performance on reasoning-intensive tasks due to the additional cognitive load of format adherence.

Constrained Decoding. To guarantee structural validity, constrained decoding techniques (Beurer-Kellner et al., 2024; Liu et al., 2024) guide the generation process using formal grammars. Tools like llama.cpp (Gerganov, 2023), Guidance (GuidanceAI, 2023), XGrammar (Dong et al., 2024), and Outlines (Willard and Louf, 2023) implement this approach. While effective at ensuring valid outputs, these methods typically increase inference latency and may impact task performance due to the restrictive nature of the constraints.

Post-Training Approaches. Task-specific fine-tuning has shown success in structured output generation, particularly for tasks like named-entity recognition (Wang et al., 2023a) and information extraction (Wang et al., 2023b). However, this approach requires separate training efforts for each task-format combination, limiting its scalability.

Hybrid Solutions. Recent work has explored combining multiple techniques to balance reliability and performance. Jiang et al. (2024) demonstrated success in fine-tuning with diverse schema datasets, while Agarwal et al. (2025) combined reinforcement learning with supervised fine-tuning for schema-constrained tasks.

As illustrated in Appx Fig. 4, these existing approaches are typically tied to specific LLMs, requiring reimplementation for each new model. Additionally, previously launched LLMs cannot benefit from these improvements without significant modification. Our proposed SLOT addresses these limitations by decoupling output formatting from the natural language task, offering a model-agnostic solution that maintains task performance while ensuring structural validity.

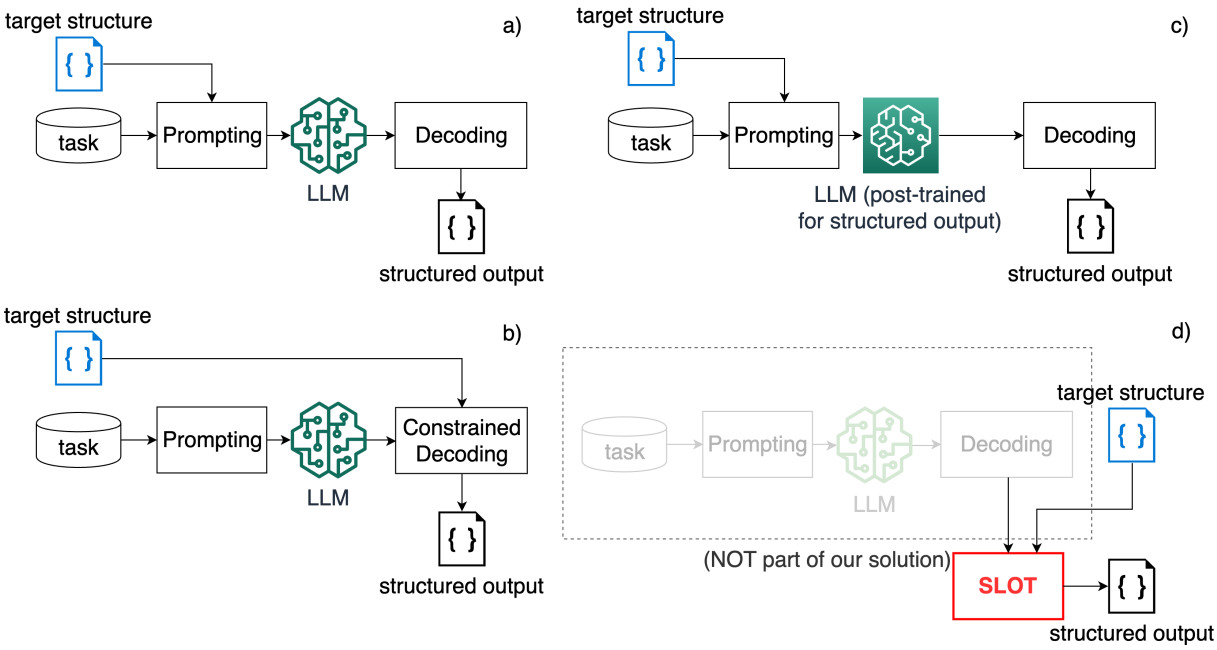

Figure 4: Approaches for LLM structured outputs. a) prompting LLM form structured output, b) constrained decoding, c) post-training, and d) SLOT.

