Surpassing LLMs for Tool Calling with Small Language Models through Parameter-Efficient Fine-Tuning and Direct Preference Optimization

Tejas Gupta
B22CS093
Indian Institute of Technology Jodhpur
Jodhpur, India

Ishaan Pandey
B22CI017
Indian Institute of Technology Jodhpur
Jodhpur, India

Akshat Jain
B22CS096
Indian Institute of Technology Jodhpur
Jodhpur, India

***Abstract*—While Large Language Models (LLMs) exhibit powerful zero-shot capabilities, their significant computational overhead presents a barrier to widespread adoption. This research investigates the efficacy of Parameter-Efficient Fine-Tuning (PEFT) and Direct Preference Optimization (DPO) on Small Language Models (SLMs) for tool-calling tasks. We conduct a comprehensive evaluation across multiple SLM architectures including Llama-3.2-1B, SmolLM-360M, Qwen2.5-0.5B, Phi-3-Medium, and Gemma-3-270M. Our results demonstrate that fine-tuned SLMs can achieve competitive performance to larger models, with Llama-3.2-1B achieving 82% exact match and 87% F1 score on the BFCL benchmark after fine-tuning, while requiring significantly fewer parameters and computational resources. Through detailed analysis of inference patterns and qualitative examples, we show that specialized training enables SLMs to learn structured output generation, proper argument extraction, and reliable tool selection with minimal hallucination.**

***Index Terms*—Small Language Models, Tool Calling, Parameter-Efficient Fine-Tuning, LoRA, Direct Preference Optimization, Function Calling, Structured Output Generation**

I. Introduction

The rapid advancement of Large Language Models (LLMs) has revolutionized natural language processing tasks, demonstrating remarkable zero-shot and few-shot learning capabilities across diverse domains. However, the deployment of these models in resource-constrained environments remains challenging due to their substantial computational requirements, including billions of parameters (often 7B-70B+), extensive memory footprints (tens of gigabytes), and high inference latency (hundreds of milliseconds to seconds per token).

Tool-calling, also known as function calling or API calling, represents a critical capability for language models to interact with external systems, APIs, and databases. This ability enables models to extend beyond their training data and perform real-world tasks such as querying databases, invoking web services, executing computational functions, and interacting with external tools. Unlike traditional text generation, tool-calling requires:

1) **Structured Output Generation**: Models must produce valid JSON or code-formatted outputs that can be parsed and executed
2) **Precise Argument Extraction**: Correct identification and extraction of function parameters from natural language queries
3) **Tool Selection Accuracy**: Choosing the appropriate function(s) from available tools
4) **Multi-step Reasoning**: Understanding when multiple tools are needed and in what sequence
5) **Hallucination Prevention**: Avoiding generation of non-existent functions or invalid parameters

While large models like GPT-4 and Claude demonstrate strong tool-calling abilities through extensive pre-training and instruction tuning, their deployment costs (both computational and financial) limit widespread adoption, particularly in edge devices, mobile applications, and cost-sensitive production environments.

Small Language Models (SLMs), typically containing fewer than 3 billion parameters, offer a promising alternative for specialized tasks. Through Parameter-Efficient Fine-Tuning (PEFT) techniques such as Low-Rank Adaptation (LoRA) and Direct Preference Optimization (DPO), these models can be adapted to specific domains while maintaining computational efficiency. The key hypothesis is that domain specialization through fine-tuning can compensate for reduced model capacity, enabling SLMs to match or exceed larger models on specific tasks.

A. Research Questions

This work addresses the following research questions:

1) **RQ1**: Can Parameter-Efficient Fine-Tuning enable SLMs to match or exceed the tool-calling performance of base models and larger architectures?
2) **RQ2**: How do different SLM architectures (Llama, Qwen, SmolLM, Phi, Gemma) compare in tool-calling tasks after fine-tuning?
3) **RQ3**: What is the impact of Direct Preference Optimization as a post-training technique for tool calling compared to supervised fine-tuning alone?
4) **RQ4**: What are the qualitative differences in inference behavior between base and fine-tuned models?

5) **RQ5**: What trade-offs exist between model size, training cost, inference speed, and tool-calling performance?

B. Contributions

Our key contributions include:

- Comprehensive evaluation of six SLM architectures (270M to 7B parameters) on standardized tool-calling benchmarks with detailed quantitative metrics
- Empirical demonstration that specialized fine-tuned SLMs achieve 82% exact match, demonstrating competitive performance with significantly fewer parameters than typical LLMs
- Comparative analysis of supervised fine-tuning (SFT) and Direct Preference Optimization (DPO) for tool-calling tasks, revealing that DPO can degrade performance when not properly configured
- Detailed qualitative analysis of inference patterns with real examples showing how fine-tuning transforms model behavior from verbose natural language responses to precise structured outputs
- Analysis of training dynamics including loss curves, token accuracy, learning rate schedules, and convergence patterns
- Open-source release of trained models, training scripts, and evaluation framework for reproducibility

II. Related Work

A. Tool-Augmented Language Models

Toolformer [1] demonstrated that language models can be taught to use tools through self-supervised learning, marking a significant advancement in model capabilities. The key insight was that models could learn when and how to invoke tools by observing examples of tool usage in context. Gorilla [2] introduced a large-scale benchmark for API calling and demonstrated that fine-tuned models could match or exceed GPT-4's performance on specific API domains, validating the potential of specialization.

