TSCG: Token-Context Semantic Grammar for Causal Prompt Optimization in Large Language Models

Furkan Sakizli
Independent Researcher
furkan.sakizli@gmail.com

Abstract

Production agentic frameworks universally transmit tool definitions as JSON schemas, yet small language models (4B–14B) parse these poorly at scale—achieving 0–49% tool-selection accuracy with >15 tools. We introduce **Token-Context Semantic Grammar (TSCG)**, a deterministic tool-schema compiler comprising eight causally-grounded principles that transforms JSON schemas into token-efficient text representations. On the TSCG-Agentic-Bench (TAB)—∼13,000 API calls across 12 models (4B–32B local, 3 frontier APIs) and 5 scenarios—TSCG recovers small-model accuracy to 65–90%, enabling them as functional tool-use agents. A systematic format decomposition (2,940 Ollama calls) identifies the mechanism: TSCG's gains arise from implicit format translation (JSON→text), not structural compression—but since all production APIs transmit JSON, this is the precisely needed intervention. For frontier models, TSCG provides genuine structural compression (+5–11 pp for Claude Sonnet 4; scenario-dependent for GPT-5.2) persisting without the format confound, with 50–72% token savings. We establish compression guarantees (≥51% on well-formed schemas), derive a predictive model (R^2=0.88 on JSON baselines, collapsing to 0.03 on text baselines), and introduce a four-class behavioral taxonomy for deployment guidance. The implementation—1,200 LOC TypeScript, zero dependencies, sub-millisecond—is open-source infrastructure for the agent framework stack.

1 Introduction

Every time Claude Code spawns a subprocess, it injects approximately 50,000 tokens of tool definitions into the context window—16 tools repeated verbatim on every API call. Open-source 7B models cannot perform reliable tool-use beyond 15–20 tools because the JSON schema overhead consumes their effective context. This is not a prompt engineering problem—it is an infrastructure problem, and it is the dominant cost driver in agentic AI deployments.

Modern agent frameworks—Claude Code, LangChain, CrewAI, AutoGen, OpenAI Assistants—embed dozens to hundreds of tool definitions in every API call. Each tool schema consumes 100–500 tokens; a typical agent invocation with 30 tools dedicates 3,000–25,000 tokens to tool descriptions alone, *on every call*. This infrastructure gap imposes three costs: (1) **Token cost**: tool schemas are pure structural redundancy transmitted identically on every call; (2) **Capability cost**: small models (4B–14B) cannot parse JSON-format schemas reliably at scale (0–49% accuracy at >15 tools), locking agentic capabilities behind expensive frontier APIs; (3) **Scaling cost**: schema overhead grows as $O(n \cdot k)$ where n is calls and k is tokens per schema.

No existing framework simultaneously satisfies five desiderata for agentic tool-schema optimization:

1. **Tokenizer awareness**: accounting for BPE boundaries (Sennrich et al., 2016)
2. **Causal attention grounding**: exploiting attention sink (Xiao et al., 2023) and recency bias
3. **Deterministic transforms**: same input → same output
4. **Black-box compatibility**: no model access required
5. **Budget-constrained anchoring**: selective reinforcement of critical information

We introduce **Token-Context Semantic Grammar (TSCG)**, a deterministic tool-schema compiler that reduces agentic token overhead from $O(n \cdot k)$ to $O(n \cdot k/c)$ where $c \approx 3.5$ for structured schemas. TSCG implements eight composable principles, each grounded in autoregressive

Model	Condition	Accuracy (%)				Savings	
		Sc. A	95% CI	Sc. B	95% CI	Tokens	ARR
Claude Sonnet 4	Natural (FC)	74.0	[62.8, 84.2]	90.0	[86.7, 93.3]	—	—
	Natural (text)	51.1	[39.1, 63.1]	70.0	[58.3, 81.7]	—	—
	TSCG	**85.2**	[77.3, 92.1]	**95.0**	[92.3, 97.3]	50.1%	1.15
	TSCG+SAD	76.7	[66.7, 86.1]	95.0	[92.3, 97.3]	49.5%	1.04
GPT-4o	Natural (FC)	55.5	[42.7, 67.6]	70.0	[64.7, 75.0]	—	—
	Natural (text)	45.7	[33.0, 58.4]	55.0	[41.7, 66.7]	—	—
	TSCG	56.5	[44.2, 68.7]	**79.7**	[75.1, 84.2]	6.2%	1.02
	TSCG+SAD	53.2	[40.3, 65.6]	81.4	[77.0, 85.5]	5.3%	0.96
GPT-5.2	Natural (FC)	51.9	[40.5, 63.5]	82.4	[78.1, 86.5]	—	—
	Natural (text)	78.3	[69.5, 86.6]	**98.2**	[96.6, 99.4]	—	—
	TSCG	**81.6**	[73.8, 88.2]	91.6	[88.9, 94.0]	11.4%	1.57
	TSCG+SAD	**85.9**	[80.1, 90.9]	90.9	[88.0, 93.6]	10.5%	1.65

Table 1: TAB Benchmark: Frontier model results on Scenario A (20 tasks, small catalog) and Scenario B (100 tasks, large catalog). Each condition evaluated with 3 runs ($n = 60$ per cell for Sc. A, $n = 300$ for Sc. B). *Natural (FC)* uses the provider's native function calling API; *Natural (text)* embeds full JSON schemas as text; *TSCG* uses balanced compression as text; *TSCG+SAD* adds aggressive compression with Selective Anchor Duplication. Token savings are relative to Natural (FC). ARR = Accuracy Retention Rate (TSCG accuracy / Natural FC accuracy; values > 1.0 indicate TSCG outperforms the baseline). **Bold** marks best accuracy per model per scenario.

Model	Format Effect		Compression Effect	
	Sc. A	Sc. B	Sc. A	Sc. B
Claude Sonnet 4	−22.9	−20.0	+34.1	+25.0
GPT-4o	−9.8	−15.0	+10.8	+24.7
GPT-5.2	+26.4	+15.8	+3.3	−6.6

Table 2: Decomposition of the TSCG advantage into format and compression effects (percentage points). *Format effect* = Natural (text) − Natural (FC): positive values indicate text-mode is better than native function calling. *Compression effect* = TSCG − Natural (text): positive values indicate compression improves accuracy beyond uncompressed text. For Claude and GPT-4o, compression is the dominant driver (overcoming a negative format penalty). For GPT-5.2, the text format itself provides the main benefit.

transformer processing:

- TAS: aligns delimiters with BPE boundaries
- CFL: exploits attention sink at position 0
- CFO: reorders operations into causal chains
- SDM: removes 104+ filler patterns
- DRO: converts verbose phrases to compact delimiters
- CCP: appends closure block for recency bias
- CAS: scores and reorders by fragility
- SAD-F: budget-constrained anchor duplication

1.1 Contributions

1. **Formal optimization framework**: eight principles with mathematical specifications linked to transformer mechanisms (§3).
2. **Compression guarantees**: deterministic ≥51% savings on well-formed schemas (Theorem 3.1), operator taxonomy with interaction effect explaining conservative vs. balanced profiles (Corollary 3.1).
3. **Predictive model**: R^2=0.88 ($p < 10^{-20}$) predicts TSCG benefit from a single baseline measurement; format-confound decomposition reveals R^2 collapses to 0.03 against text baselines (Figure 4).
4. **TAB benchmark**: ∼13,000 API calls, 12 models (4B–32B + frontier), 5 scenarios—the first tool-schema compression benchmark.
5. **JSON-API enablement**: seven small models achieve 0–49% on JSON tools at >15 tools; TSCG recovers to 65–90%. E1/E4 (2,940 calls) identifies format translation as the mechanism (§5.2).
6. **Frontier compression**: +5–11 pp for Claude Sonnet 4 persists when format confound is eliminated; GPT-5.2 shows scenario-dependent variation (§5.3).
7. **External validation**: BFCL ARR 108% at 46.8% savings; GSM8K reasoning preserved under tool-schema load.
8. **Implementation**: 1,200 LOC TypeScript, zero dependencies, sub-millisecond, 34.7 KB bundle, open-source.

