Notation Matters: A Benchmark Study of Token-Optimized Formats in Agentic AI Systems

Anonymous ACL submission

Abstract

Large language models in Agentic AI systems consume tool schemas and execution results and emit tool invocations as structured data. The default language for that exchange, JSON, was designed for application-to-application interchange rather than token efficiency, so its structural elements impose substantial token overhead. Recent work proposes token-optimized alternatives such as TOON (Token-Oriented Object Notation) and TRON (Token Reduced Object Notation) as more compact replacements, but these formats have been evaluated only on isolated comprehension or generation tasks. Whether their token reductions hold inside end-to-end agentic loops therefore remains an open question. We evaluate TOON and TRON on four agentic benchmarks (BFCL, MCPToolBenchPP, MCP-Universe, StableToolBench) and five open-weight LLMs, decoupling input compression from output compression to measure comprehension and generation independently. TRON reduces tokens by up to 27% with accuracy within 14 pp of the JSON baseline. TOON achieves up to 18% reduction at a similar 9 pp accuracy cost, but additionally cascades on multi-turn parsing failures and collapses parallel tool-call output for most models.

1 Introduction

Large Language Models (LLMs) are increasingly deployed within Agentic AI systems, where they consume structured tool descriptions, interpret execution results, and emit tool invocations to accomplish user tasks. As these systems scale to more tools, longer descriptions, and richer result payloads, token consumption grows substantially. Because both inference cost and context-window limits are measured in tokens, the encoding of structured data is a direct determinant of system cost, energy efficiency, and feasibility.

In current tool-augmented pipelines, JSON is the default data format for tool schemas, invocations, and results [2], and LLMs interpret it effectively because of extensive exposure in their training corpora. JSON was, however, designed for application-to-application data interchange and not optimized for token efficiency, so its structural elements (quotation marks, commas, braces) impose substantial token overhead when consumed by LLMs. Recent work proposes token-optimized alternatives such as TOON (Token-Oriented Object Notation) [18, 8] and TRON (Token Reduced Object Notation) [7], with initial results suggesting that LLMs can read TOON with minimal accuracy loss on isolated generation tasks [10]. To date, however, no prior work has systematically evaluated these formats inside agentic pipelines that involve tool discovery, multi-turn reasoning, and execution.

We address this gap by evaluating token-optimized alternatives to JSON (illustrated in Figure 1a) in agentic tool-augmented systems and measuring whether they reduce token consumption while preserving task accuracy. Specifically, we benchmark TOON and TRON against JSON across function-calling and Model Context Protocol (MCP)-based benchmarks on five open-weight LLMs, substituting JSON at three points of the tool-calling loop (Figure 1b). We measure how much token reduction is achievable, how the substitution affects LLM accuracy in tool selection and parameter generation, and how both effects vary across model family, parameter scale, and architecture.

Our results indicate that TRON reduces total tokens by up to 27% while keeping accuracy within 14 pp of the JSON baseline across all four benchmarks. TOON achieves up to 18% token reduction at a similar 9 pp accuracy cost, but its accuracy degrades further in multi-turn settings where parsing failures cascade into additional reasoning iterations and erode per-call gains. The token reductions scale with schema complexity, with the largest gains on benchmarks that expose many tools with detailed parameter definitions. Figure 1c shows

