CONSTRAINT TAX IN OPEN-WEIGHT LLMS: AN EMPIRICAL STUDY OF TOOL CALLING SUPPRESSION UNDER STRUCTURED OUTPUT CONSTRAINTS

Fangzheng Li[1,2,†]
[1]Focus AI Center, Focus Technology Co., Ltd.
[2]Nanjing University of Science and Technology
lifangzheng@focuschina.com
Fangzheng_Li@njust.edu.cn

Aimin Zhang[1,†*]
[1]Focus AI Center, Focus Technology Co., Ltd.
zhangaimin@focuschina.com

Chen Lv[1]
[1]Focus AI Center, Focus Technology Co., Ltd.
lvchen1018@focuschina.com

ABSTRACT

Tool Calling and Structured Output are two core capabilities of modern Agent systems, yet their interaction under joint deployment conditions remains insufficiently understood. This paper reports a reproducible phenomenon observed in a production Agent system: when Tool Calling and JSON Schema constraints are simultaneously enabled, multiple open-weight models cease invoking tools despite maintaining high schema compliance. We refer to this behavior as **Tool Suppression**.

Through controlled experiments across multiple model families and deployment settings, we consistently reproduce Tool Suppression under joint constraints, while tool execution and schema compliance remain functional when evaluated independently. Further analysis reveals that JSON Schema constraints are compiled into grammar-based token masks, causing tool-call tokens to become unreachable during decoding. This provides an implementation-level explanation for the observed behavior.

To interpret the phenomenon, we formulate the **Constraint Priority Inversion (CPI)** hypothesis, which suggests that schema satisfaction may dominate action-selection behavior under multiple simultaneous constraints. We present CPI as a behavioral hypothesis consistent with the observed evidence rather than a verified internal mechanism.

To mitigate the problem, we propose **Transparent Two-Pass Execution**, an inference-time strategy that decouples tool execution from schema-constrained response generation. Experimental results show that this approach restores tool invocation while preserving structured output guarantees without requiring model retraining.

These findings suggest that evaluating tool use and structured output separately may overlook important reliability issues in production Agent systems. Code, data, and docs will be released at https://github.com/Fzsama/Constrain-Tax-26-06.git.

Keywords Constraint Tax · Tool Suppression · Constraint Priority Inversion · Large Language Models · Agent Systems

1 Introduction

1.1 Tool-Augmented LLM Agents

As Large Language Models (LLMs) evolve from text-only interfaces to action-taking agents, tool augmentation has become a standard paradigm for enabling models to interact with external environments. With the development of

*Corresponding author: zhangaimin@focuschina.com

standardized protocols such as MCP and OpenAI-compatible tool-calling APIs, tool execution has been widely adopted in production Agent systems.

Meanwhile, Structured Output has become another critical capability in production Agent systems. Rather than returning unconstrained natural language, deployed Agents are often required to generate responses that satisfy predefined JSON schemas for downstream parsing, workflow automation, API integration, and multi-Agent coordination.

As a result, Tool Calling and Structured Output are increasingly activated simultaneously within the same execution pipeline. A typical production workflow may require an Agent to first acquire external information through tools and then organize the retrieved information into a schema-compliant response.

Although both capabilities have been extensively studied individually, relatively little attention has been paid to their interaction under joint deployment conditions. Existing evaluations typically assess tool use, structured generation, and task completion separately, implicitly assuming that capabilities that function correctly in isolation will continue to function correctly when combined.

Whether this assumption holds in production Agent environments remains insufficiently understood and motivates the investigation presented in this paper.

1.2 Constraint Tax and an Unexplored Agent Failure Mode

Recent research has begun to focus on the unintended effects of structured-output constraints on model behavior. This line of work has observed that requiring models to generate outputs in specific formats can degrade answer quality, reduce factual accuracy, and increase token usage. This phenomenon has been described as a "Constraint Tax" on model performance in text-generation tasks.

This finding indicates that structured-output constraints are not computationally neutral; they impose measurable costs on model behavior beyond format compliance.

Agent systems introduce a fundamentally different execution setting. Unlike traditional text-generation tasks, Agent workflows require models to make decisions about whether external actions should be executed, when they should be executed, and how externally acquired information should be incorporated into the final response.

Consequently, when Tool Calling and Structured Output constraints coexist, structured generation constraints may influence not only the form of the final response but also the execution process itself. One possible mechanism is that decoding-level constraint enforcement—such as grammar-based token masking—may render tool-call tokens unreachable during generation, thereby preventing tool execution at the output layer.

This observation motivates an open research question:

> When Tool Calling and Structured Output constraints are simultaneously enabled, do they interact in ways that affect Agent execution behavior? If so, at what level does this interaction occur—model preference, decoding constraint, or both?

To our knowledge, this question has received limited empirical investigation. Existing Constraint Tax studies primarily focus on answer quality degradation under structured-output requirements, whereas the potential impact on Agent action execution and the underlying implementation-level mechanisms remain largely unexplored.

This paper investigates that question through controlled experiments, inference-stack tracing, and production-system observations.

1.3 Observation from a Production Agent System

The investigation was initially motivated by an unexpected observation during the deployment of an Agent system in a production environment. The system was configured to use an open-weight model with both Tool Calling and Structured Output constraints enabled, a configuration that is common in production settings.

Under this joint-constraint configuration, the Agent repeatedly failed to invoke external tools even when tasks explicitly required external information acquisition. Tool call events were not generated, and the final response was produced without incorporating any external data.

In contrast, when the Structured Output constraint was disabled while keeping all other conditions unchanged, the same model successfully invoked tools and completed the tasks.

This behavior was unexpected from a system-design perspective because the only modification involved the presence of a schema constraint, while model weights, prompts, tool definitions, and task requirements remained identical.

The key observation was that the model appeared to satisfy the schema requirement while silently bypassing the tool execution step entirely. This pattern was initially treated as an implementation anomaly but persisted across repeated trials.

The observation described above motivates the central question investigated in this paper. Both Tool Calling and Structured Output generation remain functional when evaluated independently. However, under joint deployment conditions, tool execution behavior may disappear despite the continued presence of tool definitions, unchanged task requirements, and successful schema generation.

Understanding the origin of this behavioral inconsistency forms the primary objective of the study.

1.4 Contributions

This paper makes the following contributions:

- **Identification and characterization of Tool Suppression.**
  We report a reproducible failure pattern observed in a production Agent system in which multiple evaluated open-weight models cease invoking tools when Tool Calling and Structured Output constraints are simultaneously enabled. We define this behavior as *Tool Suppression* and characterize its observable properties through controlled experiments.
- **Root cause localization to grammar-based constrained decoding.**
  Through systematic tracing of the inference stack, we identify that JSON Schema constraints are compiled into grammar-based token masks that render tool-call tokens unreachable during decoding. This provides a concrete implementation-level explanation for the observed suppression phenomenon.
- **Formulation of the Constraint Priority Inversion (CPI) hypothesis.**
  Based on behavioral evidence collected during diagnostic experiments, we introduce CPI as a possible interpretation of the observed suppression pattern. CPI is presented as a behavioral hypothesis consistent with the observed evidence rather than a verified internal mechanism.
- **Proposal and validation of Transparent Two-Pass Execution.**
  We propose an inference-time mitigation strategy that separates tool execution from schema-constrained response generation. Experimental results show that the approach restores tool invocation behavior while preserving structured output guarantees.
- **Development of a Tool Suppression behavioral taxonomy.**
  We further summarize recurring suppression patterns into a behavioral taxonomy (TS-A to TS-E), providing a descriptive framework for analyzing suppression behavior across different models and deployment settings.

2 Background

2.1 Tool Calling in LLM Agents

Tool Calling refers to the ability of language models to interact with the external environment by generating structured function call requests. Unlike traditional language models that rely solely on parametric knowledge, tool-augmented Agents can access external resources such as search engines, databases, code interpreters, and enterprise internal systems, thereby completing tasks that require real-time information acquisition, environmental perception, or external execution [1, 2, 3]. Therefore, tool calling has become one of the important features distinguishing modern Agent systems from pure dialogue models.

Existing Agent frameworks commonly treat tool calling as an execution process composed of multiple stages, including Task Understanding, Tool Planning, Tool Selection, and Tool Execution. In this process, models not only need to determine whether external information is required but also need to generate correct tool call actions and utilize returned results for subsequent reasoning. In recent years, extensive research has been conducted on tool learning capabilities, including evaluation metrics such as Tool Selection Accuracy, Task Completion Rate, and Tool Use Success Rate [3, 4].

Existing research has identified certain limitations of open-weight models in tool use. Shen et al. [5] pointed out that small-scale language models exhibit significant disadvantages in tool learning tasks and proposed the "Weak Tool Learners" phenomenon. Wang et al. [6] further demonstrated that incorporating failure cases during Agent fine-tuning can effectively improve tool use behavior. These works indicate that tool calling capability itself has become an important research direction in Agent research.

However, existing research primarily focuses on whether tool calling capability exists and whether tool calling is correctly executed, with less attention to the interaction effects between tool calling capability and other system constraints. Particularly in scenarios where tool calling and structured output constraints coexist, whether models can maintain normal tool execution behavior remains systematically understudied. The Tool Suppression phenomenon studied in this paper occurs precisely in this joint constraint scenario.

2.2 Structured Output Generation

Structured Output Generation refers to the technical paradigm of constraining language model output format through predefined Schemas, enabling generated results to be stably parsed and consumed by programs. Unlike traditional natural language responses, structured output requires models to organize content according to specific data structures, typically manifested as JSON objects, function parameters, or text conforming to formal grammar constraints. As LLMs are increasingly integrated into production systems, structured output has become a fundamental capability for Agents, workflow orchestration systems, and multi-Agent collaboration frameworks.