B Detailed Experiment Results

We present the full experiment results in Table 3 and Table 4.

LLM & Settings	SFT dataset	Schema Accuracy (%)					
		WebNLG	E2E NLG	WikiBio	ToTTo	GitHub	Avg
Llama 3.2 11B	-	0.47	0.13	45.88	60.58	15.18	24.45
Llama 3.2 1B	-	0	0	42.20	2.07	3.52	9.56
	Public	99.28	91.86	99.44	13.49	11.26	63.07
	Synthetic	99.42	99.96	98.4	95.23	54.87	89.58
	Public & Synthetic	97.73	100	99.44	95.44	52.16	88.95
Llama 3.2 1B + Outlines	-	100	100	99.88	97.93	56.88	90.94
	Public & Synthetic	100	99.53	99.96	100	15.38	82.97
Llama 3.2 1B + XGrammar	-	100	100	99.8	91.7	78.09	93.92
	Public & Synthetic	100	100	100	100	81.21	96.24
Llama 3.2 3B	-	0	0	1.48	0	0.10	0.32
	Public	99.46	100	99.88	28.22	34.07	72.33
	Synthetic	100	99.79	98.92	94.81	82.31	95.17
	Public & Synthetic	99.75	99.7	99.96	98.55	75.58	94.71
Llama 3.2 3B + Outlines	-	100	100	100	96.47	56.88	90.67
	Public & Synthetic	99.28	99.83	99.44	98.96	36.88	86.88
Llama 3.2 3B + XGrammar	-	100	100	99.6	98.34	84.02	96.39
	Public & Synthetic	100	100	99.96	99.79	91.46	98.24
Mistral 7B (v0.2)	-	0	0	0	0	0	0
	Public	100	100	99.6	11.83	20.7	66.43
	Synthetic	99.03	99.96	97.88	98.76	88.14	96.75
	Public & Synthetic	99.5	100	99.96	98.34	93.07	98.17
Mistral 7B (v0.2) + Outlines	-	100	100	99.92	88.59	60.70	89.84
	Public & Synthetic	100	100	99.92	97.93	70.85	93.74
Mistral 7B (v0.2) + XGrammar	-	100	100	99.8	88.17	73.07	92.21
	Public & Synthetic	100	100	100	99.79	97.49	99.46

Table 3: Schema accuracy of different base and fine-tuned LLMs

LLM & Settings	SFT dataset	Content Similarity (%)					
		WebNLG	E2E NLG	WikiBio	ToTTo	GitHub	Avg
Llama 3.2 11B	-	0.42	0.20	35.63	59.30	9.90	21.09
Llama 3.2 1B	-	0.08	0	32.03	1.54	1.69	7.07
	Public	80.59	89.91	94.35	51.13	25.48	68.29
	Synthetic	74.46	75.51	82.78	90.68	48.77	74.44
	Public + Synthetic	79.84	98.15	94.28	89.95	46.24	81.69
Llama 3.2 1B + Outlines	-	72.17	76.13	85.61	93.68	51.32	75.78
	Public + Synthetic	83.34	94.62	93.87	95.73	13.43	76.20
Llama 3.2 1B + XGrammar	-	68.58	76.29	83.67	90.25	73.38	78.43
	Public + Synthetic	82.32	98.17	94.31	95.32	78.09	89.64
Llama 3.2 3B	-	0	0	0.02	0	0	0
	Public	84.87	98.3	95.5	79.48	50.99	81.83
	Synthetic	78.42	81.51	85.56	91.35	74.86	82.34
	Public + Synthetic	85.32	98.08	95.88	94.73	69.43	88.69
Llama 3.2 3B + Outlines	-	74.3	76.06	87.36	95.26	56.25	77.85
	Public + Synthetic	84.93	96.01	94.4	95.45	34.67	81.09
Llama 3.2 3B + XGrammar	-	67	77.49	84.02	94.42	77.65	80.12
	Public + Synthetic	85.53	98.06	95.63	96.73	86.14	92.42
Mistral 7B (v0.2)	-	0	0	0	0	0	0
	Public	85	97.79	95.86	37.25	25.49	68.28
	Synthetic	77.91	85.4	86	96.45	81.66	85.48
	Public + Synthetic	87	98.72	96.24	96.73	85.94	92.93
Mistral 7B (v0.2) + Outlines	-	73.76	82.96	86.28	87.61	58.44	77.81
	Public + Synthetic	86.6	94.88	94.69	96	64.66	87.37
Mistral 7B (v0.2) + XGrammar	-	73.95	82.93	85.93	87.92	70.16	80.18
	Public + Synthetic	87.42	98.73	96.25	98.01	89.71	94.02