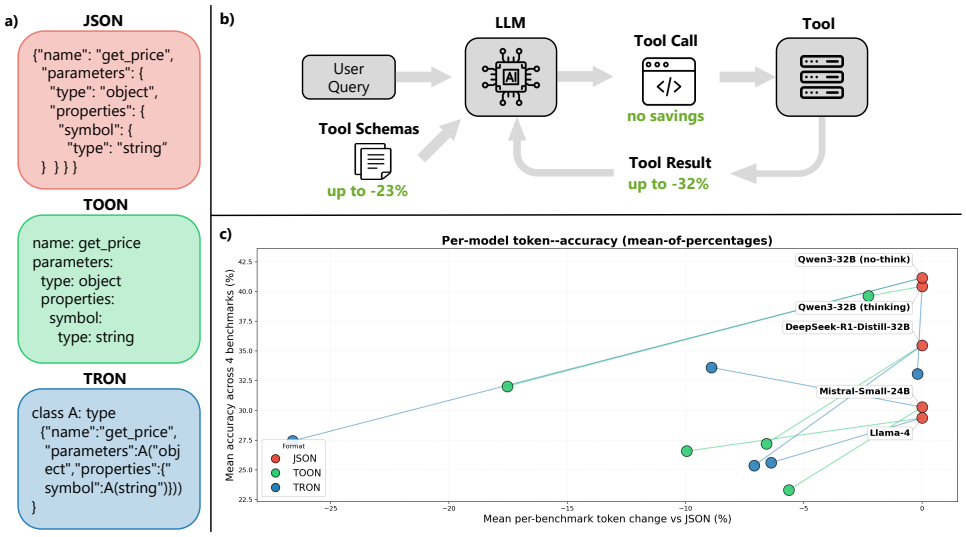

Figure 1: Format substitution in the tool-augmented pipeline. (a) The same tool schema encoded in JSON, TOON, and TRON. (b) The tool-calling loop, with green annotations marking the three format conversion points and the aggregate token saving observed at each. Tool schemas and tool results compress substantially (up to -23% and -32% respectively under TOON), while tool calls show no net saving due to parse failures in multi-turn benchmarks (see Sections 5 and 6). (c) Per-model token–accuracy view, averaged across the four benchmarks (mean of per-benchmark percentages). Each marker is one (model, format) combination, with the x axis showing mean per-benchmark token change vs JSON and the y axis showing mean accuracy across benchmarks. Lines trace the JSON $\rightarrow$ TOON and JSON $\rightarrow$ TRON movement per model. TRON arrows can reach further along the token axis (up to -27% on Qwen3-32B no-think) than TOON arrows (up to -18%), though both formats span a wide range across models (TRON 0 to -27%, TOON -2 to -18% per model). Accuracy drops also vary widely: TOON ranges from less than 1 pp on Qwen3-32B (thinking) to 9 pp on Qwen3-32B (no-think), while TRON ranges from a $+3$ pp gain on Mistral-Small-24B to a -14 pp drop on Qwen3-32B (no-think). Neither format dominates uniformly, and the format–model interaction is comparable in magnitude to the format choice itself.

the per-model token–accuracy tradeoff, and panel b annotates where each reduction accrues across the three substitution points of the tool-calling loop.

For deployments, our results identify TRON as a drop-in replacement for JSON in token-sensitive agentic systems, while TOON's stronger compression is offset by accuracy losses in multi-turn settings and is therefore not safe as a default. Researchers can use our evaluation framework to facilitate the evaluation of future serialization on the same comprehension and generation axes across four agentic benchmarks.

2 Background

Two background topics frame this study: the tool-calling loop through which LLMs interact with external systems, and the object-notation formats that encode the data flowing through it.

2.1 Tool Augmentation

Tool augmentation enables LLMs to query and modify external systems through a three-stage loop: the LLM receives structured tool definitions, generates a tool invocation, and receives the execution result. This loop runs once in single-turn tasks, where the model emits a single invocation and the interaction ends, and is iterated in multi-turn tasks, where the model interleaves reasoning with successive tool calls until producing a final answer. Across all implementations, JSON is the universal exchange format within Agentic AI. Current practice is shaped by two complementary standards: 1) function calling, the LLM-API convention for describing tools to the model and parsing its tool calls, and 2) the Model Context Protocol (MCP), the application-layer protocol that governs how applications discover and connect to remote tool servers.

Function calling is the convention in which tool definitions are provided as JSON Schema objects specifying a function name, description, and typed parameters. The LLM selects one or more functions and returns a structured invocation with the appropriate arguments. This convention originated with OpenAI's 2023 function-calling API and has since been adopted by most LLM providers and benchmarks. Evaluation typically relies on Ab-

stract Syntax Tree (AST) matching, comparing the structure and values of the predicted invocation against a reference.