Current mainstream implementation approaches mainly include two categories. The first is API-layer constraints, such as the `response_format` mechanism in OpenAI-compatible interfaces, which guides models to generate output structures meeting requirements through Schema descriptions. The second is Constrained Decoding at the decoding layer, which restricts the search space during token generation through grammar constraints, state machines, or finite automata, such as `guided_json` in SGLang and Grammar-based Decoding mechanisms in vLLM. Regardless of the implementation approach, the core objective is to improve the determinability and parsability of output format.

Existing research generally treats structured output as an engineering reliability mechanism. Liu et al. [7] found through user surveys that structured output requirements have become a prevalent demand in industrial scenarios; Deng et al. [8] further demonstrated that there exists a significant coupling relationship between output format and task solving process, and directly imposing format constraints on the generation process may affect the model's ability to complete the task itself.

This finding resonates with Constraint Tax research. Growing evidence suggests that structured output constraints do not only act on the final output stage but may alter the model's resource allocation and decision behavior during reasoning. When models simultaneously need to satisfy content generation objectives and format constraint objectives, competition may arise between the two types of objectives. However, existing research primarily focuses on the impact of structured output on answer quality, reasoning capability, and format compliance rate, with insufficient empirical research on whether it further intervenes in action selection behavior during Agent execution.

The Tool Suppression phenomenon focused on in this paper occurs precisely in this context: structured output constraints no longer merely affect "how to answer" but may further affect whether the Agent "executes actions."

2.3 Constraint Tax

In recent years, with the widespread application of structured output in industrial scenarios, researchers have begun to focus on the impact of format constraints on model behavior itself. Ray [9] first systematically proposed the **Constraint Tax** concept to describe the performance cost that language models pay when satisfying structured output requirements. Experimental results demonstrate that when strict Schema constraints are imposed on the generation process, models can achieve higher format compliance rates, but their answer accuracy may significantly decrease.

This phenomenon indicates that structured output constraints are not free engineering capabilities. Traditional views typically assume that Schema constraints only affect output form without affecting the model's ability to solve tasks themselves. However, Constraint Tax research shows that format constraints actually participate in decision competition during the model generation process: models not only need to complete task solving but also simultaneously satisfy format compliance objectives.

From a more general perspective, the Constraint Tax can be understood as a **Goal Competition** phenomenon. When models face multiple simultaneous optimization objectives, their limited reasoning resources and generation capabilities need to be allocated across different objectives [10]. For example, models may simultaneously need to:

- Generate correct content (Content Correctness);
- Satisfy format requirements (Format Compliance);
- Invoke external tools (Tool Execution);
- Control generation length (Length Constraints).

Ideally, these objectives should be simultaneously satisfied; however, in actual models, different objectives may form competitive relationships. When one objective receives excessively high priority, other objectives may be partially or even completely sacrificed. The Constraint Tax is precisely a typical manifestation of this goal competition in structured output scenarios.

Existing research primarily examines the impact of the Constraint Tax on answer accuracy and reasoning capability, so its manifestation is typically described as "format-correct but answer-incorrect." However, for Agent systems, content generation is only the final stage in the execution pipeline. In tool-augmented Agents, models also need to complete action decision processes such as tool planning, tool selection, and tool execution. Therefore, a natural but insufficiently studied question is: do structured output constraints further affect the Agent's action selection behavior, rather than merely affecting final answer quality?

This paper's research demonstrates that such influence indeed exists. In Agent scenarios, the Constraint Tax no longer only manifests as content quality degradation but may further evolve into systematic absence of tool execution behavior. We refer to this phenomenon as **Tool Suppression** and propose Constraint Priority Inversion (CPI) as a possible explanation mechanism in subsequent sections.

2.4 Grammar-Constrained Decoding

Grammar-constrained decoding is a technique used in modern inference frameworks to enforce structured output formats at the token-generation level. Rather than relying solely on model prompting or post-hoc validation, this approach actively constrains the set of tokens that the model may generate at each decoding step.

Frameworks such as SGLang and vLLM integrate grammar-based constraint engines including xgrammar and Outlines. These engines compile user-provided JSON Schemas into finite-state machines (FSMs) that define the set of valid token sequences. At each decoding step, the FSM determines which tokens are permissible given the current generation state, and the inference framework applies a vocabulary mask to set impermissible token logits to $-\infty$, effectively making them impossible to sample.

This mechanism ensures format compliance with high reliability, which is particularly valuable in production environments where downstream systems require deterministic parsing. However, the application of grammar constraints is not limited to the final output stage; it operates at the token level throughout generation.

The effect of grammar-constrained decoding on token availability has been characterized in prior work on constrained generation. For example, Garbacea and Mei [11] discuss the structural challenges introduced by vocabulary restrictions in constrained language generation. Suresh et al. [12] further examine the interaction between constrained inference and model behavior. These works suggest that grammar constraints can fundamentally alter token availability during generation, which may have implications beyond simple formatting compliance.

In the context of Agent systems that use XML-style tool-call formats, this token-level constraint mechanism introduces an important consideration: when a JSON Schema grammar is active, any token sequence that deviates from the grammar—including tool-call tags—becomes unreachable. This creates a potential tension between structured output requirements and tool execution capability at the decoding layer.

2.5 Agent Execution Pipelines

Tool-augmented Agents are typically modeled as execution pipelines composed of multiple consecutive decision stages. From a functional perspective, Agents not only need to generate natural language responses but also need to complete a series of intermediate steps such as task analysis, tool decision-making, and external environment interaction. Therefore, the Agent execution process is essentially a Multi-stage Decision Making process rather than a pure text generation task.

Although existing Agent frameworks differ in specific implementations, their execution logic can generally be abstracted as the following pipeline:

$$\texttt{User Query} \rightarrow \texttt{Task Understanding} \rightarrow \texttt{Tool Planning} \rightarrow \texttt{Tool Execution} \rightarrow \texttt{Response Generation}$$

Where:

- **Task Understanding**: The model identifies user intent, task objectives, and required information;
- **Tool Planning**: The model determines whether external tools are needed and formulates tool usage plans;
- **Tool Execution**: The model initiates tool calls and receives external return results;

- **Response Generation**: The model generates final responses based on tool results.

Existing Agent evaluation benchmarks mostly focus on tool planning and tool execution capabilities. For example, works such as ToolBench, API-Bank, and BMTools primarily evaluate whether models can correctly select tools and whether they can complete tasks [3, 13]. Meanwhile, structured output-related research mainly focuses on format compliance during the response generation stage.

However, this evaluation approach implicitly assumes that different stages in the Agent execution pipeline are independent, i.e., constraints acting on the response generation stage do not affect the decision process of preceding stages. In existing literature, this assumption is rarely explicitly validated.

The observations in this paper indicate that this assumption may not hold. When structured output constraints are imposed on the response generation stage, their influence may propagate backward to the tool execution stage, thereby changing the Agent's action selection results. In other words, a constraint that originally belongs to the output level may affect execution-level behavior.

Based on this perspective, this paper positions Tool Suppression as a Cross-Stage Interference phenomenon in the Agent execution pipeline: task understanding and tool planning processes still complete normally, but the tool execution stage is systematically skipped, while the response generation stage continues to produce format-compliant output. Subsequent sections will further demonstrate that this behavioral pattern is fundamentally different from traditional tool capability deficiency.

3 Problem Definition

3.1 Constraint Tax in Agent Contexts

Ray [9] defines the Constraint Tax as the performance cost introduced when language models are required to satisfy additional structured-output constraints under fixed model and task conditions. Existing studies primarily observe this cost through reduced answer quality, degraded reasoning performance, or increased proportions of format-compliant but content-incorrect outputs.

From a broader perspective, Constraint Tax can be viewed as a behavioral consequence of operating under multiple simultaneous objectives. In addition to solving the primary task, models may also be required to satisfy format constraints, safety requirements, workflow specifications, or other deployment-related conditions.

For Agent systems, these objectives coexist with action-oriented behaviors such as tool planning and tool execution. Consequently, structured-output constraints may influence not only response generation but potentially other stages of Agent execution as well.

The present study investigates this possibility in the context of Tool Calling and Structured Output constraints operating simultaneously within production Agent environments. As we will show, the interaction may originate not only from model-level behavioral tendencies but also from decoding-level constraint enforcement mechanisms.

3.2 Tool Suppression

We define the core phenomenon studied in this paper as **Tool Suppression**.

> **Tool Suppression** refers to the phenomenon where, in tasks with genuine tool requirements, models possess tool calling capability and can identify tool usage needs, but systematically abandon tool execution behavior under joint constraint conditions.

This definition includes three necessary conditions:

1. **Tool Requirement**. The task itself requires accessing external information beyond the model's parameters, or requires solving through external tools.
2. **Tool Capability**. The model can normally complete tool calling under conditions without joint constraints.
3. **Execution Omission**. The model ultimately does not initiate tool calls but directly generates responses.

It is worth noting that Tool Suppression is fundamentally different from traditional Tool Incapability. In Tool Incapability scenarios, models cannot correctly identify tool requirements or cannot generate valid tool calls; whereas in Tool Suppression scenarios, models possess the corresponding capability, but this capability is not activated under specific constraint conditions.

From the Agent execution pipeline perspective, Tool Suppression manifests as a Selective Failure:

Task Understanding ✓

Tool Planning ✓

Tool Execution ×

Response Generation ✓

That is, models can understand tasks, identify information gaps, and generate final responses, but exhibit systematic skipping during the tool execution stage.

This phenomenon has a similar structure to the "format-correct but answer-incorrect" in traditional Constraint Tax: both manifest as constraint conditions causing degradation of certain core capabilities. However, in Tool Suppression, the degraded object is no longer answer quality but the Agent's action execution capability. Therefore, we regard it as a Behavior-Level Manifestation of the Constraint Tax in Agent systems.

Subsequent sections will further use Tool Invocation Rate (TIR) and Suppression Rate (SR) to quantitatively characterize this phenomenon.