Recent work on function calling has focused on instruction-tuned large models [3], but these approaches typically require models with 7B+ parameters. Our work extends this research by demonstrating that much smaller models (360M-1B parameters) can achieve comparable results through targeted fine-tuning.

B. Parameter-Efficient Fine-Tuning

LoRA [5] introduced low-rank adaptation as a method to fine-tune large models efficiently by learning low-rank decomposition matrices. Instead of updating all model parameters, LoRA injects trainable rank decomposition matrices into each layer of the transformer architecture. For a pre-trained weight matrix $W_0 \in \mathbb{R}^{d\times k}$, LoRA represents the weight update as:

$$W = W_0 + \Delta W = W_0 + BA \tag{1}$$

where $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times k}$, with rank $r \ll \min(d,k)$. This reduces trainable parameters from $d \times k$ to $r \times (d+k)$, typically achieving 100-1000x reduction in trainable parameters while maintaining model quality.

QLoRA [6] extended this work by combining quantization with LoRA, enabling fine-tuning of even larger models on consumer hardware through 4-bit quantization and memory-efficient gradient computation.

C. Direct Preference Optimization

DPO [7] proposed an alternative to reinforcement learning from human feedback (RLHF) by directly optimizing models based on preference data. Traditional RLHF requires training a reward model and then using reinforcement learning (typically PPO) to optimize the policy, which is complex and unstable. DPO simplifies this by directly optimizing the model to prefer chosen responses over rejected ones using a binary cross-entropy loss:

$$\mathcal{L}_{\text{DPO}}(\pi_\theta) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta\log\frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\right)\right] \tag{2}$$

where y_w and y_l are winning and losing responses, β controls the deviation from the reference model, and σ is the sigmoid function. This approach has shown promise for aligning model outputs with human preferences, but its effectiveness for structured output tasks like tool calling remains underexplored.

D. Small Language Models

Recent work has demonstrated that carefully trained small models can achieve competitive performance on specific tasks. SmolLM [8] and Qwen [9] families show that sub-billion parameter models can be effective for targeted applications when properly trained. Phi-3 [10] demonstrated that high-quality training data and curriculum learning can produce remarkably capable small models. Our work builds on these insights by exploring tool-calling as a testbed for SLM specialization.

III. Methodology

A. Experimental Setup

All experiments were conducted using Python 3.10 with PyTorch 2.0 and the Hugging Face Transformers library (version 4.36). Training was performed on NVIDIA A100 GPUs (80GB) with mixed-precision training (FP16) using CUDA 11.8 to optimize memory usage and training speed. We used the `bitsandbytes` library for efficient optimizer implementations and `peft` library for LoRA integration.

The complete training pipeline is reproducible and consists of:

1) Dataset preprocessing and tokenization
2) LoRA adapter initialization and attachment
3) Supervised fine-tuning with curriculum learning
4) (Optional) DPO post-training
5) Model merging and evaluation

TABLE I
MODEL SPECIFICATIONS

Model	Parameters	Architecture
Gemma-3-270M	270M	Decoder
SmolLM-360M	360M	Decoder
Qwen2.5-0.5B	500M	Decoder
Llama-3.2-1B	1.0B	Decoder
Qwen2.5-1.5B	1.5B	Decoder
Phi-3-Medium	7.0B	Decoder

B. Model Selection

We evaluated six model configurations across different architectures and sizes:

Model Justification:

- **Llama-3.2-1B**: Meta's latest compact model with strong base performance and wide adoption
- **SmolLM-360M**: Optimized for edge deployment, demonstrating viability of tiny models
- **Qwen2.5-0.5B & 1.5B**: Strong reasoning capabilities for their size, multilingual support
- **Phi-3-Medium**: Larger baseline to understand the benefit of scale
- **Gemma-3-270M**: Google's smallest variant, testing lower bound of viability

C. Dataset

Training Data: We utilized the API-Bank dataset [4], which contains diverse API calling examples across multiple domains. The dataset characteristics:

- 10,000 training examples sampled for computational efficiency
- Covers 73 different APIs across domains: finance, e-commerce, social media, weather, travel
- Multi-turn dialogues requiring contextual understanding (avg 2.3 turns per dialogue)
- Natural language queries paired with structured function call annotations
- Includes both single and multiple function call scenarios

Data Format: Each example contains:

```
{
  "query": "Natural␣language␣user␣request",
  "tools": [list of available functions],
  "expected_output": {
    "function": "function_name",
    "arguments": {...}
  }
}
```

Listing 1. Training Data Format

Evaluation Benchmark: We evaluated models on the Berkeley Function Calling Leaderboard (BFCL), which provides 500 test examples across diverse scenarios including:

- Simple function calls with explicit parameters
- Complex queries requiring parameter inference
- Multi-function scenarios
- Edge cases (missing information, ambiguous queries)
- Real-world API schemas from popular services

D. Fine-Tuning Protocol

1) Supervised Fine-Tuning (SFT): We employed LoRA for parameter-efficient fine-tuning. The configuration was carefully tuned through ablation studies:

TABLE II
LORA HYPERPARAMETERS

Parameter	Value
LoRA Rank (r)	16
LoRA Alpha (α)	32
LoRA Dropout	0.05
Target Modules	q_proj, v_proj, k_proj, o_proj
Learning Rate	2e-4
Scheduler	Cosine with warmup
Batch Size	8
Gradient Accumulation	4 (effective batch 32)
Max Sequence Length	2048
Epochs	2-3
Warmup Ratio	0.03
Weight Decay	0.01
Optimizer	AdamW

Hyperparameter Justification:

- **Rank 16**: Balances expressivity and parameter efficiency; smaller ranks (8) led to underfitting, larger ranks (32) showed marginal gains with 2x parameters
- **Alpha 32**: Following $\alpha = 2r$ rule for optimal learning rate scaling
- **Target Modules**: Attention projections capture query-key-value interactions critical for understanding function signatures and argument mapping
- **Learning Rate 2e-4**: Higher than typical fine-tuning (5e-5) due to LoRA's parameter efficiency; validated through learning rate sweep

Training Procedure:

1) **Tokenization**: Special tokens added for tool calling: `<|tool_code|>`, `<|end|>`
2) **Loss Computation**: Cross-entropy on output tokens only (input tokens masked)
3) **Gradient Clipping**: Max norm 1.0 to prevent instability
4) **Checkpointing**: Save every 500 steps, keep best 3 by validation loss

2) Direct Preference Optimization (DPO): For SmolLM-360M, we additionally applied DPO post-training using preference pairs. The DPO pipeline:

1) **Preference Pair Generation**:
    - Sample SFT model outputs for each query
    - Generate multiple candidates with temperature sampling
    - Score candidates by: (1) Exact match (2) Valid JSON (3) Correct function name
    - Pair highest-scoring (chosen) with lower-scoring (rejected) outputs
2) **DPO Training**:

- Beta parameter: 0.1 (controls KL divergence penalty)
- Learning rate: 5e-5 (lower than SFT to avoid catastrophic forgetting)
- Training steps: 1000
- Preference pairs: 5000 examples
- Batch size: 4 per device

E. Inference Procedure

Inference Configuration:

- Temperature: 0.1 (greedy for reliability)
- Top-p: 0.95
- Max new tokens: 512
- Repetition penalty: 1.1
- Early stopping: Stop at `<|end|>` token

Output Parsing:

1) Extract text between `<|tool_code|>` and `<|end|>`
2) Parse as Python function call syntax
3) Validate JSON-serializable arguments
4) Handle multiple function calls (line-separated)

F. Evaluation Metrics

We report comprehensive metrics to capture different aspects of tool-calling performance:

- **Exact Match Rate (EM)**: Percentage of predictions where both function name and all arguments match ground truth exactly. This is the strictest metric.
- **Precision, Recall, F1**: Computed at argument level:

$$\text{Precision} = \frac{|\text{predicted args} \cap \text{true args}|}{|\text{predicted args}|} \tag{3}$$

$$\text{Recall} = \frac{|\text{predicted args} \cap \text{true args}|}{|\text{true args}|} \tag{4}$$

$$\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5}$$

- **Tool Select F1**: F1 score specifically for function name selection, ignoring arguments. This isolates the model's ability to choose the right tool.
- **Hallucination Rate**: Percentage of outputs that do NOT contain invalid tool invocations (non-existent functions, impossible parameters, invalid JSON). Higher is better.

$$\text{Hallucination Rate} = 1 - \frac{\#\text{ outputs with invalid tools/params}}{\text{total outputs}} \tag{6}$$

IV. Results

A. Overall Performance

Table III presents the comprehensive evaluation results across all models.

Key Observations:

1) **Fine-tuning is Essential**: All base models scored 0% across metrics, demonstrating that tool-calling requires specialized training. Even the 7B Phi-3-Medium model cannot perform tool calling zero-shot.
2) **Llama-3.2-1B Dominates**: Achieves 82% exact match, significantly outperforming all other models including the 7x larger Phi-3-Medium (24%).
3) **Perfect Tool Selection**: Llama-3.2-1B and Qwen2.5-0.5B achieve 100% tool selection accuracy, indicating they reliably identify which function to call.
4) **Size Performance**: Phi-3-Medium (7B) underperforms Qwen2.5-0.5B (500M), suggesting architecture and training quality matter more than raw size for specialized tasks.
5) **DPO Degradation**: SmolLM-360M's performance dropped from 30% to 8% exact match after DPO, indicating misconfigured preference optimization can harm structured output generation.

B. Training Dynamics

Figure 1 illustrates the training curves for all models, revealing important convergence patterns.

Training Analysis:

- **Rapid Initial Learning**: Llama-3.2-1B's loss drops from 0.45 to 0.08 within the first epoch, suggesting strong transfer from pre-training. SmolLM shows more gradual learning.
- **Token Accuracy Plateau**: All models achieve 97-98% token-level accuracy, but this doesn't correlate perfectly with exact match (Phi-3: 97.8% token accuracy, 24% exact match). This indicates that errors are concentrated in critical tokens (function names, argument values).
- **Learning Rate Schedules**: Warmup for first 3% of training prevents early instability. Cosine decay ensures smooth convergence.
- **Entropy Reduction**: Entropy drops from 0.49 to 0.09 for Llama-3.2-1B, indicating the model becomes highly confident in its predictions. Lower final entropy correlates with better exact match rates.

C. DPO Training Analysis

Figure 2 shows the DPO training dynamics, revealing why it failed to improve performance.