2 Related Work

We organize related work into four categories and position TSCG within the design space.

Model	Tools	Accuracy (%)			Δ vs Natural (pp)		
		Natural	$\text{TSCG}_{\text{cons}}$	TSCG_{bal}	$\text{TSCG}_{\text{cons}}$	TSCG_{bal}	cons − bal
Mistral 7B	10	83.5	76.0	73.3	−7.5	−10.2	+2.7
	20	35.0	80.0	80.1	+45.0	+45.1	−0.1
	50	30.0	**75.3**	65.0	+45.3	+35.0	+10.3
Gemma 3 4B	10	69.1	**80.7**	74.7	+11.6	+5.6	+6.0
	20	49.9	**87.3**	67.0	+37.4	+17.1	+20.3
	50	24.3	87.5	**87.4**	+63.2	+63.1	+0.1
Gemma 3 12B	10	90.0	—	**93.0**	—	+3.0	—
	20	85.0	—	**95.0**	—	+10.0	—
	50	85.0	—	**98.0**	—	+13.0	—
Qwen3 14B	10	94.9	**98.8**	86.2	+3.9	−8.7	+12.6
	20	90.2	**99.3**	84.1	+9.1	−6.1	+15.2
	50	94.6	**95.0**	89.6	+0.4	−5.0	+5.4

Table 3: TAB Scenario D: Small model accuracy across catalog sizes (3 runs each, $n = 60$ per cell). *$TSCG_{cons}$* = conservative profile (SDM filler removal only); *$TSCG_{bal}$* = balanced profile (full structural compression). At ≥20 tools, both TSCG profiles provide massive improvements (+17–63pp) for most models. Conservative outperforms balanced for Mistral 7B and Gemma 3 4B in 5/6 cases. For Qwen3 14B, where balanced TSCG degrades accuracy (−5 to −9pp), conservative mode *improves* accuracy (mean Δ: +4.4pp, N3 re-run with corrected profile). **Bold** marks best TSCG variant per row.

Condition	Acc.	Tool Sel.	Param F1	Savings
Natural	85.7%	86.7%	84.2%	—
TSCG	**93.2%**	**95.0%**	**91.7%**	46.8%
TSCG+SAD	89.0%	90.0%	87.5%	46.0%

Table 4: BFCL external validation on Claude Sonnet 4 ($n = 60$, 3 runs). TSCG improves accuracy by +7.5pp while saving 46.8% tokens. TSCG+SAD slightly underperforms balanced TSCG on this benchmark. 95% CI for TSCG: [86.3, 98.3].

Model	Tools	$\text{Param F1}_{\text{cons}}$	$\text{Param F1}_{\text{bal}}$
Mistral 7B	10	71.2%	68.3%
	20	**81.9%**	80.0%
	50	**78.3%**	66.7%
Gemma 3 4B	10	**74.2%**	62.6%
	20	**85.7%**	59.3%
	50	90.0%	87.8%
Qwen3 14B	10	**97.8%**	84.7%
	20	**98.3%**	81.0%
	50	**95.0%**	89.0%

Table 5: Parameter F1 comparison between conservative (SDM-only) and balanced (full structural) TSCG profiles. Conservative achieves higher param-F1 in 8/9 cases. For Qwen3 14B (N3 re-run), conservative achieves near-perfect param-F1 (>95%) while balanced degrades it (<85%). This confirms that aggressive structural compression disrupts well-learned schema patterns in strongly fine-tuned models. **Bold** marks the better profile per row.

Compression-based methods. LLMLingua (Jiang et al., 2023) and LLMLingua-2 (Pan et al., 2024) achieve up to 20× compression on natural prose through perplexity-based token importance scoring, but require GPU model inference, produce non-deterministic output, and—critically—are ineffective on structured content (80% accuracy on tool schemas vs. TSCG's 93.3%; the compound pipeline yields 0%, §5.4). Selective Context (Li et al., 2023) uses self-information thresholds but lacks tokenizer and positional awareness. CompactPrompt (Anonymous, 2025) is the only other LLM-free heuristic, achieving 1.44× via n-gram abbreviation—versus TSCG's 3.5× on structured content—limited by lexical-level operation.

Search-based methods. DSPy (Khattab et al., 2023) compiles declarative LLM calls into self-improving pipelines via bootstrapped demonstrations. SAMMO (Schlegel et al., 2024) shares TSCG's "compile-time" framing but requires an LLM for search-based mutations and produces non-deterministic output. OPRO (Yang et al., 2024), APE (Zhou et al., 2023), and EvoPrompt (Chen et al., 2024) all require iterative model interaction. These systems optimize prompt *content* (what to say); TSCG optimizes prompt *structure* (how to represent it).

Gradient and template methods. TextGrad (Yüksekgonul et al., 2024) introduces textual gradients for prompt refinement but requires model access. LangGPT (Wang

Model	Params	TSCG Δ Acc. (avg)	Avg. Token Savings	Recommended Profile
GPT-5.2	>100B	+10.9pp	11.4%	balanced
Claude Sonnet 4	>100B	+8.4pp	50.1%	balanced
GPT-4o	>100B	+5.4pp	6.2%	balanced
Qwen2.5-Coder 32B	32B	−4.7pp	—	conservative
Mistral-Small 24B	24B	−1.1pp	—	conservative
Gemma 3 12B	12B	+8.7pp†	—	conservative
Qwen3 14B	14B	+4.4pp	—	conservative
Gemma 3 4B	4B	+28.3pp†	—	conservative
Mistral 7B	7B	+26.2pp†	—	conservative

Table 6: Cross-model summary of TSCG effectiveness across 9 models. Δ Acc. uses recommended profile (conservative for all small/mid models). †JSON-baseline gains; E4 text-baseline experiments show these are format-dominated (compression gain is −7 to −9 pp). 30B models (N1): Mistral-Small −1.1 pp (neutral, Class 3), Qwen2.5-Coder −4.7 pp (Class 4). Qwen3 14B: conservative +4.4 pp (N3 re-run), vs balanced −6.6 pp. Frontier models are the *only* class with confirmed compression benefit against text baselines.

Table 7: TAB (Tool-Augmented Benchmark) overview: five core scenarios plus two external benchmarks testing TSCG across diverse tool counts, model classes, and compression profiles. Total evaluation volume: ∼13,000 API calls across 9 statistically significant comparisons (Holm–Bonferroni, $\alpha = 0.05$).

Scenario	Description	Tools	Tasks	Models
A	Claude Code Catalog	16	20	Claude Sonnet 4, GPT-4o, GPT-5.2
B	MCP Server Collection	43	100	Claude Sonnet 4, GPT-4o, GPT-5.2
C	Scaling Stress	25–100	20	Claude Sonnet 4, GPT-5.2
D	Small Model Threshold	3–50 (7 sizes)	20	7 models (4B–14B)
E	Multi-Collection Stress	57 (3 overlapping)	20	Claude Sonnet 4, GPT-5.2
BFCL	External validation	real-world	20	Claude Sonnet 4
GSM8K	Reasoning under load	0/10/25/50	200	Claude Sonnet 4, GPT-5.2

Table 8: Native function calling (via API) vs. TSCG as plain text. TSCG text outperforms native FC across all frontier models and scenarios, with deltas from +1.0 to +29.7 pp (mean +10.9 pp) while simultaneously saving 6–50% tokens.

Model	Sc.	Native FC	TSCG Text	Δ	Savings
Claude S. 4	A	74.0%	**85.2%**	+11.2	50.1%
Claude S. 4	B	90.0%	**95.0%**	+5.0	50.1%
GPT-4o	A	55.5%	**56.5%**	+1.0	6.2%
GPT-4o	B	70.0%	**79.7%**	+9.7	6.2%
GPT-5.2	A	51.9%	**81.6%**	+29.7	11.4%
GPT-5.2	B	82.4%	**91.6%**	+9.2	11.4%
Mean				**+10.9 pp**	**20.6%**

et al., 2024) provides structured templates without formal optimization. Neither addresses tokenizer alignment or causal attention grounding.