3.3 Tool Invocation Rate

To quantitatively characterize tool calling behavior, we define the **Tool Invocation Rate (TIR)** as follows:

$$TIR = \frac{N_{\text{tool}}}{N_{\text{total}}} \tag{1}$$

Where:

- N_{tool} represents the number of requests that successfully initiated at least one tool call;
- N_{total} represents the total number of requests.

Therefore, $TIR \in [0, 1]$, with higher values indicating that the model is more inclined to execute tool calling behavior.

It should be emphasized that TIR measures **Tool Execution Behavior**, not tool calling quality. As long as the model successfully triggers at least one valid tool call, it counts as one tool execution event; whether the tool returns correct results and the quality of the final answer do not affect TIR statistics.

This design is consistent with this paper's research objectives. Since Tool Suppression focuses on "whether to execute tools" rather than "whether to correctly use tools," TIR can directly reflect whether Agent execution behavior is suppressed.

Ideally, for a task set with genuine tool requirements, we should have:

$$TIR_{\text{baseline}} \approx 1 \tag{2}$$

That is, the model can stably execute tool calls.

If after introducing joint constraints we observe:

$$TIR_{\text{constrained}} \ll TIR_{\text{baseline}} \tag{3}$$

This indicates that tool execution behavior is significantly affected. In extreme cases, when

$$TIR_{\text{constrained}} = 0 \tag{4}$$

This means the model completely stops tool calling behavior, manifesting as Complete Tool Suppression.

3.4 Suppression Rate

Although TIR can reflect the model's tool execution behavior, its absolute value is difficult to directly characterize the additional impact caused by joint constraints. For example, different models may inherently have different tool calling tendencies, so comparing only TIR under constraint conditions cannot accurately measure the severity of Tool Suppression.

To this end, we further define the **Suppression Rate (SR)**:

$$SR = 1 - \frac{TIR_{\text{constrained}}}{TIR_{\text{baseline}}} \tag{5}$$

Where:

- TIR_{baseline} represents the tool invocation rate without joint constraints;
- $TIR_{\text{constrained}}$ represents the tool invocation rate with joint constraints.

When $TIR_{\text{baseline}} > 0$,

$$SR \in [0, 1]$$

Higher values indicate stronger suppression of tool calling behavior.

According to this definition:

- When

  $$TIR_{\text{constrained}} = TIR_{\text{baseline}}$$

  then

  $$SR = 0$$

  indicating that joint constraints have no observable impact on tool execution behavior;
- When

  $$0 < TIR_{\text{constrained}} < TIR_{\text{baseline}}$$

  then

  $$0 < SR < 1$$

  indicating Partial Tool Suppression;
- When

  $$TIR_{\text{constrained}} = 0$$

  then

  $$SR = 1$$

  indicating Complete Tool Suppression.

Compared to directly using TIR, SR can eliminate the influence of the model's inherent tool calling tendency, making it more suitable as a standardized indicator for cross-model comparison and constraint effect analysis.

In summary, Tool Suppression defines the core phenomenon studied in this paper, while TIR and SR provide quantifiable measurement methods from behavioral and effect levels respectively. In subsequent experiments, this paper uses TIR to characterize tool execution behavior and SR to measure the degree of suppression caused by joint constraints.

3.5 Taxonomy of Tool Suppression Behaviors

This taxonomy is constructed based on three dimensions:

- **Tool Awareness**: Whether the model identifies the need for external tools;
- **Tool Execution**: Whether the model actually initiates tool calls;
- **Content Generation Strategy**: How the model constructs responses when unable to obtain tool results.

Based on the above dimensions, we propose the following taxonomy.

TS-A: Empty Compliance The model's generated response fully complies with the required schema, but key fields contain null values, default values, or placeholders.

```
{"recommendations": [], "key_findings": []}
```

This pattern indicates that the model minimizes content generation workload while prioritizing format compliance. This phenomenon was mainly observed in GPT-OSS-20B.

TS-B: Simulated Retrieval The model generates information that appears to originate from external tools without ever calling any tools.

```
{"buyer_background":
"Simulated Web Search Results"}
```

Compared to TS-A, this pattern actively fabricates content, generating data that appears to be retrieval results. Since its output is superficially plausible, downstream systems have greater difficulty detecting it. This behavior was mainly observed in Qwen3.5-122B-A10B.

TS-C: Intent Without Action The model explicitly expresses the need to use tools but ultimately fails to execute the corresponding operations.

```
{
    "need_search": true,
    "tool_requirements": [...]
}
```

This category is particularly informative because it reveals a behavioral separation between explicit recognition of tool requirements and the absence of actual tool execution.

In the evaluated examples, models occasionally produced outputs indicating awareness of external information requirements while no corresponding tool calls were observed. This pattern motivates further investigation into how action-selection decisions are made under joint constraints—specifically, whether the failure originates from model-level preferences, decoding-level constraints, or their interaction.

TS-D: Tool-Free Hallucination The model neither calls tools nor acknowledges insufficient information, but directly generates content lacking substantiation. This pattern represents typical tool-free hallucination and is the highest-risk failure mode in deployment environments. Since the generated content is usually fluent and seemingly plausible, it may mislead users or downstream systems.

TS-E: Frozen Required Tool Even when explicitly forced to use tools (e.g., through `tool_choice="required"` settings), the model still fails to call tools. This behavior indicates that Tool Suppression cannot always be explained by incorrect assessment of tool requirements. Instead, it may reflect deeper failures in the action selection mechanism. Table 1 summarizes these five categories of Tool Suppression behavior. The taxonomy presented above is intended as a descriptive framework rather than a causal explanation.

The five categories capture different observable manifestations of Tool Suppression across models and deployment settings. Some categories primarily reflect omission behavior (TS-A, TS-E), while others involve content-generation strategies that compensate for missing tool results (TS-B, TS-D). TS-C is distinctive because it exhibits explicit acknowledgement of tool requirements without corresponding execution behavior, motivating deeper investigation into the underlying constraint mechanisms. This taxonomy provides a common vocabulary for analyzing suppression behavior in subsequent experiments.

Table 1: Taxonomy of Tool Suppression Behaviors

Category	Output Quality	Tool Awareness	Tool Execution	Deceptiveness
TS-A Empty Compliance	Low	Absent	Blocked	Low
TS-B Simulated Retrieval	Medium	Absent	Blocked	Medium
TS-C Intent Without Action	Medium to High	**Present**	Blocked	Low
TS-D Tool-Free Hallucination	High	Absent	Blocked	**High**
TS-E Frozen Required Tool	Medium	Absent	Blocked (Forced)	Low

4 Experimental Setup

4.1 Research Questions

This study investigates whether the coexistence of Tool Calling and Structured Output constraints introduces a systematic degradation in tool execution behavior for open-weight LLM agents.

Rather than evaluating general reasoning quality, we focus specifically on action execution under joint constraints.

The study addresses four research questions:

- **RQ1 (Reproducibility)**: Can tool suppression be consistently reproduced across different open-weight model families under identical task settings?
- **RQ2 (Constraint Interaction)**: Does the suppression behavior originate from the interaction between Tool Calling and Structured Output constraints rather than from either capability individually?
- **RQ3 (Alternative Explanations)**: Can the observed phenomenon be explained by inference frameworks, deployment environments, model scale, quantization methods, schema design, or tool invocation settings?
- **RQ4 (Mitigation)**: Can production-oriented engineering strategies restore tool execution behavior while preserving structured output guarantees?

The above research questions correspond to four levels: phenomenon verification, scale effect analysis, causal exclusion verification, and engineering mitigation exploration.

4.2 Controlled Experimental Design

To isolate the effect of joint constraints, we adopt a controlled three-condition design. The specific configurations and objectives for each condition are defined as follows:

- **T1 (Tool-Only Baseline)**: Tools = ON, Response Format = OFF. This condition measures the model's baseline ability to invoke tools when no structured output constraints are present.
- **T2 (Joint Constraint Condition)**: Tools = ON, Response Format = ON. This condition represents the production setting where tool execution and JSON Schema compliance are simultaneously required.
- **T3 (Schema-Only Control)**: Tools = OFF, Response Format = ON. This condition verifies whether models can independently satisfy structured output constraints without tool execution requirements.

Table 2: T1/T2/T3 Experimental Conditions

Condition	Tools	Response Format	Purpose
T1 (Baseline)	ON	OFF	Measure baseline tool calling capability
T2 (Joint Constraint)	ON	ON	Detect Tool Suppression phenomenon
T3 (Schema Control)	OFF	ON	Verify independent Schema compliance capability

The three experimental conditions share identical System Prompt, User Prompt, tool definitions, and output parsing logic, with only the `tools` and `response_format` API parameters adjusted. Through this design, observed behavioral differences can be attributed to joint constraints themselves rather than Prompt changes or task differences.

Each model runs 5 test rounds independently under each condition with fixed parameter settings:

- `temperature = 0.5`

- `stream = true`
- `max_completion_tokens = 4096`

According to the definitions in Section 3, Tool Invocation Rate (TIR) and Suppression Rate (SR) are recorded as primary evaluation metrics during experiments.

4.3 Task Set Construction

A common threat in tool-use evaluation is over-reliance on a single prompt instance.

To reduce this risk, evaluation tasks were sampled from representative production scenarios that require external information acquisition.

The task set covers five business-oriented categories:

- Buyer Background Analysis
- Company Verification
- Market Intelligence Search
- Product Knowledge Retrieval
- Compliance and Risk Investigation

All tasks require information that is unavailable in model parameters alone and therefore necessitate at least one external tool invocation.

The same task set is executed across all T1/T2/T3 conditions, ensuring that observed differences originate from constraint configurations rather than task variation.