MCP is an open-source standard for connecting LLMs to external tools, databases, and systems, analogous to how HTTP standardized web communication [13]. MCP uses a client-server architecture in which servers host and execute tools or data resources. Clients are LLM-based applications that discover and trigger these tools to fulfill user requests. Communication is managed via a data layer using JSON-RPC for structured exchange and a transport layer that defines message delivery. Each server exposes its capabilities through an MCP schema consisting of three core primitives: tools (executable functions), resources (contextual information sources), and prompts (reusable templates). This standardization reduces integration overhead and provides predictable interfaces for tool discovery and invocation.

2.2 Object Notation Formats

Tool schemas, invocations, and results are structured data, and the serialization format determines how that data appears as text in the prompt. We consider three formats. JSON is the universal baseline in current pipelines. TOON [18] and TRON [7] are two recently proposed formats that reduce token counts relative to JSON on the same data [8, 10, 11]. Figure 4 in Appendix E shows the three formats applied to the same example data.

TOON. Token-Oriented Object Notation (TOON) is a serialization format designed as a lossless, drop-in alternative to JSON, optimized for token efficiency [18]. TOON eliminates quotation marks, braces, and brackets by using YAML-like indentation for nested objects and CSV-style tabular notation for uniform arrays. This combination is particularly effective for structured data with repeated schemas, such as lists of objects sharing the same keys, and reduces token count substantially on such data [8, 10, 11]. The gains are largest on tabular structures and smaller for deeply nested or non-uniform data [8]. However, structural-correctness rates degrade for models without native support [10], which is the caveat our agentic-loop evaluation builds on.

TRON. Token Reduced Object Notation (TRON) is a compact serialization format that reduces redundancy through class definitions [7]. When multiple objects share the same structure, TRON defines the schema once as a named class and emits each object as a compact instance. Unlike TOON, TRON's compression scales with the number of repeated structures. This makes it effective for tool schemas, where multiple tools share similar parameter patterns. To exploit this scaling, tool schemas must be serialized as a single batch so that TRON can identify shared structures across tools. We refer to this requirement as the TRON batching condition. JSON and TOON do not benefit from such batching and are serialized individually.

3 Related Work: Alternative Data Formats for LLMs

Several recent works have proposed token-optimized serialization formats as alternatives to JSON for structured data exchanged with LLMs. Lafalce [8] provided a mathematical evaluation of TOON, showing that it reduces byte size compared to JSON, with particularly strong compression on tabular data. Masciari et al. [10] extended this evaluation to generation tasks, benchmarking TOON against JSON, XML, and YAML across multiple LLMs. Their results show that TOON yields more compact outputs and lower carbon emissions, but structural correctness decreases for models that were not exposed to TOON during pre-training, with larger models closing this gap.

The accuracy impact of TOON is highly model-dependent. McMillan [12] ran 9,649 prompt-completion trials across 11 models and 4 formats (YAML, Markdown, JSON, TOON) on SQL generation tasks, finding that format does not significantly affect aggregate accuracy (χ^2=2.45, p=0.484), though individual models show format-specific sensitivities with accuracy changes ranging from -7.7% to +2.7%. Consistent with Masciari et al. [10], the dominant factor was model capability, with a 21 pp gap between frontier and open-source tiers, dwarfing any format effect. Matveev [11] compared TOON generation against plain JSON and constrained-decoding JSON, finding that TOON has a favorable accuracy-to-token ratio for in-domain tasks but that the prompt overhead needed to teach the model TOON syntax can wipe out the per-token savings on short outputs. TOON therefore only becomes cost-effective once the output is large enough for cumulative per-token savings to outweigh the fixed prompt cost.

Concurrent work has explored further JSON alternatives. JTON [15], like TRON, factors shared

structure across repeated objects through a Zen Grid tabular encoding. It reports 15–60% token reduction across seven domains, with comprehension and one-shot generation tests on 10–12 LLMs. ONTO [3] uses pipe-delimited columnar notation declaring each entity's fields once, reducing tokens by 46–51% on IoT data, with comprehension verified on Qwen2.5-7B. Neither JTON nor ONTO are evaluated inside multi-turn agentic loops, and JTON's batching is restricted to homogeneous arrays whereas TRON generalizes to repeated structural patterns across heterogeneous tool definitions.

