Article

Not peer-reviewed version

Think Inside the JSON: Reinforcement Strategy for Strict LLM Schema Adherence

Bhavik Agarwal , Ishan Joshi , Viktoria Rojkova *

Posted Date: 20 February 2025

doi: 10.20944/preprints202502.1390.v2

Keywords: Large Language Models (LLMs); Schema Adherence; LLM Reasoning; DeepSeek R1; Reinforcement Learning (RL); 1.5B-parameter Model; Structured Reasoning; Synthetic Reasoning Dataset; Custom Reward Functions; Group Relative Policy Optimization (GRPO); Supervised Fine-Tuning (SFT); Schema Consistency; GPU Training (H100, A100); ThinkJSON; Distilled Models (Qwen-1.5B, Qwen-7B); Gemini 2.0 Flash (70B); Resource-Efficient Framework; Schema-Constrained Text Generation

 doi:10.20944/preprints202502.1390.v2

Article

Think Inside the JSON: Reinforcement Strategy for Strict LLM Schema Adherence

Bhavik Agarwal *, Ishan Joshi and Viktoria Rojkova

Master Control AI Research
* Correspondence: bagarwa4@jhu.edu

Abstract: In this paper, we address the challenge of enforcing strict schema adherence in large language model (LLM) generation by leveraging LLM reasoning capabilities. Building on the DeepSeek R1 reinforcement learning framework, our approach trains structured reasoning skills of a 1.5B parameter model through a novel pipeline that combines synthetic reasoning data set construction with custom reward functions under Group Relative Policy Optimization (GRPO). Specifically, we first perform R1 reinforcement learning on a 20K sample unstructured to structured data set, mirroring the original DeepSeek R1 methods, to establish core reasoning abilities. Subsequently, we performed supervised fine-tuning on a separate 10K reasoning sample dataset, focusing on refining schema adherence for downstream tasks. Despite the relatively modest training scope, requiring approximately 20 hours on an 8×H100 GPU cluster for GRPO training and 3 hours on 1xA100 for SFT, our model demonstrates robust performance in enforcing schema consistency. We compare our ThinkJSON approach against the original DeepSeek R1 (671B), distilled versions of DeepSeek R1 (Qwen-1.5B and Qwen-7B) and Gemini 2.0 Flash (70B), showcasing its effectiveness in real-world applications. Our results underscore the practical utility of a resource-efficient framework for schema-constrained text generation.

Dataset: Hugging Face Model; R1-Reasoning-Unstructured-To-Structured Dataset; JSON-Unstructured-Structured Dataset

Dataset License: License under which the data set is made available (CC0, CC-BY, CC-BY-SA, CC-BY-NC, etc.)

Keywords: large language models (LLMs); schema adherence; LLM reasoning; DeepSeek R1; reinforcement learning (RL); 1.5B-parameter model; structured reasoning; synthetic reasoning dataset; custom reward functions; group relative policy optimization (GRPO); supervised fine-tuning (SFT); schema consistency; GPU training (H100; A100); ThinkJSON; distilled models (Qwen-1.5B; Qwen-7B); Gemini 2.0 Flash (70B); resource-efficient framework; schema-constrained text generation

1. Introduction

In the highly regulated domain of bio-manufacturing quality, there is a growing need to convert legacy production records into structured digital formats for compliance and analysis. Biomanufacturing has historically been 'steeped in a paper culture', and even incremental moves toward electronic batch records are significant steps in industry digitalization [1]. A key prerequisite of this digital migration is schema adherence: AI systems, such as large language models (LLMs) used to transcribe or summarize production logs, must output data that fit a predefined schema exactly. Any deviation (missing fields, incorrect format) could violate data integrity standards and render the generated records unusable for regulatory compliance [2]. This introduces a critical challenge: While modern LLMs are extraordinarily powerful in free-form text generation, ensuring that they produce strictly structured, schema-valid outputs is not trivial.

LLMs by default generate text probabilistically, with no built-in guarantee of conforming to a given format [3]. This unpredictability poses risks when structured output is required for machine

 doi:10.20944/preprints202502.1390.v2

consumption or auditing. Empirical studies have found that even state-of-the-art models can *fail* to consistently follow format instructions – success rates in producing correct JSON, for example, can vary widely from 0% to 100% depending on the task complexity and model used [4]. Such inconsistency is problematic in any setting, but in regulated bio-manufacturing, an output that does not exactly match the schema (e.g., a misformatted timestamp or an extra delimiter) might lead to compliance issues or require costly manual correction. Developers report that substantial effort is spent on prompt tuning and post-processing to coerce LLMs into the desired format [5]. From a user perspective, unreliable formatting undermines trust – constraints help prevent nonsense or hallucinated fields, thereby ensuring the output remains credible and useful [5]. In short, structured output generation is both a technical and a governance challenge: the model must be reliable in content as well as form.

