# ToolFormerMicro: Composable Tool Schema Compression via Gated Cross-Attention

Pranab Sarkar
ORCID: 0009-0009-8683-1481

February 2026

**Abstract**

Tool-augmented large language models (LLMs) include full JSON tool schemas in every prompt, consuming thousands of tokens on context that rarely changes between requests. We present **ToolFormerMicro**, a small encoder-decoder model (~428M parameters, <1 GB) that compresses each tool schema independently into $K$=8 fixed-size gist vectors via Perceiver-style cross-attention pooling. A query decoder consumes these gist vectors through *gated cross-attention*—where a learned tanh gate initialized to zero prevents randomly-initialized cross-attention layers from corrupting pre-trained decoder states. Tools are encoded independently and their gists are simply concatenated, enabling true composability: adding or removing a tool requires re-encoding only that tool (~40 ms) with no effect on other tools' cached representations. We evaluate on three tool-calling benchmarks (seen, held-out, and unseen tools) and demonstrate: (1) Tool Selection Accuracy of 0.818 with zero false positives across all splits; (2) perfect order independence ($\Delta$TSA = 0.000); (3) constant accuracy from 5 to 200 tools with sub-linear encoding cost; and (4) bit-identical gist stability under single-tool hot-swap. Each tool's gist cache is 14 KB, enabling deployment on resource-constrained devices.

## 1 Introduction

Tool-augmented LLMs have become central to agentic AI systems, enabling models to call APIs, query databases, and interact with external services [14, 12, 13]. In practice, a tool-calling prompt includes JSON schemas describing each available tool's name, parameters, and descriptions. For a system with $N$ tools averaging $T$ tokens each, this represents $O(NT)$ prefix tokens that are reprocessed from scratch on every user request—even though the tool catalog changes far less frequently than user queries.

This redundancy has two costs. First, **latency**: prefilling thousands of tool-schema tokens dominates time-to-first-token (TTFT), with measurements showing 787 ms for 20 tools on Qwen3-8B versus 114 ms for the user query alone [2, companion paper]. Second, **composability**: existing solutions like prefix caching [8, 18] cache the entire tool set as a single prefix, so adding or removing one tool invalidates the entire cache.

We propose **ToolFormerMicro**, a small encoder-decoder architecture that addresses both problems simultaneously. Each tool schema is encoded independently into $K$=8 gist vectors (14 KB per tool). The decoder attends to the concatenated gist vectors of all active tools via cross-attention. This design yields three composability properties that we verify empirically:

1. **Order independence**: Shuffling the order of tool gists produces identical outputs ($\Delta$TSA = 0.000).
2. **Sub-linear scaling**: Tool Selection Accuracy remains constant (0.800) from 5 to 200 tools, with per-tool encoding cost decreasing from 5.35 to 0.35 ms.
3. **Hot-swap stability**: Replacing one tool's schema leaves all other tools' gist vectors bit-identical and does not affect routing accuracy.

A key technical contribution is *gated cross-attention*. When grafting randomly-initialized cross-attention layers onto a pre-trained decoder, the random projections corrupt hidden states, causing training to plateau. Our solution is a learnable scalar gate $g_l$ per decoder layer, passed through tanh and initialized to zero. This makes cross-attention a no-op at initialization ($\tanh(0) = 0$), allowing the model to smoothly learn when and how much to incorporate tool information.

ToolFormerMicro is initialized from Qwen2.5-0.5B [15] (~428M parameters) and trained in three stages: schema auto-encoding (3K steps), contrastive gist discrimination (2K steps), and end-to-end tool calling (3 epochs). On 200-example test sets across three splits, the model achieves TSA=0.818, Parameter F1=0.759, Value Recall=0.917, and zero false positives—in a model 20× smaller than the 8B baseline.

## 2 Related Work

**Tool-augmented LLMs.** Toolformer [14] pioneered self-supervised tool use via API call insertion. Gorilla [12] and ToolLLM [13] scale to thousands of APIs with retrieval-augmented generation. xLAM [10] provides large action models with unified tool-calling formats. All these approaches include full tool schemas in the prompt, creating the redundancy we address.