Beyond general-purpose formats, domain-specific notations have emerged for time-series data. TSLN [14] achieves 68–73% token reduction through schema-first architecture, relative timestamps, and differential encoding. TOKON [19] reduces tokens by 2–3× through tokenization-optimized normalization, improving forecasting root mean square error by 7–18%. Both TSLN and TOKON are orthogonal to our work: they target time-series payloads specifically, whereas tool-augmented pipelines exchange heterogeneous tool-schema, tool-call, and tool-result data. Alshaer [1] identified that TOON's lack of explicit delimiters allows attacker-controlled string content to be reparsed as schema fields, and proposed the S-TOON protocol, which reintroduces explicit boundary markers around untrusted input. Although our work does not target adversarial inputs, the existence of this attack underlines that format choice has consequences beyond efficiency.

The works above show that token-optimized formats can reduce token consumption with varying accuracy trade-offs. Several of them already evaluate format substitution systematically across multiple LLMs and formats [10, 12, 11], but only for isolated comprehension or generation tasks. Our work is, to our knowledge, the first systematic evaluation of token-optimized formats inside agentic tool-calling pipelines, the first to decouple input-side compression from output-side compression as separate experimental conditions, and the first to report TRON's effect on LLM accuracy.

4 Experimental Setup

To evaluate whether LLMs can both consume and produce token-optimized formats inside agentic pipelines, we substitute JSON with TOON and TRON at three points of the tool-calling loop:

1) Tool schemas (input): The list of available tool definitions, including each tool's name, description, and typed parameter specification, is serialized in the target format and included in the system prompt at session start. This block is typically the largest single piece of structured data in the prompt, since it scales with the number of available tools.

2) Tool calls (output): At each agent step, the LLM emits a structured invocation in the target format (JSON, TOON, TRON) specifying the tools to call and its arguments. The agent deserializes this and re-encodes the arguments as JSON before invoking the tool, since function-calling APIs and MCP's JSON-RPC transport both require JSON-encoded arguments.

3) Tool results (input): Execution results returned by tools or MCP servers are serialized in the target format and appended to the conversation context for the next reasoning step. Results grow across iterations in multi-turn execution, since each tool call appends fresh content to the conversation context for the next reasoning step.

From the LLM's perspective, we decouple input and output formats. The input format governs how tool schemas and results are presented to the LLM, while the output format governs how the LLM must structure its tool invocations. This decoupling enables two experimental conditions. First, **input-only** compression, where the LLM consumes schemas and results in the target format but still emits tool calls as JSON. Second, **full** compression, where the LLM works in the target format end-to-end and also emits tool calls in the same format. All format conversions are routed through a unified serialization library (shared_format) that centralizes encoding and decoding behavior across the four benchmarks. At every substitution point, the LLM emits the target format as a stand-alone document: the entire per-turn response is a single TOON or TRON object with explicit `thought`, `action`, and `arguments` keys, rather than wrapped inside a labeled plain-text field such as the ReAct convention's `Action Input` [20] (the Action-Input anti-pattern, Appendix B).

4.1 Benchmarks

We use four tool-augmented benchmarks that together cover function calling and MCP, each in a single-turn and a multi-turn setting. BFCL [16] and StableToolBench [6] provide single- and multi-turn function calling, and MCPToolBenchPP [5] and

MCP-Universe [9] provide the same two settings for MCP. We run all four benchmarks in prompt mode, since the native function-calling APIs of inference providers serialize tool schemas internally and emit tool calls only as JSON, which prevents the format substitution we study.

These four benchmarks offer diversity in multiple characteristics, that are essential to our evaluation: the shape of data, the structure of the interaction, tool-catalog size, result payload depth and single-/multi-turn settings. Appendix A compares these benchmarks and explains their diversity in the mentioned aspects.

4.2 Evaluation Metrics

Token Consumption Metrics. We decompose token consumption into schema tokens, tool-call tokens, and result tokens, all measured with the cl100k_base tokenizer from tiktoken. Total prompt and completion tokens per call are reported by the inference API (vLLM for self-hosted runs). Token-consumption metrics are defined consistently across all four benchmarks. For cross-benchmark comparison we also report each format's relative change against the JSON baseline as a percentage gain or loss, which yields a benchmark-agnostic measure of format effect.