Table 4: Content similarity of different base and fine-tuned LLMs

C Evaluation Details

C.1 Existing evaluation metrics for JSON structured outputs from LLMs

Task performance. This approach treats structured output as an additional requirement alongside the original task (e.g., reasoning or information extraction). Researchers measure the impact of this requirement by comparing performance metrics (such as answer accuracy) with and without structured output constraints, or across different required formats. However, this evaluation method is task-specific and measures the LLM's ability to provide structured outputs for particular tasks rather than its general

formatting capability (e.g., in Tam et al. (2024a), Beurer-Kellner et al. (2024), Jiang et al. (2024), Shorten et al. (2024), Geng et al. (2025) etc.).

Latency or speed-up. Structured output requirements demand additional processing during next-token generation and/or constrained decoding. Works focusing on algorithmic efficiency often measure either latency (compared to baseline without structured output requirements) or speed-up (compared to other structured output tools) (e.g., in Willard and Louf (2023), Geng et al. (2025) etc.).

JSON validity. Many studies evaluate whether the LLM's responses are valid JSON, e.g. Zhou et al. (2023), Beurer-Kellner et al. (2024), Jiang et al. (2024), Agarwal et al. (2025). Some examine schema compliance (e.g. Geng et al. (2025)) or value accuracy using exact match (Agarwal et al. (2025)) or edit distance (Li et al. (2024)). While some benchmarks like Xia et al. (2024) relied on LLM-as-a-Judge (LLMaaJ) to evaluate the generated structured output's format and/or content validity, this approach's non-deterministic nature makes it unsuitable for industrial applications involving LLM agents or function calling.

C.2 Details on content similarity metric

The content similarity score in this work is defined as the harmonic mean of soft-precision and soft-recall:

$$sim_C(y, y') = 2 \times \frac{sim_P(y, y') \cdot sim_R(y, y')}{sim_P(y, y') + sim_R(y, y')}$$

There are several rationales behind the design choices of our content similarity. First, if we simply concatenate all the values into one large string, not only the order of concatenation is not trivial, such metric could be dominated by values of very long strings. Consider:

Gold:

```
{
  "error": "ValueError: This is a sample error
     message",
  "traceback": "Traceback (most recent call
    last): ......"
  // a very long error message
}
```

Prediction:

```
{
  "error": "ValueError: This is a sample error
     message",
  "traceback": ""
}
```

In contrary, the pairwise SBERT works even when the numbers of entities in ground truth and prediction are different, and also weighs each value uniformly disregarding its length.

Second, requiring exact key matches, rather than using edit distances as in Li et al. (2024), helps identify cases where values are incorrectly swapped between similar keys. Here is one example we observed from the E2E NLG dataset:

Text:

```
"Near The Rice Boat you can visit coffee shop called Giraffe."
```

Gold:

```
{
  "name": "Giraffe",
  "eatType": "coffee shop",
  "near": "The Rice Boat"
}
```

Prediction:

```
{
  "name": "The Rice Boat",
  "eatType": "coffee shop",
  "near": "Giraffe"
}
```

In such cases, particularly for agent chaining and function calling applications, attributing values to incorrect keys represents a failure to follow user intent and should be penalized.