Tool-calling benchmarks. BFCL (Patil et al., 2024) evaluates function-calling accuracy but always with full, uncompressed schemas. ToolBench (Qin et al., 2023) tests multi-step API planning across 16,000+ APIs without considering schema token overhead. The prompt compression survey (Li et al., 2024) evaluates no system on function-calling schemas. TAB fills this gap: the first benchmark measuring tool-schema compression effects on LLM tool-use performance.

2.1 Design-Space Comparison

Table 10 summarizes TSCG's positioning along six axes.

No existing system combines deterministic output, zero dependencies, theoretical grounding in attention mechanics, and specialization for tool schemas. TSCG's niche—structured prompt content in agentic systems—is underserved: the NAACL 2025 compression survey evaluates no system on function-calling schemas. On natural prose, LLMLingua-2 dominates (20× vs. TSCG's 1.07×); on prompt content optimization, DSPy achieves complementary gains. TSCG is the strongest system for a specific, growing problem—not a general-purpose replacement.

3 The TSCG Framework

TSCG comprises eight deterministic operators organized into three classes (Definitions 3.1–3.3), applied as a fixed-order pipeline of 10 transforms (Figure 1). Let $\mathrm{tok}(p)$ denote the BPE tokenization of prompt p, $\mathrm{sem}(p)$ the set of semantic atoms, and

Table 9: Four-class behavioral taxonomy from Scenario D, text-baseline experiments (E1, E4), and 30B benchmark (N1, 840 calls). Classes reflect distinct baseline capabilities and TSCG response patterns, enabling targeted deployment recommendations. Format-dominated classes show large JSON-baseline gains that vanish or reverse against text baselines.

Class	Models	JSON Δ	Text Δ	Recommendation
1: Format-dom.	Phi-4, Mistral 7B, Gemma 4B, Qwen3 4B	+17–90 pp	−7 to −23 pp	conservative; gain is format only
2: Compression	Claude, GPT-4o, GPT-5.2	+5–11 pp	+5–11 pp	balanced profile
3: Neutral	Llama 8B, Gemma 12B, Mistral-Sm. 24B	+6–9 pp	≈0 pp	conservative profile
4: Cons.-only	Qwen3 14B, Qwen2.5-Coder 32B	−7 pp	−5 to −13 pp	conservative only (+4.4 pp)

Table 10: Design-space comparison of representative prompt optimization systems.

System	Det.	0-Dep	Theory	Schema	Compr.	Speed
LLMLingua-2	✗	✗	✗	✗	20×	42 s
DSPy	✗	✗	✗	✗	—	min
TextGrad	✗	✗	✗	✗	—	min
Sel. Context	✗	✗	✗	✗	5×	ms
LangGPT	✓	✓	✗	✗	—	ms
TSCG	✓	✓	✓	✓	**3.5×**	**<1 ms**

$\mathrm{Attn}(i, j)$ the causal attention weight from position i to j ($= 0$ for $j > i$). Every transform satisfies semantic preservation: $\mathrm{sem}(\tau(p)) \supseteq \mathrm{sem}(p)$.

3.1 Eight Operators

TAS (Tokenizer-Aligned Syntax). Selects delimiter variants minimizing token count: $d^* = \arg\min_{d_i \in D} |\mathrm{tok}(d_i)|$. Example: `->` (2 tokens) → ↦ (1 token).

CFL (Constraint-First Layout). Repositions output constraints to position 0, exploiting the attention sink (Xiao et al., 2023): $\mathrm{CFL}(p) = c(p) \oplus (p \setminus c(p))$. Implemented as `[ANSWER:type]` at the prompt start.

CFO (Causal-Forward Ordering). Reorders multi-step operations into topological order: $o_i \prec o_j \implies \mathrm{pos}(o_i) < \mathrm{pos}(o_j)$, ensuring prerequisites are causally accessible.

SDM (Semantic Density Maximization). Removes filler tokens (104+ patterns: politeness markers, hedging, redundant connectives) to maximize $\mathcal{D}(p) = |\mathrm{sem}(p)|/|\mathrm{tok}(p)|$.

DRO (Delimiter-Role Optimization). Replaces verbose structural phrases with compact delimiters: "the following items" → enumeration markers, "X corresponds to Y" → "X→Y".

CCP (Causal Closure Principle). Appends a closure block recapitulating key atoms at position n, exploiting recency bias in autoregressive generation: $\mathrm{CCP}(p) = p \oplus \kappa(A(p))$.

CAS (Causal Access Score). Scores atoms by fragility $\mathcal{F}(a) = \alpha \cdot \mathrm{importance}(a) + (1 - \alpha) \cdot \mathrm{distance\_penalty}(a)$ and places high-fragility atoms at positions 0 and n (attention sink + recency).

SAD-F (Selective Anchor Duplication with Fragility). Duplicates top-k atoms by fragility/token ratio within budget B, reinforcing critical information in the closure block. Fragility scoring, budget allocation, and parameter sensitivity are detailed in Appendix E.

3.2 Operator Taxonomy

Definition 3.1 (Token-Reducing Operators)**.** $\mathcal{T}_R = \{SDM, DRO, TAS, CFL\}$*: each satisfies* $|\mathrm{tok}(T_i(S))| \leq |\mathrm{tok}(S)|$.

Definition 3.2 (Structure-Reordering Operators)**.** $\mathcal{T}_S = \{CAS, CFO\}$*: preserve token count but change position order.*

Definition 3.3 (Token-Expanding Operators)**.** $\mathcal{T}_E = \{SAD, CCP\}$*: add tokens (anchor duplications, closure blocks) within budget.*

Corollary 3.1 (Operator Interaction Effect)**.** *For models with <10B parameters,* $\mathrm{Acc}(\mathcal{T}_R(S)) > \mathrm{Acc}((\mathcal{T}_R \cup \mathcal{T}_S)(S))$*: adding structure-reordering operators to token-reducing operators* degrades *accuracy. Conservative profile ($\mathcal{T}_R$ only) is recommended below 10B.*

3.3 Compression Guarantee

Theorem 3.1 (Deterministic Compression Bound)**.** *For a well-formed JSON-Schema tool collection S with the TSCG pipeline* Π*:*

$$|\mathrm{tok}(\Pi(S))| \leq |\mathrm{tok}(S)| \cdot \Big(1 - \sum_{T_i \in \mathcal{T}_R} r_i \cdot f_i(S)\Big) \tag{1}$$

where r_i is the per-token reduction factor and $f_i(S)$ the fraction of tokens affected by operator T_i.

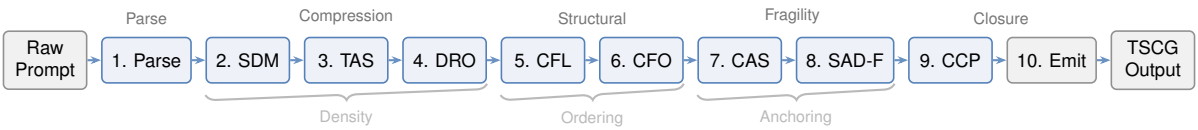

Figure 1: The TSCG 10-pass transform pipeline, executing left-to-right in five phases: *Parse* (segmentation), *Compression* (SDM, TAS, DRO), *Structural* (CFL, CFO), *Fragility* (CAS, SAD-F), and *Closure* (CCP, Emit). Each transform is a pure function; the composition $\Pi = \tau_{10} \circ \cdots \circ \tau_1$ is deterministic.