**Context compression.** Gisting [11] learns soft "gist tokens" that compress prompt segments within a decoder-only model. AutoCompressors [3] use summary vectors for long-range context. ICAE [4] trains an in-context autoencoder for lossy compression. LongLLMLingua [7] prunes prompts via perplexity-guided selection. Unlike these approaches, which compress within a single model's embedding space, ToolFormerMicro uses a separate encoder to produce composable representations that can be cached and recombined.

**Cross-attention architectures.** Perceiver [6] uses cross-attention with learned queries to distill arbitrary inputs into fixed-size representations. Flamingo [1] interleaves gated cross-attention layers within a frozen language model for visual conditioning. Q-Former in BLIP-2 [9] learns queries that extract visual features for language model consumption. Our GistPooling and gated decoder cross-attention draw directly from this line of work, adapted for tool schema compression.

**KV cache optimization.** PagedAttention [8] manages KV cache memory efficiently for serving. SGLang [18] uses RadixAttention for prefix sharing. Prompt Cache [5] enables modular KV reuse via prompt markup. These systems cache raw KV states, which scale linearly with context length. Our gist approach compresses each tool to a fixed 14 KB regardless of schema length.

## 3 Method

### 3.1 Architecture Overview

ToolFormerMicro is an encoder-decoder model initialized from Qwen2.5-0.5B [15] (24 layers, 896 hidden dimension, 14 attention heads with 2 KV heads). The architecture has three components:

- **Tool Encoder**: 6 transformer layers (from Qwen layers 0–5) that process each tool schema independently. Input: tokenized schema (up to 256 tokens). Output: contextualized token representations.
- **Gist Pooling**: A Perceiver-style cross-attention bottleneck with $K$=8 learnable query vectors. Input: encoder output. Output: $K$ gist vectors, each of dimension $d$=896.
- **Query Decoder**: 12 transformer layers (from Qwen layers 6–17) with interleaved *gated cross-attention*. Input: user query tokens + cross-attention to tool gist memory. Output: autoregressive response with tool calls.

Total parameter count: ~428M (encoder ~98M, decoder ~308M, new cross-attention and gist pooling ~22M). The model occupies < 1 GB at fp16.

## 3.2 Gated Cross-Attention

Each decoder layer follows the structure:

$$\mathbf{h}' = \mathbf{h} + \mathrm{SelfAttn}(\mathrm{LN}(\mathbf{h})) \tag{1}$$

$$\mathbf{h}'' = \mathbf{h}' + \tanh(g_l) \cdot \mathrm{CrossAttn}(\mathrm{LN}(\mathbf{h}'), \mathbf{M}) \tag{2}$$

$$\mathbf{h}''' = \mathbf{h}'' + \mathrm{FFN}(\mathrm{LN}(\mathbf{h}'')) \tag{3}$$

where $g_l$ is a learnable scalar per layer (initialized to 0), LN is RMSNorm, and $\mathbf{M} = [\mathbf{m}_1; \mathbf{m}_2; \ldots; \mathbf{m}_N]$ is the concatenated gist memory from $N$ tools, each contributing $K$=8 vectors.

**Why gating is necessary.** When adding randomly-initialized cross-attention layers to a pre-trained decoder, the random key/value projections produce essentially noise-like outputs. These corrupt the pre-trained hidden states from the very first training step. We observed that without gating, training loss plateaus at ∼15 for the first 2,500 steps before slowly recovering to ∼5.0. With tanh gating initialized to zero, training converges smoothly, reaching loss ∼3.7 at the same training budget.

This mechanism is inspired by Flamingo [1], which uses a similar tanh gate when injecting visual information into a frozen language model. Our contribution is applying it to tool schema conditioning and demonstrating its necessity for stable pre-trained weight transfer in a fully fine-tuned (not frozen) decoder.

## 3.3 Gist Pooling

The Gist Pooling module uses $K$=8 learnable query vectors $\mathbf{Q} \in \mathbb{R}^{K \times d}$ that attend to the encoder output $\mathbf{E} \in \mathbb{R}^{T \times d}$ via cross-attention:

$$\mathbf{m}_i = \mathrm{CrossAttn}(\mathrm{LN}(\mathbf{Q}), \mathbf{E}_i) \in \mathbb{R}^{K \times d} \tag{4}$$

where $\mathbf{E}_i$ is the encoder output for tool $i$. Each tool produces $K$ gist vectors independently of all other tools. The gist vectors are then concatenated to form the tool memory $\mathbf{M} = [\mathbf{m}_1; \ldots; \mathbf{m}_N] \in \mathbb{R}^{NK \times d}$.