Accuracy Metrics. Task accuracy uses each benchmark's own standard evaluation, since each benchmark defines a task-appropriate notion of correctness. BFCL uses AST-matching accuracy against a reference invocation. MCPToolBenchPP reports a hierarchical pass@1 over five trials, with overall pass@1, tool_pass@1 for tool selection, and param_pass@1 for parameter correctness given a correct tool. MCP-Universe reports a strict pass rate, where a task counts as solved only if all of its evaluators succeed. StableToolBench reports the Solvable Pass Rate (SoPR), the fraction of solvable queries judged to fulfill the user's intent.

4.3 Model Configuration

Format sensitivity may depend on the model. To check whether observed effects are general or model-specific, we evaluate five open-weight LLM configurations against all four benchmarks. The five configurations span four model families (Qwen3, DeepSeek, Mistral, Llama) and vary along four axes: parameter scale (17B–32B active parameters), architecture (dense versus Mixture-of-Experts), training emphasis (Qwen3 and Mistral are explicitly trained for tool calling, while DeepSeek-R1-Distill is reasoning-distilled from Qwen-2.5-32B, and Llama-4-Scout is the MoE variant in the Llama-4 family), and reasoning mode (Qwen3-32B is evaluated with thinking on and off as two separate configurations). Table 2 in Appendix C lists the full configuration.

5 Results

We address the three research questions across the format–model matrix:

RQ1 How much token reduction is achievable when replacing JSON with token-optimized alternatives in agentic tool-augmented systems?

RQ2 Does format substitution affect LLM accuracy in tool selection and parameter generation?

RQ3 How does the effect of token-optimized formats vary across open-weight LLMs that differ in family, parameter scale, and architecture?

We evaluate two settings. **Experiment 1 (input-only compression)** substitutes JSON with TOON or TRON for schemas and results while tool calls remain JSON, isolating format comprehension from generation. **Experiment 2 (full compression)** additionally emits tool calls in the target format, testing end-to-end production. Figure 2 reports mean accuracy per model and format for both settings, and Figure 3 reports the per-model token–accuracy tradeoff. Per-cell numbers are in Tables 3 and 4.

Token reduction (RQ1). Under Experiment 1, averaged per model across the four benchmarks, TOON reduced total tokens by 2–18% and TRON by 0–27% (Figure 3, left), with both maxima reached on Qwen3-32B (no-think). The TRON batching condition (Section 2.2) determines where TRON's savings concentrate: the class-definition mechanism delivers consistent reductions on benchmarks whose tasks expose many tools per query (MCPToolBenchPP saves roughly 10% per model, StableToolBench up to 22%) but *increases* tokens by up to 21% on BFCL's non-live categories, where each task references only a few unique tool schemas and the per-batch class header costs more than the body it deduplicates. Under Experiment 2, adding output-side compression does not uniformly increase savings. Non-reasoning

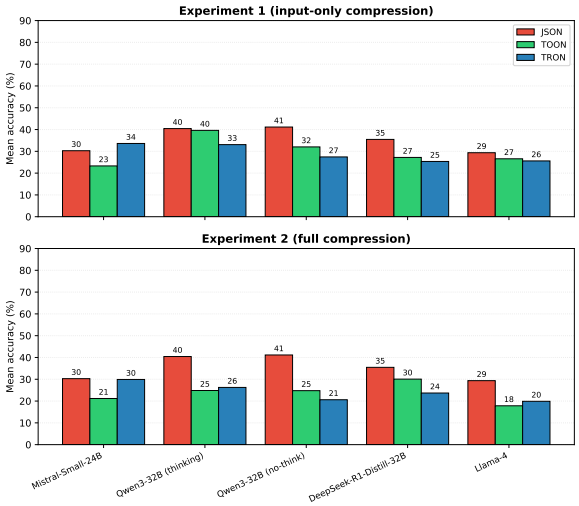

Figure 2: Mean accuracy per model and format, averaged across benchmarks. Each triple of grouped bars compares JSON, TOON, and TRON within one model. Top: **input-only** compression. Bottom: **full** compression, including tool calls. In general, all models suffer accuracy losses under compression, with TOON dropping more than TRON. Exceptions are Qwen3-32B with thinking whose TOON accuracy matches its JSON baseline under input-only compression, and Mistral-Small-24B, which achieves JSON accuracy with TRON.