2. Relevant Work

Researchers and practitioners are exploring several approaches to address these challenges and enforce schema adherence in LLM outputs. Key strategies include:

2.1. Supervised Fine-Tuning

An LLM can be fine-tuned on domain-specific data with the required output schema, so it learns to produce the correct structure. Fine-tuning on curated input–output pairs (e.g., historical records mapped to structured entries) can significantly improve format fidelity [6]. However, this approach is resource-intensive – training large models on specialized data is complex and costly, often requiring techniques like low-rank adaptation to be feasible [6]. Fine-tuning also risks making the model too domain-specific or rigid outside the training distribution.

2.2. Reinforcement Learning with Human Feedback (RLHF)

RLHF has proven effective in aligning LLMs with human instructions and preferences [7]. By training a model with feedback signals that reward correct adherence to the desired format, one can encourage structured outputs. Notably, the instruction-following abilities of models like ChatGPT/GPT-4 are largely attributed to such alignment techniques [7], enabling them to obey fine-grained formatting requests (e.g. "output as JSON"). In regulated settings, RLHF could incorporate compliance-specific criteria into the reward model. The downside is that RLHF requires extensive high-quality feedback data and careful reward design; even then, smaller open-source models often still lag behind in format obedience despite alignment efforts [7].

2.3. Constraint-Based Decoding

Rather than relying on the model to *choose* the right format, constraint-based methods *force* compliance by integrating schema rules into the generation process. Techniques like grammar- or regex-guided decoding intercept the model's token output, only allowing continuations that keep the output valid according to a formal schema [3,8]. This guarantees 100% schema adherence by construction. Recent frameworks implement fast, non-invasive constrained decoding that can guide LLMs to produce, for example, JSON that matches a given schema exactly [6]. Industry adoption of these ideas is rising; for instance, OpenAI's API now accepts developer-provided JSON schemas and assures that the model's response will conform to them [3]. The trade-off here is potential complexity in setup and slight inference latency overhead, as well as the challenge of designing schemas that are neither over- nor under-constraining. Nonetheless, when correctness is paramount, constrained decoding is a powerful approach.

2.4. Prompt Engineering

The most accessible technique is to craft the input prompt in a way that strongly cues the desired structure. This can involve giving the model explicit formatting instructions, examples of correctly formatted output, or even "layout hints" in the prompt. A well-designed prompt can often induce a model to produce a nearly perfect structured output [6]. Prompt engineering requires no model

 doi:10.20944/preprints202502.1390.v2

training and can be iteratively refined. However, it demands significant manual effort and expertise, and even then does not *guarantee* consistency [6]. Models may still err on edge cases or as the prompt complexity grows, and maintaining long, complex prompts (especially across different models or updates) can be cumbersome. In practice, prompt-based solutions might be combined with lightweight validation or post-processing in high-stakes applications.

2.5. Hybrid Constraint-Based Decoding and Prompt Engineering

By embedding knowledge of the schema at the prompt level and using a specialized procedure to keep the generation on track (via tagging, iterative re-checks, or extra control tokens), hybrid systems achieve schema adherence more reliably than a vanilla LLM approach [9]. This structured, schema-first method is key to guaranteeing the outputs are valid, parseable, and aligned with downstream consumption requirements. Schema acts as a blueprint for how the final text must be organized while controllable generation mechanism conditions the model's decoding process on these schema constraints. Instead of free-form text generation, the model is guided to fill in required slots, adhere to the correct format, and avoid extraneous or malformed outputs [9].

Each of these approaches comes with effectiveness trade-offs, especially under the stringent demands of regulated industries. Fine-tuning and RLHF can deeply instill format compliance into a model but at high development cost and with less transparency. Prompt engineering is more flexible and avoid retraining, but it relies on the base model's capacity to follow instructions. Constraint-based decoding offers hard guarantees on structure, appealing for compliance, though it requires integrating external constraint logic with the model's output stream. The choice often depends on the specific use case and constraints – for instance, biomanufacturers must consider not only technical accuracy but also validation, auditing, and data governance. Ensuring that LLM-generated records are *both* accurate in content and precise in format is vital to meet quality and regulatory standards. Recent work underlines that reliable structured generation remains an open challenge, calling for continued research into methods that can robustly align LLM outputs with predefined schemas [4].

3. Method

Although the strategies outlined above—ranging from prompt engineering to constraint-based decoding—can improve structured output, they often require specialized tooling or large-scale fine-tuning. In regulated domains such as bio-manufacturing, these approaches must also be cost-effective and robust. In this section, we describe a reasoning-driven methodology that leverages synthetic data construction and iterative LLM reasoning to ensure schema adherence with minimal overhead. Specifically, we demonstrate how to:

- **Build RL reasoning dataset**

  Create synthetic unstructured and structured data [10],[11] in tandem using controlled prompts and Qwen 14B/32B [12],

  Reverse-engineer how unstructured text can map onto an empty JSON schema by engaging a distilled DeepSeek R1 Qwen 32B [13] to explain—step by step—how each schema field is populated.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 20 February 2025 doi:10.20944/preprints202502.1390.v2