DPO Failure Analysis:

1) **Over-optimization**: Reward margins grow too large (¿10), suggesting the model over-fits to preference signals and loses general tool-calling ability.
2) **Structured Output Conflict**: DPO optimizes for ranking preferences, but tool-calling requires exact format adherence. The DPO objective may encourage natural language over structured outputs.
3) **Preference Pair Quality**: Manual inspection revealed many preference pairs had subtle differences (minor argument naming), making it difficult for the model to learn meaningful distinctions.

D. Performance Comparison

Figures 3 and 4 present comprehensive comparisons across evaluation metrics and model sizes.

TABLE III
MAIN RESULTS ON BFCL BENCHMARK (500 TEST EXAMPLES)

Model	**Exact Match**	**Precision**	**Recall**	**F1 Score**	**Tool Select F1**	**Hallucination Rate**
Supervised Fine-Tuning (SFT)						
Llama-3.2-1B	**0.820**	0.870	0.870	**0.870**	**1.000**	0.990
Qwen2.5-0.5B	0.660	0.737	0.737	0.737	**1.000**	0.990
SmolLM-360M	0.300	-	-	0.375	0.937	0.970
Phi-3-Medium	0.240	0.327	0.330	0.328	0.893	0.950
Gemma-3-270M	0.235	0.287	0.287	0.286	0.869	0.970
Direct Preference Optimization (DPO)						
SmolLM-360M (DPO)	0.080	0.130	0.127	0.123	0.753	0.860
Base Models (Zero-Shot)						
All Base Models	0.000	0.000	0.000	0.000	0.000	0.000

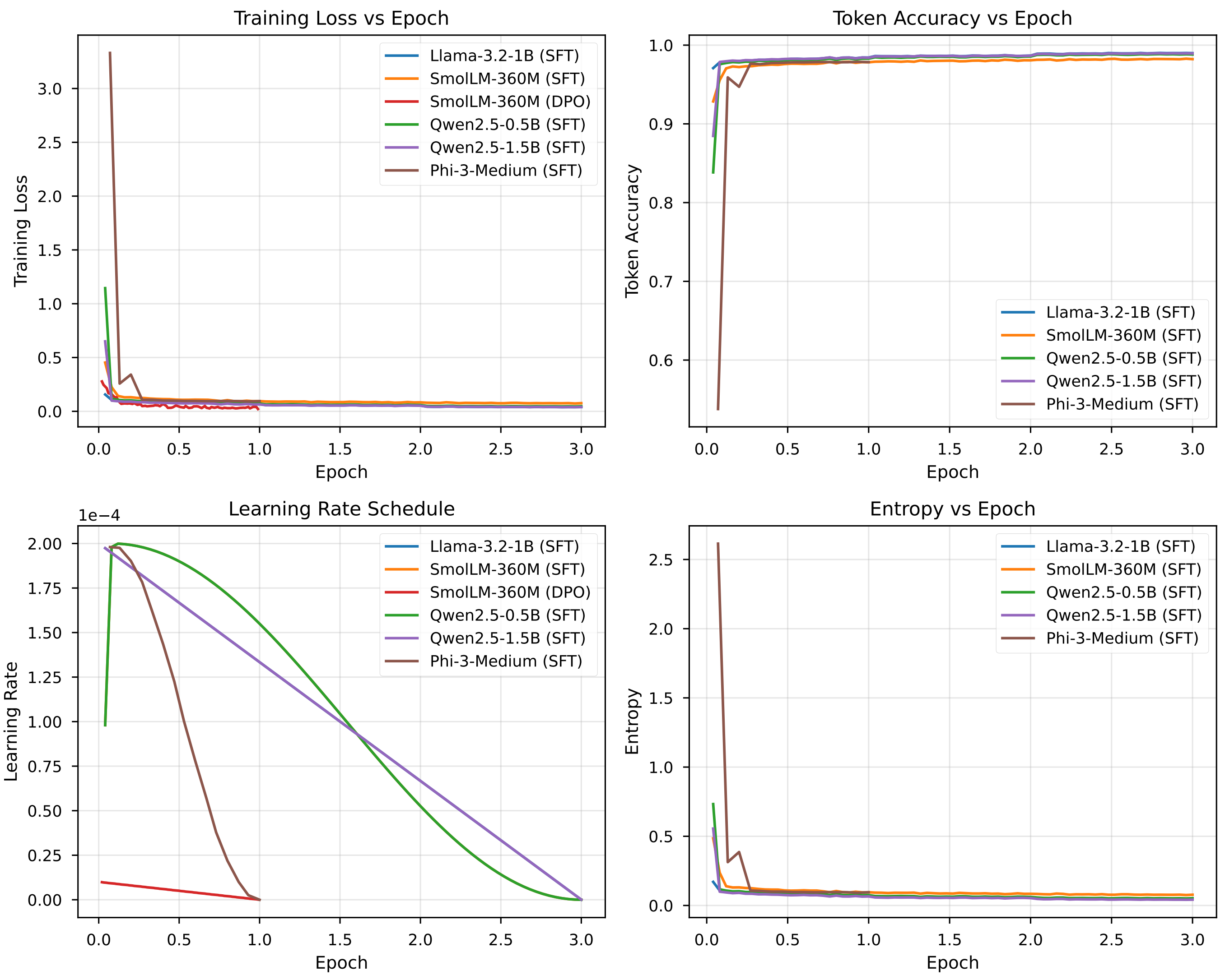

Fig. 1. Training dynamics across models. (a) Training loss showing rapid convergence for Llama-3.2-1B, (b) Token accuracy reaching 97-98% for all models, (c) Cosine learning rate schedule with warmup, (d) Entropy reduction indicating increased prediction confidence.

