Description-Free Tool Planning for Fixed Catalogs via QLoRA Fine-Tuning

Anonymous ACL submission

Abstract

Tool-using language agents commonly include full tool schemas in every prompt, even when the available tools are fixed across many queries. This repeated schema context increases input length and can make small models unreliable planners. We study whether small language models can internalize a fixed tool catalog through supervised parameter-efficient fine-tuning, enabling structured tool planning without explicit tool descriptions at inference time.

Using AssetOpsBench, an industrial asset-operations benchmark with MCP-style tools, we fine-tune Gemma 4 E4B and Qwen3-4B with 8-bit QLoRA on approximately 1,700 tool-use examples. Under description-free inference, the fine-tuned models outperform an unfine-tuned schema-informed baseline that receives the full tool catalog, while reducing prompt length by 94.7%. The best Gemma run achieves an AT-F1 of 0.65 and an overall LLM-judge score of 3.88, compared with 0.47 and 2.88 for the informed baseline.

These results suggest that, for fixed tool catalogs, supervised QLoRA fine-tuning can shift tool knowledge from prompt context into adapter weights, reducing repeated schema overhead while maintaining or improving tool-planning quality.

1 Introduction

Large language models (LLMs) are increasingly used as planning components in tool-using agents, where they decompose user requests, select tools, construct arguments, and coordinate multi-step workflows. In many current tool-use pipelines, including MCP-style systems (Anthropic, 2026), the model receives a serialized tool catalog in the prompt so that it can infer which tools exist and how they should be called. This schema-informed approach is flexible: new tools can be added by updating the prompt. However, it also repeats the same catalog across queries, creating substantial input overhead.

This paper studies a narrower but practically important regime: *fixed-catalog tool planning*. In many enterprise and industrial deployments, the available tool catalog changes slowly relative to the number of user queries. In such settings, repeatedly prompting with full tool descriptions may be unnecessary. We ask whether this stable catalog knowledge can instead be amortized into a small model through supervised fine-tuning, allowing the model to generate structured tool-use plans without receiving tool descriptions at inference time.

This setting targets two related inefficiencies. First, small language models often require extensive tool-context prompting to produce valid plans. Second, repeated schema prompting can dominate the input budget. In our AssetOpsBench setting (Patel et al., 2025), the serialized tool catalog accounts for approximately 2,200 of the 2,400 prompt tokens, while the user query itself is short. Description-free inference removes this catalog section entirely, reducing the prompt to approximately 128 tokens and asking the fine-tuned model to recover server routing, tool selection, argument keys, and dependency structure from its learned parameters.

1.1 Contributions

We propose and evaluate *description-free* tool planning: a fixed-catalog setting in which the model must generate structured tool-use plans without receiving tool descriptions at inference time. Our contributions are:

1. We formalize *static-catalog description-free tool planning*, where a model must select MCP servers, tools, arguments, and dependencies for a fixed catalog while omitting the serialized tool catalog from the inference prompt.

2. We develop a catalog-grounded supervised

fine-tuning protocol in which a teacher model has access to full tool schemas during data construction, while the student model is trained and evaluated with the tool catalog removed.

3. We show that 8-bit QLoRA fine-tuning on approximately 1,700 examples enables ∼4B parameter models to outperform an unfine-tuned schema-informed baseline on AssetOpsBench, despite receiving no tool descriptions at inference time.

4. We evaluate whether the resulting specialization preserves broader model capability, showing that LoRA rank induces a quality–retention trade-off between tool-planning performance and general multiple-choice accuracy.

2 Related Work

2.1 Tool-Use and Function-Calling Evaluation

Tool-use evaluation has evolved from measuring whether models can call individual APIs to testing tool selection, argument construction, dependency tracking, and multi-step execution. API-Bank (Li et al., 2023) introduced a runnable benchmark for tool-augmented LLMs with 73 API tools. BFCL (Patil et al., 2024a) evaluates function calling across serial and parallel invocations using AST-based evaluation. MCP-Bench (Wang et al., 2025b) extends this to MCP-style agents, evaluating multi-step tasks that require cross-tool coordination across live MCP servers. Our work does not propose a new benchmark; instead, we study a stricter inference condition within a fixed-domain benchmark: the model must plan without receiving tool descriptions at inference time.

2.2 Schema-Informed and Retrieval-Augmented Tool Learning

A dominant approach to tool use is to expose tool information at inference time through prompt schemas or retrieved documentation. Gorilla (Patil et al., 2024b) fine-tunes models for API invocation with retrieval-aware training. ToolLLM (Qin et al., 2024) constructs ToolBench over 16,464 real-world APIs with a neural API retriever. ToolACE (Liu et al., 2024) shows that high-quality synthetic function-calling data with automatic verification can train strong small models. These approaches are well suited to broad or changing API ecosystems. Our setting makes the opposite trade-off: we assume a fixed, mature tool catalog and ask whether its schema information can be amortized into model parameters.

2.3 Tool and Skill Internalization

Recent work has begun to move tool or skill knowledge from prompt context into model parameters. ToolGen (Wang et al., 2025a) represents each tool as a unique vocabulary token, integrating tool knowledge into the language model's generation process and enabling unified tool retrieval and calling across over 47,000 tools. SKILL0 (Lu et al., 2026) studies skill internalization for embodied agents by providing skill context during RL training and progressively withdrawing it until the agent operates without runtime skill retrieval. Our work is motivated by the same observation, prompt-time tool context is expensive, but uses a different approach: ordinary supervised fine-tuning with QLoRA on a fixed industrial catalog, without special tool tokens, vocabulary expansion, or reinforcement learning.