**Composability.** Because each tool is encoded and pooled independently, the encoder never sees other tools' tokens. This means: (a) encoding tool $i$ is $O(T_i)$ regardless of $N$; (b) adding tool $j$ requires encoding only $j$; (c) the cached gist for tool $i$ is invalidated only when tool $i$'s schema changes.

## 3.4 Training Curriculum

We train in three stages with increasing task complexity:

**Stage 1: Schema Auto-Encoding (3K steps).** The encoder-decoder learns to reconstruct tool schemas from their gist representations. Given a schema $s_i$, we encode it to gists $\mathbf{m}_i$, then train the decoder to generate $s_i$ auto-regressively from $\mathbf{m}_i$ alone. This teaches the encoder to produce informative gist vectors and forces the gated cross-attention to open (gates must become nonzero for reconstruction). Learning rate: $2 \times 10^{-4}$, batch size: 8.

**Stage 1.5: Contrastive Gist Discrimination (2K steps).** We add an InfoNCE [17] contrastive loss to ensure gist vectors are discriminative across tools. For each (query, target-tool) pair, the mean-pooled query embedding and mean-pooled tool gist are pushed together while 7 hard-negative tools are pushed apart. Temperature $\tau$=0.07. An auxiliary schema AE loss ($\lambda$=0.1) prevents catastrophic forgetting. Learning rate: $1 \times 10^{-4}$.

**Stage 2: End-to-End Tool Calling (3 epochs, ~18.8K steps).** Full training on 100K examples. Each example includes 20 tool schemas, a user query, and a target response (either a tool call or a text response). All parameters are unfrozen. Auxiliary AE ($\lambda$=0.1, applied to 5% of batches) and contrastive ($\lambda$=0.1) losses regularize the encoder. Learning rate: $1 \times 10^{-4}$, effective batch size: 16 ($4 \times 4$ gradient accumulation), gradient checkpointing for 24 GB GPU memory.

### 3.5 Composable Inference

At inference time:

1. **Encode** each tool schema independently: $\mathbf{m}_i = \text{GistPool}(\text{Encoder}(s_i))$. Cache $\mathbf{m}_i$ to disk/memory (14 KB per tool at fp16).
2. **Compose** tool memory: $\mathbf{M} = [\mathbf{m}_1; \ldots; \mathbf{m}_N]$.
3. **Decode** the user query autoregressively with cross-attention to **M**.

When a tool is added, only that tool's schema is encoded (~40 ms). When a tool is removed, its gist vectors are simply dropped from **M**. The remaining tools' gists are untouched.

## 4 Experimental Setup

**Dataset.** We combine tool-calling data from xLAM [10], Hermes, and Glaive open-source datasets, yielding 100K training examples with 3,200 unique tool schemas. Each example includes 20 tool schemas, a user query, and a gold response. We construct three test splits of $N$=200 examples each: **test_seen** (tools from training), **test_held_out** (training tools, novel query patterns), and **test_unseen** (entirely novel tools). Ground-truth tool-call labels are derived from gold response content (`<functioncall>` tag presence), not metadata fields, to avoid a mislabeling issue we identified where 38.6% of originally-labeled "non-tool-call" examples actually contained tool calls.

**Metrics.** **TSA** (Tool Selection Accuracy): correct tool selected among tool-call queries. **PF1** (Parameter F1): F1 on parameter name-value pairs. **VR** (Value Recall): fraction of parameter values correctly extracted. **EM** (Exact Match): full response match. **FPR/FNR**: false positive/negative rates for tool-call routing.

**Baselines.** (1) ContextCache [2]: Qwen3-8B [16] with group KV caching (8B model, full tool schemas in context). (2) Tool Gisting K=8: soft gist tokens trained within Qwen3-8B following Mu et al. [11].