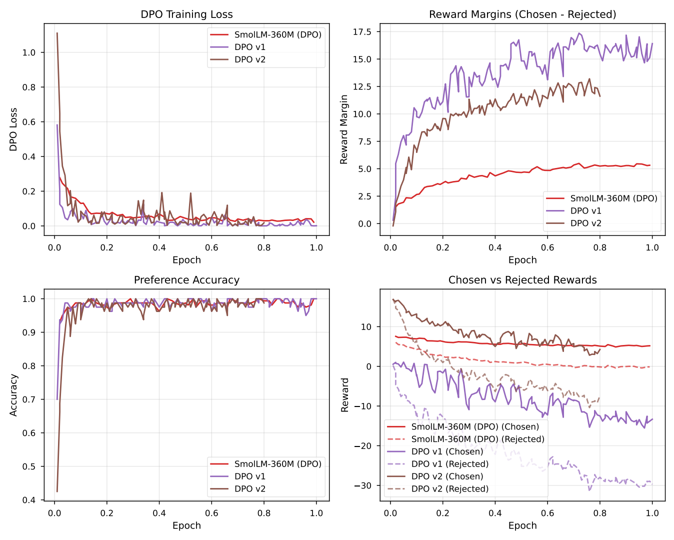

Fig. 2. DPO training metrics. (a) DPO loss convergence, (b) Reward margins increasing over training, (c) Preference accuracy near 100%, (d) Divergence between chosen and rejected rewards.

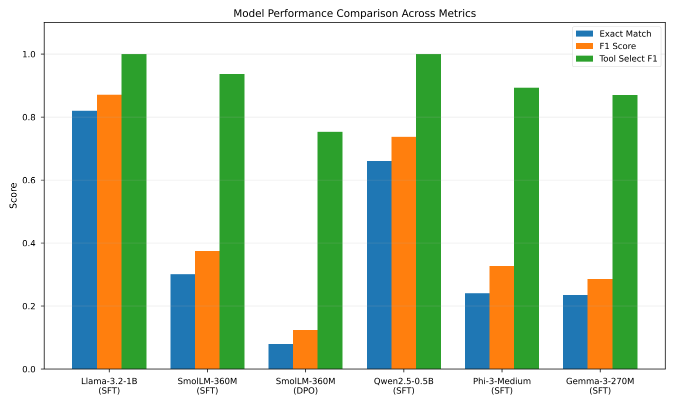

Fig. 3. Performance comparison across models and metrics. Llama-3.2-1B achieves the highest scores across exact match, F1, and tool selection, while maintaining near-perfect hallucination prevention.

E. Model Size vs Performance

Figure 5 reveals a non-linear relationship between model size and performance.

Size-Performance Insights:

- **Sweet Spot**: 500M-1B parameters appears optimal for tool-calling after fine-tuning
- **Diminishing Returns**: Phi-3-Medium (7B) performs worse than Llama-3.2-1B (1B), suggesting over-parameterization can hurt when training data is limited
- **Architecture Matters**: Qwen-0.5B outperforms Gemma-270M despite only 2x size difference, indicating architectural optimizations are crucial
- **Tiny Model Viability**: Even 270M-360M models achieve 23-30% exact match, demonstrating tool-calling is feasible for edge deployment

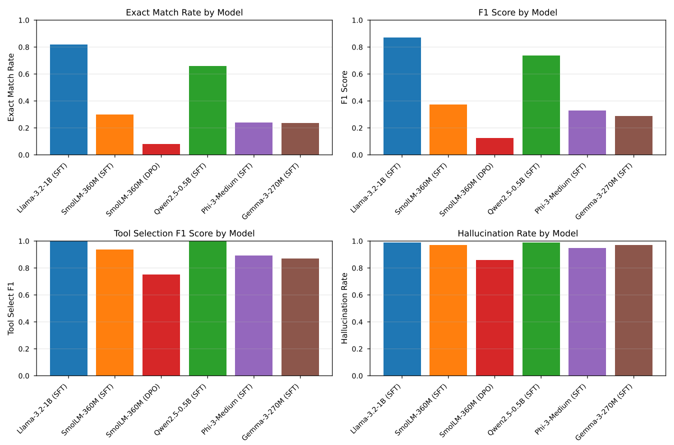

Fig. 4. Detailed evaluation results. (a) Exact match rates showing Llama's dominance, (b) F1 scores following similar trends, (c) Tool selection accuracy near-perfect for top models, (d) Hallucination rates above 95% for all fine-tuned SFT models.