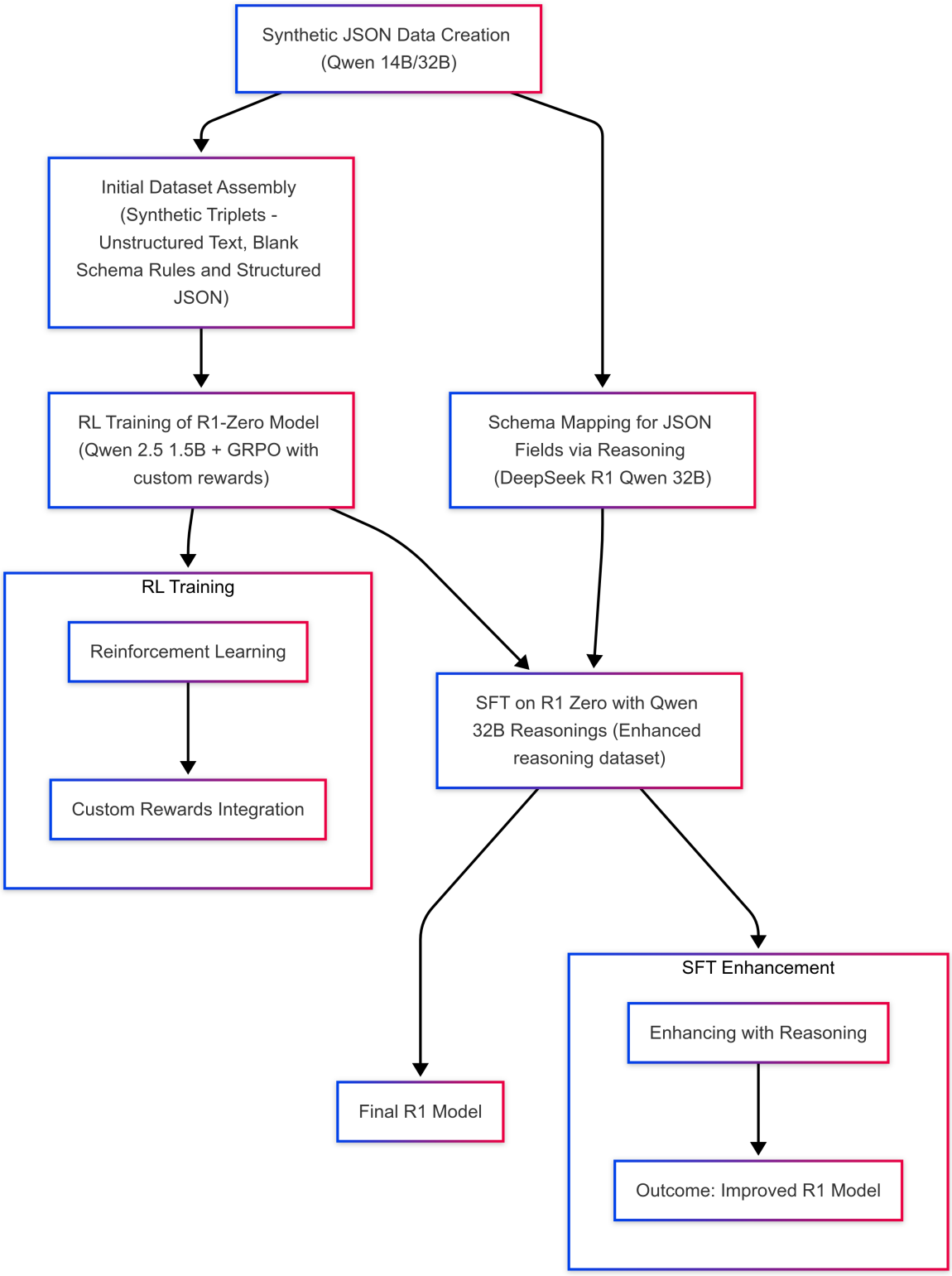

Figure 1. "Think inside the JSON" pipeline.

- **Train reasoning model with RL and SFT**.

  Develop custom reward mechanisms that directly evaluate how well the outputs adhere to a predefined schema while balancing fluency, completeness, and correctness.

  Train R1-Zero reasoning model from Qwen 2.5 1.5B base model using RL [14],[15] and synthetic unstructured-structured pair dataset, integrate custom rewards into GRPO [14] without

 doi:10.20944/preprints202502.1390.v2

altering the core policy optimization loop. The combined reward drives the training so that the model produces outputs that score highly on all relevant criteria.

Fine tune R1-Zero model into R1 with supervised fine-tuning using reasoning dataset.