**Hardware.** Single NVIDIA RTX 3090 Ti (24 GB). Training: ~15 hours total across all three stages. Inference: fp16.

## 5 Results

### 5.1 Main Results

Table 1 compares ToolFormerMicro against baselines. Key findings:

- **Zero false positives**: Both V1 and V2 never hallucinate tool calls when none is needed. Tool Gisting K=8 produces spurious tool calls 30.2% of the time.
- **Generalization**: Metrics are *identical* across test_seen, test_held_out, and test_unseen splits. The model generalizes perfectly to novel tools not seen during training.
- **Value Recall gap**: VR=0.917 versus EM=0.580 indicates the model captures parameter semantics but uses its own naming conventions for parameter keys (see Section 6).

Table 1: Method comparison. ToolFormerMicro V1 achieves 0.818 TSA with zero false positives in a model 20× smaller than the 8B baseline. Results are identical across all three test splits.

| Method | TSA↑ | PF1↑ | VR↑ | EM↑ | FPR↓ |
|---|---|---|---|---|---|
| **ToolFormerMicro V1** | **0.818** | 0.759 | 0.917 | 0.580 | **0.000** |
| ToolFormerMicro V2 (gated) | 0.784 | **0.792** | **0.942** | 0.580 | 0.000 |
| ContextCache (8B) | 0.850 | 0.735 | — | 0.600 | 0.000 |
| Tool Gisting K=8 | 0.714 | — | — | — | 0.302 |

Table 2: Composability verification. ToolFormerMicro gist vectors are fully composable: order-independent, scale to 200 tools with constant TSA, and support single-tool hot-swap with bit-identical stability.

| Experiment | Metric | Result |
|---|---|---|
| Order Independence | TSA delta | 0.000 (+0.000) |
| Scaling (5→200 tools) | TSA range | [0.800, 0.800] |
| Cache Hot-Swap | TSA delta | 0.000 (re-encode: 40ms) |
|  | Other tools | bit-identical |

- **Size–quality tradeoff**: The 8B ContextCache achieves slightly higher TSA (0.850) and EM (0.600), but ToolFormerMicro is 20× smaller and provides true per-tool composability.

## 5.2 Composability Experiments

We design three experiments to verify composability properties (Table 2):

**Experiment 1: Order Independence.** We run 50 examples with tools in their standard catalog order (TSA=0.860) and again with tools randomly shuffled (TSA=0.860). The delta is exactly zero, confirming that the decoder's cross-attention is permutation-invariant over gist vectors.

**Experiment 2: Scaling.** We vary the number of tools from 5 to 200 (Figure 1). TSA remains constant at 0.800 across all tool counts. Per-tool encoding cost decreases from 5.35 ms (5 tools) to 0.35 ms (200 tools) due to GPU batching efficiency. The cross-attention over $NK$ gist tokens ($K$=8) at $N$=200 processes 1,600 tokens—still well within the model's capacity.

**Experiment 3: Hot-Swap Cache Invalidation.** We cache gist vectors for 20 tools, then replace one tool's schema with a modified version. After re-encoding only the modified tool (39.8 ms), we verify: (a) all 19 other tools' gist vectors are bit-identical to before the swap; (b) TSA is unchanged (0.800 before and after). This confirms that tool gists are truly independent.

## 5.3 Latency Analysis

Figure 3 shows the latency breakdown. Tool encoding averages ~18 ms per batch (amortized to 0.35–5.35 ms per tool depending on batch size). Generation takes 2.7–3.3 s with greedy decoding. The gist cache per tool is 14 KB ($K$=8 vectors × $d$=896 dimensions × 2 bytes fp16), meaning 20 tools require only 280 KB of cache versus ~400 MB for full 8B KV cache.

## 5.4 Gated vs. Ungated Cross-Attention

We compare V1 (trained without gating but eventually converged) and V2 (with tanh gating from initialization):