2.4 Prompt Compression and Schema Removal

Prompt-compression methods such as LLMLingua (Jiang et al., 2023) reduce inference cost by removing less important prompt tokens. Description-free tool planning can be viewed as an extreme fixed-catalog variant of prompt compression: rather than compressing the tool catalog, we remove the entire serialized schema from the inference prompt. The missing catalog information must therefore be recovered from the fine-tuned model weights rather than from a shorter prompt. Unlike standard knowledge distillation (Hsieh et al., 2023), we do not require teacher model inference at test time.

2.5 Parameter-Efficient Fine-Tuning and Retention

LoRA (Hu et al., 2022) freezes the pretrained model and trains low-rank adapter updates, while QLoRA (Dettmers et al., 2023) further reduces memory by quantizing the frozen base model. Although adapter-based fine-tuning generally reduces catastrophic forgetting compared with full fine-tuning (Han et al., 2025), adapter updates can still bias the effective model toward the fine-tuning distribution and degrade prior capabilities. This is especially relevant for tool planning, where the

training distribution consists of structured plans rather than general natural language. We therefore evaluate both tool-planning quality and general multiple-choice retention after fine-tuning.

2.6 Industrial Tool-Planning Benchmarks

We use AssetOpsBench (Patel et al., 2025) because it provides a bounded but realistic industrial tool-planning environment with domain-specific MCP servers, simulated telemetry, human-authored scenarios, and multi-step workflows. This differs from broad benchmarks such as ToolBench (Qin et al., 2024) and MCP-Bench (Wang et al., 2025b), which evaluate large-scale tool discovery or cross-domain MCP usage. Our goal is not broad generalization to unseen tools, but controlled evaluation of whether a stable MCP-style catalog can be internalized into a small model.

3 Problem Definition

We formalize the setting studied in this paper as *description-free tool planning* for a fixed tool catalog. Let $\mathcal{T}$ denote a tool catalog containing a set of MCP servers, tools, argument schemas, return types, and natural-language tool descriptions. A user query is denoted by x, and the desired output is a structured plan $y = (s_1, \ldots, s_n)$ consisting of ordered planning steps. Each step s_i specifies a task description, an MCP server, a tool, a JSON argument object, dependencies on previous steps, and an expected output.

In standard schema-informed tool planning, the model receives both the user query and a serialized representation of the tool catalog, denoted $d(\mathcal{T})$. The model therefore generates plans according to

$$p_\theta(y \mid x, d(\mathcal{T})), \tag{1}$$

where θ are the parameters of the pretrained model. This is the common prompting paradigm used by many tool-use systems: every query is accompanied by the full or partial tool catalog so the model can infer which tools exist and how they should be called.

In contrast, description-free tool planning removes the serialized catalog from the inference-time prompt. The model must instead generate a valid plan using only the user query and the structured output format:

$$p_{\theta'}(y \mid x), \tag{2}$$

where θ' denotes the parameters of the model after tool-use fine-tuning. The goal is for θ' to encode enough information about $\mathcal{T}$ to recover correct server routing, tool selection, argument construction, and dependency ordering without seeing $d(\mathcal{T})$ at inference time.

This setting differs from general open-ended tool use in an important way: the tool catalog is assumed to be fixed during evaluation. We do not attempt to generalize to unseen tools. Instead, we study whether knowledge of a stable catalog can be shifted from prompt context into model parameters. This creates a trade-off between adaptability and efficiency. Prompt-based methods can immediately support new tools by adding their descriptions to the prompt, while description-free fine-tuning reduces prompt length and inference overhead but requires additional training or adapter updates when the tool catalog changes.

Following the AssetOpsBench output format, generated plans use the field name `#Agent` to identify the responsible component. In our MCP setting, this field corresponds to the MCP server responsible for the tool call. We use "server" in prose and preserve `#Agent` when referring to the literal output format.

We evaluate this setting using four criteria. First, the generated plan must select the correct MCP servers and tools. Second, it must construct valid argument keys and values for each selected tool. Third, it must order tool calls into a coherent dependency structure. Fourth, it should achieve these objectives while reducing prompt length relative to schema-informed prompting. The central question of this work is therefore:

> Can a small language model internalize a fixed tool catalog through parameter-efficient fine-tuning, allowing it to outperform schema-informed prompting while omitting tool descriptions at inference time?

4 Methodology

We evaluate whether supervised fine-tuning can substitute for prompt-time tool descriptions in a fixed-catalog setting. The experiments compare schema-informed prompting, description-free prompting without fine-tuning, and description-free prompting after QLoRA fine-tuning. We then analyze adapter rank and capability retention to quantify the cost of specialization.