3.1. Generating Structured and Unstructured Data

We begin by prompting a language model (Qwen 14B and 32B) to produce diverse, fully populated JSON schemas (including nested and complex fields). These filled schemas emulate real-world documentation (e.g., QA checklists, batch records) while showcasing variations in schema hardness and domain.

```
You are an expert in building a hierarchical JSON schema and object for the domain {DOM
Your task is to create:

1. A multi-level JSON Schema describing:
   - ROOT (level 0),
   - SECTION (level 1),
   - SUBSECTION (level 2),
   - $DETAIL_{N}$ (level 3+).
   Each level may contain tables (2D data layouts) and checkbox elements (MCQs, confirm
   with nested components reflecting complex structures.

2. A JSON Object that strictly matches this schema, including:
   - \"id\" and \"title\"
   - \"level\" and \"$level_{type}$\"
   - An array of \"component\" objects (paragraphs, tables, or checkboxes)
   - A recursive \"children\" array
   - Special \"properties\" (e.g., \"variables\", \"content\") for data, logs, metrics,

Formatting Requirements:
- Escape all quotes (\"), replace newlines with \\n
- No trailing commas, single quotes, or extra data
- Enclose the final output with no extra explanations:
```

In parallel, we generate corresponding blank schemas—retaining structural outlines but omitting values. This gives us a "before and after" pair for each schema: an empty template and a filled instance. Such pairs are crucial for teaching LLMs how unstructured text should be systematically transformed into the exact JSON schema. We then produce unstructured text reflecting the same content as the filled schema—but presented in varying layouts (e.g., sequential paragraphs, parallel sections, combined strategies) and table formats (ASCII art, XML/HTML-like snippets, simulated PDF extraction, etc.). These multi-format "narratives" mimic the real challenge of reading and interpreting inconsistent legacy documents.

```
You are an expert in generating hierarchical text documents from JSON Object data point

**Task**: Convert the JSON Object into an unstructured, paragraph-based document.

**Given Data**: **Domain**; **JSON Schema**; **JSON Object**

**OUTPUT FORMAT** (enclosed strictly within <text>):
<text>
[Insert formatted hierarchical text from JSON object here]
</text>
```

 doi:10.20944/preprints202502.1390.v2


```
**Layout References**:
- Layout options for components/levels: \n{RANDOM_LAYOUT}
- Table styles: \n{RANDOM_TABLE_STYLE}
- Checkbox styles: '''[ ], YES, NO, N/A, etc.'''

**RULES**:
1. Map every JSON level, component, and attribute to the correct layout/style.
2. Surround JSON data points with additional words/sentences to obscure parsing.
3. Include all data (title, variables, metadata, content); no extra sections.
4. End each data point with a brief, unrelated remark.
5. Add filler paragraphs (definitions, domain info, etc.) not directly tied to the JSON
```

In doing so, we create a synthetic corpus that covers a broad range of domain contexts, from general manufacturing logs to specialized quality assurance frameworks. Each piece of unstructured text is logically equivalent to a filled JSON schema, yet differs in structure, style, and formatting.

3.2. Reasoning Dataset: Reverse-Engineering from Text to Schema

We employ Distilled DeepSeekR1 Qwen 32B with the following prompt:

```
You are an AI assistant tasked with extracting structured data from a piece of text.

Inputs:
1. Text (source of information)
2. Blank Schema (unfilled JSON schema)
3. Filled Schema (final populated JSON)

Goals:
1. Compare Text + Blank Schema to the Filled Schema.
2. Explain step by step (chain-of-thought) how the text populates the blank schema.
3. Output only the reasoning steps (thought process).
4. Cross-verify that this reasoning exactly produces the Filled Schema.

Format your final response as:
Chain of Thought Explanation: """
```

The LLM is instructed to output only its chain-of-thought reasoning, explicitly describing the mapping from text to schema. Such self-explaining prompts push the model to maintain strict schema fidelity while revealing the logic behind each structural decision. Because the prompt demands an explicit reasoning path, the LLM self-checks how each field is filled, minimizing random or malformed output. The chain-of-thought not only ensures correctness but also documents how the text was interpreted which is vital for regulated environments. By varying the domain (e.g., different types of QA reports) and text layout styles, we create a dataset that fosters LLM resilience to formatting quirks.

3.3. GRPO Training on a Small Foundation Model

Once we finalize the reasoning dataset, we proceed to train a small foundation model—mirroring the minimalistic DeepSeek R1 Zero approach—using GRPO [13]. We employ a 1.5B-parameter base model "to develop reasoning capabilities without any supervised data, focusing on their self-evolution through a pure reinforcement learning process" [13]. By leveraging a group-based advantage calculation and carefully designed reward signals (e.g., schema compliance, correctness), we efficiently instill structured reasoning capabilities within a resource-constrained pipeline. By incorporating multiple reward functions [16] into the GRPO framework, we can simultaneously encourage format correctness (via r_format) and content/domain correctness (via r_equation). The combined reward drives training so that the model produces outputs that score highly on all relevant criteria. The entire

 doi:10.20944/preprints202502.1390.v2

process remains computationally light (e.g., 20 hours on an 8×H100 cluster), demonstrating that strict schema adherence can be achieved even with compact, low-overhead foundation models.

3.3.1. JSON-Based Reward

This reward algorithm balances two aspects: (1) schema faithfulness via the key-value matching fraction, and (2) structural completeness via JSON length similarity. A high final reward indicates that the predicted JSON object closely matches the ground truth both in field contents and overall size.