Input (JSON schema, ∼120 tokens)

```
{"name": "search_files",
 "description": "Search project
files
 by content or filename pattern",
 "parameters": {"type": "object",
 "properties": {"query": {"type":
 "string", "description": "The
search
 query string"}, "path": {"type":
 "string", "description": "Optional
 directory path to search in"}}}}
```

Output (TSCG balanced, ∼45 tokens, 62.5% reduction)

```
search_files(query:str path?:str)
|Search files by content or pattern
```

Figure 2: TSCG compression: a single tool schema before and after compilation. SDM removes filler; DRO compresses delimiters; TAS aligns to BPE boundaries.

Proof sketch. Each operator in $\mathcal{T}_R$ acts on disjoint token subsets (SDM: filler patterns; DRO: verbose delimiters; TAS: suboptimal tokenizer splits; CFL: positional overhead). Structure-reordering operators preserve count; setting SAD-F budget $B{=}0$ excludes expansion. Individual reductions compose additively. Full proof in Appendix E. □

Empirical validation. The bound predicts ≥51% savings; empirically we observe 61% (Scenario A), 66% (BFCL), and 75% (tool descriptions)—conservative by 10–24 pp due to the pessimistic disjoint-subset assumption.

3.4 Scoring Metric

All TAB results use a composite accuracy metric:

$$\text{Overall} = 0.6 \times \text{TSA} + 0.4 \times \text{Parameter\_F1} \quad (2)$$

where TSA is tool selection accuracy and Parameter_F1 measures parameter extraction quality.

3.5 Compiler Characterization

TSCG is a *compiler*, not a search-based optimizer. The pipeline $\Pi = \tau_{10} \circ \cdots \circ \tau_1$ applies 10 deterministic transforms in fixed order: same input always produces the same output. No model access is required. This distinguishes TSCG from DSPy (Khattab et al., 2023) (search-based, non-deterministic), TextGrad (Yüksekgonul et al., 2024) (gradient-based, requires model access), and LLMLingua (Pan et al., 2024) (requires GPU model inference, non-deterministic). TSCG executes in <1 ms on commodity hardware; LLMLingua-2 requires 42.5 s on the same prompts (∼40,000× slower).

4 Theoretical Foundations

We connect TSCG's operators to three foundational properties of causal autoregressive transformers.

4.1 Causal Attention and Information Flow

In a standard autoregressive transformer (Vaswani et al., 2017), attention at layer ℓ satisfies $\text{Attn}^{(\ell)}(i,j) = 0$ for $j > i$ (causal mask), creating asymmetric information flow: early tokens cannot access later ones.

Implication for CFO. If step o_2 depends on o_1 but o_1 appears *after* o_2, the model must rely on parametric knowledge rather than direct attention. CFO ensures $\text{pos}(o_1) < \text{pos}(o_2)$, guaranteeing prerequisites are causally accessible.

Implication for CAS. Total attention to position i follows a U-shaped distribution (Xiao et al., 2023): positions near 0 and n receive disproportionate attention, while middle positions form an "attention valley." CAS places high-fragility tools at positions 0 and n.

4.2 Attention Sink Exploitation

The attention sink phenomenon (Xiao et al., 2023)—position 0 receives $\text{Attn}(i,0) > 1/i$ for most i and layers—provides a natural amplification mechanism.

Implication for CFL. Placing the output constraint at position 0 ensures it receives elevated attention from every subsequent token. Combined with CCP/SAD-F at position n (recency bias), this creates a "bookend" strategy: critical information

occupies both extremes of the attention distribution.

4.3 BPE Non-Monotonicity

Theorem 4.1 (Tokenization Non-Monotonicity)**.** *For BPE tokenizer $\mathcal{T}$ and strings s_1, s_2 with $|s_1|_{chars} < |s_2|_{chars}$, it is not necessarily the case that $|\text{tok}(s_1)| < |\text{tok}(s_2)|$.*

TAS exploits this by selecting surface forms aligned with learned BPE merges, achieving token reduction without semantic change. Proof in Appendix E.

4.4 Attention Dilution and SDM

Proposition 4.1 (SDM Improves Effective Attention)**.** *Removing k filler tokens from a prompt of length n increases average effective attention per semantic atom by at least $n/(n-k)$.*

This follows from softmax normalization: filler tokens compete for attention weight with semantic tokens. SDM removes this competition, providing formal justification beyond token cost savings.

4.5 Fragility and Causal Accessibility

Definition 4.1 (Causal Accessibility)**.** $\mathcal{A}(a) = \frac{1}{L}\sum_{\ell=1}^{L} \text{Attn}^{(\ell)}(n, i)$*: average attention from the generation position to atom a at position i.*

Definition 4.2 (Fragility)**.** *An atom a is fragile when importance exceeds accessibility: $\mathcal{F}(a) \propto$ importance$(a) - \mathcal{A}(a)$.*

High-importance, low-accessibility atoms are the targets for CAS reordering and SAD-F duplication. Budget-constrained duplication is preferable to naive repetition because duplicating all atoms would increase length and dilute attention (Proposition 4.1).

5 Experiments

5.1 TAB: TSCG-Agentic-Bench

TAB is the first benchmark measuring tool-schema compression effects on LLM tool-use, comprising five scenarios across ∼13,000 API calls and 12 models (4B–32B local + 3 frontier; Table 11). Each scenario compares Natural (uncompressed JSON), TSCG, and TSCG+SAD across four task categories (single_tool, multi_tool, parameter_extraction, no_tool). Overall accuracy = $0.6 \times \text{TSA} + 0.4 \times \text{PF1}$; we weight tool selection higher because correct tool identification is prerequisite for parameter extraction (conclusions robust to alternative weightings 0.5–0.8 for TSA).

Table 11: TAB: five scenarios, ∼13,000 calls.

Sc.	Focus	Tools	Calls
A	Claude Code (16 real tools)	16	∼540
B	MCP Servers	43	∼2,700
C	Scaling (25–100 tools)	var.	∼1,080
D	Small models × 7 configs	var.	∼5,000
E	Multi-collection	57	∼540

Table 12: Scenario D: accuracy at 20 and 50 tools.

Model	20 Tools		50 Tools		Shift
	Nat.	TSCG	Nat.	TSCG	
Phi-4 14B	0.0	84.4	0.0	90.3	3→50
Mistral 7B	35.0	80.1	30.0	65.0	20→50
Gemma 3 4B	49.9	67.0	24.3	87.4	15→50
Gemma 3 12B	85.0	95.0	85.0	98.0	—
Llama 3.1 8B	78.4	81.0	75.1	86.3	—
Qwen3 4B	44.3	69.3	90.0	75.0	volatile
Qwen3 14B	90.2	84.1	94.6	89.6	—

Experimental conditions. All scenarios compare up to four conditions: *Natural (FC)*—tool schemas via native function-calling API (JSON, model-specific decoding); frontier models only. *Natural-Text*—full JSON schemas as plain text in the user prompt; the production-realistic baseline for local models. *TSCG*—schemas compressed by TSCG (balanced profile unless noted); always text-mode. *Naive Truncation*—signatures only: `name(param:type, param?:type)`, no descriptions; text-mode. Format effect = Natural-Text − Natural (FC); compression effect = TSCG − Natural-Text.

Scenario A (Claude Code). Claude Sonnet 4 achieves 74.0% natural (FC), 85.2% TSCG (50.1% savings, ARR 115%). Text-mode: TSCG 87.8% vs. 76.9% natural—+10.9 pp genuine compression. GPT-5.2 shows the largest frontier effect: +29.6 pp (CI [+15.9, +43.3], d=0.78).

Scenario D (Small-Model Unlock). Seven models (4B–14B) at 3–50 tools: JSON-baseline accuracy 0–49% at >15 tools; TSCG recovers to 65–90% (Table 12). Models exceeding the 65% threshold at 50 tools rise from 4/7 to 7/7 (Figure 3).

N1 extends to 30B: Mistral-Small 24B (−1.1 pp, Class 3); Qwen2.5-Coder 32B (−4.7 pp, Class 4)—Qwen sensitivity persists across sizes.