Table 3: Task Categories and Example Requirements

Category	**Example Requirement**
Buyer Background	Analyze a buyer company
Company Verification	Verify company legitimacy
Market Search	Search target market demand
Product Knowledge	Retrieve product specifications
Compliance Check	Investigate regulatory risks

For detailed task design, tool definitions, and schema specifications, see Appendix A.

4.4 Model Selection

To verify the universality of the Tool Suppression phenomenon, this paper selects seven model instances from different model families, parameter scales, and deployment methods, covering both closed-source baseline models and open-weight models.

Table 4: Evaluated Models

Model	Parameters	Architecture	Deployment
GPT-5.4-mini	—	Proprietary	Cloud
Qwen3.6-35B-A3B	35B	MoE (3B Active)	Local
Qwen3.5-122B-A10B	122B	MoE (10B Active)	Local
GPT-OSS-20B	20B	MoE (3.6B Active)	Local
Nemotron 3 Super	120B	Hybrid MoE-Mamba	Local
Qwen3.5-397B-A17B	397B	MoE (17B Active)	Cloud
Qwen3-VL-235B-Thinking	235B	MoE + Vision-Language	Cloud

Model selection follows these principles:

1. Covering different parameter scales (20B to 397B);
2. Covering different model architectures (MoE [14] and Hybrid architectures);
3. Covering local deployment and cloud deployment environments;
4. Including one closed-source commercial model as a reference baseline.

This model matrix supports verification of RQ1 and RQ2 and provides a basis for subsequent cross-model comparisons. The purpose of this model selection is not to represent all open-weight LLMs, but to evaluate whether the observed behavior can be reproduced across multiple model families, parameter scales, deployment modes, and inference stacks. The complete experimental environment configuration, including inference framework settings and parser configurations, is provided in Appendix A.

4.5 Evaluation Protocol

All experiments are executed through OpenAI-compatible API interfaces to ensure consistency in invocation methods across different models. Streaming generation mode is adopted during testing, with real-time parsing of tool call events and text content returned by models.

According to the definitions in Section 3, this paper adopts the following core metrics:

- **Tool Invocation Rate (TIR)**: The proportion of sessions that contain at least one valid tool call.
- **Suppression Rate (SR)**: The relative reduction of tool invocation behavior under joint constraints.
- **JSON Compliance Rate (JCR)**: The proportion of responses that satisfy the required schema.
- **Average Tool Calls per Session (ATC)**: The average number of tool invocations generated within a successful session.
- **End-to-End Success Rate (ESR)**: The proportion of sessions that both invoke tools and produce valid structured outputs.

For each model, 5 test rounds are run independently under each experimental condition. Test tasks remain consistent with fixed random sampling parameters to reduce the impact of random fluctuations on results.

During streaming responses, the experimental framework automatically records the following information:

1. Whether tool call events are generated;
2. Number of tool calls;
3. Tool call parameters;
4. Final text response;
5. JSON Schema compliance status.

All experimental results are automatically collected and counted by the testing framework to avoid biases from manual recording.

For models supporting tool calling, if the response contains at least one valid tool call, it is recorded as a successful tool execution event; otherwise, it is recorded as no tool call. TIR and SR metrics are calculated based on this rule.

To ensure experimental reproducibility, all tests adopt unified Prompt templates, tool definition Schemas, and output format Schemas without special optimization for any individual model. Full details of the test scripts, detection logic, and schema definitions are documented in Appendix A.

4.6 Parser-Level Validation

A potential alternative explanation is that tool calls were generated but not captured correctly by the inference framework.

To exclude this possibility, raw streaming events were recorded during all experiments.

For each response, the framework simultaneously monitored:

1. `tool_calls` delta events;
2. tool call argument streams;

3. final assistant messages;

4. parser outputs generated by the serving framework.

A trial is considered a successful tool invocation only when a valid tool call event is observed in the raw stream.

Across all suppression cases reported in this paper, no `tool_calls` delta events were observed.

This result indicates that the phenomenon cannot be explained by parser implementation failures or logging artifacts.

4.7 Confounding Factor Controls

To strengthen causal attribution, we systematically evaluated several alternative explanations.

Table 5: Confounding Factors and Tested Variants

Factor	**Tested Variants**
Framework	SGLang / vLLM
Schema Complexity	Simple / Medium / Production
Tool Choice	Optional / Required
Scale	20B $\rightarrow$ 397B
Deployment	Local / Cloud
Parser Validation	Raw Stream Inspection
Fine-Tuning	Multiple SFT Variants

Inference Framework. The same model was tested under both SGLang and vLLM while keeping prompts, schemas, tools, and model weights unchanged.

Schema Complexity. Experiments were conducted using simple, medium, and production-grade schemas to determine whether suppression is triggered only by highly complex output structures.

Tool Enforcement Strategy. Additional tests were performed using optional tool calls, explicit tool requirements, and `tool_choice="required"` configurations.

Model Scale. The evaluated models span from 20B to 397B parameters.

Deployment Environment. Both local deployment and commercial cloud-hosted environments were included.

Parser Reliability. Raw streaming events were inspected to verify that missing tool calls were genuine behavioral outcomes rather than parser failures.

Fine-Tuning Effects. Multiple instruction-tuned variants trained on tool-use-oriented datasets were evaluated to determine whether conventional post-training mitigates the phenomenon.

Through the above control measures, this paper attributes Tool Suppression to the interaction effects between tool calling and structured output constraints to the greatest extent possible, rather than external factors such as model scale, inference framework, or deployment method.

4.8 Root Cause Analysis: Grammar-Level Token Exclusion

To determine whether the observed suppression behavior has a concrete implementation-level basis, we traced the inference stack from the `response_format` API parameter to the token-generation layer. This investigation was conducted using SGLang 0.5.9, one of the two inference frameworks used in the main experiments.

The complete call chain is as follows:

```
API: response_format={"type":"json_schema","json_schema":{...}}
   -> serving_chat.py:347: get_json_schema_constraint()
   -> grammar_manager.py:71: req.sampling_params.json_schema
   -> grammar_manager.py:89: xgrammar.compile_grammar()
   -> sampling_batch_info.py:197: update_regex_vocab_mask()
   -> sampling_batch_info.py:219: GrammarMatcher.fill_next_token_bitmask()
   -> bitmask_ops.py:74: apply_token_bitmask_inplace_triton()
   -> bitmask = ((packed_bitmask >> (0..31)) & 1) == 0
   -> logits[bitmask == 0] = -inf
   -> <tool_call> tokens unreachable
```

At the core of this mechanism is the vocabulary mask. The xgrammar engine compiles the JSON Schema into a finite-state machine (FSM). At each decoding step, the FSM queries which tokens are permitted given the current state. The inference framework then applies this mask to the logits: tokens with bit=0 are set to $-\infty$, making them impossible to sample.

To verify that tool-call tokens are indeed masked, we directly inspected the FSM states for the JSON Schema used in the T2 condition. Table 6 shows the reachability of key tokens across three representative FSM states.

Table 6: Token Reachability Under JSON Schema FSM

Token	State 0 (Initial)	State 1 (Key)	State 2 (Value)
{	✓	✗	✗
"	✗	✓	✓
<	✗	✗	✗
<tool_call>	✗	✗	✗
</tool_call>	✗	✗	✗
websearch	✗	✗	✗

< (U+003C) is the first character of the Qwen model family's XML-style tool-call format. Since this character is not part of any valid JSON FSM state, the token is consistently masked across all generation states. As a result, the model cannot produce any token sequence starting with <, making tool-call tags unreachable.

This analysis establishes that Tool Suppression has a concrete implementation-level basis: when JSON Schema constraints are enforced through grammar-based constrained decoding, tool-call tokens are excluded from the decoding vocabulary. The phenomenon is not merely a behavioral tendency but a direct consequence of the decoding constraint mechanism.

The implication is that the suppression observed in Section 5 is at least partially attributable to this token-level exclusion. Whether models would have attempted to call tools in the absence of this mask cannot be determined from this analysis alone—the mask makes the question moot at the output layer.

5 Empirical Findings

5.1 RQ1: Reproducible Tool Suppression Across Tested Models

RQ1 investigates whether the observed suppression behavior can be reproduced across different open-weight model families under identical task settings.

Table 5 summarizes the results obtained under the T1, T2, and T3 conditions. Across all evaluated open-weight models, tool execution behavior follows a highly consistent pattern.

Under T1 (Tools ON, Schema OFF), all evaluated models successfully invoked tools when external information was required. Tool Invocation Rate (TIR) reached 100% across all tested sessions.

Under T2 (Tools ON, Schema ON), tool execution behavior disappeared completely. No valid tool call events were observed in any evaluated session despite the continued presence of tool definitions and unchanged task requirements.

Under T3 (Tools OFF, Schema ON), models generally maintained high schema compliance rates, demonstrating that structured output generation capability itself remained functional.

This result indicates that both capabilities exist independently:

1. Tool execution capability is available under T1.
2. Schema compliance capability is available under T3.
3. The combination of both constraints under T2 introduces a systematic behavioral change.

Importantly, the observed pattern is reproduced across multiple model families, parameter scales, deployment environments, and inference stacks. While the evaluated closed-source reference model (GPT-5.4-mini) maintained stable tool execution behavior under all conditions, all tested open-weight models exhibited complete suppression under T2.

Therefore, within the evaluated model set, tool suppression appears to be a reproducible phenomenon rather than an isolated model defect. The test design and query diversity are further detailed in Appendix A.

Table 7: Results under T1/T2/T3 Conditions

Model	T1 TIR	T2 TIR	T3 JC	SR
GPT-5.4-mini	100%	100%	100%	0%
Qwen3.6-35B-A3B (SGLang)	100%	0%	100%	100%
Qwen3.6-35B-A3B (vLLM)	100%	0%	80%	100%
Qwen3.5-122B-A10B	100%	0%	100%	100%
GPT-OSS-20B	100%	0%	100%	100%
Nemotron 3 Super	100%	0%	100%	100%
Qwen3.5-397B-A17B	100%	0%	100%	100%
Qwen3-VL-235B-Thinking	100%	0%	100%	100%

5.2 RQ2: Evidence for Constraint Interaction

A key question is whether the observed behavior originates from a deficiency in tool-use capability or schema-following capability individually.