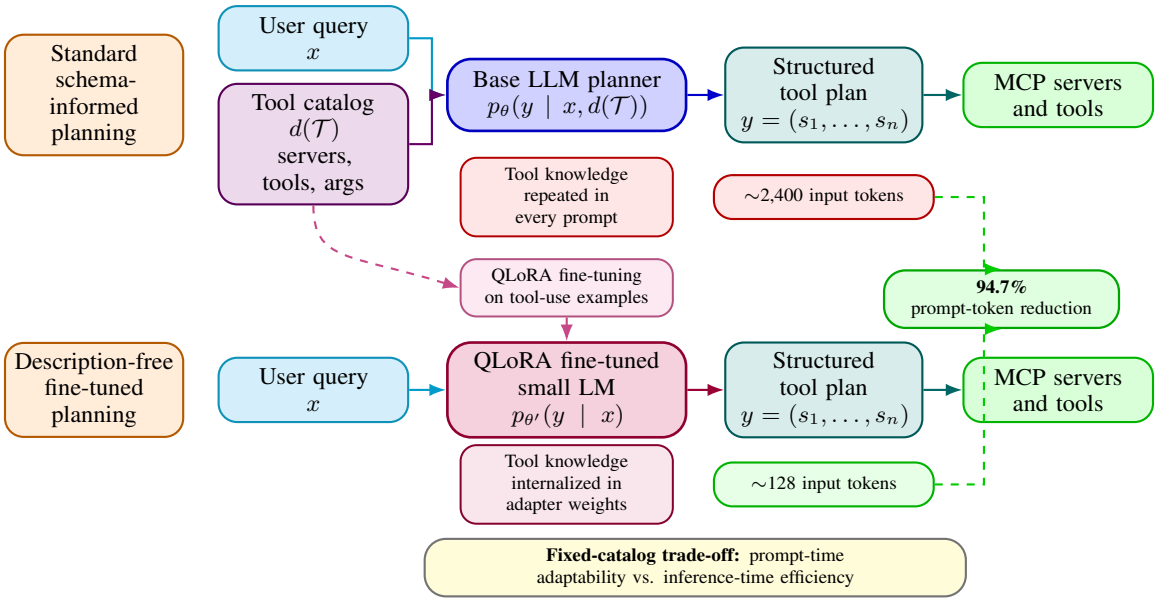

Figure 1: Schema-informed versus description-free tool planning. Standard prompting supplies the user query and serialized tool catalog $d(\mathcal{T})$ at each call, while QLoRA internalizes fixed-catalog tool knowledge into adapter weights, enabling MCP tool-use planning from the query alone.

4.1 Benchmark and Data

Catalog-grounded description-free SFT. We use a teacher–student data construction protocol. During data construction, the teacher model (Gemini 2.5 Flash (Google DeepMind, 2025)) receives the full serialized catalog $d(\mathcal{T})$ and produces schema facts, question-to-plan mappings, and execution-style traces. During student training and evaluation, the serialized catalog is removed from the prompt. The student observes only the user query, output-format instructions, and target plan, forcing the adapter to encode tool names, server ownership, argument keys, and common dependency patterns in its parameters rather than relying on prompt-time schemas.

AssetOpsBench. We use AssetOpsBench (Patel et al., 2025) as the primary benchmark for both data construction and evaluation. The benchmark contains 152 natural-language scenarios, each requiring the model to select appropriate domain tools, generate valid arguments, and order tool calls into a structured execution plan. Each plan consists of sequenced steps in a structured format specifying a task description, the assigned MCP server, the tool to call, JSON arguments, and inter-step dependencies. Table 1 gives a compact overview of the MCP servers and tool counts.

Training data. The resulting data comprises three types of supervised fine-tuning examples (see Appendix A for composition). *Tool and server knowledge* examples teach the model which MCP servers and tools exist, how they are associated, what arguments each tool requires, and how to distinguish between near-miss tools such as `get_failure_modes` and `get_failure_mode_sensor_mapping`. *Question-to-plan* examples, based on the 152 AssetOpsBench scenarios together with generated paraphrases, teach the model to map a natural-language query to a structured plan; each plan is validated to follow the required `#Task/#Agent/#Tool/#Args` format. *Execution-style* examples combine planning steps with execution traces and placeholder-resolution patterns such as `{step_N}`.

Table 1: AssetOpsBench MCP server inventory used in this work

MCP Server	Tools
IoT	4
FMSR	2
TSFM	6
Utilities	3
WorkOrder	8

Prompt structure. At inference time, each query is embedded in a structured planning prompt (shown in full in Appendix G). The prompt contains four components: a system preamble describing the model's role as a planning assistant;

Table 2: Base models used for fine-tuning

	Gemma 4 E4B	Qwen3-4B
Total params	∼8B w/ PLE	4.0B
Active params	4.5B	3.6B
Layers	42	36
Attention	Hybrid	GQA
Hidden size	2560	2560
Vocab size	262K	152K

the complete tool catalog listing all MCP server names, tool signatures, argument types, and descriptions (∼2,200 tokens); the structured output format specification (`#Task`/`#Agent`/`#Tool`/`#Args`/`#Dependency`/`#ExpectedOutput`); and the user question. The full prompt totals approximately 2,400 tokens. In the description-free setting, we remove the complete tool-catalog component while keeping the system preamble, output-format specification, and user question, reducing the prompt to ∼128 tokens—a 94.7% prompt-token reduction.