5.2 Format Translation vs. Compression (E4)

E4 decomposes TSCG's JSON-baseline gains into format gain (text−JSON) and compression gain

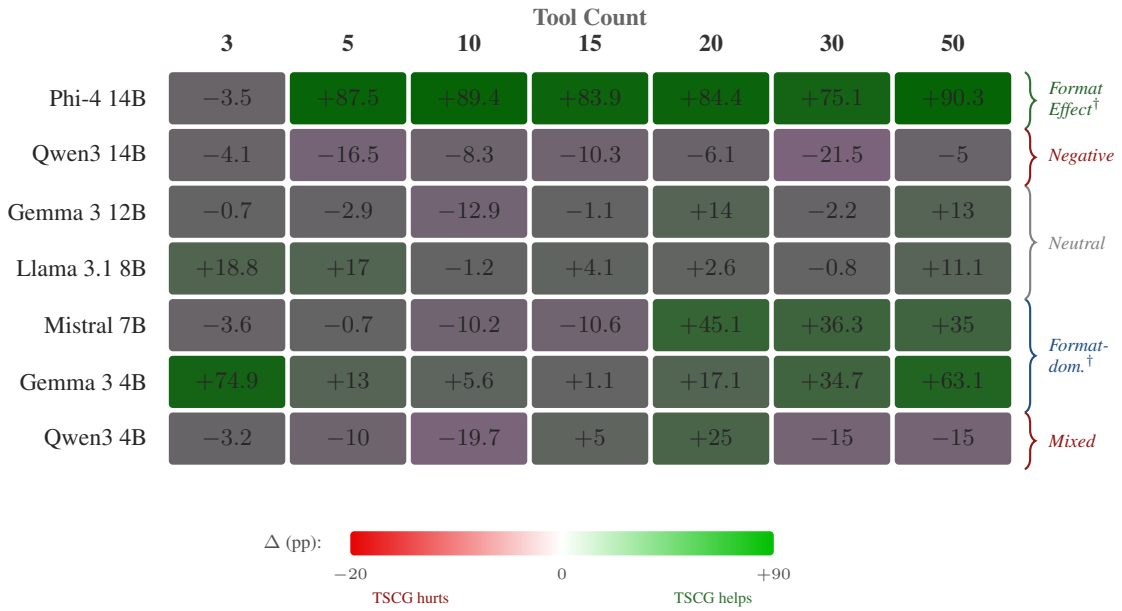

Figure 3: TSCG delta heatmap (Δ = TSCG accuracy − JSON-baseline accuracy, in percentage points) across seven models and seven tool-count conditions. Green cells indicate conditions where TSCG improves over JSON schemas; red cells indicate degradation. Five behavioral clusters emerge: *Format Effect*†—Phi-4's +75–90 pp gains are JSON→text translation, not compression benefit (§5.2); *Format-dominated*†—Mistral and Gemma 4B show large deltas driven primarily by format translation; *Neutral*—Llama 3.1 8B and Gemma 3 12B show modest mixed effects; *Negative*—Qwen3 14B shows uniform degradation (−4 to −22 pp); *Mixed*—Qwen3 4B exhibits chaotic response. †E1/E4 text-baseline experiments confirm format contribution (§5).

Table 13: E4: format vs. compression decomposition. No small model shows genuine compression.

Model	Format Δ	Compr. Δ	Class
Phi-4 14B	+92.0	−7.0	1: Format
Mistral 7B	+44.6	−7.4	1: Format
Gemma 3 4B	+47.3	−8.9	1: Format
Qwen3 4B	+16.7	−23.4	1: Format
Llama 3.1 8B	+5.6	+0.3	3: Neutral
Gemma 3 12B	+9.2	−0.9	3: Neutral

Table 14: Naive truncation vs. TSCG (text-mode, Claude Sonnet 4).

Sc.	Condition	Savings	TSA	Overall
A-16	Natural-Text	—	85.0	76.9
	TSCG	17%	95.0	87.8
	Naive trunc.	93%	95.0	87.0
C-50	Natural-Text	—	95.0	95.0
	TSCG	23%	100.0	100.0
	Naive trunc.	87%	100.0	98.5

(TSCG−text) across six small models (2,520 Ollama calls; Table 13).

TSCG's gains on small models arise from format translation (JSON→text), not structural compression. Since every production API transmits JSON, this translation *is* the needed intervention. Naive truncation outperforms TSCG in hypothetical text-mode (mean +10.6 pp), but no production framework offers text-mode tools.

5.3 R^2 Decomposition and Naive Truncation

The predictive regression (n=49, 7 models × 7 sizes) yields:

$$\Delta_{\text{TSCG}} \approx -0.93 \times \text{Acc}_{\text{JSON}} + 0.76, \quad R^2 = 0.88 \tag{3}$$

Against text baselines (E4): R^2 collapses to 0.03 (p=0.24)—a 97% drop confirming format sensitivity, not compression (Figure 4). LOO RMSE = 12.8 pp (detail in Appendix D).

At 16 tools, naive truncation matches TSCG (87.0% vs. 87.8%; Table 14). At 50 ambiguous tools, TSCG achieves 100% vs. 98.5%, with multi-tool sequencing driving the gap (87.5% vs. 75.0%).

5.4 External Validation

BFCL: 93.2% on Berkeley Function Calling Leaderboard schemas (vs. 85.7% natural; ARR 108%, 46.8% savings)—TSCG *improves* accuracy on third-party benchmarks. GSM8K-Under-Load: Claude Sonnet 4 reasoning preserved (∼81% across 0–50 tools; GPT-5.2 shows more degrada-

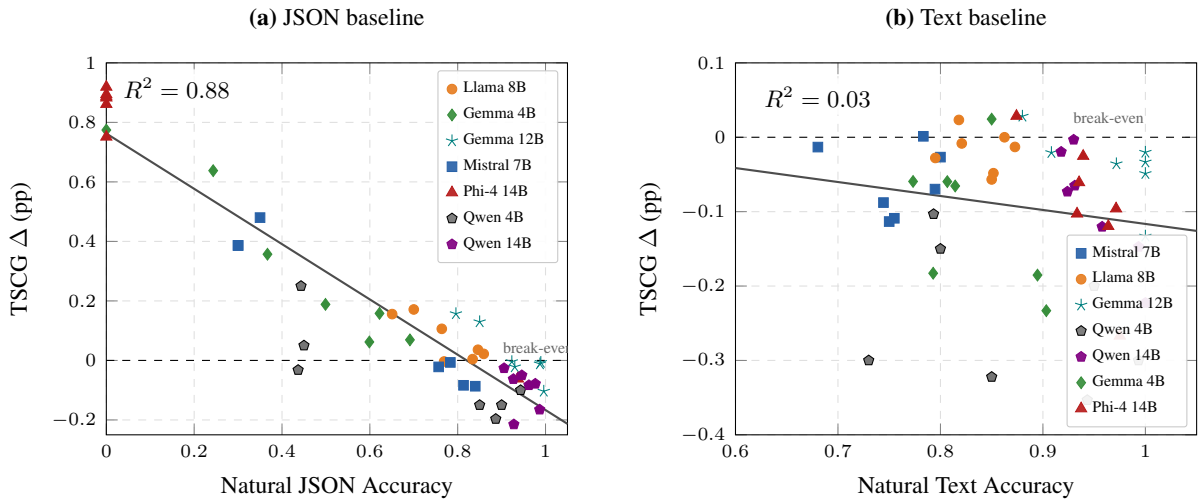

Figure 4: Format-confound decomposition of the predictive model. **(a)** Against JSON baselines (n=49, 7 models × 7 catalog sizes), natural accuracy strongly predicts TSCG improvement (R^2=0.88, slope ≈−0.93): models that fail at JSON parsing gain the most. **(b)** Against text baselines (n=49), the correlation vanishes (R^2=0.03, p=0.24): once the format confound is removed, baseline accuracy has no predictive power. The 97% drop in R^2 confirms that the JSON-baseline regression captures format sensitivity, not compression benefit.

tion); details in Appendix B. Conservative SDM on Qwen3-14B: +4.4 pp (balanced: −6.6 pp); ablation in Appendix C and Figure 5.