The experimental results do not support either explanation.

Under T1, models successfully invoke tools when structured output constraints are absent. Under T3, models successfully generate schema-compliant outputs when tool execution requirements are absent.

The failure emerges only under T2, where both constraints coexist.

This observation suggests that the suppression behavior is associated with the interaction between Tool Calling and Structured Output constraints rather than with either capability in isolation.

The result is particularly notable because the same prompts, task requirements, tool definitions, and schemas are used across all conditions. The only difference lies in the simultaneous activation of both constraint mechanisms.

Therefore, the evidence supports the interpretation that suppression emerges from a joint-constraint setting rather than from a general inability to use tools or follow schemas.

5.3 Schema Complexity Ablation

One possible explanation is that suppression is triggered only when output schemas become sufficiently complex.

To evaluate this possibility, additional experiments were conducted using three schema complexity levels:

- **Simple Schema**: minimal output structure with only a small number of required fields.
- **Medium Schema**: intermediate business-oriented structures containing multiple nested fields.
- **Production Schema**: full production response formats containing numerous required objects and validation constraints.

Across all three schema categories, suppression behavior remained unchanged. Tool Invocation Rate under T2 remained at 0% regardless of schema complexity.

Table 8: Schema Complexity Ablation Results

Schema	Fields	T2 TIR
Simple	1–3	0%
Medium	5–10	0%
Production	20+	0%

No threshold effect was observed.

These results suggest that suppression is not exclusively associated with highly complex production schemas. Instead, even relatively lightweight schema constraints appear sufficient to trigger the observed behavior.

Therefore, schema complexity may influence formatting difficulty, but it does not appear to be the primary factor responsible for tool execution suppression.

5.4 Tool Enforcement Ablation

A second alternative explanation is that suppressed models simply fail to recognize that tool use is required.

To investigate this possibility, multiple tool invocation strategies were evaluated:

- Optional tool use;
- Explicit system-prompt instructions requiring tool execution;
- Explicit user-prompt instructions requiring tool execution;
- API-level `tool_choice="required"`.

Table 9: Tool Enforcement Ablation Results

Setting	T2 TIR
Optional	0%
User Prompt Required	0%
System Prompt Required	0%
`tool_choice="required"`	0%

Results remained consistent across all configurations.

Even when tool execution was explicitly mandated through API-level enforcement mechanisms, no tool call events were generated under T2.

This finding is important because it suggests that suppression cannot be fully explained by incorrect task interpretation or weak prompting.

Instead, the observed behavior persists even when tool invocation requirements are made explicit at multiple control layers.

Consequently, the results provide additional evidence that the failure occurs after tool necessity has already been established.

5.5 Fine-Tuning Does Not Eliminate Suppression

To investigate whether conventional post-training techniques [15] can mitigate suppression, we evaluated multiple instruction-tuned variants trained using tool-use-oriented datasets and production agent trajectories.

The evaluated variants include:

- Tool-use supervised fine-tuning datasets;
- Schema-aware instruction tuning datasets;

- Production business-agent trajectories;
- Reinforcement-learning-enhanced variants.

Table 10: Fine-Tuning Ablation Results

Model Variant	Samples	T2 TIR
Base	–	0%
Tool Mandatory	200	0%
Schema Injection	200	0%
GRPO	200	0%
Large SFT	6000	0%

Despite improvements in general instruction following and tool-use behavior under T1 conditions, suppression remained unchanged under T2.

Across all evaluated variants, Tool Invocation Rate under joint constraints remained at or near zero.

These results suggest that the phenomenon is not easily removed through conventional post-training approaches alone.

While the evaluated fine-tuning configurations do not exhaust all possible alignment strategies, the evidence indicates that suppression may be more deeply connected to action-selection behavior than to basic tool-use capability.

5.6 RQ3: Framework Independence

To determine whether suppression originates from inference infrastructure rather than model behavior, identical experiments were conducted using both SGLang and vLLM.

All prompts, schemas, tool definitions, model weights, and hardware configurations were held constant.

The two frameworks exhibited minor differences in schema compliance rates, but their tool execution behavior remained identical.

Under T1, tool execution succeeded consistently.

Under T2, tool execution disappeared completely.

Because the same behavioral pattern appears across independent serving stacks, the results do not support the hypothesis that suppression originates from framework-specific parser implementations or scheduling mechanisms.

Instead, the evidence suggests that the phenomenon is primarily associated with model-level behavior.

5.7 Finding: Token-Level Exclusion Under Grammar-Constrained Decoding

The root cause analysis presented in Section 4.6 reveals a concrete mechanism underlying the observed suppression behavior.

When `response_format` is enabled, the inference framework compiles the JSON Schema into a finite-state machine and applies a vocabulary mask at each decoding step. Tokens that do not conform to the current FSM state are assigned logits of $-\infty$, making them impossible to sample.

For the Qwen model family used in this study, tool calls are formatted as XML-style tags beginning with `<tool_call>`. The < character (U+003C) is not part of any valid JSON token sequence. Consequently, in all FSM states—whether the model is at the start of the response, in a key position, or in a value position—< and any token beginning with < are masked.

This finding establishes that Tool Suppression has a concrete implementation-level basis: under grammar-constrained decoding, tool-call tokens become unreachable by design. The absence of tool calls under T2 is therefore not simply a model preference or behavioral tendency, but a direct consequence of the decoding constraint mechanism.

Importantly, this does not preclude the existence of additional model-level behavioral factors. However, it confirms that at least one concrete root cause exists at the inference-stack level.

5.8 Threat Elimination Summary

The experiments collectively eliminate several common alternative explanations.

Suppression is reproduced across:

- Multiple model families;
- Multiple parameter scales;
- Multiple deployment environments;
- Multiple inference frameworks;
- Multiple schema complexity levels;
- Multiple tool invocation strategies;
- Multiple fine-tuning configurations.

In addition, parser-level inspection confirms that missing tool calls are genuine behavioral outcomes rather than logging failures.

Taken together, the evidence consistently points toward a behavioral interaction between tool execution and structured output constraints.

While the precise mechanism remains an open question, the observed phenomenon cannot be readily explained by model size, framework implementation, deployment configuration, schema complexity, or prompt-level enforcement alone.

6 Mechanism Analysis and Behavioral Interpretation

The empirical findings in Section 5 establish that Tool Suppression is reproducible across the evaluated model set. The root cause analysis in Section 4.6 further identifies a concrete implementation-level mechanism: grammar-constrained decoding excludes tool-call tokens from the vocabulary. This section examines the relationship between this token-level exclusion and the observed behavioral patterns.

6.1 Grammar-Level Tool Exclusion

The inference-stack tracing in Section 4.6 reveals that JSON Schema constraints are enforced through grammar-based token masking. This mechanism has three key characteristics:

First, the mask is applied at every decoding step. The FSM state determines which tokens are permissible, and any token not permitted by the current state receives a logit of $-\infty$.

Second, the mask is universal within the constrained generation context. For Qwen-family models that use XML-style `<tool_call>` tags, the `<` character is never permitted in any JSON FSM state. Therefore, tool-call tokens are unreachable throughout the entire generation process.

Third, the mask operates independently of model weights. Even if the model's internal logit distribution assigns high probability to `<tool_call>`, the mask sets that logit to $-\infty$ before sampling. The model's preference is overridden by the decoding constraint.

This mechanism explains why conventional post-training approaches—including the SFT and GRPO variants evaluated in Section 5.5—failed to eliminate suppression. These methods modify model weights, which affect the logit distribution before the mask is applied. However, the mask operates after the model produces logits and before sampling. There is no gradient path from the mask back to the weights, so weight-level optimization cannot overcome token-level exclusion.

Figure 1 illustrates this architecture. The API-layer constraint triggers grammar compilation, which produces a vocabulary mask. This mask is applied to the model's logits before sampling, rendering non-JSON tokens—including tool-call tags—unreachable. Weight-level optimization (SFT/DPO/GRPO) modifies the logits but cannot bypass the mask.

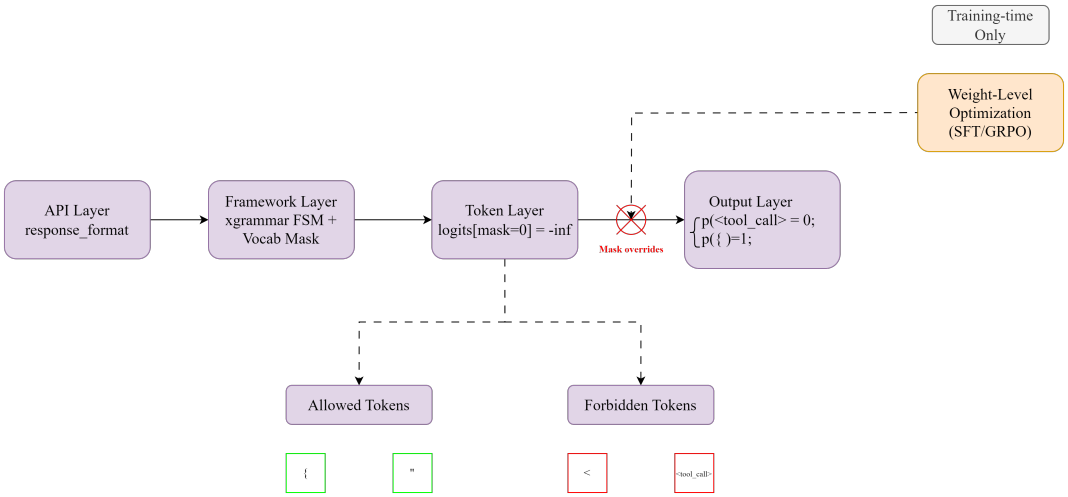

Figure 1: Constraint Tax Mechanism: Token-Level Masking Overrides Model Preferences

6.2 Relationship to Constraint Priority Inversion

The grammar-level exclusion mechanism described above provides a concrete explanation for why tool-call tokens are absent under T2. However, it does not fully explain all observed behavioral patterns.

