Proceedings of CONF-CDS 2026 Symposium: Computer Vision-Based Multimodal Cognitive Load Estimation for Adaptive Media Communication
DOI: 10.54254/2755-2721/2026.GL37071

Research on Structured Output of Large Language Models Based on Dual-Dimensional Testing

Chao Zhang

School of Software Engineering, Chengdu University of Information Technology, Chengdu, China
zhangchao050530@gmail.com

Abstract. This study evaluates multiple constraint methods and output formats for producing structured data with large language models. To move beyond JSON-centered discussions, it builds a dual-dimensional testing framework in which output formats are compared horizontally and constraint methods are compared vertically. The experiments use practical extraction tasks and measure syntactic correctness, semantic correctness, response time, and token consumption across local open-source models and commercial lightweight models. The results show that JSON, although widely used in practice, is not always the most efficient structured output format. CSV and TSV perform well for flat tabular tasks, while JSON with validation or constrained decoding is more suitable when stability and schema compliance are critical. The findings indicate that output formats and constraint methods should be selected according to task shape, latency budget, model capability, and tolerance for downstream repair, rather than from a fixed preference for a single format.

Keywords: Large language models, structured output, dual-dimensional evaluation, JSON, benchmark

1. Introduction

Large language models are now being embedded in many routine software workflows [1-3], but the usefulness of these systems often depends on whether the returned text can be consumed by another program without manual repair. Automated bookkeeping is a typical example. After a user uploads a bill screenshot, the model may need to identify the merchant, amount, date, category, and remarks, and then pass these fields to a database or an accounting module. A fluent natural-language answer is not enough in this setting; the output must follow a stable schema. However, much existing discussion still treats JSON as the default choice or examines only one constraint strategy [4-8]. This leaves two practical questions insufficiently compared: whether the format itself changes token cost and latency, and whether stronger constraints actually improve correctness enough to justify their overhead. To address these questions, this paper compares output formats in one dimension and constraint methods in the other, using token consumption, response time, format validity, and field correctness as the main indicators.

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2755-2721/2026.GL37071

2. Overview

2.1. Overview of large language models

Most current large language models are built on the Transformer architecture and are trained on large-scale corpora with a next-token prediction objective. During inference, the model estimates a probability distribution for the next token under the given context, and the final text is produced through decoding strategies such as sampling or search. This generation mechanism is powerful, but it also means that the model is not naturally a deterministic data serializer. As LLMs move from chat interfaces into application back ends, this gap between probabilistic generation and program-readable output becomes a concrete engineering problem.

2.2. Definition and importance of structured output

In this paper, structured output refers to model responses that follow explicit fields, delimiters, and syntax rules, so that software can parse them directly. JSON and YAML are common examples, but the same issue also appears in CSV, TSV, TOML, and other formats. The difficulty is that these formats require local details to remain exact: a missing quotation mark, an extra key, or a wrong data type may be enough to break a downstream parser. Because an LLM generates tokens probabilistically rather than by executing formal grammar, such errors may still occur even when the prompt describes the expected format clearly. Recent work on reliable structured output and schema-based generation has explored similar failure modes [4, 5, 8].

For engineering use, this requirement cannot be avoided. Stable fields allow the result to be checked, stored, queried, and passed to later modules. They also reduce the amount of manual inspection needed when a model is used repeatedly in a production workflow.

3. Dual-dimensional testing framework

3.1. Concept and construction of dual-dimensional testing

To systematically evaluate the structured output capability of large models, this study constructs a dual-dimensional structured output evaluation framework: horizontal × vertical. The horizontal dimension keeps the constraint method unchanged and evaluates the structured output capability of large models under different output formats. The vertical dimension keeps the output format unchanged and evaluates the structured output capability of large models under different constraint methods.

3.2. Selection and classification of testing dimensions

3.2.1. Prompt engineering

Prompt engineering is currently the most widely used, flexible, and low-cost structured constraint method. The model generates the target format based solely on natural language prompts, meaning that its output is entirely self-regulated by the model and does not include any mandatory validation mechanism. This constraint method is highly universal and can be applied to any format. It also has the lowest cost, as it does not require additional inference mechanisms or inference frameworks. However, under this constraint, formatting errors or deviations are most likely to occur, and the output cannot be guaranteed to be fully structured.