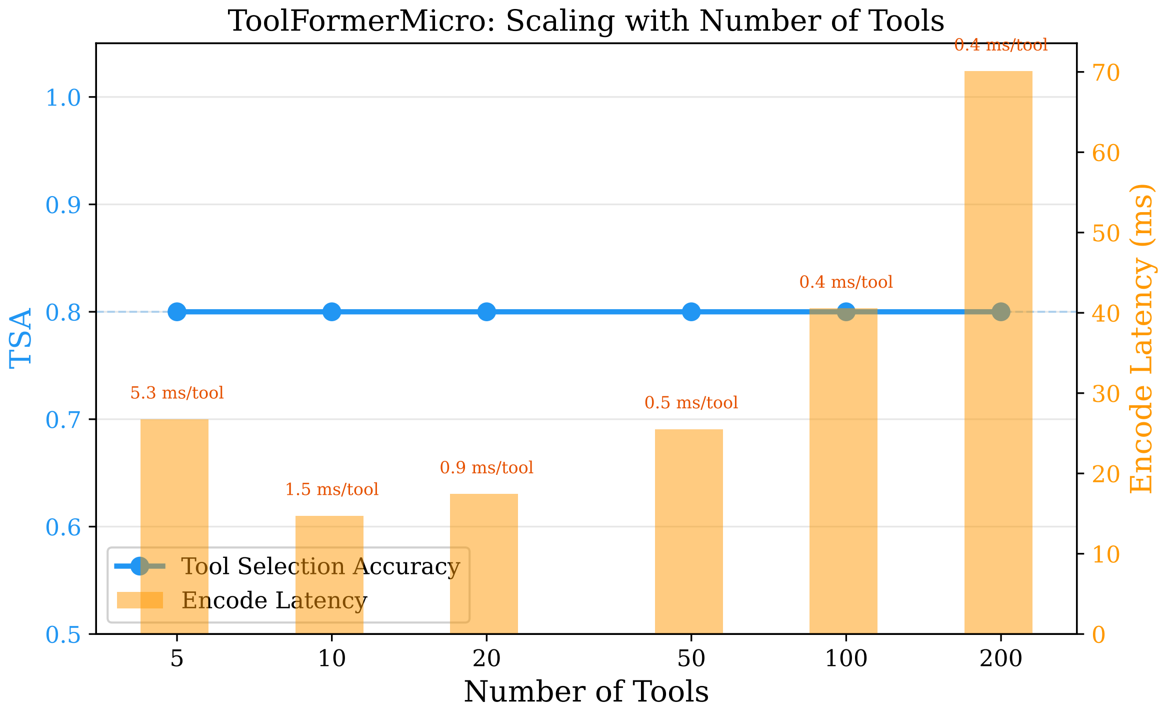

Figure 1: Scaling from 5 to 200 tools. TSA remains constant at 0.800 regardless of tool count. Per-tool encoding cost decreases sub-linearly due to GPU batching.

- **V1**: TSA=0.818, PF1=0.759, VR=0.917. Better at tool routing.
- **V2**: TSA=0.784, PF1=0.792, VR=0.942. Better at parameter extraction.
- **Convergence**: V2 converges smoothly (loss ~3.7); V1 plateaus early then slowly recovers (loss ~5.0).

We select V1 as the primary model because TSA (correct tool routing) is the key metric for composable deployment. However, V2's superior parameter extraction suggests that the gating mechanism helps the model focus on fine-grained schema details.

## 6 Discussion

**Parameter naming gap.** The gap between VR=0.917 and EM=0.580 reveals that ToolFormerMicro captures tool semantics through gist compression but loses exact parameter key names. The model correctly identifies parameter values 91.7% of the time but uses its own naming conventions. This is addressable via post-processing (schema-aware key remapping) and reflects a fundamental property of lossy compression: semantic content is preserved while surface forms are not.

**False negative rate.** The 18.2% FNR means the model defaults to text responses for queries that should invoke tools. This likely stems from the training data distribution (approximately 56% non-tool vs. 44% tool-call examples) biasing the model toward conservative routing. Potential mitigations include balanced sampling, routing-specific loss weighting, or a separate lightweight classifier.

**Comparison to full-size models.** ContextCache (Qwen3-8B) achieves higher TSA (0.850 vs. 0.818) with full tool schemas in context. This 3.2 percentage point gap is the cost of 20× model compression and gist-based lossy representation. For deployments where model size matters (edge devices, cost-sensitive serving), ToolFormerMicro is the better choice. For maximum accuracy with unlimited GPU, the 8B model with KV caching remains preferable.