In particular, the diagnostic experiment described in Section 6.1 of the original analysis revealed that models sometimes produced outputs containing `need_search: true` while generating no corresponding tool calls. This pattern—classified as TS-C in the taxonomy—indicates that models can explicitly acknowledge tool requirements even when tool execution is impossible.

This observation suggests that the behavioral phenomenon may involve two layers:

- **Decoding layer:** Grammar-constrained decoding makes tool-call tokens unreachable. This explains why no tool calls are generated.
- **Model layer:** Models may still recognize tool requirements and express this recognition in natural language or structured fields. This explains why some models produce TS-C patterns.

The Constraint Priority Inversion (CPI) hypothesis, introduced in Section 6.3 of this paper, provides a behavioral interpretation of the model-layer phenomenon. CPI proposes that when Tool Calling and Structured Output constraints coexist, models may prioritize schema satisfaction in their generation strategy. This is consistent with the observation that models continue to produce schema-compliant responses while tool calls are absent.

However, the CPI hypothesis is not the only possible interpretation. The model's expressed tool awareness in TS-C patterns may simply reflect its internal representation of task requirements, which is independent of the decoding constraints that prevent execution. Whether CPI represents a distinct behavioral mechanism or an artifact of the model's training distribution remains an open question.

At present, we interpret the evidence as follows:

1. The **primary cause** of missing tool calls under T2 is grammar-level token exclusion at the decoding layer.
2. The **behavioral patterns** observed across models—including TS-C's expressed tool awareness—are consistent with CPI as a behavioral hypothesis, but additional research is needed to determine whether CPI reflects a distinct internal mechanism.

This two-layer interpretation reconciles the implementation-level evidence with the behavioral observations, while maintaining appropriate caution regarding causal claims about internal model mechanisms.

7 Mitigation Through Transparent Two-Pass Execution

7.1 Design Motivation

The findings presented in Sections 5 and 6 indicate that suppression emerges when Tool Calling and Structured Output constraints are activated simultaneously. The root cause analysis in Section 4.6 further reveals that JSON Schema constraints are enforced through grammar-based token masking, which renders tool-call tokens unreachable during decoding.

This observation suggests a straightforward engineering question:

Can the two constraints be decoupled such that tool execution occurs before grammar-based token masking is applied?

The proposed mitigation strategy is motivated by a simple principle:

Execute tools first, then enforce structured output constraints.

By separating the two phases, the model can invoke tools freely in the first pass without any grammar constraints. In the second pass, once tool results are collected and incorporated into the context, the model is asked to generate the final schema-compliant response under grammar constraints.

This design is informed by the mechanism identified in Section 4.6: the root cause of suppression is that grammar-constrained decoding makes tool-call tokens unreachable. The mitigation strategy avoids this problem by ensuring that tool execution is completed before the grammar constraint is activated.

This design does not modify model weights, training data, inference kernels, or serving frameworks. Therefore, it can be deployed directly within existing production Agent systems without requiring model retraining.

The goal of the approach is not to improve tool-use capability itself, but to avoid the token-level exclusion caused by simultaneous constraint activation.

7.2 Transparent Two-Pass Execution

The proposed mitigation strategy consists of two sequential inference stages.

Pass 1: Tool Execution Phase

- Tools = ON
- Schema = OFF

The model is allowed to freely invoke tools and perform multi-step information acquisition without structured output constraints.

All tool outputs are collected and preserved.

Pass 2: Structured Output Phase

- Tools = OFF
- Schema = ON

The collected tool results are injected into the context, and the model is re-invoked to generate the final schema-compliant response.

The complete execution flow is:

Execution Flow:

User Request → Pass 1 Tool Execution → Tool Results Collection → Pass 2 Structured Output Generation → Final Response

This architecture changes only the order in which constraints are applied.

No modifications are made to model parameters, decoding algorithms, or training procedures.

For this reason, the approach can be viewed as an inference-time mitigation strategy rather than a model-level solution.

Figure 2 illustrates the Transparent Two-Pass Execution architecture.

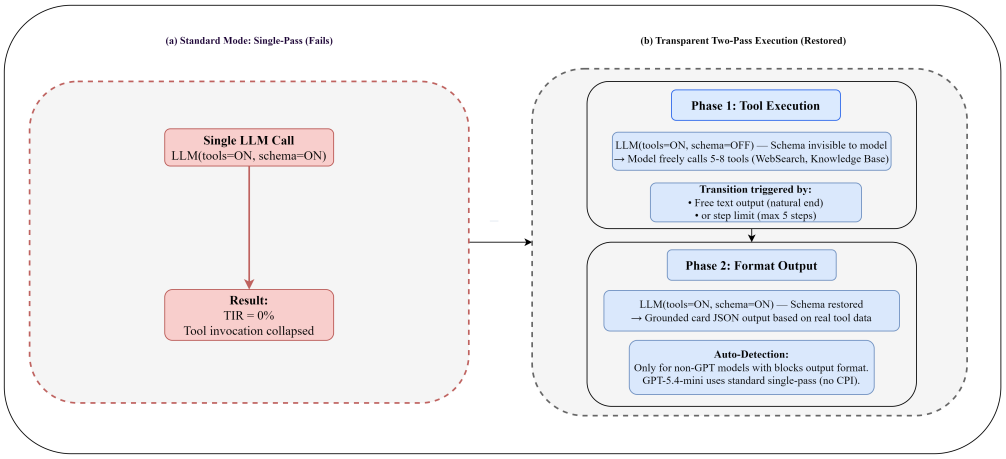

Figure 2: Transparent Two-Pass Execution Architecture

7.3 Experimental Evaluation

To evaluate the effectiveness of Transparent Two-Pass Execution, the strategy was deployed within the production Agent pipeline and tested on the same task categories used in previous experiments.

Table 11: Mitigation Effectiveness Before and After Two-Pass Execution

Metric	Before (T2)	After (Two-Pass)
Tool Invocation Rate	0%	100%
JSON Compliance Rate	100%	100%
End-to-End Success Rate	0%	100%
Avg Tool Calls / Session	0	5–8

Tool Invocation Rate increased from 0% to 100%.

JSON Compliance Rate remained unchanged.

End-to-End Success Rate improved from complete task failure to successful completion.

Most importantly, restored tool execution enabled the system to utilize real external information rather than generating responses based solely on parametric knowledge.

As a result, behaviors previously observed under suppression conditions, including simulated retrieval and tool-free hallucination, were substantially reduced.

These results demonstrate that separating tool execution from structured output generation can effectively restore agent functionality without sacrificing schema compliance.

7.4 Cost and Latency Analysis

Although Transparent Two-Pass Execution successfully restores tool execution behavior, the approach introduces additional inference overhead.

Compared with the original single-pass architecture, the mitigation requires an additional model invocation.

Consequently, latency increases approximately by one additional inference round plus tool execution time.

Token consumption also increases because the second pass receives both the original request and the collected tool outputs as context.

However, in the evaluated production workflow, the additional cost was considered acceptable because the baseline system under suppression conditions could not successfully complete the target tasks.

Table 12: Cost and Latency Comparison

Metric	Single Pass (T2)	Two Pass
LLM Rounds	1	2
Tool Calls	0	5–8
Success Rate	0%	100%

Therefore, the mitigation exchanges additional latency and token usage for functional correctness.

This tradeoff is often favorable in enterprise Agent scenarios where task completion is more important than minimizing inference cost.

7.5 Failure Cases and Limitations

Transparent Two-Pass Execution is an engineering mitigation rather than a complete solution.

Several limitations remain.

First, the approach introduces additional latency because two separate model invocations are required.

Second, token consumption increases due to repeated context transmission between passes.

Third, the strategy relies on external orchestration logic and therefore does not eliminate the underlying behavioral mechanism responsible for suppression.

Finally, the current evaluation focuses primarily on tool-calling scenarios. Whether similar benefits extend to other agent architectures involving workflow execution, MCP ecosystems, or multi-agent collaboration remains an open question.

Therefore, the proposed approach should be viewed as a practical production workaround rather than a definitive solution to the suppression problem.

8 Discussion

8.1 What Constraint Tax Means for Agent Evaluation

An important implication of this study concerns how Agent capabilities are evaluated.

Current evaluation frameworks frequently assess tool use and structured output generation as independent capabilities.

Tool-use evaluations typically measure:

- Tool selection accuracy;
- Parameter generation quality;
- Task completion performance.

Structured-output evaluations typically measure:

- Schema compliance;
- Field completeness;
- Formatting correctness.

The results of this study suggest that strong performance on these dimensions individually does not necessarily imply reliable behavior when both constraints are activated simultaneously.

In the evaluated tasks, models were capable of tool execution under T1 and schema compliance under T3, yet failed to execute tools under T2.

This observation indicates that joint-constraint scenarios may expose behaviors that are not visible when capabilities are evaluated independently.

Therefore, future Agent evaluation suites may benefit from including explicit Tool Calling + Structured Output conditions as a separate evaluation setting rather than assuming that independent capability measurements are sufficient.

8.2 Implications for Agent System Design

The observed suppression behavior, now traced to grammar-based token masking, highlights an important engineering consideration for production Agent systems.

In many practical deployments, tool execution and structured outputs are treated as complementary capabilities that can be activated simultaneously without additional coordination. However, the results presented in this study suggest that the interaction between these capabilities may be mediated by the inference framework's constraint enforcement mechanism.