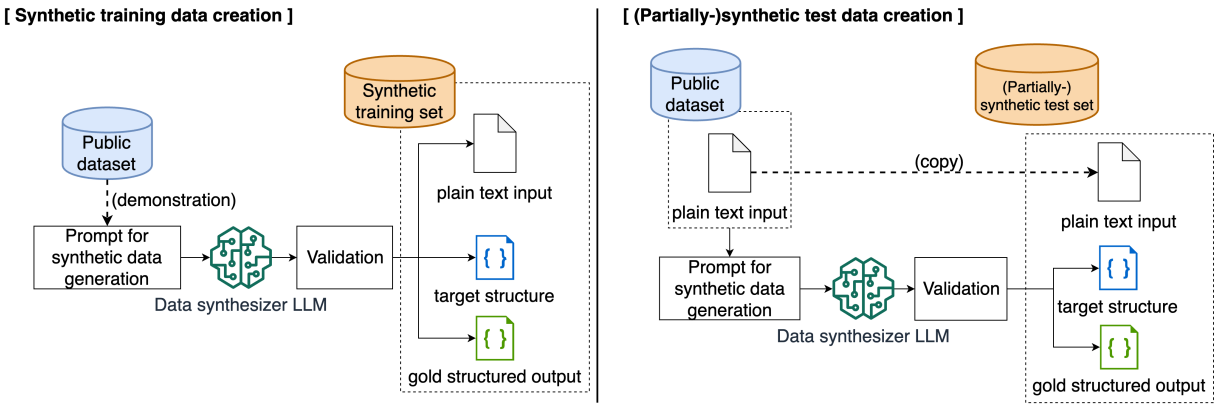

Figure 5: Data curation pipeline for synthetic training data and partially synthetic test data.

D Dataset Details

We benchmark SPOT on five public datasets, recast into a text-to-JSON formulation.

D.1 Dataset Curation Details and Examples

WebNLG [License: CC BY-NC-SA 4.0] (DBpedia Triples → Text) In the original dataset, each instance contains data/text pairs where the data is a set of triples extracted from DBpedia, and the text is a verbalisation of these triples. We use the WebNLG v3.0 dataset of 25K RDF triple sets each paired with 3–7 verbalizations. We converted the triplets into a JSON which described the text / subject.

```
{
  "input_text": "The album 1969: The Velvet Underground Live is preceded by the Velvet Underground album
       Squeeze, which was followed by The Quine Tapes.",
  "json_schema": {
    "type": "object",
    "properties": {
      "category": {
        "type": "string"
      },
      "subject": {
        "type": "string"
      },
      "properties": {
        "type": "object",
        "properties": {
          "precededBy": {
            "type": "string"
          },
          "followedBy": {
            "type": "string"
          }
        },
        "required": ["precededBy", "followedBy"],
        "additionalProperties": false
      }
    },
    "required": ["category", "subject", "properties"],
    "additionalProperties": false
  },
  "gold": {
    "category": "MusicalWork",
    "subject": "Bootleg Series Volume 1: The Quine Tapes",
    "properties": {
      "precededBy": "Squeeze (The Velvet Underground album)",
      "followedBy": "1969: The Velvet Underground Live"
    }
  }
}
```

E2E NLG [License: CC-BY-4.0] (Restaurant Attributes → Text) The original dataset is an English benchmark dataset for data-to-text models that verbalize a set of 2-9 key-value attribute pairs in the restaurant domain. Contains restaurant domain descriptions paired with attribute-value structures. This is also one of the dataset that was reversed engineered to text-to-json.

```
{
```

```
  "input_text": "The Blue Spice is a pub located by the riverside, near the Rainbow Vegetarian Cafe. It is
      not child friendly.",
  "json_schema": {
    "type": "object",
    "properties": {
      "name": {
        "type": "string"
      },
      "eatType": {
        "type": "string"
      },
      "area": {
        "type": "string"
      },
      "familyFriendly": {
        "type": "string"
      },
      "near": {
        "type": "string"
      }
    },
    "required": ["name", "eatType", "area", "familyFriendly", "near"],
    "additionalProperties": false
  },
  "gold": {
    "name": "Blue Spice",
    "eatType": "pub",
    "area": "riverside",
    "familyFriendly": "no",
    "near": "Rainbow Vegetarian Cafe"
  }
}
```


WikiBio [License: CC-BY-SA-3.0] (Infobox → Intro Paragraph) The original Dataset contains 728K biographies extracted from Wikipedia containing the first paragraph of the biography and the tabular infobox. This dataset was in json format but not in out desired format to json schema format. We convert the infobox key–value table into a nested JSON schema aligned with first-paragraph content. We randomly sampled 2,500 examples from its test set, providing evaluation on biographical information extraction with diverse schema complexity.


```
{
  "input_text": "miroslav popov -lrb- born 14 june 1995 in dvur kralove nad labem -rrb- is a czech grand
      prix motorcycle racer . he currently races in the fim cev moto2 championship for montaze broz racing
      team aboard a suter .",
  "json_schema": {
    "type": "object",
    "properties": {
      "birth_date": {
        "type": "string"
      },
      "name": {
        "type": "string"
      }
    },
    "required": ["birth_date", "name"],
    "additionalProperties": false
  },
  "gold": {
    "birth_date": "14 june 1995",
    "name": "miroslav popov"
  }
}
```