configurations (Mistral-Small-24B, Llama-4-Scout, DeepSeek-R1-Distill) continued to save 1–10% of total tokens on average, but the reasoning configurations (Qwen3-32B with and without thinking) drifted toward or above the JSON baseline (Figure 3, right). The drift is concentrated on the multi-turn benchmarks (MCP-Universe and StableToolBench), where the parsing-cascade applies: a failed parse of a tool call triggers an additional reasoning iteration whose own thought, action, and observation payload outweighs the per-call savings. After partial offset by savings on the single-turn benchmarks, the per-model means resolve to $+8\%$ (Qwen3-32B with thinking, TOON), $+11\%$ (Qwen3-32B with thinking, TRON), and $+11\%$ (Qwen3-32B without thinking, TOON).

Accuracy (RQ2). Under Experiment 1, TRON's accuracy averaged across the four benchmarks varied between $+3$ pp on Mistral-Small-24B and -14 pp on Qwen3-32B (no-think) (Figure 3). TOON's accuracy varied more widely. The single largest drop was Mistral-Small-24B on BFCL's non-live categories, where AST-matching accuracy fell from 89% under JSON to 53% under TOON (visible as the lower-left Mistral point in the BFCL panel of Figure 5, Table 3). We hypothesise that smaller models with limited pre-training exposure to TOON syntax are particularly vulnerable on BFCL's strict AST evaluation, consistent with the structural-correctness gap that Masciari et al. [10] report for smaller models without native TOON support. The same Mistral configuration, however, *gains* accuracy on MCPToolBenchPP under both TOON ($+13$ pp at -29% tokens) and TRON ($+17$ pp at -10% tokens), suggesting that the format's effect interacts strongly with task type rather than acting uniformly. Under Experiment 2, full compression exposes a sharp degradation on BFCL's `parallel` and `parallel_multiple` categories (`parallel`: the same function called multiple times in parallel, `parallel_multiple`: several different functions called in parallel), where accuracy collapses to near zero for most (model, format) combinations. This is the most significant negative finding of the study: our models, which are pre-trained predominantly on JSON tool-call output, do not reliably generate parallel invocations in alternative formats because the structural cues that disambiguate sequential from parallel calls are not consistently preserved when the format is substituted. On the multi-turn benchmarks the parsing-cascade compounds under full compression, because parsing failures during the output stage themselves trigger further reasoning iterations. DeepSeek-R1-Distill-32B is the only configuration whose mean TOON accuracy improved under Experiment 2 relative to Experiment 1 ($+3$ pp), driven entirely by StableToolBench where its SoPR jumped from 22% to 46%. We flag this as an outlier without a mechanistic explanation.

Per-model variation (RQ3). Under Experiment 1, Qwen3-32B with thinking enabled was the only configuration whose TOON accuracy matched its JSON baseline. Toggling thinking off on the same base model dropped TOON accuracy by 8 pp and TRON accuracy by 6 pp, indicating that the explicit reasoning step helps the model recover from format-driven structural surprises and that model size and family are not the only drivers for robustness. The other reasoning-capable configuration, DeepSeek-R1-Distill-32B, did not inherit this robustness: its TRON accuracy dropped by 10 pp on average, the second-largest TRON drop in the matrix after Qwen3-32B (no-think, Table 3). Llama-4-Scout-17B's MoE architecture did not show a systematic advantage on input compression over comparable dense models. Under Experiment 2,