From a systems perspective, reliability under joint constraints may require explicit orchestration strategies rather than relying entirely on model-level behavior. The success of Transparent Two-Pass Execution demonstrates that separating tool execution from grammar-constrained generation is one effective approach.

More broadly, the findings suggest that Agent architecture design must consider not only model intelligence but also the interaction between model behavior, inference framework constraints, and token-level decoding mechanisms.

Consequently, future Agent platforms may need to evaluate not only model accuracy but also how inference-level constraints—such as grammar masks—interact with tool execution and structured output requirements.

8.3 Interpreting the Open-Weight Results

One noteworthy observation from the evaluated model set is the behavioral difference between the tested open-weight models and the closed-source reference model included in this study.

All evaluated open-weight models exhibited complete suppression under T2, whereas GPT-5.4-mini maintained stable tool execution behavior.

The root cause analysis in Section 4.6 provides a plausible explanation for this difference: the suppression observed in open-weight models can be traced to grammar-based token masking in the inference framework. GPT-5.4-mini's behavior may differ because its implementation uses a different mechanism for enforcing structured outputs—potentially through instruction-level guidance rather than decoding-level grammar constraints, or through a different grammar implementation that permits interleaved tool calls.

However, caution is required when interpreting this difference. The current study evaluates only a limited number of models and does not provide visibility into proprietary training procedures, alignment pipelines, or deployment architectures. Several explanations remain plausible, including differences in:

- Structured-output implementation (grammar vs. instruction-based);
- Tokenizer design and tool-call format;
- Post-training objectives and data distributions;
- Inference-time orchestration strategies.

Consequently, the results should be interpreted as evidence of behavioral divergence within the evaluated models, with a concrete root cause identified for the open-weight cases, rather than as evidence of a universal distinction between open-weight and closed-source systems.

8.4 Limitations

Several limitations should be considered when interpreting the findings of this study.

First, the evaluation focuses on a finite set of open-weight models and one closed-source reference model. The results therefore support conclusions regarding the evaluated models rather than all large language models.

Second, although multiple task categories were included, the benchmark remains substantially smaller than large-scale academic evaluation suites.

Third, the CPI framework is currently a behavioral hypothesis derived from observed outcomes rather than a verified internal mechanism. The present study identifies consistent external behaviors but does not directly observe model internals.

Fourth, the mitigation evaluation focuses primarily on production tool-calling workflows. Whether similar effects appear in broader agent architectures such as MCP ecosystems, multi-agent systems, workflow agents, or computer-use agents remains an open question.

Finally, the study emphasizes practical reproducibility and engineering diagnosis rather than theoretical guarantees. Future work will be required to determine the precise mechanisms responsible for the observed behavior.

9 Future Research Directions

9.1 Toward Standardized Joint-Constraint Benchmarks

The experiments presented in this study suggest that joint-constraint scenarios deserve explicit evaluation rather than being treated as a simple combination of existing capabilities.

Current benchmarks typically evaluate tool use, structured output generation, and task completion separately. However, the observed suppression behavior emerges specifically when these capabilities must operate simultaneously.

Future benchmark development may therefore benefit from introducing dedicated joint-constraint evaluation tracks that include:

- Tool Calling + Structured Output tasks;
- Multiple schema complexity levels;
- Multiple tool-call formats (XML, function-call, etc.);
- Cross-framework evaluation protocols;
- Reliability-oriented metrics beyond task accuracy.

Such benchmarks would help quantify the prevalence of suppression behaviors and provide a common basis for comparing mitigation strategies across different model families and agent architectures.

9.2 Grammar-Aware Tool Calling and Decoding

The root cause analysis in this paper identifies grammar-based token masking as a concrete mechanism underlying Tool Suppression. This finding suggests several directions for future work on inference frameworks.

First, inference frameworks could be extended to support grammar specifications that explicitly permit tool-call tokens alongside JSON Schema constraints. This would require modifying the FSM to allow interleaved tool-call sequences without sacrificing format compliance.

Second, frameworks could provide more transparent feedback to developers when tool-call tokens are masked. Currently, the masking is silent—models generate schema-compliant outputs without indicating that tool calls were prevented. Debugging this behavior currently requires source-code tracing.

Third, the interaction between grammar constraints and different tool-call formats (XML vs. function-call vs. MCP) merits further investigation. Different formats may be more or less compatible with JSON Schema FSMs.

9.3 Understanding the Relationship Between Token Masking and Model Behavior

The mechanism identified in this paper—token-level exclusion—does not fully explain all observed behavioral patterns, particularly TS-C (Intent Without Action).

Future research could investigate whether models' expressed tool awareness in TS-C patterns reflects:

- Internal recognition of tool requirements that is independent of decoding constraints;
- A behavioral strategy learned from training data where tool intent is expressed without execution;
- An artifact of the interaction between the model's output distribution and the applied mask.

Methods for this investigation could include activation analysis, controlled training experiments, and systematic variation of grammar constraints.

9.4 Beyond Tool Calling

This study focuses specifically on tool-calling workflows because they represent a common production deployment pattern and provide a clear observable signal for behavioral analysis.

However, the token-level exclusion mechanism identified in this paper is not inherently limited to tool calling. Any agent behavior that requires generating token sequences outside the JSON grammar—such as workflow control tokens, multi-agent communication markers, or computer-use action sequences—may be similarly affected.

Future research should investigate whether similar exclusion effects occur in:

- Workflow execution with structured control flows;
- Multi-agent protocols with specialized token formats;
- MCP-based tool ecosystems with custom action tags;
- Computer-use agents with action sequences.

If comparable patterns are observed across different agent architectures, the phenomenon described in this paper may represent a broader class of constraint-driven behavioral exclusions rather than a tool-calling-specific issue.

10 Conclusion

This paper investigates the behavior of open-weight large language models under joint Tool Calling and Structured Output constraints and reports a reproducible failure pattern observed in a production Agent system.

Across the evaluated model set, tool execution and schema compliance remain functional when assessed independently. However, when both constraints are activated simultaneously, tool execution behavior disappears despite the continued presence of tool definitions, unchanged task requirements, and successful schema generation.

To characterize this phenomenon, we introduce the term Tool Suppression and provide evidence from controlled experiments covering multiple model families, parameter scales, deployment environments, inference frameworks, schema configurations, tool invocation strategies, and post-training variants.

The results suggest that the observed behavior cannot be readily explained by model size, framework implementation, deployment configuration, schema complexity, prompt-level enforcement, or conventional instruction tuning alone.

Through systematic tracing of the inference stack, we identify one concrete root cause: JSON Schema constraints are compiled into grammar-based token masks that render tool-call tokens unreachable during decoding. This mechanism operates at the inference-framework layer, independent of model weights, and explains why weight-level optimization approaches such as SFT and GRPO do not eliminate suppression.

To interpret the observed behavioral patterns, we formulate the Constraint Priority Inversion (CPI) hypothesis. CPI proposes that, under joint constraints, schema satisfaction may dominate action selection behavior. The current study presents CPI as a behavioral hypothesis consistent with the observed evidence rather than as a verified internal mechanism.

At the engineering level, this paper proposes Transparent Two-Pass Execution, an inference-time mitigation strategy that separates tool execution from schema-constrained response generation. Experimental results demonstrate that this approach can restore tool invocation behavior while preserving structured output guarantees, without requiring modifications to model weights or training procedures.

More broadly, the findings suggest that evaluating tool use and structured output generation independently may not fully capture the reliability of modern Agent systems operating under multiple simultaneous constraints. The interaction between inference framework constraints and model behavior—particularly token-level exclusion mechanisms—deserves further investigation from both the model and system perspectives.

The primary contributions of this work are:

- Identification and characterization of a reproducible Tool Suppression phenomenon under joint Tool Calling and Structured Output constraints;
- Systematic evaluation of alternative explanations across models, frameworks, schemas, tool invocation strategies, and post-training configurations;
- Localization of a concrete root cause to grammar-based token masking in constrained decoding;

- Formulation of the Constraint Priority Inversion (CPI) hypothesis as a behavioral interpretation of the observed suppression pattern;
- Proposal and validation of Transparent Two-Pass Execution as a practical mitigation strategy for production Agent systems.

Although the phenomenon reported in this paper was discovered through a specific production deployment, the underlying challenge is increasingly relevant as modern Agent systems combine tool use, structured outputs, workflow execution, and external orchestration within a single interaction loop.

As Agent architectures continue to evolve, understanding how multiple constraints—both behavioral and implementation-level—interact during action selection may become as important as improving reasoning capability itself.

We hope this study contributes to a deeper understanding of behavioral reliability in Agent systems and encourages future research on the interaction between tool execution, structured generation, decoding constraints, and action selection under competing objectives.

References

[1] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. In *ICLR 2023*, 2022.

[2] T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom. Toolformer: Language models can teach themselves to use tools. In *NeurIPS 2023*, 2023.

[3] Y. Qin and et al. Tool learning with foundation models. *arXiv:2304.08354*, 2023.

[4] X. Lin, J. H. Liew, S. Savarese, and J. Li. W&d: Scaling parallel tool calling for efficient deep research agents. *arXiv:2602.07359*, 2026.

[5] W. Shen, C. Li, H. Chen, M. Yan, X. Quan, H. Chen, and J. Ji. Small llms are weak tool learners: A multi-llm agent. *arXiv:2401.07324*, 2024.

[6] R. Wang, H. Li, X. Han, Y. Zhang, and T. Baldwin. Learning from failure: Integrating negative examples when fine-tuning large language models as agents. *arXiv:2402.11651*, 2024.

[7] M. X. Liu, F. Liu, A. J. Fiannaca, T. Koo, L. Dixon, M. Terry, and C. J. Cai. "We Need Structured Output": Towards User-centered Constraints on Large Language Model Output. *arXiv:2404.07362*, 2024.