6 Discussion

6.1 Three Mechanisms of TSCG Improvement

TSCG's accuracy improvements arise from three distinct mechanisms. **M1: Format Translation** converts JSON schemas to structured text—the dominant mechanism for Class 1 models, where JSON parsing is the bottleneck (e.g., Phi-4: 0%→90%, entirely format-driven per E1/E4). **M2: Structural Reorganization** (CAS reordering, CFL constraint positioning, CFO causal ordering) improves accuracy beyond format change—the dominant mechanism for Class 2 frontier models (+10.9 pp on Claude Sonnet 4 in text-mode, +24.4 pp on GPT-5.2). **M3: Token Reduction** decreases attention dilution as a secondary effect; GPT-4o achieves positive accuracy improvement even with negative token savings on certain scenarios, proving M2 operates independently of M3.

The mechanisms contribute differently per model class: M1 dominates for small models (Class 1), M2 for frontier models (Class 2), and neither contributes meaningfully for neutral models (Class 3). For Qwen architectures (Class 4), balanced M2 transforms actively harm accuracy, but conservative filler removal preserves benefits.

6.2 Four-Class Taxonomy

E4 text-baseline experiments (2,520 calls, 6 small models), N3 conservative ablation (180 calls), and N1 30B benchmarks (840 calls, 2 models) yield a four-class taxonomy across 12 models (Table 15):

Class 1 models gain dramatically from JSON-baseline TSCG but show *negative* compression against text baselines—the benefit is format translation (JSON→text), confirmed by E4 decomposition. **Class 2** frontier models show genuine structural compression (+5–11 pp for Claude Sonnet 4; GPT-5.2 is scenario-dependent) that persists when format confound is eliminated. **Class 3** models are robust across all formats; TSCG neither helps nor harms. **Class 4** Qwen models degrade under balanced TSCG but improve +4.4 pp under conservative SDM (N3)—the degradation is architecture-specific, persisting from 14B to 32B (N1), implicating CAS/DRO structural transforms rather than capacity limitations.

Deployment guidance. Since every production API transmits JSON: (1) Class 1 models achieve 0–49% at >15 tools but 65–90% under TSCG—format translation is the needed intervention; (2) Class 2 benefits from balanced profiles; (3) Class 4 requires conservative mode exclusively; (4) Class 3 can use TSCG without risk.

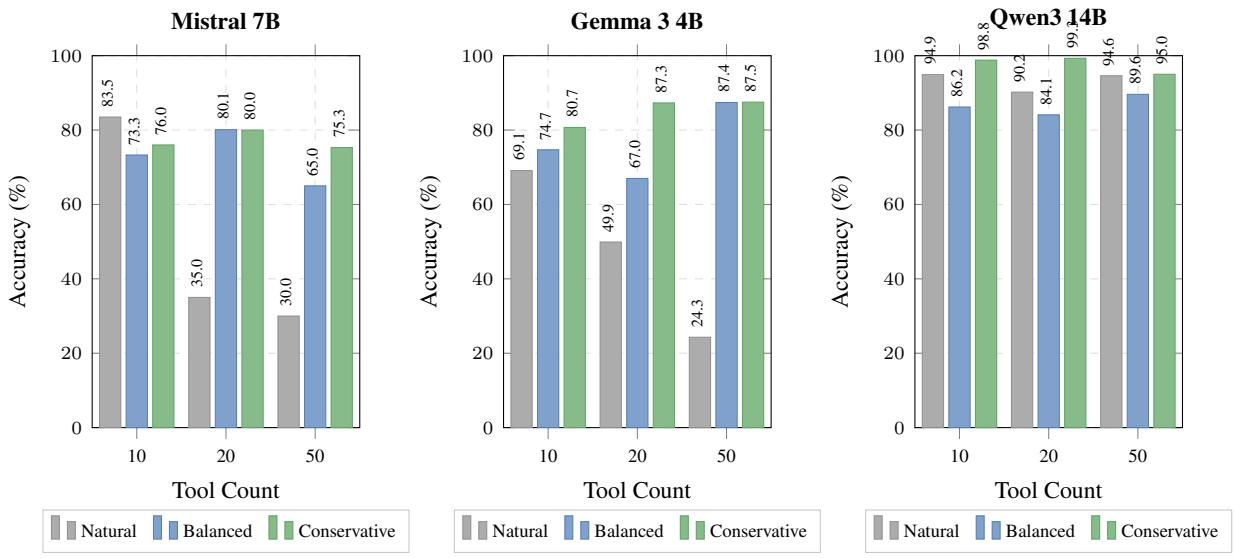

Figure 5: SDM mode ablation across model sizes. **Left:** Mistral 7B (¡10B). **Center:** Gemma 3 4B (¡10B). **Right:** Qwen3 14B (14B, N3 re-run with corrected conservative profile). For small models (left, center), Conservative SDM matches or outperforms Balanced in 5 out of 6 conditions. For Qwen3 14B, Balanced TSCG degrades accuracy by –6.6 pp mean, while Conservative mode *improves* accuracy (mean Δ: +4.4 pp), *surpassing* Natural at 10 and 20 tools. Conservative SDM is universally safe and often beneficial.

Table 15: Four-class taxonomy of TSCG effects (12 models, 4B–32B + frontier). Text Δ: compression gain (TSCG minus text baseline).

Class	Models	JSON Δ	Text Δ	Recommendation
1: Format-dom.	Phi-4, Mistral 7B, Gemma 4B, Qwen3 4B	+17 to +90 pp	−7 to −23 pp	Conservative
2: Compression	Claude, GPT-4o, GPT-5.2	+5 to +11 pp	+5 to +11 pp[†]	Balanced
3: Neutral	Llama 8B, Gemma 12B, Mistral-Sm. 24B	+5 to +9 pp	≈0 pp	Conservative
4: Cons.-only	Qwen3 14B, Qwen2.5-Coder 32B	−5 to −7 pp	−5 to −13 pp	Conservative only

[†]Range reflects Claude Sonnet 4 (+5–11 pp); GPT-5.2 compression is scenario-dependent (Sc. A: +3.3 pp, Sc. B: −6.6 pp).

6.3 Naive Truncation Honesty

At 16 well-named tools (Scenario A), naive truncation (retaining only `ToolName(param:type,...)`) matches TSCG: 87.0% vs. 87.8% overall. Claude Sonnet 4 recognizes tools by distinctive names, making descriptions dispensable. The gap emerges at scale: at 50 tools with ambiguous names (Scenario C-50), TSCG achieves 100% while naive truncation drops to 98.5%, concentrated in multi-tool sequencing (87.5% vs. 75.0%, −12.5 pp) and parameter extraction (100% vs. 92.5%). TSCG's structural reorganization preserves the semantic cues needed for disambiguation—parameter descriptions, usage constraints, return types—that naive truncation discards.

6.4 Infrastructure Impact

TSCG reduces per-call schema overhead from $O(n \cdot k)$ to $O(n \cdot k/c)$ where $c \approx 3.5\times$ for structured schemas. At production volumes (100k calls/day with 16 tools at $3/MTok), the savings exceed $30,000/month. Combined with sub-millisecond deterministic execution, zero dependencies, and a 34.7 KB bundle, TSCG deploys as transparent middleware in any agent framework—Claude Code, LangChain, CrewAI, or MCP servers—without GPU infrastructure or external API calls.

For small models (4B–14B), the infrastructure impact extends beyond cost: TSCG enables deployment of local models as functional tool-use agents in edge, cost-sensitive, and privacy-constrained environments where frontier APIs are unavailable or unacceptable.

7 Limitations

Benchmark scope. TAB is self-constructed; we mitigate this with BFCL external validation (ARR 108%), but independent third-party evaluation on

diverse tool catalogs is needed.