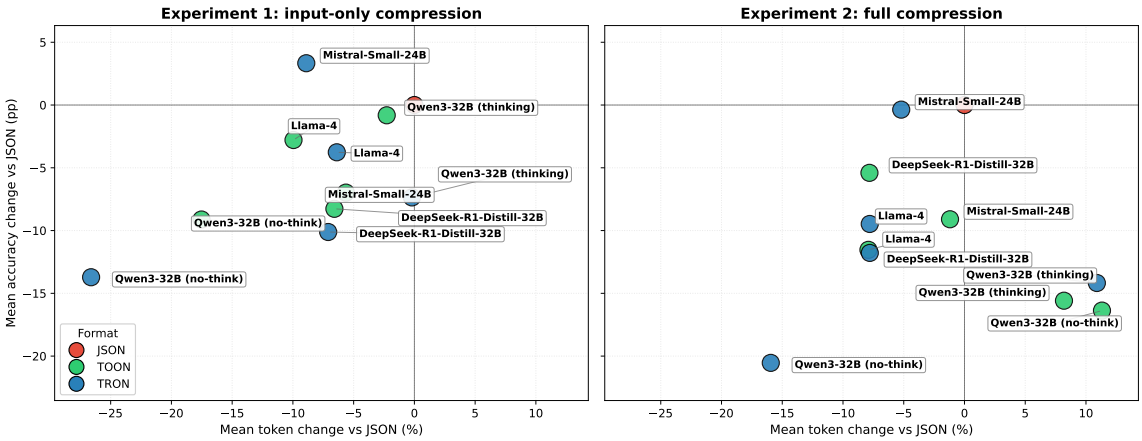

Figure 3: Per-model token–accuracy tradeoff against the JSON baseline, averaged across benchmarks. **Left:** input-only compression. **Right:** full compression, including tool calls. Each marker is one (model, format) combination, and the red JSON marker sits at the origin in both panels. Under input-only compression, format effects are model-dependent: Mistral-Small-24B gains accuracy under TRON, while reasoning configurations (Qwen3-32B no-think, DeepSeek-R1-Distill) lose more accuracy on TRON than on TOON. Under full compression, several configurations drift to the right of the origin because the parsing-cascade pushes total tokens above the JSON baseline, and accuracy losses widen for both formats.

the thinking-mode advantage changes character. For TRON output, the gap between Qwen3-32B with and without thinking persists at 5 pp (26% versus 21% mean accuracy), but for TOON output the gap vanishes entirely: both configurations land at 25%. Explicit reasoning thus helps the model generate well-formed TRON under full compression but does not rescue its TOON generation. Llama-4-Scout's MoE architecture again showed no systematic advantage on alternative output formats. Across the five configurations, no single model succeeded under full compression: every configuration lost at least 4 pp of accuracy under TOON, and only Mistral-Small-24B retained TRON accuracy within 0.4 pp of JSON.

6 Discussion

Token cost: when compression delivers and when it backfires (RQ1). The token effects of TOON and TRON in our matrix are governed by two interacting mechanisms. TRON's class-definition mechanism only pays off when the serialization batch contains repeated structural patterns. In our setup, this condition is met when a task exposes many tools whose parameter schemas share fields. This is the typical shape of MCP servers and of StableToolBench's API bundles, and TRON reduces tokens consistently on these benchmarks (up to 27% on Qwen3-32B no-think). The same mechanism becomes a liability on BFCL's non-live categories. Each task there references only a handful of unique tool schemas, and the per-batch class header costs more than the body it deduplicates, so total tokens *increase* by up to 21%. A second backfire mechanism is specific to multi-turn pipelines: a parse failure on a single tool call triggers an additional reasoning iteration, and that iteration carries its own thought, action, and observation payload. For reasoning configurations this extra payload includes the model's full chain of thought. Two or three triggered iterations are then sufficient to consume the per-call savings accumulated over an entire trajectory. Qwen3-32B with thinking ends at $+8\%$ tokens under full TOON and $+11\%$ under full TRON, and Qwen3-32B without thinking at $+11\%$ under TOON (Tables 3 and 4). Whether a token-optimized format saves tokens therefore depends on the shape of the workload and on the model's parse-success rate, not on per-call format compression in isolation.