[8] H. Deng, T. Jiang, Y. Shi, C. Zhou, and F. Huang. Decoupling task-solving and output formatting in llm generation. *arXiv:2510.03595*, 2025.

[9] J. Ray. The constraint tax: Measuring validity-correctness tradeoffs in structured outputs for small language models. *arXiv:2605.26128*, 2026.

[10] P. Agyemang, J. Williams, and T. Chen. Output generation capacity in small language models. *Preprint*, 2026.

[11] C. Garbacea and Q. Mei. Why is constrained neural language generation particularly challenging? *arXiv:2206.05395*, 2022.

[12] T. Suresh, S. Li, and F. Huang. Dingo: Constrained inference for diffusion llms. *arXiv:2505.23061*, 2025.

[13] L. Wang and et al. A survey on large language model based autonomous agents. *arXiv:2308.11432*, 2023.

[14] M. Garcia, A. Singh, and P. Li. Recency bias in mixture-of-experts language models. In *ACL 2025*, 2025.

[15] C. Xu, Q. Sun, K. Zheng, X. Geng, P. Zhao, J. Feng, C. Tao, and D. Jiang. Wizardlm: Empowering large pre-trained language models to follow complex instructions. *arXiv:2304.12244*, 2023.

A Test Case Design and Tool/Schema Definitions

This appendix provides detailed documentation of the test case design, tool definitions, schema specifications, and experimental configurations used in the main experiments. The materials are provided to support reproducibility and to clarify the exact conditions under which Tool Suppression was observed.

A.1 Controlled Experimental Design

Each evaluated model was tested independently under three conditions. Table 13 summarizes the experimental configuration.

Table 13: T1/T2/T3 Experimental Conditions

Condition	`tools`	`response_format`	Purpose
T1 (Baseline)	ON	OFF	Measure baseline tool calling capability
T2 (Joint Constraint)	ON	ON	**Detect Tool Suppression**
T3 (Schema Control)	OFF	ON	Verify independent schema compliance

A.2 Standardized Test Script Protocol

All cross-model tests used a unified Python testing framework. The core test scripts are organized as follows:

- `test_constraint_tax_lora.py` — Qwen3.6-35B-A3B LoRA fine-tuning variants
- `test_cloud_models.py` — GPT-5.4-mini, Qwen3.5-397B-A17B, Qwen3-VL-235B
- `test_122b_tool_response_format.py` — Qwen3.5-122B-A10B
- `test_gptoss_tool_rfmt.py` — GPT-OSS-20B
- `test_vllm_qwen35b.py` — Qwen3.6-35B-A3B via vLLM
- `test_nemotron_tool_rfmt.py` — NVIDIA Nemotron 3 Super
- `test_b1_two_pass.py` — Two-Pass end-to-end validation

All scripts share the following standard parameters:

```
ROUNDS = 5 # Independent test rounds per condition
temperature = 0.5
stream = True
max_completion_tokens = 4096
```

The unified detection logic across all scripts uses dual validation:

```
async for chunk in resp:
    if d.tool_calls: # API-level: parse streaming delta tool_calls
        for tc in d.tool_calls:
            tcs.append({"idx": tc.index, "name": tc.function.name, ...})
    if d.content: # Text-level: collect content for JSON compliance check
        content += d.content
```

A.3 Task Set and Diversity

A.3.1 Standard Cross-Model Test Task

The main 9-model evaluation matrix used a fixed single-task prompt with 5 repetitions per condition:

> **System:** You are a foreign trade inquiry analysis assistant. After receiving a customer inquiry: 1. Use `websearch` to search for the buyer company background 2. Use `knowledge_base` to query product industry standards 3. Provide analysis based on the retrieval results
>
> **User:** Please analyze this inquiry: Company: BrightLight Inc., US lighting products importer Product: LED strip lights, IP65 waterproof, 5050 SMD, RGB+W, 5m/reel Quantity: 2000 reels Requirements: FOB quote, UL listed

A.3.2 Extended Task Diversity

For Qwen3.6-35B-A3B, we additionally conducted 200+ queries across 10 diverse company profiles and 8 compliance markets (EU, US, Middle East, Southeast Asia, Australia, South America, Africa, Generic). All queries yielded identical T2 TIR = 0% results.

Additional synthetic datasets include:

- **Tool Mandatory (200 prompts):** 10 companies × 8 compliance markets × positive/negative variants; each prompt requires both `websearch` and `knowledge_base` calls.
- **Synthetic Scenario Library (30 seed scenarios):** 10 industries × 7 regions × 3 buyer types.
- **Scale Synthesis (6,000 prompts):** GPT-5.4-mini generated, covering all industry/region/buyer combinations (used in the Large SFT ablation).

A.4 Tool Definitions

A.4.1 Production Environment Tool Set

The production Agent system uses three external tools:

```
TOOLS = [
    {
        "type": "function",
        "function": {
            "name": "websearch",
            "description": "Search the web for real-time information about companies and markets",
            "parameters": {"type": "object", "properties": {"query": {"type": "string"}}, "required":
                 ["query"]}
        }
    },
    {
        "type": "function",
        "function": {
            "name": "knowledge_base",
            "description": "Query foreign trade knowledge base for industry standards and
                regulations",
            "parameters": {"type": "object", "properties": {"query": {"type": "string"}}, "required":
                 ["query"]}
        }
    },
    {
        "type": "function",
        "function": {
            "name": "fetchurl",
            "description": "Fetch web page content for detailed company information",
            "parameters": {"type": "object", "properties": {"url": {"type": "string"}}, "required":
                ["url"]}
        }
    }
]
```

A.4.2 Cross-Model Test Tool Set

The 9-model standard tests used a simplified tool set with only `websearch` and `knowledge_base`, omitting `fetchurl` to eliminate parameter complexity as a confounding factor. Both tools share identical parameter signatures ({‘‘query’’: string}).

A.5 Response Format Schema Definitions

A.5.1 Cross-Model Test Schema (4-field)

Used for all 9-model standard tests reported in Table 5:

```
{
    "type": "json_schema",
    "json_schema": {
        "name": "inquiry_analysis",
        "strict": true,
        "schema": {
            "type": "object",
```

```
            "properties": {
                "buyer_background": {"type": "string"},
                "product_analysis": {"type": "string"},
                "recommendations": {"type": "string"},
                "key_findings": {"type": "array", "items": {"type": "string"}}
            },
            "required": ["buyer_background", "product_analysis", "recommendations", "key_findings"],
            "additionalProperties": false
        }
    }
}
```

A.5.2 Minimal Schema (3-field)

Used in fine-tuning ablation experiments and GRPO training:

```
{
    "type": "json_schema",
    "json_schema": {
        "name": "company_info",
        "strict": true,
        "schema": {
            "type": "object",
            "properties": {
                "company_name": {"type": "string"},
                "company_info": {"type": "string"},
                "compliance_notes": {"type": "string"}
            },
            "required": ["company_name", "company_info", "compliance_notes"],
            "additionalProperties": false
        }
    }
}
```

A.5.3 Production-Grade Schema

The full production schema used in the EvoAgent deployment includes a 4-card block structure with tool dependency tracking:

```
{
    "type": "json_schema",
    "json_schema": {
        "name": "inquiry_analysis",
        "strict": true,
        "schema": {
            "type": "object",
            "properties": {
                "blocks": {
                    "type": "array",
                    "items": {
                        "oneOf": [
                            {"type": "object", "properties": {"type": {"const": "text"}, "content": {"
                                type": "string"}}},
                            {"type": "object", "properties": {"type": {"const": "card"}, "card": {"
                                type": "object"}}}
                        ]
                    }
                },
                "tool_dependency": {
                    "type": "object",
                    "properties": {
                        "required": {"type": "boolean"},
                        "tools_used": {"type": "array", "items": {"type": "string"}},
```

```
                    "claims": {"type": "array"},
                    "reason": {"type": "string"}
                }
            }
        },
        "required": ["blocks"],
        "additionalProperties": true
    }
  }
}
```

A.6 Inference Framework and Tool Call Parser Configuration

A.6.1 SGLang Server Configuration

The primary inference framework used in this study:

```
python -m sglang.launch_server
    --model-path <path>
    --served-model-name <name>
    --port 8082 --host 0.0.0.0
    --tp-size 2 # 2x A800 80GB
    --mem-fraction-static 0.85
    --max-total-tokens 128144
    --max-running-requests 64
    --chunked-prefill-size 8192
    --enable-flashinfer
    --log-level warning
    --reasoning-parser qwen3
    --tool-call-parser qwen3_coder # Qwen3 XML to OpenAI format
    --trust-remote-code
```

A.6.2 vLLM Server Configuration

Used for framework independence validation:

```
python -m vllm.entrypoints.openai.api_server
    --model <path>
    --tensor-parallel-size 2
    --gpu-memory-utilization 0.85
    --max-model-len 4096
```

A.6.3 Tool Call Detection Mechanism

Dual detection ensures no missed or false-positive tool calls:

- **Method 1 — API structured** `tool_calls`**:** Parses the `tool_calls` array from streaming deltas
- **Method 2 — Content text-level check:** Scans for `<tool_call>` XML tags in generated content

In over 200 T2 queries, both detection methods yielded identical results (zero tool calls), ruling out parser-level false negatives.

A.7 Experimental Environment

Table 14: Experimental Environment Configuration

Component	**Configuration**
GPU	2x NVIDIA A800-SXM4-80GB
NVIDIA Driver	535.183.06
CUDA	12.6
SGLang	0.5.9
vLLM	0.22.0
Tool Parser (SGLang)	`qwen3_coder`
Tool Parser (vLLM)	`qwen3_coder`
Guided Decoding	xgrammar (SGLang), xgrammar (vLLM)
Python	3.12 / 3.13