3.3.2. Format Verification Reward

The format check enforces correct usage of specialized tags, crucial for downstream tasks that rely on clearly separated reasoning (<think> block) and final answers (<answer> block). The binary reward (0 or 1) simplifies reinforcement signals, focusing exclusively on structural correctness rather than content fidelity. The optional logging step enables sampling a small fraction of completions for qualitative inspection, aiding diagnostic or future training data curation.

 doi:10.20944/preprints202502.1390.v2

Algorithm 1 JSON-Based Reward Computation.

Given:
A list of completions $\mathcal{C} = \{c_1, \ldots, c_n\}$ from the model.
A list of ground-truth JSON objects $\mathcal{G} = \{g_1, \ldots, g_n\}$.
Each g_i is a valid JSON string.
procedure COMPUTEREWARD($\mathcal{C}, \mathcal{G}$)
  $\mathcal{R} \leftarrow \emptyset$ ▷ Initialize rewards list
  for each pair (c_i, g_i) in $(\mathcal{C}, \mathcal{G})$ **do**
    $c_i' \leftarrow$ `"<think>"` $\| \, c_i$ ▷ Insert `<think>` prefix
    $ans \leftarrow$ substring$(c_i', \text{``<answer>''}, \text{``</answer>''})$
    if ans is empty **then**
      $r_i \leftarrow 0$
      append r_i **to** $\mathcal{R}$; **continue**
    end if
    ▷ Parse as JSON
    parse ans into `answer_json`; parse g_i into `gt_json`
    if either parse fails **then**
      $r_i \leftarrow 0$
      append r_i **to** $\mathcal{R}$; **continue**
    end if
    ▷ Compute field overlap
    $\mathcal{K}_a \leftarrow$ keys(`answer_json`)
    $\mathcal{K}_g \leftarrow$ keys(`gt_json`)
    $total\_fields \leftarrow |\mathcal{K}_a \cup \mathcal{K}_g|$
    $matching\_fields \leftarrow \sum_{k \in (\mathcal{K}_a \cap \mathcal{K}_g)} \mathbf{1}[\texttt{answer\_json}[k] = \texttt{gt\_json}[k]]$
    if $total\_fields > 0$ **then**
      $key\_match\_score \leftarrow \frac{matching\_fields}{total\_fields}$
    else
      $key\_match\_score \leftarrow 0$
    end if
    ▷ Compare JSON lengths
    $\ell_a \leftarrow$ length(`answer_json`) or 1
    $\ell_g \leftarrow$ length(`gt_json`) or 1
    $length\_ratio \leftarrow \frac{\min(\ell_a, \ell_g)}{\max(\ell_a, \ell_g)}$
    ▷ Calculate final reward
    $r_i \leftarrow \frac{key\_match\_score + length\_ratio}{2}$
    clamp r_i to $[0, 1]$; round to 1 decimal place
    if $r_i \geq 0.6$ **then**
      log c_i' with 60% probability
    end if
    append r_i **to** $\mathcal{R}$
  end for
  return $\mathcal{R}$
end procedure

 doi:10.20944/preprints202502.1390.v2

Algorithm 2 Format Verification Reward.

Goal: Assign a reward of 0 or 1 depending on whether a generated completion follows an expected structure using `<think>...</think>` and `<answer>...</answer>`.

Inputs:
A list of completions $\mathcal{C} = \{c_1, \ldots, c_n\}$ (model-generated).
A list of ground-truth objects $\mathcal{G} = \{g_1, \ldots, g_n\}$ (not directly used here, but included for extensibility).
A small probability p (e.g., 0.1) for selectively logging completions.

Output: A list of scalar rewards $\mathcal{R} = \{r_1, \ldots, r_n\}$, with $r_i \in \{0, 1\}$.

Initialize an empty rewards list: $\mathcal{R} \leftarrow [\,]$.
for each pair (c_i, g_i) **in** $(\mathcal{C}, \mathcal{G})$ **do**
  Synthesize prompt format: $c_i' \leftarrow$ `"<think>"` $\| \, c_i$ ▷ Prepend "`<think>`"
  Probabilistic logging: Draw $x \sim U[0, 1]$.
  if $x < p$ **then**
    Log c_i' to file for future analysis.
  end if
  Format check via regex:
  Define $\mathcal{R} =$ `"^<think>([^<]*(?:<(?!/?think>)[^<]*)*)</think>\n<answer>([\\s\\S]*?)</answer>$"`
  Match $\mathcal{R}$ against c_i'.
  if match fails (no correct grouping) **then**
    $r_i \leftarrow 0$
  else
    $r_i \leftarrow 1$
  end if
  Append r_i to $\mathcal{R}$.
end for
return $\mathcal{R}$

Algorithm 3 GRPO with Multiple Reward Functions.

Notation and Setup
Define a combined reward:

$$R_{\text{comb}}(c) \;=\; f\big(r_1(c),\, r_2(c),\, \ldots,\, r_K(c)\big),$$

where f can be a weighted sum, mean, or any aggregator, π_θ be the current policy (a language model parameterized by θ), $\{r_k\}_{k=1}^{K}$ be K reward functions (e.g., r_{format}, r_{equation})

Group-Based Relative Advantage
Let $G = \{\, c_1, \ldots, c_M\}$ be a group of M outputs sampled from π_θ.
For each c_i, compute a combined reward $R_i = R_{\text{comb}}(c_i)$.
Define the relative (rank-based) advantage:

$$A^{(\text{rel})}(c_i) \;=\; \frac{1}{M-1} \sum_{j \neq i} \mathbf{1}\big(R_i > R_j\big),$$

which is the fraction of samples in G that have lower reward than c_i.