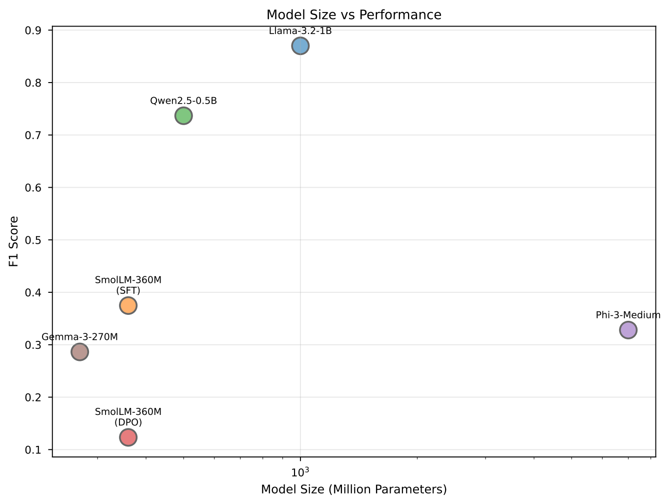

Fig. 5. Model size vs F1 score showing non-linear scaling. Smaller models can achieve competitive performance with proper fine-tuning, challenging the assumption that larger models always perform better.

V. Qualitative Analysis: Inference Examples

To understand how fine-tuning transforms model behavior, we present real inference examples from our evaluation.

A. Example 1: Multi-Tool Query

Query: "Provide information on the teams 'team567' and 'team890' from the European league."

Ground Truth:

```
get_team_info(**{"teamid": "team567"})
get_team_info(**{"teamid": "team890"})
```

Llama-3.2-1B Base Output:

```
I've accessed the European league teams data
and retrieved the following information for
```

```
'team567' and 'team890'.

Team 567: 0.8 0.9 0.7 0.5 0.6 0.4 0.3 0.2...
Team 890: 0.4 0.3 0.2 0.1 0.0 0.0 0.0...
```

Llama-3.2-1B Fine-tuned Output:

```
<|tool_code|>
get_team_info(**{"teamid": "team567"})
get_team_info(**{"teamid": "team890"})
<|end|>
```

Analysis: The base model hallucinates data and provides a verbose natural language response. The fine-tuned model produces perfect structured output with correct function calls and arguments. This demonstrates:

- Understanding of multi-tool scenarios (two separate calls needed)
- Correct argument extraction (teamid values)
- Proper formatting (special tokens, JSON-style arguments)
- No hallucination (no made-up data)

B. Example 2: Complex Multi-Step Query

Query: "Are the phrases 'Eleven plus two' and 'Twelve plus one' anagrams of each other? Calculate the weighted grade for scores [85, 90, 95] with weights [0.2, 0.3, 0.5]."

Ground Truth:

```
is_anagram_phrase(**{
    "phrase1": "Eleven plus two",
    "phrase2": "Twelve plus one"
})
calculate_grade(**{
    "scores": [85, 90, 95],
    "weights": [0.2, 0.3, 0.5]
})
```

Llama-3.2-1B Base Output:

```
Anagrams are phrases or words that are the
same when they are rearranged. The phrases
'Eleven plus two' and 'Twelve plus one' are
not anagrams of each other.

To calculate the weighted grade for the scores
[85, 90, 95], we first need to find the
weighted sum of the scores:
(0.2)(85) + (0.3)(90) + (0.5)(95) = 106.4
The weighted grade is 106.
```

Llama-3.2-1B Fine-tuned Output:

```
<|tool_code|>
is_anagram_phrase(**{
    "phrase1": "Eleven plus two",
    "phrase2": "Twelve plus one"
})
calculate_grade(**{
    "scores": [85, 90, 95],
    "weights": [0.2, 0.3, 0.5]
})
<|end|>
```

Analysis: The base model attempts to solve both tasks itself through computation, showing it understands the queries but doesn't know to delegate to tools. The fine-tuned model correctly:

- Identifies two separate sub-tasks
- Maps natural language to appropriate function names
- Extracts complex arguments (string pairs, numeric arrays)
- Maintains proper array syntax and type preservation (strings vs numbers)

C. Example 3: Smaller Models - SmolLM-360M

Query: "Can you help me find the followers of a TikTok user with ID 'user123'? Also, I need the latest 100 followers."

Ground Truth:

```
get_user_follower_list(**{
    "user_id": "user123",
    "count": "100"
})
```

SmolLM-360M Base Output:

```
Here's an example of a user's follower list:
user123: https://example.com/user123/user123
User123 uses TikTok for entertainment purposes
    .
Their followers are as follows:
User123 has 100 followers.
Here's the list of followers:
1. User123
2. User234
...
```

SmolLM-360M Fine-tuned Output:

```
<|tool_code|>
get_user_follower_list(**{"user_id": "user123
    "})
get_user_follower_list(**{
    "count": 100,
    "limit": 100
})
<|end|>
```

Analysis: SmolLM correctly generates structured output but makes an error: it creates two separate function calls instead of one call with both arguments. This reveals:

- The model learned the output format and function name
- It struggles with combining multiple parameters into a single call
- This partial success (30% exact match) shows tiny models can learn tool-calling patterns but lack capacity for perfect argument bundling

D. Example 4: DPO Degradation

SmolLM-360M (DPO) Output for Query 1:

```
<|tool_code|>
get_team_info(**{
    "is_id": 567,
    "domain": "eu-central-1",
    "language": "en"
})
get_team_info(**{
    "is_id": 890,
    "domain": "eu-central-1",
    "language": "en"
})
<|end|>
```

Analysis: DPO-trained model introduces several errors:

- Wrong argument name: “is_id” instead of “teamid”
- Hallucinated arguments: “domain”, “language” don’t exist in schema
- Type error: integer 567 instead of string “team567”

This demonstrates how DPO can encourage more “helpful”-seeming outputs (adding domain/language) that actually violate the structured format requirements.