ToTTo [License: CC-BY-SA-3.0] (Table → Text) We sampled 500 examples from the challenge split of ToTTo, a table-to-text dataset where reference texts are annotated based on Wikipedia tables. Since the source tables often contain information beyond the referenced text, we employed LLM annotation to synthesize JSON schema and structured output with further validation and filtering to ensure the structured data strictly adhere to the text content.


```
{
  "input_text": "Cypresses are numbered B.152 and date from 1887.",
  "json_schema": {
    "type": "object",
    "properties": {
      "composition": {
        "type": "object",
        "properties": {
          "title": {"type": "string"},
```

```
          "catalog_number": {"type": "string"},
          "year": {"type": "integer"}
        },
        "required": ["title", "catalog_number", "year"]
      }
    },
    "required": ["composition"]
  },
  "gold": {
    "composition": {
      "title": "Cypresses",
      "catalog_number": "B.152",
      "year": 1887
    }
  }
}
```


HF GitHub Issues [License: Apache-2.0] (Issue Text → Structured Issue Report) We sampled 1,000 GitHub issues from Hugging Face transformers repository up to Feb 2025. For each issue, we combined the title and body text as input, then synthesized the corresponding JSON schema and structured output following the same procedure for ToTTo. The structured outputs capture diverse aspects of software issues including system / error information, reproduction steps, code snippets and expected behaviors, making it an excellent test of model's capability to handle intricate nested structures and technical input contents.


```
{
  "input_text": "## qwen2_5_vl processor padding side is wrong.\n### System Info\n\n![Image](https://github.
      com/user-attachments/assets/6ecbc96d-d34a-4164-903a-0ef65ea65fb0)\n\n![Image](https://github.com/user
      -attachments/assets/e92f3446-3e81-4887-9f9c-0b5cb3047683)\n\n![Image](https://github.com/user-
      attachments/assets/8bef88c1-40ba-413b-8444-d018c9691787)\nthe padding side should be left as qwen2 vl
       do .\n\n### Information\n\n- [ ] The official example scripts\n- [x] My own modified scripts\n\n###
      Tasks\n\n- [ ] An officially supported task in the `examples` folder (such as GLUE/SQuAD, ...)\n- [x]
       My own task or dataset (give details below)\n\n### Reproduction\n\nrun conditional generation using
      qwen2_5_vl  using flash attention 2 .\n\n### Expected behavior\n\n![Image](https://github.com/user-
      attachments/assets/e92f3446-3e81-4887-9f9c-0b5cb3047683)\n",
  "json_schema": {
    "type": "object",
    "properties": {
      "issue_title": {
        "type": "string"
      },
      "system_info": {
        "type": "array",
        "items": {
          "type": "string"
        }
      },
      "information": {
        "type": "object",
        "properties": {
          "official_example_scripts": {
            "type": "boolean"
          },
          "modified_scripts": {
            "type": "boolean"
          }
        }
      },
      "tasks": {
        "type": "object",
        "properties": {
          "official_task": {
            "type": "boolean"
          },
          "own_task": {
            "type": "boolean"
          }
        }
      },
      "reproduction": {
        "type": "string"
      },
      "expected_behavior": {
        "type": "string"
      }
    }
  },
  "gold": {
    "issue_title": "qwen2_5_vl processor padding side is wrong.",
    "system_info": [
      "https://github.com/user-attachments/assets/6ecbc96d-d34a-4164-903a-0ef65ea65fb0",
      "https://github.com/user-attachments/assets/e92f3446-3e81-4887-9f9c-0b5cb3047683",
      "https://github.com/user-attachments/assets/8bef88c1-40ba-413b-8444-d018c9691787"
```

```
    ],
    "information": {
      "official_example_scripts": false,
      "modified_scripts": true
    },
    "tasks": {
      "official_task": false,
      "own_task": true
    },
    "reproduction": "run conditional generation using qwen2_5_vl using flash attention 2.",
    "expected_behavior": "https://github.com/user-attachments/assets/e92f3446-3e81-4887-9f9c-0b5cb3047683"
  }
}
```


D.2 Training Data Diversity Dimensions

Industry Vertical. Defines the domain context spanning 30 categories such as "Healthcare", "Financial Services", etc.