Task coverage. TAB measures tool selection accuracy and parameter extraction only—not generation quality, multi-turn coherence, or end-to-end task completion.

Small-model degradation. Class 1 models show negative compression gains against text baselines (-7 to -23 pp), meaning TSCG actively harms accuracy in hypothetical text-mode deployments with <10 tools.

Language. The filler pattern library and all evaluations are English-only. BPE tokenizations and filler patterns differ across languages; multilingual extension requires language-specific libraries.

Model coverage. 12 models tested across 4 architecture families. Generalization to untested architectures (e.g., Mamba, RWKV) is uncertain. Thinking models (o1, o3-mini, Gemini 2.5 Pro) are excluded—they produce empty responses on TSCG-compressed prompts.

Predictive model precision. LOO RMSE of 12.8 pp provides coarse triage only; Phi-4's zero-accuracy leverage points inflate R^2 (excluding Phi-4: R^2=0.81).

Serialization alternatives. We compare against JSON (production standard) and naive truncation (minimal baseline). Alternative text serializations (YAML, key-value lists) are not evaluated; the Natural-Text vs. TSCG comparison isolates compression beyond format.

AI assistance. AI language models assisted with code quality checks, statistical verification, and manuscript editing. All experimental design, data collection, analysis, and scientific conclusions are the sole work of the author.

8 Conclusion

We have presented Token-Context Semantic Grammar (TSCG), a deterministic tool-schema compiler that addresses the dominant token cost in agentic AI deployments.

JSON-API enablement. Every production framework transmits tools as JSON, yet small models (4B–14B) achieve only 0–49% accuracy at >15 JSON-format tools. TSCG recovers accuracy to 65–90% by implicitly translating JSON schemas to structured text—enabling small models as functional tool-use agents in edge, cost-sensitive, and privacy-constrained environments. E1/E4 (2,940 Ollama calls across 6 models) identifies format translation as the dominant mechanism for this class.

Frontier compression. For frontier models (Claude Sonnet 4, GPT-4o, GPT-5.2), TSCG provides genuine structural compression (+5–11 pp for Claude Sonnet 4; GPT-5.2 shows scenario-dependent variation), with 50–72% token savings that persist when the format confound is eliminated. At production volumes, this exceeds $30,000/month in savings.

Mechanistic decomposition. The first format-vs-compression analysis for tool-schema optimization yields a four-class taxonomy across 12 models (4B–32B + frontier), with actionable deployment guidance: conservative SDM for small and Qwen models, balanced for frontier. The predictive model (R^2=0.88, collapsing to 0.03 against text baselines) quantifies format sensitivity and enables deployment triage from a single baseline measurement.

TSCG is released as an open-source npm package—1,200 LOC TypeScript, zero dependencies, sub-millisecond execution, 34.7 KB bundle.[1] TAB, the first tool-schema compression benchmark (∼13,000 calls, 12 models, 5 scenarios), and all evaluation code are released as open-source infrastructure for the agent framework stack.

Acknowledgements

This work was conducted independently without institutional funding or affiliation. Portions of the experimental analysis, quality control, and manuscript preparation were assisted by AI language models (Claude, GPT-4), consistent with ACL's policy on AI writing assistance. All scientific claims, experimental design, data analysis, and final editorial decisions are solely the author's responsibility.

Data and Code Availability

The TSCG compiler, TAB benchmark suite (task definitions, tool schemas, evaluation harness), and all evaluation scripts are released as open-source software at: https://github.com/SKZL-AI/tscg. Raw evaluation logs are available on request.

[1] https://github.com/SKZL-AI/tscg

References

Anonymous. 2025. CompactPrompt: LLM-free heuristic prompt compression via n-gram abbreviation and quantization. *arXiv preprint*.

Md Adnan Arefeen, Biplob Debnath, and Srimat Chakradhar. 2024. LeanContext: Cost-efficient domain-specific question answering using LLMs. *arXiv preprint arXiv:2309.00841*.

Qingyan Chen, Yongqi Zhao, Bing Nan, Jinyuan Fan, Hung Bui, David Barber, Jia-Jie Gu, and Qi Liu. 2024. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. 2023. Adapting language models to compress contexts. *arXiv preprint arXiv:2305.14788*.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2024. PromptBreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*.

Junyi Guo, Minhwa Lee, and Elias Bareinboim. 2024. Connecting the dots: LLMs can infer and verbalize latent structure from disparate training data. *arXiv preprint arXiv:2406.14546*.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. LLMLingua: Compressing prompts for accelerated inference of large language models. *arXiv preprint arXiv:2310.05736*.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2024. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. *arXiv preprint arXiv:2310.06839*.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Mober, and 1 others. 2023. DSPy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*.

Yucheng Li, Bo Dong, Frank Guerin, and Chenghua Lin. 2023. Compressing context to enhance inference efficiency of large language models. *arXiv preprint arXiv:2310.06201*.

Zongqian Li and 1 others. 2024. A survey on prompt compression for large language models. *arXiv preprint*. NAACL 2025 survey; evaluates no system on function-calling schemas.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. 2023. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Ruhle, Yuqing Yang, Chin-Yew Lin, and 1 others. 2024. LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. *arXiv preprint arXiv:2403.12968*.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive APIs. *arXiv preprint arXiv:2305.15334*. Berkeley Function Calling Leaderboard (BFCL).

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. *arXiv preprint arXiv:2305.03495*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2023. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv preprint arXiv:2307.16789*.

Viktor Schlegel and 1 others. 2024. SAMMO: A general-purpose framework for prompt optimization. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. EMNLP.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30.

Xingchen Wan, Ruoxi Sun, Hootan Nakhost, Hanjun Dai, Dale Schuurmans, Zoltan Gyenes, and Ed H Chi. 2024. Universal self-adaptive prompting. *arXiv preprint arXiv:2305.14926*.

Ming Wang, Yuanzhong Peng, Yixin Cai, Yuzhi Chen, Wenjie Xiao, Yasheng Li, Binyuan Wang, and Jianfei Li. 2024. LangGPT: Rethinking structured reusable prompt design framework for LLMs from the programming language. *arXiv preprint arXiv:2402.16929*.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P Xing, and Zhiting Hu. 2023. PromptAgent: Strategic planning with language models enables expert-level prompt optimization. *arXiv preprint arXiv:2310.16427*.

Yang Wen, Xindi Liu, Yujia Qin, Yiheng Wu, Wanjun Chen, Zhiyuan Zhang, and Xiao-Shan Cai. 2024. On the multi-turn instruction following for conversational web agents. *arXiv preprint arXiv:2402.15057*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*.

Mert Yüksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. 2024. TextGrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*.

A Pre-TAB Evaluation

The pre-TAB evaluation comprised 19 core tasks across 7 categories (Factual n=4, Reasoning n=4, Classification n=2, Extraction n=1, OptFirst n=3, Complex n=2, NearDup n=3), 6 compression strategies, 11 independent runs on Claude Sonnet 4, and domain-specific benchmarks across 4 phases. This evaluation established TSCG's baseline effectiveness before the multi-model TAB benchmark. Full results are available in the supplementary materials.

B GSM8K-Under-Load

Tool-schema overhead does not degrade reasoning: Claude Sonnet 4 maintains ∼81% GSM8K accuracy across 0–50 irrelevant tool definitions (∼400 calls). TSCG provides a consistent ∼4.5 pp advantage over natural schemas, suggesting that reduced schema overhead frees attention capacity for mathematical reasoning. The effect is statistically significant (+4.5 pp, $p = 0.020$) but practically small (Cohen's $d = 0.11$).

C SDM Ablation Detail

Conservative SDM (filler removal only) is compared with Balanced (full structural compression) on Qwen3-14B (180 calls, N3 corrected re-run at sizes 10/20/50):

- Balanced TSCG degrades accuracy by −6.6 pp mean (90.2%→84.1% at 20 tools, 94.6%→89.6% at 50 tools).
- Conservative SDM *improves* accuracy by +4.4 pp mean (90.2%→99.3% at 20 tools, 94.6%→95.0% at 50 tools).
- CAS reordering and bracket elision—not SDM filler removal—trigger the degradation.