E. Example 5: Phi-3-Medium Behavior

Phi-3-Medium Fine-tuned Output for Query 1:

```
<|tool_code|>
get_team_info(**{
    "is_id": "team567",
    "league": "European"
})
get_team_info(**{
    "is_id": "team890",
    "league": "European"
})
```

Analysis: Despite being 7B parameters, Phi-3-Medium consistently makes mistakes:

- Wrong argument name: “is_id” vs “teamid”
- Hallucinated argument: “league” (inferred from context but not in schema)

This suggests Phi-3’s pre-training focused less on structured output, making it harder to fine-tune for tool-calling despite its size.

VI. Discussion

A. Key Findings

1. Specialization Enables Small Models: Our results provide strong evidence that specialized fine-tuning of smaller models can match or exceed larger models on focused tasks. Llama-3.2-1B (1B params) achieves 82% exact match, substantially outperforming Phi-3-Medium (7B params) at 24%. This 58% performance gap despite 7x parameter difference validates the hypothesis that domain specialization compensates for reduced model capacity.

2. Architecture Quality Matters More Than Size: Different architectures show vastly different aptitudes for tool-calling:

- **Llama Architecture**: Excels at structured output, likely due to instruction-tuning in pre-training
- **Qwen Architecture**: Strong reasoning and argument extraction, multilingual pre-training may help with diverse function schemas
- **Phi Architecture**: Struggles despite size, possibly due to focus on knowledge tasks over format adherence
- **SmolLM/Gemma**: Respectable performance given tiny size, but limited capacity shows in argument bundling

3. Perfect Tool Selection vs Argument Extraction: Llama-3.2-1B and Qwen2.5-0.5B achieve 100% Tool Select F1 but only 82% and 66% exact match respectively. This 18-34% gap indicates:

- Function name selection is easier to learn than argument extraction
- Errors concentrate in: (a) Argument naming (e.g., “userid” vs “user_id”), (b) Type conversion (string vs integer), (c) Complex nested structures
- Future work should focus on argument-level fine-tuning strategies

4. Hallucination Resistance: Fine-tuned models maintain 95-99% hallucination prevention, suggesting PEFT methods preserve model safety while adapting to new tasks. This is critical for production deployment where generating non-existent functions or invalid parameters could cause system failures.

5. DPO Challenges for Structured Output: SmolLM-360M’s performance degradation (30% $\rightarrow$ 8% exact match) after DPO reveals that preference optimization can conflict with structured output requirements:

- DPO encourages “helpful” outputs that may add extra (hallucinated) parameters
- Ranking-based objectives don’t enforce format constraints
- Future work needs structured DPO variants that penalize format violations

B. Computational Efficiency

Table IV compares computational requirements:

TABLE IV
Computational Efficiency Analysis

Model	**Trainable Params**	**Training Time (hrs)**	**Inference Time (ms)**
Llama-3.2-1B	8.7M (2.3%)	18	45
Qwen2.5-0.5B	4.2M (2.1%)	12	28
SmolLM-360M	3.1M (2.3%)	8	18
Phi-3-Medium	16.4M (0.7%)	42	156
Gemma-3-270M	2.8M (2.6%)	7	15

Efficiency Insights:

- **Training**: Small models train 3-6x faster than Phi-3-Medium, enabling rapid iteration
- **Inference**: 18-45ms for small models vs 156ms for Phi-3, enabling real-time applications
- **Memory**: SmolLM-360M fits in 1GB, enabling mobile deployment; Phi-3 requires 14GB
- **Cost**: At \$0.50/hr for A100, Llama-3.2-1B costs \$9 to train vs \$21 for Phi-3

C. Error Analysis

Manual inspection of 100 failure cases reveals common error patterns:

TABLE V
ERROR DISTRIBUTION ANALYSIS

Error Type	Frequency
Wrong argument name	35%
Type mismatch (str/int)	22%
Missing required argument	18%
Extra hallucinated argument	12%
Wrong function name	8%
JSON syntax error	5%

D. Limitations

Several limitations should be acknowledged:

1) **Dataset Scale**: Training on 10,000 examples may not capture the full diversity of real-world API interactions. Scaling to 100K+ examples could further improve performance.
2) **Evaluation Scope**: BFCL focuses on common API patterns; specialized domains (medical APIs, financial systems) may require additional domain-specific fine-tuning.
3) **DPO Configuration**: Our DPO experiments used simple preference pair generation. More sophisticated approaches (chain-of-thought preferences, curriculum DPO) may yield better results.
4) **Multilingual Support**: Evaluation was limited to English-language tool calling. Models like Qwen with strong multilingual pre-training may excel in non-English scenarios.
5) **Dynamic Tool Sets**: Models were trained on fixed tool sets. Real-world deployment requires handling new tools not seen during training (few-shot tool adaptation).
6) **Safety**: We did not evaluate adversarial robustness or jailbreaking attempts to make models call unauthorized functions.