JSON Complexity. Specifies 10 levels of structural complexity from "Basic" (3-5 key-value pairs) to "Comprehensive" (multiple nested objects)

Text Length Style. Covering 10 input text length styles from "Brief Snippets" (15-30 words) to "Extended" (200-300 words)

Genre Includes 40 text styles (e.g., from news articles, technical reports, etc.)

Text Type. Specifies 10 types of text structures that are widely present in LLM responses (e.g., "Bullet points", "Code snippets", "Dialogue")

D.3 Json Complexity Dimensions

D.3.1 Depth (d)

Maximum nesting level of the JSON structure, calculated recursively:

$$d = \max_{v \in \text{values}} \text{depth}(v)$$

where depth is calculated as:

$$\text{depth}(v) = \begin{cases} 0 & \text{if } v \text{ is a primitive value} \\ 1 + \max_{c \in \text{children}(v)} \text{depth}(c) & \text{if } v \text{ is an object or array} \end{cases}$$

D.3.2 Number of Keys (k)

Total count of keys at all levels in the JSON structure, calculated as:

$$k = \sum_{o \in \text{objects}} |\text{keys}(o)|$$

where $|\text{keys}(o)|$ is the number of keys in object o, summed across all nested objects.

D.3.3 Size in Bytes (s)

Size of the JSON string when encoded in UTF-8, calculated as:

$$s = |\text{UTF-8}(\text{json.dumps}(\text{obj}))|$$

where $|\text{UTF-8}(x)|$ represents the length of string x when encoded in UTF-8. This has a small weight to avoid dominating the score.

D.3.4 Number of Elements (e)

Total count of all values in the JSON structure, including primitive values, objects, and arrays. Incremented during traversal:

$$e = |\{v : v \text{ is any value in the JSON structure}\}|$$

D.3.5 Cyclomatic Complexity (c)

Measure of structural decision points (branches) in the JSON, calculated as:

$$c = \sum_{o \in \text{objects}} |\text{keys}(o)| + \sum_{a \in \text{arrays}} [|a| > 0]$$

where:

- $|\text{keys}(o)|$ is the number of keys in object o
- $[|a| > 0]$ is 1 if array a is non-empty, 0 otherwise

D.3.6 Schema Complexity (sc)

Complexity of the JSON structure considering types and nesting. For a value v:

$$sc(v) = \begin{cases} 1 + \frac{\min(|v|,100)}{10} & \text{if } v \text{ is a string} \\ 1 + |\text{keys}(v)| + \sum_{k \in \text{keys}(v)} sc(v[k]) & \text{if } v \text{ is an object} \\ 1 + |v| + \begin{cases} sc(v[0]) & \text{if homogeneous} \\ \sum_i sc(v[i]) & \text{if heterogeneous} \end{cases} & \text{if } v \text{ is an array} \\ 1 & \text{otherwise} \end{cases}$$

where:

- $|v|$ is the length of a string or array, or number of keys in an object
- An array is homogeneous if all items have the same type
- For homogeneous arrays, complexity is calculated using the first item's schema

D.3.7 Content Complexity (cc)

Complexity of the actual data values. For a value v:

$$cc(v) = \begin{cases} \sum_{k \in \text{keys}(v)} cc(v[k]) & \text{if } v \text{ is an object} \\ \sum_i cc(v[i]) & \text{if } v \text{ is an array} \\ lf + ef + sf + tf & \text{if } v \text{ is a string} \\ \min(\frac{|\text{digits}(v)|}{5}, 1) & \text{if } v \text{ is a number} \\ 0 & \text{otherwise} \end{cases}$$

where for strings:

- $lf = \min(\frac{|v|}{20}, 5)$ (length factor)
- $ef = \frac{|\text{unique}(v)|}{|v|} \cdot 3$ (entropy factor)
- $sf = \min(\frac{|\text{special}(v)|}{|v|} \cdot 5, 3)$ (special characters factor)
- $tf = 2$ if contains code-like patterns, 0 otherwise (structure factor)
- $\text{special}(v)$ counts non-alphanumeric, non-space characters

E Experiment Details

E.1 Training and Inference Hyperparameters

Hyperparameters and configurations used during training and inference are presented in Table 5.