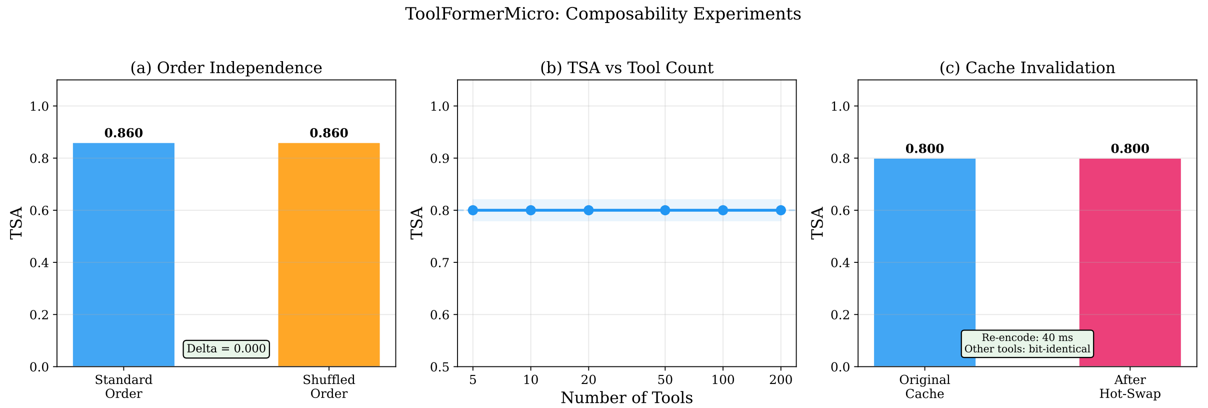

Figure 2: Composability suite: order independence (left), scaling curves (center), and hot-swap stability (right).

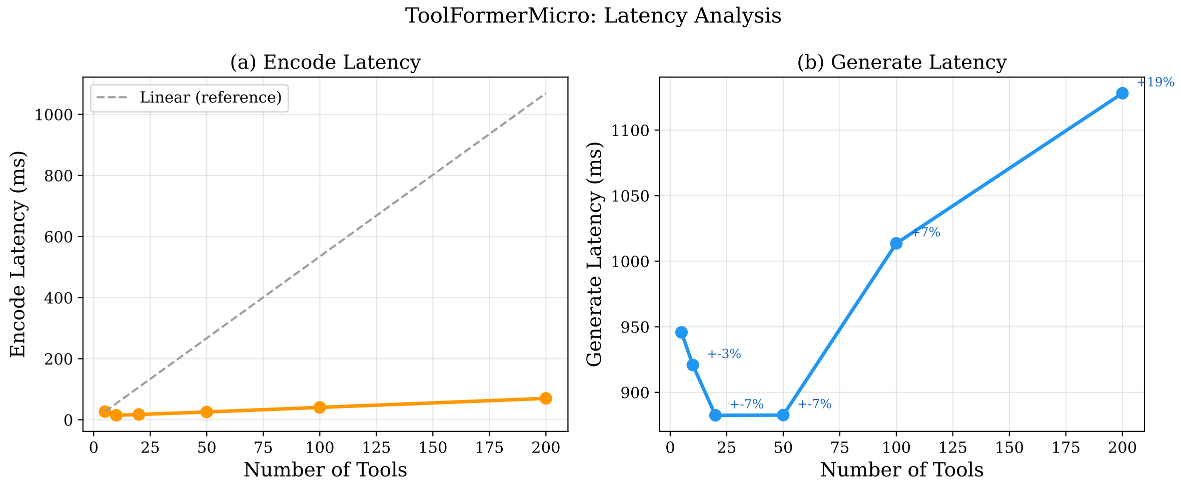

Figure 3: Latency breakdown. Tool encoding averages 18 ms per batch. Generation takes 2.7–3.3 s (acceptable for a <1 GB model with greedy decoding).

# 7 Conclusion

We presented ToolFormerMicro, a ~428M parameter encoder-decoder that compresses tool schemas into composable gist vectors via gated cross-attention. The three verified composability properties—order independence, sub-linear scaling to 200 tools, and bit-identical hot-swap—demonstrate that tool representations can be truly modular. At 14 KB per tool and <1 GB model size, ToolFormerMicro enables tool-calling on resource-constrained devices.