Accuracy can improve under compression (RQ2). Several configurations in our matrix improved accuracy under compression rather than degrading. The clearest case is Mistral-Small-24B on MCPToolBenchPP, where accuracy rose by $+13$ pp under TOON at -29% tokens and by $+17$ pp under TRON at -10% tokens (visible in the MCPToolBenchPP panel of Figure 5; per-cell numbers in Table 3). We hypothesise that compression reduces noise on the model's attention surface, freeing attention budget for the task con-

tent. Removing JSON's structural elements (braces, brackets, repeated key strings) drops a substantial share of tokens that carry no task-relevant signal. This reading is consistent with the long-context literature, which reports smaller models benefiting from denser prompts more than larger ones. It also explains why the gain was largest on the smallest model in our matrix and on the benchmark with the most verbose JSON schemas. The effect was not uniform across the matrix, so we treat it as a hypothesis for follow-up rather than as a deployment recommendation.

Accuracy losses follow a coherent pattern (RQ2). The accuracy losses we observe under compression cluster on two failure modes. Under input-only compression, the losses concentrate on models whose pre-training exposure to the target format is presumably low, with the largest single drop on Mistral-Small-24B's BFCL accuracy (89% under JSON to 53% under TOON, Table 3). Under full compression the failure mode shifts from comprehension to generation. BFCL's `parallel` and `parallel_multiple` categories collapse to near-zero accuracy across most (model, format) combinations, suggesting that LLMs trained predominantly on JSON tool-call output do not reliably reproduce the parallel-call structure in alternative formats. The Action-Input anti-pattern surfaced during pilot runs reinforces this reading: when TOON content was embedded inside a labelled plain-text field, models defaulted back to JSON-shaped output regardless of the surrounding format instruction. The shared mechanism is that JSON output is the deeper prior, and alternative formats require either explicit reasoning at inference time or explicit training-time exposure.

Training emphasis and reasoning mode (RQ3). The five model configurations differ along four axes (parameter scale, architecture, training emphasis, reasoning mode), and the dominant variable in our matrix is reasoning mode. Qwen3-32B with thinking enabled is the only configuration whose TOON accuracy matches its JSON baseline under input-only compression. Toggling thinking off on the same base model drops TOON accuracy by 8 pp and TRON accuracy by 6 pp. We hypothesise that explicit reasoning lets the model recover when it encounters an unfamiliar input format, by reasoning over the structure before committing to an answer. The other reasoning-capable configuration, DeepSeek-R1-Distill-32B, did not inherit this robustness and dropped 10 pp on TRON (Table 3). This suggests that distillation from Qwen-2.5 does not transfer format-side robustness, and that explicit chain-of-thought at inference time is the load-bearing variable. Llama-4-Scout-17B's MoE architecture did not show a systematic advantage over comparable dense models, indicating that MoE routing did not select format-specialised experts on our matrix. Tool-calling training emphasis, present in both Qwen3 and Mistral, did not protect Mistral from BFCL's strict AST evaluation under TOON, indicating that tool-call training transfers to comprehension but not to robust generation in unfamiliar formats. The pattern across the four axes is that no single architectural or training choice is decisive on its own, and that an inference-time reasoning step is the only variable that yields a consistent advantage in our matrix.

Practical guidance. For deployments, our results identify TRON as a defensible drop-in for JSON in token-sensitive agentic systems when the workload exposes many structurally similar tools, and otherwise as a workload-dependent choice that should be measured rather than assumed. TOON is not safe as a default in multi-turn agentic systems, because its stronger per-call compression is offset by parsing-cascade effects and by structural failures on parallel tool-call output.

7 Conclusions

We evaluated TOON and TRON inside agentic tool-calling pipelines on four benchmarks and five open-weight LLM configurations, decoupling the input and output sides of the loop so that comprehension and generation could be measured independently. TRON behaves as a drop-in replacement for JSON under input-only compression, reducing total tokens by up to 27% with accuracy within 14 pp of the JSON baseline, provided the workload exposes enough structurally similar tools for its class-definition mechanism to amortize. TOON yields up to 18% reduction at a similar 9 pp accuracy cost. However, TOON loses accuracy further in multi-turn settings, where parsing failures cascade into additional reasoning iterations and erode per-call gains. Follow-up work should characterise the interaction between format substitution and tool-call training emphasis on frontier-scale models, and evaluate whether targeted post-training on tool-call outputs in alternative formats removes the structural failures we observe under full compression.