GRPO Update
Update θ to favor completions with higher relative advantage.
The GRPO loss for group G is:

$$\mathcal{L}_{\text{GRPO}}(\theta) \;=\; -\sum_{c_i \in G} A^{(\text{rel})}(c_i)\; \log \pi_\theta(c_i) \;+\; \text{Reg}(\theta),$$

where $\text{Reg}(\theta)$ includes regularization terms (e.g., entropy bonus, KL-divergence).

 doi:10.20944/preprints202502.1390.v2

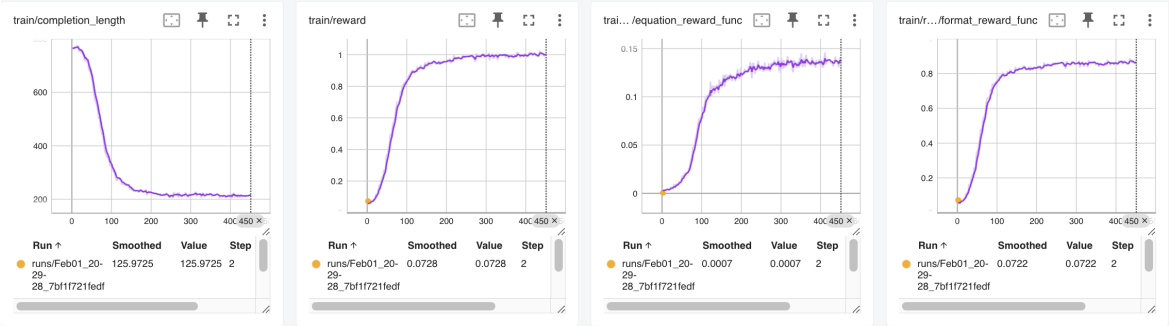

Figure 2. GRPO Training Metrics.

3.4. Supervised Fine-Tuning

While reinforcement learning confers advanced reasoning capacities, but supervised fine-tuning provides the final task- and schema-specific "polish" that ensures outputs are both logically grounded and robustly aligned with real-world standards. [13]. Reinforcement learning (RL) optimizes a policy for broad correctness or format adherence but can overlook rare or domain-specific intricacies (e.g., specialized field naming conventions, unusual data types). SFT exposes the model to explicit examples that emphasize precisely how to handle real-world edge cases, ensuring no field or condition is left under-represented. Although RL fosters adaptability, the learned policy may still exhibit variability in ambiguous contexts or unrepresented task scenarios [13]. SFT, by contrast, anchors the final policy to concrete labeled examples, reducing output drift. By overlaying a final SFT stage, ThinkJSON tightly aligns its already-developed reasoning to the strict output requirements (e.g., correct JSON keys, mandatory fields), producing outputs suitable for audit or compliance. For SFT (and SFT+LoRA) we used the Unsloth training framework on an A100 GPU, completing the process in about 3 hours.

```
### Role:
You are an expert data extractor mapping hierarchical text to a given JSON Schema.

### DATA INPUT: Text; Blank JSON Schema

### TASK REQUIREMENT:
1. Map all relevant text to the JSON Schema.
2. Output in two sections:
   - <think>: Reasoning
   - <answer>: Filled JSON

### STRICT RULES:
1. Provide both <think> and <answer>.
   - If minimal reasoning, say: \"Direct mapping from text to schema.\"
2. Map text exactly to the JSON Schema (no omission/alteration).
3. Preserve hierarchy (ROOT $\to$ SECTION $\to$ SUBSECTION $\to$ DETAIL\_N)
4. Correctly set attributes (id, idc, idx, level\_type, component\_type, etc.).
5. JSON Format:
   - Escape quotes as \"
   - Replace newlines with \\n
   - No trailing commas
   - Only double quotes
6. Explain key decisions in <think>.

### IMPORTANT:
If <think> or <answer> is missing, response is incomplete.\"),axis=1)
```