Data split. We use a pattern-aware stratified 80/20 split over the 152 scenarios, where each scenario is assigned a pattern based on its dominant task family, tool set, and step count. The split is performed at the scenario level *before* paraphrase expansion, so all paraphrases and derived traces for a given scenario remain exclusively in either training or test. This produces 122 training scenarios and 30 held-out test scenarios. The full training corpus contains approximately 1,833 examples. We define three training configurations: Config A (Plan-only, ∼1,200 examples), Config B (Tool-knowledge only, ∼500 examples), and Config C (Tool+Plan, ∼1,741 examples). We use Config C as the primary configuration because it provides the broadest tool-knowledge coverage. Data-composition ablations are in Appendix B.

4.2 Models and Training

We compare two open-source instruction-tuned models: Gemma 4 E4B-it (Google AI for Developers, 2026) and Qwen3-4B (Yang et al., 2025). Table 2 summarizes their architectural differences. Both share a hidden size of 2560 but differ in depth, attention mechanism, and vocabulary size.

Both models are fine-tuned using QLoRA with 8-bit quantization, which keeps the base model weights fixed while training a small set of adapter parameters. We use the same Config C dataset for both models (1,741 training, 92 evaluation examples) and train with identical hyperparameters unless otherwise stated: LoRA rank $r = 32$, $\alpha = 64$, dropout 0.05, learning rate 2×10^{-4} with cosine schedule, batch size 2 with gradient accumulation of 4, 2 epochs, early stopping with patience 2, and 436 total training steps. All linear layers are targeted.

Although the two models use the same LoRA rank, the resulting trainable-parameter fraction differs due to model-size differences. For Gemma, rank 32 corresponds to approximately 0.63% trainable parameters, whereas for Qwen3 it corresponds to approximately 1.62% and about $2.6\times$ higher adaptation intensity. This difference directly affects retention, as discussed in Section 5.3.

4.3 Evaluation Protocol

We evaluate plans using two complementary approaches. Structural metrics provide deterministic, reproducible scores but can penalize valid plans that deviate from the gold reference. LLM-based judging captures semantic quality and accepts valid alternative formulations, but may exhibit grading bias. Using both approaches and checking for agreement provides more robust evidence than either alone.

For structural evaluation, we report AT-F1, which extracts the set of (MCP server, tool) pairs from actionable steps (excluding steps where agent or tool is "none") in both the gold and candidate plans and computes set-based F1 (precision = matched pairs / candidate pairs, recall = matched pairs / gold pairs). We also report ArgKey-F1, which measures whether the model predicts the correct argument field names for matched tool calls. For semantic evaluation, we use Gemini 2.5 Flash as an LLM-as-judge (Zheng et al., 2023) to rate each plan on six dimensions on a 1–5 scale: correctness, server routing, tool selection, argument quality, efficiency, and dependency correctness. The judge overall score is the mean of all six dimensions. Full judge prompts are in Appendix E.

We evaluate models under three prompting conditions: *informed*, where full tool descriptions are included in the prompt; *description-free*, where the base model receives only the user query with no tool schemas; and *fine-tuned description-free*, where the fine-tuned model generates plans without tool schemas at inference time.

We also evaluate capability retention using 100 multiple-choice questions drawn from MMLU (Hendrycks et al., 2021), ARC-Challenge (Clark et al., 2018), and HellaSwag (Zellers et al., 2019).

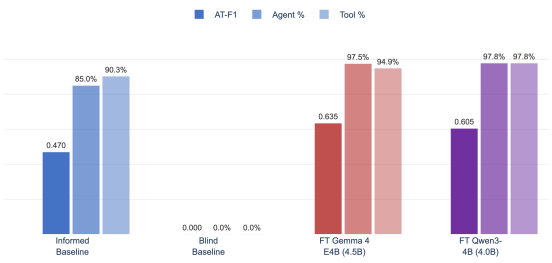

Figure 2: Structural planning metrics across all four configurations. Fine-tuned models operating without tool descriptions surpass the informed baseline on AT-F1, server routing, and tool selection.

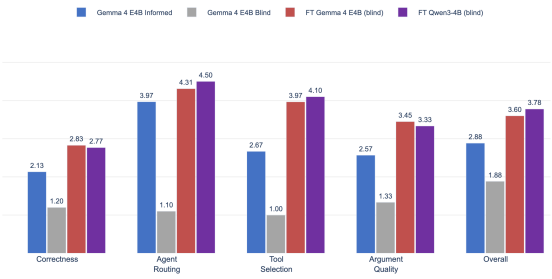

Figure 3: LLM-as-judge scores (1–5 scale) across five evaluation dimensions. Fine-tuned models operating without tool descriptions outperform the informed baseline on every dimension.

Figure 4: Training and evaluation loss over 436 steps (2 epochs). Both models converge smoothly to comparable final evaluation losses (Gemma 0.331, Qwen3 0.347), despite Qwen3 starting from a higher initial loss.

This measures whether tool-use specialization degrades general reasoning behavior that may still be needed by a multi-purpose planner. Details are in Appendix F.

5 Experimental Results

5.1 Main Result: Description-Free Fine-Tuning Improves Tool Planning

We first evaluate whether fine-tuning can replace prompt-time tool descriptions. The unfine-tuned Gemma 4 E4B model is evaluated in two conditions. In the *informed* condition, the prompt contains the system preamble, full serialized tool catalog (∼2,200 tokens of schema text), the output format specification, and the user query, totaling approximately 2,400 input tokens per query (see Appendix G). In the *description-free* condition, the same unfine-tuned model receives only the system preamble, output format, and user query (∼128 tokens). The description-free baseline achieves zero AT-F1, confirming that the base model cannot reliably produce valid tool-use plans without explicit tool descriptions.