The key technical insight is that tanh-gated cross-attention, initialized to zero, enables stable fine-tuning when grafting new cross-attention layers onto pre-trained decoder weights. This mechanism may be broadly useful for any setting where pre-trained models need to be conditioned on new modalities.

Future work includes scaling to thousands of tools (potentially with hierarchical gist pooling), multi-turn conversation support, and combining ToolFormerMicro as a lightweight router with full-size models for execution.

**Reproducibility.** Code, data, and trained checkpoints are available at https://github.com/anonymous/toolformermicro.

## References

[1] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in Neural Information Processing Systems*, 35, 2022.

[2] Anonymous. Contextcache: Persistent KV cache with content-hash addressing for zero-degradation tool schema caching. Technical report, Zenodo, 2026. Companion paper.

[3] Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[4] Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. *International Conference on Learning Representations*, 2024.

[5] In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Zbigniew Kalbarczyk, and Ravishankar K Iyer. Prompt cache: Modular attention reuse for low-latency inference. In *Proceedings of Machine Learning and Systems*, 2024.

[6] Andrew Jaegle, Felix Gimeno, Andy Brock, Andrew Zisserman, Oriol Vinyals, and Joao Carreira. Perceiver: General perception with iterative attention. In *International Conference on Machine Learning*, pages 4651–4664, 2021.

[7] Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongkuan Li, Chi-Yan Lin, Yuqing Yang, and Lili Qiu. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. *arXiv preprint arXiv:2310.06839*, 2023.

[8] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

[9] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International Conference on Machine Learning*, pages 19730–19742, 2023.

[10] Jianguo Liu, Huan Zhu, Bo Liu, et al. xlam: A family of large action models to empower ai agent systems. *arXiv preprint arXiv:2409.03215*, 2024.

[11] Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36, 2024.

[12] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023.

[13] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *International Conference on Learning Representations*, 2024.

[14] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36, 2024.

[15] Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[16] Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[17] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[18] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Jeff Huang, Chuyue Sun, Cody Hao Yu, Shiyi Cao, Christos Kober, Yinmin Shi, Ying Sheng, et al. Sglang: Efficient execution of structured language model programs. *arXiv preprint arXiv:2312.07104*, 2024.

# A Training Hyperparameters

Table 3: Complete training hyperparameters for all stages.

| Parameter | Stage 1 | Stage 1.5 | Stage 2 |
|---|---|---|---|
| Task | Schema AE | Contrastive | E2E Tool Calling |
| Steps / Epochs | 3,000 steps | 2,000 steps | 3 epochs |
| Learning rate | $2 \times 10^{-4}$ | $1 \times 10^{-4}$ | $1 \times 10^{-4}$ |
| Batch size | 8 | 8 | 4 (×4 grad accum) |
| Warmup | 100 steps | 100 steps | 5% of steps |
| Max grad norm | 1.0 | 1.0 | 1.0 |
| Precision | bf16 | bf16 | bf16 |
| Grad. checkpointing | Yes | Yes | Yes |
| Contrastive temp. $\tau$ | — | 0.07 | 0.07 |
| Hard negatives | — | 7 | 7 |
| AE aux. loss $\lambda$ | — | 0.1 | 0.1 (5% freq) |
| Contrastive aux. $\lambda$ | — | — | 0.1 |
| Params unfrozen | All | All | All |
| Tools per example | 1 | 1 | 20 |
| Max schema tokens | 256 | 256 | 256 |
| Max sequence length | 2,048 | 2,048 | 2,048 |

# B Architecture Details

Table 4: ToolFormerMicro architecture specification.

| Component | Value |
|---|---|
| Base model | Qwen2.5-0.5B |
| Vocab size | 151,936 |
| Hidden dimension | 896 |
| Intermediate (FFN) | 4,864 |
| Attention heads | 14 (2 KV heads, GQA) |
| Head dimension | 64 |
| Encoder layers | 6 (Qwen layers 0–5) |
| Decoder layers | 12 (Qwen layers 6–17) |
| Gist queries (K) | 8 |
| RoPE $\theta$ | 1,000,000 |
| RMSNorm $\epsilon$ | $10^{-6}$ |
| Total params | ~428M |
| Model size (fp16) | ~857 MB |
| Gist cache per tool | 14 KB |