DOI: 10.54254/2755-2721/2026.GL37071

At this constraint level, we horizontally compare multiple structured formats and systematically analyze the influence of format itself on model output.

3.2.2. Post-validation

Post-validation is also a flexible and low-cost structured constraint method. There are already many frameworks on the market that support post-validation. This experiment uses the PydanticAI framework. Post-validation mainly takes two forms. The first, and most common, is to use the model itself to validate the output through mechanisms such as MCP or Function Calling. The second is to process the output through fixed mechanisms, such as regular expressions. To highlight the structured output capability of the large language model itself, this experiment adopts the former approach for post-validation constraints.

3.2.3. Constrained decoding

Constrained decoding fundamentally addresses the uncertainty of model generation. Unlike the two constraint methods above, constrained decoding intervenes during the generation process. Its core technical principle is to dynamically constrain the model's output space through an external, predefined rule set or grammar while the large language model generates each token [9]. This method can fully ensure that the large model outputs formatted content as expected. Related methods such as sketch-guided constrained decoding also use external structure to reduce invalid generations [6].

Since this experiment includes black-box commercial models, constrained decoding theoretically cannot be directly implemented. However, according to relevant documentation from OpenAI and Anthropic, API-based structured output is essentially implemented through constrained decoding inside the model. Therefore, this experiment can use this constraint method to vertically compare different models.

Because this constraint only supports JSON output, the vertical comparison research design specifies JSON as the output format for comparison.

3.3. Experimental framework overview

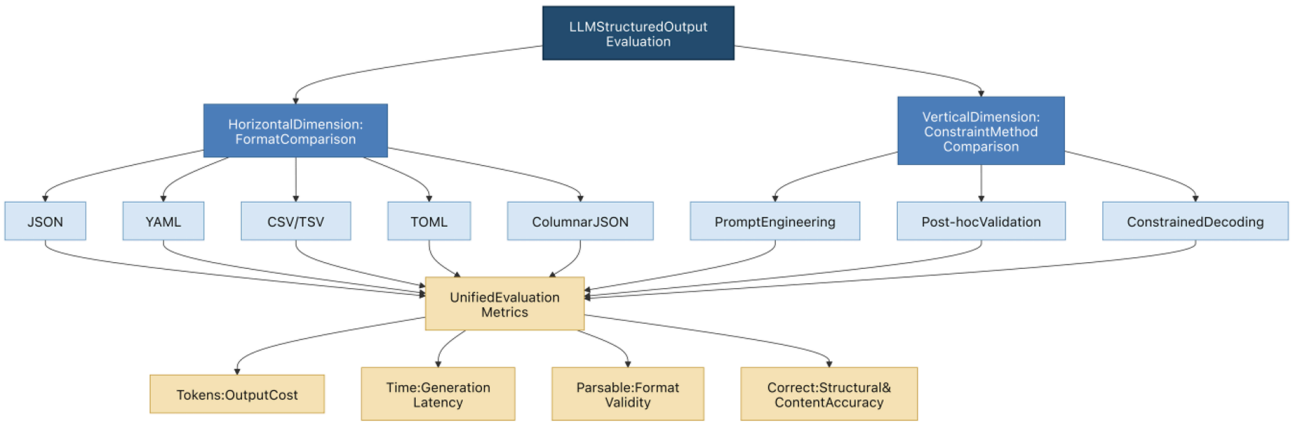

Figure 1. Overview of the dual-dimensional evaluation framework

This figure 1 illustrates the dual-dimensional evaluation framework used in this study. The horizontal dimension compares different structured output formats, including JSON, YAML,

DOI: 10.54254/2755-2721/2026.GL37071

CSV/TSV, TOML, and Columnar JSON, while the vertical dimension compares different constraint methods, including prompt engineering, post-hoc validation, and constrained decoding. All experimental settings are evaluated using unified metrics such as token consumption, generation latency, parsable format validity, and structural/semantic correctness.

4. Experimental design

4.1. Design and selection of evaluation metrics

The evaluation uses four indicators. Syntactic correctness records whether the returned text can be parsed as the required format. Semantic correctness checks whether the parsed fields are consistent with the expected information, rather than merely well formed. This distinction is consistent with broader work on LLM-based evaluation methods [10]. Response time measures the elapsed generation time observed in each run, and total token consumption records the amount of model output and interaction overhead. These four indicators were selected because a format that is easy to parse may still be slow or costly, while a fast response may be unusable if field values are wrong.