Training Parameters		Inference Parameters	
Learning rate	1e-5	Maximum generation length	2048
Training epochs	2	Temperature	0
Per-device batch size	2	Top-p	0.9
Gradient accumulation	2	Top-k	50
Context length	16,384	Generation timeout	60 s
Optimizer	paged_adamw_32bit	GPU memory utilization	0.8
Maximum gradient norm	0.1	Tensor parallel size	1
LoRA Configuration			
Rank (r)	32		
Alpha	64		
Dropout	0.05		

Table 5: Hyperparameters used in our experiments

E.2 Error Analysis

We conducted a detailed analysis of the errors made by different model configurations to better understand their failure modes. Non-fine-tuned models primarily failed by completely ignoring the target schema, either generating free-form text or incorrect JSON structures that did not match the requirements. This matches our expectation that base models lack the specialized knowledge needed to follow complex structural constraints without additional training or guidance.

SFT models showed more sophisticated error patterns, typically involving missing fields, incorrect field types, or hallucinatory content for complex examples. The GitHub Issues dataset accounted for the majority of errors, with failures often occurring in deeply nested structures. We observed that errors frequently appeared at deeper nesting levels (beyond 3-4 levels), suggesting that even fine-tuned models struggle to maintain structural coherence across extended hierarchical dependencies. This points to a fundamental limitation in how standard decoder-only transformer architectures represent and track deeply nested structures during generation.

Constrained decoding approaches (Outlines and XGrammar) guaranteed structural correctness for the examples they could complete but sometimes produced semantically incorrect content, particularly for ambiguous inputs or when multiple valid interpretations were possible. This semantic mismatch manifested in different ways between the two algorithms. XGrammar sometimes over-constrains the model's generative capabilities when the schema is highly structured, particularly affecting the nuanced language expression in fields with longer texts like descriptions. Outlines, with its regex-based validation, occasionally creates token selection pressures that lead to less fluent text in certain fields.

The combined SFT + constrained decoding approach yielded the fewest errors. The few remaining errors were predominantly in the GitHub Issues dataset and typically involved either timeout issues or extreme edge cases where the model struggled to extract the correct content for very complex nested structures. We identified two main error categories in this combined approach: (1) timeout failures when compiling extremely complex schemas with many optional fields and alternatives, and (2) cases where the model must make subjective decisions about how to map ambiguous content to structured fields.

This error analysis reinforces the complementary nature of SFT and constrained decoding approaches: SFT addresses semantic understanding and content organization, while constrained decoding enforces structural correctness. It also highlights an important direction for future research: developing more efficient algorithms for handling extremely complex schemas and improving models' ability to maintain coherent structural representations across deep hierarchies.

F Prompts

F.1 Prompts for Training and Evaluation

F.2 Prompts for Data Generation and Validation

The prompt template used for synthetic table generation is listed as below. The variables are sampled from different data diversity dimensions as mentioned in Appx. D.2.

```
Convert the following text into JSON format according to the specified schema. Ensure that both keys
and values are strings, even for numerical values.

Text: {input_text}

Provide your response in the following JSON format: {json_schema}

Please output ONLY the JSON structure and extract the attributes only present in the schema.

Output:
```

Figure 6: Prompt template for direct LLM prompting

```
Be an impartial judge to identify whether the structured data accurately reflect the input text. If the
structured data contains anything that unsupported by the ‘input_text’, return False. If everything
can be found or inferred from the input text, return True. Enclose your answer in <validity></validity>
xml tags.

<input_text>{input_text}</input_text>

<structured_data>{gold}</structured_data>

Your answer:
```