Fine-tuning changes this substantially. Figs. 2 and 3 show that fine-tuned models perform well in the description-free setting and outperform the informed baseline on both structural and semantic evaluation metrics.

The fine-tuned Gemma model (Config C, r=32) reaches an AT-F1 of 0.635 and an overall judge score of 3.60, compared with 0.47 and 2.88 for the informed baseline. Server routing and tool-selection accuracy remain high, reaching approximately 95–98% across the held-out evaluation set. The fine-tuned Qwen3 model achieves an AT-F1 of 0.605 and an overall judge score of 3.78, while using substantially less memory. Both models substantially outperform the informed baseline, confirming that fine-tuned description-free planning is viable. As shown in the rank sweep (Section 5.2), a separate Gemma training run with the same configuration achieves a judge score of 3.88, indicating that the improvement over the informed baseline is robust across runs despite some run-to-run variance on the 30-scenario test set.

Metric agreement. Structural and judge-based metrics agree directionally: both rank the configurations in the same broad order—description-free baseline worst, informed baseline substantially better, and fine-tuned description-free models best. This agreement is useful because AT-F1 is deterministic but strict, while LLM judging captures semantically valid alternatives but may introduce grading bias.

Because the 1–5 judge scale has a floor of 1.0, the description-free baseline's score of 1.88 corresponds to 0.88/4.0 on a zero-anchored scale, confirming near-zero semantic planning quality in the absence of either tool descriptions or fine-tuning.

Both models converge smoothly within two epochs (Fig. 4), reaching comparable evaluation

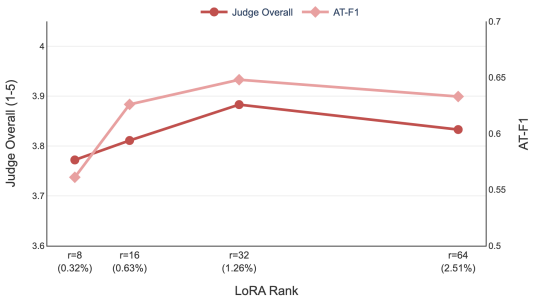

Figure 5: Effect of LoRA rank on planning quality (Gemma 4 E4B). Both Judge Overall (left axis) and AT-F1 (right axis) peak at r=32 and decline at r=64, suggesting diminishing returns at higher adapter capacity.

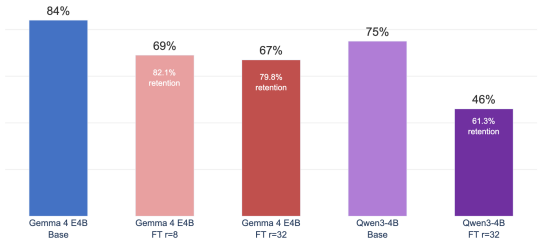

Figure 6: Overall MCQ accuracy before and after tool-use fine-tuning. Gemma 4 E4B retains 79.8–82.1% of base performance; Qwen3-4B retains only 61.3%. Retention percentages shown inside bars. Per-benchmark breakdown in Appendix F.

losses (Gemma 0.331; Qwen3 0.347). The prompt-token savings are substantial: description-free inference after fine-tuning reduces the planning prompt from ∼2,400 tokens to ∼128 tokens, a 94.7% reduction.

5.2 LoRA Rank Selection

Having established that fine-tuning enables description-free inference that surpasses the informed baseline, we next study how adapter capacity affects planning quality. LoRA rank r controls the number of trainable parameters and thus the model's capacity to absorb new tool-use knowledge. We sweep $r \in \{8, 16, 32, 64\}$ on Gemma 4 E4B, corresponding to trainable-parameter fractions from 0.32% to 2.51%.

As shown in Fig. 5, both metrics rise steeply from r=8 (Judge 3.77, AT-F1 0.56) to r=16 (Judge 3.81, AT-F1 0.63) and peak at r=32 (Judge 3.88, AT-F1 0.65). At r=64, performance slightly degrades (Judge 3.83, AT-F1 0.63), suggesting diminishing returns at higher adapter capacity. This finding is important for the retention analysis in Section 5.3: while r=32 maximizes planning quality, smaller ranks preserve more general knowledge.

5.3 Capability Retention

A planner deployed inside a broader agent may still need to recognize unrelated requests, answer simple questions, or reason outside the tool catalog. We therefore measure whether tool-use fine-tuning degrades general multiple-choice performance. We use 100 questions from MMLU, ARC-Challenge, and HellaSwag, independent of AssetOpsBench. We compare Gemma at r=8 and r=32 to test the rank trade-off, and Qwen3 at r=32.

As shown in Fig. 6, Gemma shows moderate degradation: the base model scores 84.0% MCQ accuracy, dropping to 69.0% at r=8 (82.1% retention) and 67.0% at r=32 (79.8% retention). Per-question analysis reveals that Gemma at r=32 forgot 21 of 100 questions while learning 4 new ones. This reveals a quality–retention trade-off: Section 5.2 showed that r=32 achieves the best judge score (3.88 vs. 3.77 for r=8), but at the cost of 2.3 percentage points of retention. For deployment where general reasoning matters, the smaller r=8 adapter may be preferable.