4.2. Experimental design and data collection

The experiments are based on the self-developed Bloom dataset. The dataset was organized around practical extraction tasks and designed so that additional cases can be appended without changing the evaluation pipeline. Its current cases cover field recognition, format compliance, and common error types that appear when semi-structured information is converted into machine-readable records.

The model group contains both local open-source models and commercial lightweight models. The local side uses qwen:14b and gemma3:12b through Ollama, while the commercial side uses gpt-5-mini and claude-haiku-4.5 through OpenRouter. Each model-format or model-constraint setting was executed ten times, and the reported values are the averages of these repeated runs. This repetition was used to reduce the influence of occasional decoding fluctuations.

5. Results analysis

5.1. Horizontal dimension

Table 1. Experimental results in the horizontal dimension

Target	Model	Time	Tokens	Success	Correct
Columnar JSON	gpt-5-mini	85.1	4850	TRUE	TRUE
	claude-haiku-4.5	9.31	3140	TRUE	TRUE
	qwen3:14b	100.23	4060	TRUE	TRUE
	gemma3:12b	237.03	3322	TRUE	FALSE *2
CSV	gpt-5-mini	47.71	5182	TRUE	TRUE
	claude-haiku-4.5	7.27	2631	TRUE	TRUE
	qwen3:14b	36.28	2847	TRUE	TRUE
	gemma3:12b	161.94	2845	FALSE *5	FALSE *5

DOI: 10.54254/2755-2721/2026.GL37071

Table 1. (continued)

JSON	gpt-5-mini	35.33	4506	TRUE	TRUE
	claude-haiku-4.5	12.08	3981	TRUE	TRUE
	qwen3:14b	88.3	4044	TRUE	TRUE
	gemma3:12b	334.82	4053	TRUE	TRUE
TOML	gpt-5-mini	32.03	4035	TRUE	TRUE
	claude-haiku-4.5	9.65	3586	TRUE	TRUE
	qwen3:14b	100.9	4231	FALSE *8	FALSE *8
	gemma3:12b	301.99	3730	TRUE	FALSE *2
TSV	gpt-5-mini	33.68	4269	TRUE	TRUE
	claude-haiku-4.5	8.23	2648	TRUE	TRUE
	qwen3:14b	35.67	2858	TRUE	FALSE *3
	gemma3:12b	349.56	2833	FALSE *2	FALSE *6
YAML	gpt-5-mini	35.55	4090	TRUE	TRUE
	claude-haiku-4.5	9.17	3417	TRUE	TRUE
	qwen3:14b	74.9	3454	TRUE	TRUE
	gemma3:12b	254.99	3446	TRUE	TRUE

As shown in Table 1, commercial lightweight models can output various target formats exactly as expected. Open-source lightweight models can output mainstream formats, such as JSON and YAML, completely correctly. However, their performance on other formats is weaker, with frequent parsing errors and semantic errors.

Considering both time and token metrics, the following patterns can be summarized: table-like formats consume the least time and tokens. This is because the model hardly needs to generate additional punctuation or structure, resulting in the lightest inference burden. Hierarchical formats have medium consumption because nested structures require more characters, but models are relatively familiar with JSON and YAML, so overall output remains stable. Weakly structured but syntactically complex TOML becomes the "worst-performing format," which is consistent with the experimental results of Jialin Yang et al. [7].

TSV and CSV are the lightest and most efficient structured output formats under the current constraints. They require the fewest tokens, have the shortest response time, and impose the lowest pressure on models. Therefore, they are very suitable for tabular tasks, such as bookkeeping and form filling. JSON and YAML are "medium-cost formats" and serve as practical balance points: they are well represented in model training distributions, have clear structures, and strong parsability, achieving a good balance between correctness and stability. Finally, Columnar JSON and TOML impose higher requirements on models and introduce additional overhead.