Practical recommendation: Conservative SDM for all models <10B and all Qwen architectures (including Qwen2.5-Coder 32B, confirmed by N1); Balanced for frontier models where genuine compression persists.

D Predictive Model and LOO Cross-Validation

Table 16 reports robustness checks on the predictive regression (Equation 3).

Table 16: R^2 robustness checks for the accuracy-gap regression.

Configuration	n	R^2	**95% CI**	p
7-model full set	49	0.885	[0.78, 0.93]	$< 10^{-20}$
7-model sans Phi-4	42	0.813	[0.58, 0.94]	$< 10^{-10}$
Model-level (7 means)	7	0.951	[0.62, 1.00]	< 0.001
Text baseline (E4)	49	0.028	—	0.24

Phi-4 14B is the strongest leverage point ($\Delta R^2 = -0.07$ when removed), but R^2 remains above 0.80 without it. The linear relationship holds at the model level ($R^2 = 0.95$). LOO cross-validation (holding out one model, refitting on remaining six) yields RMSE = 12.8 pp—adequate for deployment triage but insufficient for precise predictions. Qwen3 4B is the only model where the prediction direction is wrong (predicted positive, actual negative), reflecting its volatile behavior.

E Theorem Proofs

Proof of Theorem 3.1. Each token-reducing operator $T_i \in \mathcal{T}_R$ acts on a disjoint subset of tokens:

- **SDM**: filler patterns (104+ curated, non-overlapping with delimiters/tokenizer targets).
- **DRO**: verbose delimiter phrases (structural role markers).
- **TAS**: suboptimal tokenizer splits (multi-token → single-token Unicode).
- **CFL**: positional relocation of output constraint to position 0.

Disjoint action $\Longrightarrow$ additive composition. $\mathcal{T}_S$ preserves count; B=0 excludes expansion:

$$|\mathrm{tok}(\Pi(S))| \;\leq\; |\mathrm{tok}(S)| \cdot \Big(1 - \sum_{T_i \in \mathcal{T}_R} r_i \cdot f_i(S)\Big).$$

□

F Implementation and Deployment

TSCG is implemented as ∼1,200 lines of TypeScript with zero external dependencies. Three deployment modes: CLI (npm), Chrome Extension (Manifest V3), and Web Application (React + Vite). Production bundle: 34.7 KB (11.7 KB gzipped). Sub-millisecond execution for prompts under 4,096 tokens. Five optimization profiles (Table 17):

Table 17: TSCG optimization profiles.

Profile	Active Principles
`minimal`	SDM, CFL
`balanced`	SDM, DRO, CFL, CFO, TAS
`max_compress`	SDM, DRO, CFL, CFO, TAS
`max_accuracy`	SDM, CFL, DRO, TAS, CCP, SAD-F
`full`	All 8 principles

G Extended Prior-Art Comparison

H Benchmark Configurations and Statistical Detail

Table 19 documents configuration parameters for all benchmark runs.

H.1 Holm-Bonferroni Correction

107 pairwise McNemar tests, family-wise $\alpha = 0.05$. 9/107 achieve significance (Table 20), all from Scenario D with weak JSON baselines.

H.2 Bootstrap Confidence Intervals

System	Det.	Tok.	Caus.	B-Box	Bdgt.	Cmp.	NoTr.	0-Dep
Compression-Based								
LLMLingua (Jiang et al., 2023)	✗	✗	✗	✓	✓	✓	✗	✗
LLMLingua-2 (Pan et al., 2024)	✗	✗	✗	✓	✓	✓	✗	✗
Selective Context (Li et al., 2023)	✓	✗	✗	✓	✗	✓	✓	✗
LongLLMLingua (Jiang et al., 2024)	✗	✗	✗	✓	✓	✓	✗	✗
Gist Tokens (Mu et al., 2023)	✗	✗	✗	✗	✗	✓	✗	✗
AutoCompressors (Chevalier et al., 2023)	✗	✗	✗	✗	✗	✓	✗	✗
LeanContext (Arefeen et al., 2024)	✗	✗	✗	✓	✗	✓	✗	✗
Search-Based								
DSPy (Khattab et al., 2023)	✗	✗	✗	✓	✗	✗	✓	✗
OPRO (Yang et al., 2024)	✗	✗	✗	✓	✗	✗	✓	✗
APE (Zhou et al., 2023)	✗	✗	✗	✓	✗	✗	✓	✗
EvoPrompt (Chen et al., 2024)	✗	✗	✗	✓	✗	✗	✓	✗
PromptBreeder (Fernando et al., 2024)	✗	✗	✗	✓	✗	✗	✓	✗
COSP (Wan et al., 2024)	✗	✗	✗	✓	✗	✗	✓	✗
Gradient-Based								
TextGrad (Yüksekgonul et al., 2024)	✗	✗	✗	✗	✗	✗	✓	✗
ProTeGi (Pryzant et al., 2023)	✗	✗	✗	✗	✗	✗	✓	✗
PromptAgent (Wang et al., 2023)	✗	✗	✗	✓	✗	✗	✓	✗
Template / Theoretical								
LangGPT (Wang et al., 2024)	✓	✗	✗	✓	✗	✗	✓	✗
SSR++ (Wen et al., 2024)	✓	✗	✗	✓	✗	✗	✓	✗
CFPO (Guo et al., 2024)	✗	✗	✓	✓	✗	✗	✓	✗
TSCG (ours)	✓	✓	✓	✓	✓	✓	✓	✓

Table 18: Comparison of 18 prompt optimization systems and TSCG across eight desiderata. **Determ.**: deterministic output for the same input. **Tok-Aware**: optimizes with respect to BPE tokenizer boundaries. **Causal Th.**: grounded in causal attention theory. **Black-Box**: requires no access to model weights or gradients. **Budg. Anch.**: budget-constrained anchor duplication. **Compress.**: achieves measurable token compression. **No Train**: requires no model training or fine-tuning. **Zero-Dep**: zero external runtime dependencies. TSCG is the only system satisfying all eight criteria.

Table 19: Experimental configurations.

Parameter	Core/Domain	TAB	Ollama
Temperature	0	0	0
Seed	N/A	42	42
Max tokens	4096	1024	1024
Mode	Text + FC	Text-mode	Text-mode
SDM profile	full	balanced	balanced
API provider	Anthropic/OpenAI	Anthropic	Ollama (local)
Eval. method	Automated	TABEvaluator	TABEvaluator

Table 20: Significant comparisons (Holm-Bonferroni, α=0.05, 107 tests).

Model	Tools	Δpp	p_{adj}	d
Phi-4 14B	5/10/15	+95.0	0.003	6.11
Phi-4 14B	20	+84.4	0.003	4.21
Phi-4 14B	30	+80.0	0.011	2.81
Phi-4 14B	50	+90.3	0.003	6.11
Gemma 3 4B	3	+85.0	0.007	3.34
Gemma 3 4B	50	+63.1	0.004	1.68
Gemma 3 4B	50[c]	+65.0	0.004	1.73

Table 21: Bootstrap 95% CIs for key deltas (1,000 resamples, seed=42).

Scenario	Model	Δpp	95% CI	p
Sc. A (16)	Claude S. 4	+10.0	[0.6, 18.3]	0.200
Sc. A (16)	GPT-5.2	+29.7	[16.1, 42.2]	0.004
Sc. B (43)	Claude S. 4	+5.3	[−0.3, 11.3]	0.063
Sc. D (50)	Phi-4 14B	+90.3	—	<0.001
Sc. D (50)	Gemma 3 4B	+63.1	[45.0, 80.0]	<0.001
GSM8K (50)	Claude S. 4	+4.5	[0.5, 8.5]	0.020