Qwen3 shows larger degradation: its base accuracy drops from 75.0% to 46.0% after fine-tuning at r=32, or 61.3% retention. One likely contributor is adaptation intensity: at the same LoRA rank, Qwen3 trains 1.62% of its parameters compared with 0.63% for Gemma. However, architecture, tokenizer, and pretraining differences are confounded, so this should not be interpreted as a causal explanation. The per-benchmark breakdown (Fig. 9 in Appendix F) reveals that Qwen3's degradation is especially severe on logical reasoning (75%→25%) and commonsense completion (50%→27%), while Gemma's degradation is more evenly distributed.

We verified that degradation was not caused by inference quantization: the base Gemma model achieved 84% in bf16 and 87% in 8-bit, confirming that quantization alone does not explain the 84%→67% drop observed after fine-tuning.

5.4 Deployment Characterization

We report basic deployment measurements to characterize the practical cost of description-free fine-tuning. Table 3 summarizes the results on a single

Table 3: Profiling comparison on A100 80GB

Metric	Gemma	Qwen3
Base memory	11.5 GB	4.42 GB
Peak train memory	24.1 GB	16.06 GB
Train time	56 min	39.7 min
Inference speed	1.4 tok/s	3.49 tok/s
Eval time	59 min	40.7 min
Best eval loss	0.331	0.347

A100 80GB GPU.

Qwen3 uses 62% less base memory, trains 29% faster, and runs 2.5× faster at inference while achieving a comparable planning judge score. Qwen3's speed advantage comes directly from having fewer total parameters (4.0B vs. 8.0B), which reduces the size of every 8-bit matrix multiplication. Full profiling details, including the CUDA operator breakdown and cost estimation, are provided in Appendix D.

Additional data-composition and quantization ablations are provided in Appendices B and C.

6 Discussion

Fixed-catalog internalization. The main result shows that small models can learn a fixed tool catalog well enough to plan without prompt-time tool descriptions. This does not replace schema-informed prompting for open-world or rapidly changing tools. Rather, it targets mature deployments where the same catalog is reused across many queries and schema knowledge can be amortized into adapter weights.

Why can description-free fine-tuning beat schema-informed prompting? One surprising result is that fine-tuned description-free models outperform an unfine-tuned model that receives the full schema. We hypothesize two mechanisms. First, long schema prompts create context competition: the serialized tool catalog dominates the input, and small models may fail to attend to the relevant schema fragment. Second, schema parsing is itself an inference-time task. Supervised fine-tuning converts this into a learned mapping from user intents to tool sequences, arguments, and dependencies. We do not directly isolate these mechanisms, and future work should evaluate fine-tuned models both with and without schemas at inference time to separate internalization effects from context-length effects.

Specialization trade-offs. The retention results show that tool internalization has a cost. Higher LoRA rank improves planning quality but can degrade general multiple-choice performance. This matters when the planner is part of a multi-purpose agent that must handle both tool-related and unrelated requests. Lower-rank adapters, replay data, or modular adapter architectures may reduce this trade-off. A preliminary 10-scenario end-to-end execution pilot validated that plan quality correlates with task completion (70% success), but full closed-loop evaluation remains for future work. Future work should also study continual learning for adding new tools without forgetting old ones, larger benchmarks, multi-seed runs with confidence intervals, and optimized quantized kernels such as GPTQ or AWQ.

7 Conclusion

We studied description-free tool planning for fixed catalogs, where a model must generate structured MCP tool-use plans without receiving tool descriptions at inference time. On AssetOpsBench, 8-bit QLoRA fine-tuning enables ∼4B parameter models to outperform an unfine-tuned schema-informed baseline while reducing prompt length by 94.7%. The best Gemma run achieves 0.65 AT-F1 and a 3.88 judge score, compared with 0.47 and 2.88 for the informed baseline.

The gains come with a specialization cost: LoRA rank affects both planning quality and capability retention. Gemma retains approximately 80–82% of its base MCQ performance after fine-tuning, while Qwen3 shows larger degradation. Overall, these results suggest that supervised adapter fine-tuning is a practical mechanism for amortizing fixed tool-catalog knowledge in small models, but that retention and catalog-update costs remain important deployment considerations.

8 Limitations

This study is a proof of concept on AssetOpsBench. The held-out evaluation set contains only 30 scenarios, and most results are based on single fine-tuning runs rather than multiple random seeds. Small differences between model variants should therefore be interpreted cautiously.

Our evaluation focuses on plan quality rather than full closed-loop execution. A deployed agent may fail at execution time even when the generated plan is structurally correct. We do not evaluate

generalization to unseen tools; this is by design, as description-free planning assumes a fixed catalog. Benchmark coverage is incomplete: not all AssetOpsBench MCP servers, scenarios, or asset classes are evaluated.

9 Use of Generative AI

We used generative AI tools for limited drafting, editing, proofreading assistance, and exploratory scripting. The authors reviewed, edited, and verified the final manuscript, experiments, results, and references, and take full responsibility for the content of the paper.