DOI: 10.54254/2755-2721/2026.GL37071

5.2. Vertical dimension

Table 2. Experimental results in the vertical dimension

Constraint	Model	Time	Tokens	Success	Correct
Prompt Engineering	gpt-5-mini	35.33	4506	TRUE	TRUE
	claude-haiku-4.5	12.08	3981	TRUE	TRUE
	qwen3:14b	88.3	4044	TRUE	TRUE
	gemma3:12b	334.82	4053	TRUE	TRUE
Post Validation	gpt-5-mini	164.39	4240	TRUE	TRUE
	claude-haiku-4.5	76.69	4747	TRUE	TRUE
	qwen3:14b	186.42	5120	TRUE	TRUE
	gemma3:12b	206.01	5088	TRUE	TRUE
Constrained Decoding	openai/gpt-5-mini	63.35	5381	TRUE	TRUE
	anthropic/claude-haiku-4.5	12.6	3616	TRUE	TRUE
	qwen3:14b	482.18	4786	TRUE	FALSE *1
	gemma3:12b	657.95	3989	TRUE	TRUE

As shown in Table 2, under prompt engineering and post-validation constraints, the three models can mostly output the target format correctly. Under constrained decoding, qwen3:14b produces one semantic error. This indicates that the constraint can fully enforce correctness at the syntactic level, but still cannot fully guarantee semantic correctness.

In terms of token consumption, one key phenomenon appears in the experiment for post-validation: the token consumption of open-source lightweight models is significantly higher than that of commercial models. The reason behind this can be explained from the generation process. The current constraint may trigger multi-round generation, self-correction, and guided completion. Therefore, the additional token overhead comes from the initial output, supplementary generation after validation failure, and possible multi-round interactions. For constrained decoding, commercial models consume the most tokens, while lightweight models consume fewer tokens. The reason is that under the current constraint, models must generate complete structures and cannot perform semantic compression or structural simplification; all fields must be explicitly expanded.

In terms of response time, under post-validation, the response times of different models are relatively close but generally high under the current constraint. This is determined by the characteristics of the constraint method itself, including extra latency introduced by validation, implicit retries, and validation time. For constrained decoding, this method introduces a token-by-token constraint mechanism: each generated token must pass syntactic constraints, and illegal tokens are pruned. This is equivalent to adding "search restrictions" during decoding. Therefore, compared with other constraint methods, this constraint generally consumes more tokens.

Considering correctness, token consumption, and response time, the characteristics of the three constraint methods can be summarized as follows. Prompt engineering has the lowest time and token consumption, but in principle it cannot guarantee parsability or correctness of structured output. Post-validation is suitable for scenarios that require high correctness but do not require extremely low latency, and for systems that can tolerate additional token costs. It is not suitable for real-time systems or ultra-large-scale calls. Constrained decoding can fully guarantee the parsability

DOI: 10.54254/2755-2721/2026.GL37071

of output, but it does not guarantee correctness and significantly amplifies differences among models in inference optimization.

6. Case study: structured output in practical applications

The automated bookkeeping example can illustrate how the experimental results translate into engineering choices. A bill-recognition module usually needs to extract fields such as amount, merchant, time, category, and payment method from screenshots or OCR text. Different deployments do not require the same level of reliability: a personal note-taking tool, a customer-facing expense assistant, and a financial review system tolerate very different risks.

For a low-cost setting, CSV with prompt engineering is sufficient when the task is close to a flat table and occasional manual correction is acceptable. The prompt can ask the model to return one row per bill and fixed columns for the required fields. This approach adds little engineering complexity and keeps the generation burden low, so it is more appropriate for small data volumes, trial systems, or workflows where users can quickly notice and correct mistakes.

For a more common application scenario, JSON with post-validation is a safer choice. The model first produces a JSON object, and the application then checks whether the amount is numeric, whether the date can be normalized, whether required fields are missing, and whether the category belongs to an allowed range. This design keeps the format familiar to developers while adding a repair layer for predictable errors. It is therefore suitable for most bookkeeping modules that need stable data but can accept some additional latency.

For high-reliability tasks, JSON with constrained decoding is more appropriate. By binding generation to a predefined schema, the system can prevent many format-level failures before the response is completed. For example, the amount field can be restricted to a numeric value and the category field can be limited to a fixed enumeration. This method is especially useful in financial reconciliation, reimbursement review, or audit-related workflows, where an invalid structure may cause downstream processing errors or require costly manual intervention.