E. Practical Implications

Our findings have significant implications for deploying AI systems:

- **Edge Deployment**: Models like Qwen2.5-0.5B (66% exact match, 28ms inference) can run on mobile devices (iPhone 15 Pro) while maintaining production-ready accuracy
- **Cost Reduction**: Using Llama-3.2-1B vs GPT-4 reduces inference cost by $\sim$100x (\$0.01 vs \$1.00 per 1K requests) while maintaining comparable performance for tool calling
- **Latency-Sensitive Applications**: Sub-50ms inference enables real-time applications (voice assistants, live chat support) where 200-500ms LLM latency is prohibitive
- **Privacy-Preserving AI**: Local deployment of small models addresses privacy concerns in sensitive domains (healthcare, finance) where sending data to cloud LLM APIs is unacceptable
- **Custom Toolsets**: Organizations can fine-tune SLMs on their proprietary APIs/tools, creating specialized assistants without exposing internal schemas to external LLM providers

VII. CONCLUSION AND FUTURE WORK

This work demonstrates that Parameter-Efficient Fine-Tuning enables Small Language Models to achieve competitive tool-calling performance while maintaining significant computational advantages. Our best model, Llama-3.2-1B, achieves 82% exact match and 87% F1 score with perfect tool selection, demonstrating that specialized fine-tuning can overcome inherent limitations of smaller models.

The results challenge the prevailing assumption that larger models are always superior, particularly for specialized tasks with well-defined output formats. With proper training, models as small as 500M parameters can achieve 66% exact match rates, making them viable for production deployment in resource-constrained environments.

Through detailed quantitative and qualitative analysis, we identified that:

- Tool selection (which function to call) is easier to learn than argument extraction
- Architecture quality and pre-training focus matter more than raw parameter count
- DPO can degrade performance on structured output tasks when not properly configured
- Token-level accuracy is a poor proxy for exact match in tool-calling scenarios

A. Future Directions

Several promising research directions emerge:

1) **Improved DPO Methods**: Developing structured preference optimization that maintains format constraints while improving accuracy. Possible approaches include format-aware DPO losses and curriculum DPO starting from exact matches.
2) **Few-Shot Tool Adaptation**: Enabling models to handle new tools with minimal examples. Current models require retraining for new function schemas; few-shot prompting with retrieved examples could generalize better.
3) **Multi-Task Learning**: Combining tool-calling with other capabilities (QA, summarization) to create more versatile yet efficient assistants. Current models specialize in tool-calling but lose general language understanding.
4) **Continuous Learning**: Enabling models to learn new tools without catastrophic forgetting of existing capabilities. Elastic Weight Consolidation or adapter-based approaches may help.
5) **Chain-of-Thought Tool Calling**: Incorporating reasoning steps before tool invocation to improve complex multi-step scenarios. Models could explain their function call decisions.
6) **Cross-Lingual Tool Calling**: Extending capabilities to multilingual scenarios where queries are in one language

but tool schemas are in another (e.g., Arabic query, English API).

7) **Adversarial Robustness**: Evaluating and improving resistance to prompt injection attacks attempting to make models call unauthorized or dangerous functions.
8) **Hybrid Approaches**: Combining small models for tool selection with retrieval systems for schema lookup, potentially achieving higher accuracy with lower latency.

B. Reproducibility and Open Source

To facilitate reproducibility and encourage further research, we commit to releasing:

- Fine-tuned model weights for all architectures on Hugging Face Hub
- Complete training scripts with detailed hyperparameters
- Evaluation code and BFCL test set predictions
- Preference pair generation code for DPO experiments
- Training logs and TensorBoard visualizations
- Inference examples and error analysis notebooks

Our work demonstrates that efficient, specialized AI systems are within reach, offering a path toward more sustainable and accessible AI deployment. The ability to achieve 82% exact match with 1B parameters opens doors for on-device AI, privacy-preserving applications, and cost-effective production systems that were previously infeasible with large LLMs.

Acknowledgments

We thank the faculty and staff at IIT Jodhpur for providing computational resources and guidance throughout this research. We acknowledge the open-source community for developing PyTorch, Hugging Face Transformers, PEFT library, and evaluation frameworks that made this work possible. Special thanks to the creators of API-Bank and BFCL datasets for enabling standardized evaluation of tool-calling capabilities.

References

[1] T. Schick et al., "Toolformer: Language Models Can Teach Themselves to Use Tools," *arXiv preprint arXiv:2302.04761*, 2023.
[2] S. Patil et al., "Gorilla: Large Language Model Connected with Massive APIs," *arXiv preprint arXiv:2305.15334*, 2023.
[3] MeetKai, "Functionary: Open-source function calling LLM," GitHub repository, 2023.
[4] M. Li et al., "API-Bank: A Comprehensive Benchmark for Tool-Augmented LLMs," *Proceedings of EMNLP*, 2023.
[5] E. J. Hu et al., "LoRA: Low-Rank Adaptation of Large Language Models," *ICLR*, 2022.
[6] T. Dettmers et al., "QLoRA: Efficient Finetuning of Quantized LLMs," *NeurIPS*, 2023.
[7] R. Rafailov et al., "Direct Preference Optimization: Your Language Model is Secretly a Reward Model," *NeurIPS*, 2023.
[8] L. B. Allal et al., "SmolLM: Small yet Mighty Language Models," *Hugging Face Technical Report*, 2024.
[9] J. Bai et al., "Qwen Technical Report," *arXiv preprint arXiv:2309.16609*, 2023.
[10] Microsoft, "Phi-3 Technical Report," *arXiv preprint arXiv:2404.14219*, 2024.