Figure 7: Prompt template for synthetic training data validation

```
You are an advanced AI assistant specialized in data augmentation for text-to-JSON conversion tasks. Your goal is to generate
diverse and high-quality input-output pairs that will be used to train machine learning models for structured information extraction.

First, review these examples of input-output pairs:

<examples>{examples}</examples>

Your task is to generate 3 additional input-output pairs in JSONL format. Each pair should consist of:
1. Input:
- A text description
- A desired JSON schema with a brief explanation
2. Output:
- The text converted into JSON following the given schema

Please adhere to the following specifications:

<industry_vertical> {industry_vertical} </industry_vertical>
<genre> {genre} </genre>
<json_complexity_description> {json_complexity_description} </json_complexity_description>
<text_length_style> {text_length_style} </text_length_style>
<text_type> {text_type} </text_type>

Before generating each pair, wrap your thought process in <planning> tags. Consider the following:

1. Industry relevance: How can you make the text diverse and representative of the specified industry - *{industry_vertical}*?
- List 1-5 key topics or scenarios relevant to the industry.
- Consider how these topics can be incorporated into the text descriptions.
2. JSON schema complexity: How can you adhere to the complexity level of the JSON schema - *{json_complexity_description}*, while
staying within the other given constraints?
- Brainstorm 1-5 different schema structures of such complexity.
- Ensure each schema adheres to the complexity description and other constraints provided.
3. Text length adherence: How can you ensure the text length adheres to the specified style - *{text_length_style}*?
- Outline a strategy for maintaining consistent text length across all pairs.
- Consider using a word count check for each generated text.
4. Unique aspects and edge cases: What unique aspects or edge cases can you incorporate to make the training data more robust?
- List 2-3 potential edge cases or unusual scenarios relevant to the industry.
- Plan how to integrate these into some of the pairs.
5. Diversity tracking: How will you ensure diversity across all 5 pairs?
- Create a simple tracking system to ensure you're varying topics, schema complexity, and text length across the pairs.
- Number each pair as you plan it (1/5, 2/5, etc.) to keep track of your progress.
6. JSON content alignment: How will you ensure the gold JSON only contains information present in the input text?
- Implement a strict check to verify that every piece of information in the gold JSON can be directly traced back to the input text.
- Plan a review process to eliminate any potential extra content in the gold JSON that is not explicitly stated in the input text.
7. Genre: Ensure that the generated text adheres to the specified genre *{genre}*. This will influence the tone, style, and content
of the 'input_text', but does not affect the JSON structure.
8. Text Type: Generate text that aligns with the specified text type *{text_type}*. This will determine the format, structure, or
purpose of the 'input_text' you create, separate from the JSON output.
9. Follow the format in the example below for your 'input_text', 'json_schema' and 'gold':

{{ "input_text": "Acme Motors, a major automaker has announced plans to build a new electric vehicle manufacturing plant in
Greenville, South Carolina. The state-of-the-art facility will produce the company's latest line of battery-powered cars and SUVs,
with an initial annual capacity of 200,000 units.", "json_schema": {{ "type": "object", "properties": {{ "company": {{ "type":
"string" }}, "location": {{ "type": "string" }}, "production_capacity": {{ "type": "number" }} }} }}, "gold": {{ "company": "Acme
Motors", "location": "Greenville, South Carolina", "production_capacity": 200000 }} }}

Notes:
- "input_text": Contains a string value, represents the raw text input that needs to be processed
In this case, it's a news-like paragraph about a company announcement.
- "json_schema": Defines the structure of the expected output. must include "type" and "properties", defined as below:
"type": "object" - specifies that the output should be a JSON object
"properties" - defines the expected fields:
- "company": expects a string value
- "location": expects a string value
- "production_capacity": expects a number value
Each property has its own type definition
- "gold": Contains the correct/expected output that matches the schema. Has the exact same structure as defined in json_schema.
Contains the actual values extracted from the input_text:
- "company": contains the company name
- "location": contains the city and state
- "production_capacity": contains the numerical value
Values must match the types specified in the schema (strings for company and location, number for production_capacity)
- *IMPORTANT* Do not replicate any of the content in the given example, it's just used as a reference for a specified answer structure.

After your planning process, generate the input-output pair and format it in JSONL. Here's an example of the expected format:

{{"input_text": "Your generated text here", "json_schema": {{"Your": "JSON", "schema": "here"}}, "gold": {{"Your": "gold", "JSON":
"here"}}}

Remember to generate 5 unique and diverse pairs, each following this format. Ensure that each pair adheres to the industry vertical,
complexity description, and text length style specified above. EACH response should be enclosed in <JSONL> </JSONL> XML tags.
```

Figure 8: Prompt template synthetic training data generation