7. Conclusion

This paper examined structured generation from two directions: the choice of output format and the choice of constraint method. The results suggest that structured output should not be treated as a single JSON-centered problem. The best solution changes with the task shape, latency budget, model type, and tolerance for downstream repair.

At the constraint-method level, the three approaches show different trade-offs. Prompt engineering is the easiest to deploy and can be used with almost any target format, but it depends heavily on the model following instructions exactly. Post-validation adds an external checking step and can repair many predictable mistakes, although this benefit comes with extra latency and sometimes additional generation rounds. Constrained decoding offers the strongest control over syntax because invalid tokens can be blocked during generation. Even so, the experiment also shows its boundary: a schema can force a valid structure, but it cannot by itself prove that every field has the right meaning.

The output-format comparison also indicates that theoretical compactness is not the same as practical generation efficiency. CSV and TSV performed well because their structure is close to a flat table and requires little punctuation. JSON and YAML were more expensive but remained stable, likely because these formats are common in code and documentation data seen during training. TOML and Columnar JSON did not consistently benefit from their design advantages in

DOI: 10.54254/2755-2721/2026.GL37071

this setting. For lightweight models in particular, less familiar syntax or less typical field organization can increase both errors and generation cost.

Combining the two dimensions gives a more practical selection rule. If the system is mainly constrained by cost and response time, lightweight formats with prompt engineering should be considered first. If the system needs a balance between stability and development convenience, JSON plus post-validation is usually the most practical baseline. If structural validity is the primary requirement, constrained decoding should be introduced, while its latency and token overhead need to be budgeted in advance.

Overall, the structured output ability of an LLM is shaped jointly by the model, the target format, and the constraint mechanism around generation. Engineering design should therefore start from the downstream use case rather than from a fixed format preference. The central decision is how much generation freedom can be allowed before structural certainty becomes more valuable than speed or low cost.

References

[1] Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., et al. (2024). A survey on large language model based autonomous agents. *Frontiers of Computer Science, 18*, Art. no. 186345. https://doi.org/10.1007/s11704-024-40231-1

[2] Luo, Z., Xu, C., Zhao, P., Sun, Q., Geng, X., Hu, W., et al. (2024). WizardCoder: Empowering code large language models with Evol-Instruct. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/2306.08568

[3] Chiang, W.-L., Zheng, L., Sheng, Y., Angelopoulos, A. N., Li, T., Li, D., et al. (2024). Chatbot Arena: An open platform for evaluating LLMs by human preference. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*. https://arxiv.org/abs/2403.04132

[4] Hu, R., & Wu, S. (2025). RL-Struct: A lightweight reinforcement learning framework for reliable structured output in LLMs. *arXiv preprint arXiv:2512.00319*. https://doi.org/10.48550/arXiv.2512.00319

[5] Lu, Y., Li, H., Cong, X., Zhang, Z., Wu, Y., Lin, Y., et al. (2025). Learning to generate structured output with schema reinforcement learning. *arXiv preprint arXiv:2502.18878*. https://doi.org/10.48550/arXiv.2502.18878

[6] Geng, S., Döner, B., Wendler, C., Josifoski, M., & West, R. (2024). Sketch-guided constrained decoding for boosting blackbox large language models without logit access. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*. https://doi.org/10.48550/arXiv.2401.09967

[7] Yang, J., Jiang, D., He, L., et al. (2026, January). StructEval: Benchmarking LLMs' capabilities to generate structural outputs. *Transactions on Machine Learning Research*. https://openreview.net/forum?id=buDwV7LUA7

[8] Geng, S., Cooper, H., Moskal, M., Jenkins, S., Berman, J., Ranchin, N., et al. (2025). JSONSchemaBench: A rigorous benchmark of structured outputs for language models. *arXiv preprint arXiv:2501.10868*. https://doi.org/10.48550/arXiv.2501.10868

[9] Geng, S., Josifoski, M., Peyrard, M., & West, R. (2023). Grammar-constrained decoding for structured NLP tasks without finetuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 10932–10952). https://doi.org/10.18653/v1/2023.emnlp-main.674

[10] Li, H., Dong, Q., Chen, J., Su, H., Zhou, Y., Ai, Q., et al. (2024). LLMs-as-judges: A comprehensive survey on LLM-based evaluation methods. *arXiv preprint arXiv:2412.05579*. https://doi.org/10.48550/arXiv.2412.05579