References

Anthropic. 2026. What is the model context protocol (MCP)? https://modelcontextprotocol.io/docs/getting-started/intro. Accessed: Apr. 30, 2026.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: efficient finetuning of quantized llms. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Google AI for Developers. 2026. Gemma 4 model card. Google AI for Developers Documentation. Accessed: Apr. 30, 2026.

Google DeepMind. 2025. Gemini 2.5 flash and 2.5 flash image model card. https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-2-5-Flash-Model-Card.pdf. Accessed: Apr. 30, 2026.

Jiayi Han, Liang Du, Hongwei Du, Xiangguo Zhou, Yiwen Wu, Yuanfang Zhang, Weibo Zheng, and Donghong Han. 2025. SLIM: Let LLM learn more and forget less with soft LoRA and identity mixture. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4792–4804, Albuquerque, New Mexico. Association for Computational Linguistics.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-Kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8003–8017.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. LLMLingua: Compressing prompts for accelerated inference of large language models. In *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. API-bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, and 8 others. 2024. ToolACE: Winning the points of LLM function calling. *arXiv preprint arXiv:2409.00920*.

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. 2026. SKILL0: In-context agentic reinforcement learning for skill internalization. *arXiv preprint arXiv:2604.02268*.

Dhaval Patel, Shuxin Lin, James Rayfield, Nianjun Zhou, Chathurangi Shyalika, Suryanarayana R Yarrabothula, Roman Vaculin, Natalia Martinez, Fearghal O'donncha, and Jayant Kalagnanam. 2025. Assetopsbench: Benchmarking ai agents for task automation in industrial asset operations and maintenance. *arXiv preprint arXiv:2506.03828*.

Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2024a. The Berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Advances in Neural Information Processing Systems*.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024b. Gorilla: large language model connected with massive apis. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA. Curran Associates Inc.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang,

Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. 2025a. ToolGen: Unified tool retrieval and calling via generation. In *The Thirteenth International Conference on Learning Representations*.

Zhenting Wang, Qi Chang, Hemani Patel, Shashank Biju, Cheng-En Wu, Quan Liu, Aolin Ding, Alireza Rezazadeh, Ankit Shah, Yujia Bao, and Eugene Siow. 2025b. MCP-bench: Benchmarking tool-using LLM agents with complex real-world tasks via MCP servers. In *Workshop on Scaling Environments for Agents*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy. Association for Computational Linguistics.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

A Training Data Composition

Table 4: Three training datasets

Dataset	N	Source	Description
Tool Knowledge	∼500	Gemini 2.5 Flash	Tool taxonomy, ownership, args, routing, hard negatives
Planning	∼1,200	Gold plans + paraphrases	Scenario → concise planning steps
Execution	∼100	Gold plans + traces	Scenario → planning + execution links

Note. Total ∼1,833 examples. Config C (Tool+Plan) uses ∼1,741 after 95/5 train/eval split.

B Training Data Ablation

We compare four data configurations using 8-bit QLoRA with r=32 on Gemma 4 E4B. Fig. 7 shows the results.

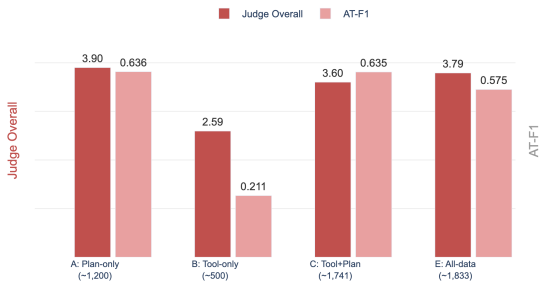

Figure 7: Effect of training data composition on planning quality. Plan-only data achieves the highest judge score (3.90), while Tool-only data performs poorly (2.59).

Plan-only training (Config A, ∼1,200 examples) achieves the highest judge score of 3.90 and AT-F1 of 0.636. Adding tool-knowledge examples (Config C) does not improve the judge score in this run (3.60), though tool selection accuracy is higher (94.9% vs. 92.9%). Tool-only training (Config B) performs poorly (2.59). One possible explanation is that planning examples expose tool usage in realistic query contexts, whereas tool-knowledge examples are more declarative. We treat these results as exploratory because the configurations also differ in size and coverage, and run-to-run variance on the 30-scenario test set can be substantial (see Limitations).

C Quantization Ablation

Table 5: Quantization ablation (Gemma 4 E4B, r=32)

Quantization	AT-F1	Judge	Train Time
8-bit	0.617	3.78	56 min
4-bit NF4	0.642	3.74	∼50 min

Note. Differences are marginal. We use 8-bit as the default for its higher judge score.

D Profiling Details

All profiling was conducted on a single NVIDIA A100 80GB GPU. Fig. 8 provides a visual comparison; Tables 6 and 7 provide additional detail.

For profiling, we use the PyTorch Profiler, `torch.cuda` memory tracking, and Weights & Biases. Training throughput scaled from ∼886 tok/s at batch size 1 to ∼1,415 tok/s at batch size 4, beyond which the model exceeded available memory. The main CUDA bottleneck was `MatMul8bitLt`, which accounted for 56.3% of total CUDA time, reflecting the dominance of 8-bit matrix multiplication in quantized inference.