 doi:10.20944/preprints202502.1390.v2

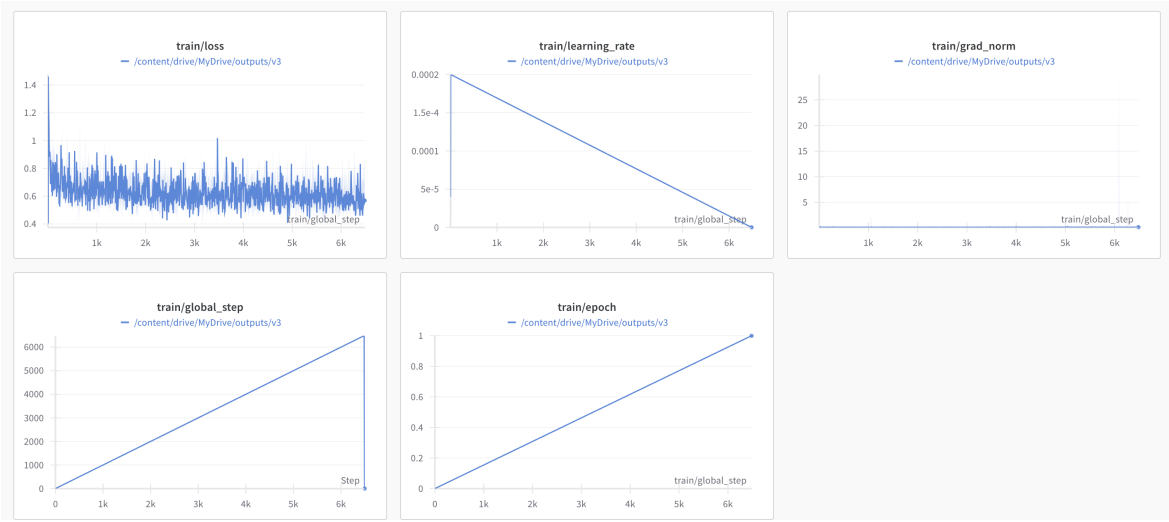

Figure 3. SFT Training Metrics.

4. Evaluation

We evaluated five models: ThinkJSON, Original DeepSeek R1 (671B), Distilled DeepSeek R1 (Qwen-1.5B / Qwen-7B) and Gemini 2.0 Flash (70B) which specializes on structured output generation [17], on a structured data extraction benchmark involving 6.5K rows. Each row was processed to produce or omit a valid JSON object, and we measured metrics including:

- **Rows With No Output**: Number of rows for which the model produced no structured output.
- **Rows With Valid JSON:** Number of rows resulting in syntactically valid JSON objects.
- **Mean Match Percentage:** Average proportion of fields correctly mapped.
- **Mean Noise Percentage:** Average proportion of extraneous or malformed tokens within the extracted JSON.

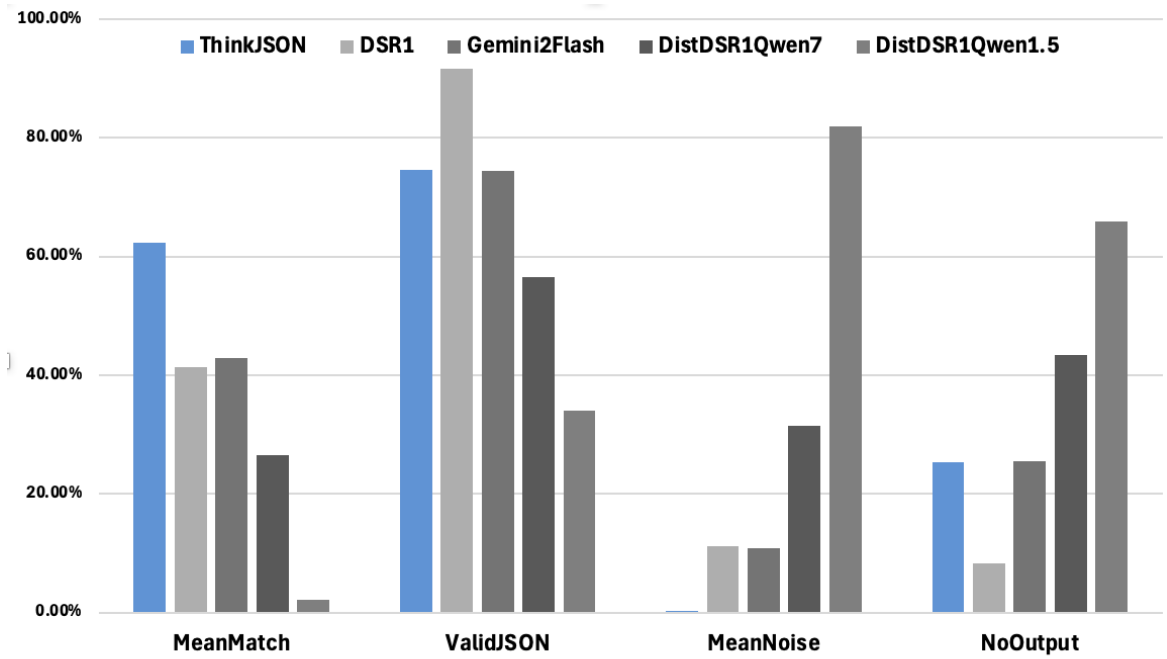

Figure 4. Performance Comparison.

As illustrated, ThinkJSON yields strong results, with a 62.41% mean match (highest of all five models) and the lowest 0.27% mean noise, indicating minimal extraneous output. The Original DeepSeek R1 also achieves relatively high valid-JSON coverage but shows a lower mean match (41.43%) and higher noise (11.14%). The two distilled variants of DeepSeek R1—Qwen 1.5B and Qwen 7B—exhibit weaker performance overall, with high

 doi:10.20944/preprints202502.1390.v2

rates of no extracted JSON or large amounts of noise. Meanwhile, Gemini 2.0 Flash achieves a midrange mean match of 42.88% but suffers from significant noise at 10.86%. These findings underscore the effectiveness of our structured reasoning approach in producing concise, schema-valid outputs.