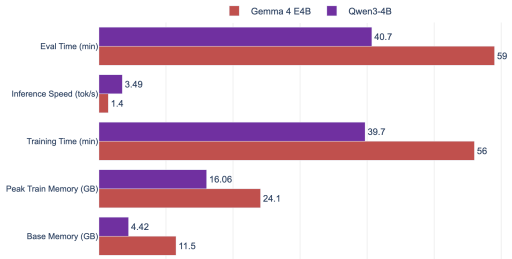

Figure 8: Profiling comparison between Gemma 4 E4B and Qwen3-4B. Qwen3 uses less memory, trains faster, and achieves 2.5× higher inference throughput.

Table 6: CUDA operator breakdown (Gemma inference, 64 tokens)

Operation	% CUDA Time
`MatMul8bitLt`	56.3%
`aten::mm`	18.2%
Attention kernel	8.7%
Other ops	16.8%

Note. Attention kernel refers to `scaled_dot_product_attention`. Other ops include softmax, layer normalization, and related kernels.

Table 7: Experiment cost breakdown (A100 at $3.93/hr)

Component	A100 Hours	Cost
Gemma ablations	3.1	$12.18
Qwen3 train + eval	1.5	$5.78
Retention analysis	11.2	$44.00
Total	**∼15.8**	**∼$62**

E Judge Prompts

E.1 Plan Quality Judge

We use Gemini 2.5 Flash (temperature=0, max_tokens=8192) to evaluate each candidate plan against the gold reference. The prompt provides the question, gold plan, candidate plan, and tool inventory, and asks the judge to rate on a 1–5 scale across six dimensions: correctness, server routing, tool selection, argument quality, efficiency, and dependency correctness. Each dimension includes a rubric (5 = matches gold reference, 3 = partially correct, 1 = major errors). The judge returns a JSON object with integer scores.

E.2 MCQ Retention Judge

For the retention benchmark, we use Gemini 2.5 Flash (temperature=0, max_tokens=1024). The prompt provides the question, answer choices, correct answer, and the model's full response. The judge is instructed to look at the *final* answer the model commits to, ignoring intermediate reasoning, and return a JSON object with a binary correct/incorrect grade. This approach is more reliable than token-level evaluation because instruction-tuned models produce chain-of-thought reasoning before committing to an answer letter.

F Retention Benchmark Details

Table 8: Retention benchmark composition

Source	N	Description
MMLU (5 subjects)	40	Knowledge and reasoning
ARC-Challenge	30	Grade-school science
HellaSwag	30	Commonsense completion
Total	100	

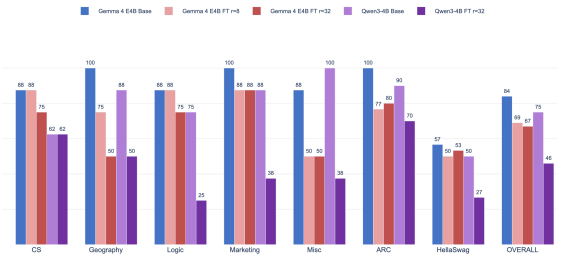

Figure 9: Per-benchmark MCQ accuracy for base and fine-tuned models. Qwen3-4B shows severe degradation on Logic (75%→25%), Marketing (88%→38%), and HellaSwag (50%→27%). Gemma degradation is more uniform across benchmarks.

MMLU subjects are selected for a suitable difficulty range on ∼4B models (base accuracy 40–100%): High School Computer Science, High School Geography, Logical Fallacies, Marketing,

and Miscellaneous. We avoid niche subjects where the base model scores near zero, making retention difficult to interpret.

The evaluation protocol is fixed across all models: 100 MCQ examples loaded with a fixed random seed; the model is prompted to think step by step and provide a final answer letter (A/B/C/D); generation is capped at 512 tokens; Gemini judges correctness. Overall retention = FT accuracy / base accuracy.

G Prompt Example

We summarize the informed and description-free prompts used for AssetOpsBench scenario 114.

Question: "What are the failure modes of Chiller 6 that can be identified by analyzing the data from the available sensors?"

Informed prompt. The informed prompt contains four components:

1. a system instruction describing the model as a planning assistant for industrial asset operations;
2. the full tool catalog, including MCP server names, tool signatures, argument schemas, and natural-language descriptions;
3. the required structured output format: `#Task`, `#Agent`, `#Tool`, `#Args`, `#Dependency`, and `#ExpectedOutput`;
4. the user question.

This prompt is approximately 2,400 tokens, of which roughly 2,200 tokens come from the serialized tool catalog.

Gold plan. The gold plan contains four tool calls:

1. call `IoTAgent.assets` with `{"site_name": "MAIN"}` to identify the asset ID for "Chiller 6";
2. call `IoTAgent.sensors` with the site name and asset ID to retrieve available sensors;
3. call `FMSRAgent.get_failure_modes` with `{"asset_name": "Chiller 6"}` to retrieve known failure modes;
4. call `FMSRAgent.get_failure_mode_sensor_mapping` using the failure modes and sensors from the previous steps to determine which failures can be detected by which sensors.

Description-free prompt. In the description-free setting, the entire tool catalog is removed. The model receives only the system instruction, output-format specification, and user question, reducing the input to approximately 128 tokens. The fine-tuned model must therefore recover the same tool sequence from internalized tool knowledge rather than from prompt-time tool descriptions.