5. Discussion and Future Direction

Our experimental findings confirm that the reasoning-driven, schema-constrained generation pipeline is both broadly applicable—capable of handling diverse reasoning tasks beyond purely mathematical or scientific domains—and budget-conscious, as it requires comparatively moderate GPU resources and a modest dataset of reasoning examples. This balanced approach addresses a critical need in bio-manufacturing compliance , where AI systems must deliver not only correct structure but also reliable, domain-specific reasoning to meet regulatory standards [18,19].

The hallmark of our framework is integrating compliance considerations at the core of the generation process. Rather than relying on prompt-based or post-hoc solutions, our pipeline combines schema adherence objectives with iterative reasoning loops, thus reducing the need for manual oversight. This focus on strict output validation resonates with bio-manufacturing's regulatory requirements—where precise field mappings and hierarchical consistency are crucial for electronic batch records and industry audits.

While we have employed a 1.5B-parameter foundation model, our method is readily scalable to bigger backbones (e.g., 7B parameters). Larger models could potentially yield richer context interpretation and more robust handling of rare or domain-specific phenomena. In future work, we plan to explore how increased capacity further expands the set of reasoning scenarios the model can tackle while maintaining resource efficiency—a pivotal benefit in industrial adoption.

Overall, this reinforcement + fine-tuning pipeline for structured text generation offers a flexible, compliance-aware approach that applies universal reasoning principles—spanning regulated bio-manufacturing tasks and broader domains—without incurring prohibitive computational overhead. This synergy of versatility and cost-effectiveness positions our method as a significant step forward in delivering reliable, schema-adherent AI-driven solutions.

References

1. Labant, M. Smart Biomanufacturing: From Piecemeal to All of a Piece. https://www.genengnews.com/topics/bioprocessing **2025**.
2. et all, M.L. "We Need Structured Output": Towards User-centered Constraints on Large Language Model Output. *https://lxieyang.github.io/assets/files/pubs/llm-constraints-2024/llm-constraints-2024.pdf* **2024**.
3. et all, S.G. Generating Structured Outputs from Language Models: Benchmark and Studies. *https://arxiv.org/html/2501.10868* **2025**.
4. et all, C.S. StructuredRAG: JSON Response Formatting with Large Language Models. *https://arxiv.org/abs/2408.11061* **2024**.
5. Souza, T. Taming LLMs **2024**.
6. et all, D.L. Large Language Model-Driven Structured Output: A Comprehensive Benchmark and Spatial Data Generation Framework. *https://www.mdpi.com/2220-9964/13/11/405* **2024**.
7. et all, Z.W. Verifiable Format Control for Large Language Model Generations. *https://arxiv.org/html/2502.04498* **2025**.
8. et all, Y.D. XGRAMMAR: FLEXIBLE AND EFFICIENT STRUCTURED GENERATION ENGINE FOR LARGE LANGUAGE MODELS. *https://arxiv.org/pdf/2411.15100* **2024**.
9. Brandon T. Willard, R.L. Efficient Guided Generation for Large Language Models. *https://arxiv.org/pdf/2307.09702* **2023**.
10. et all, A.M. Self-Refine: Iterative Refinement with Self-Feedback. *https://arxiv.org/abs/2303.17651* **2023**.
11. et all, Y.W. Self-Instruct: Aligning Language Models with Self-Generated Instructions. *https://arxiv.org/abs/2212.10560* **2022**.
12. Team, Q. Qwen2.5: A Party of Foundation Models. *https://qwenlm.github.io/blog/qwen2.5* **2024**.
13. DeepSeek-AI. Deepeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *https://arxiv.org/pdf/2501.12948* **2025**.
14. at all, Z.S. Pushing the Limits of Mathematical Reasoning in Open Language Models. *https://arxiv.org/pdf/arXiv:2402.03300* **2024**.
15. et all, P.W. Math-Shepherd: A Labelfree Step-by-Step Verifier for LLMs in Mathematical Reasoning. *https://arxiv.org/pdf/2312.08935* **2023**.

 doi:10.20944/preprints202502.1390.v2

16. et all, C.D. Reinforcement Learning Can Be More Efficient with Multiple Rewards. *https://proceedings.mlr.press/v202/dann23a.html* **2023**.
17. team, G.A. Generate Structured Output with the Gemini API. *https://ai.google.dev/gemini-api/docs/structured-output?lang=python* **2025**.
18. et all, N.E. AI Maturity Model for GxP Application: A Foundation for AI Validation. *https://ispe.org/pharmaceutical-engineering/march-april-2022/ai-maturity-model-gxp-application-foundation-ai* **2022**.
19. et al., V.A. An Overview on Pharmaceutical Regulatory Affairs Using Artificial Intelligence. *https://www.ijpsjournal.com/article/An+Overview+on+Pharmaceutical+Regulatory+Affairs+Using+Artificial+Intelligence+* **2025**.