ToolReflection: Improving LLMs for Real-World API Calls with Self-Generated Data

Gregory Polyakov[1], **Ilseyar Alimova**[2], **Dmitry Abulkhanov** , **Ivan Sedykh**[4], **Andrey Bout**[5], **Sergey Nikolenko**[6], **Irina Piontkovskaya**[7]

[1]University of Tubingen, [2]Skoltech, [3]MTS AI, [4]Yandex, [5]Steklov Institute of Mathematics, [6]AI Foundation and Algorithm Lab

Abstract

Large language models increasingly call external tools and APIs to solve real tasks. However, real-world APIs are noisy, underspecified, and return diverse error messages. Benchmarks typically miss this complexity. We propose *ToolReflection*, a simple, general strategy: collect the models own failed API calls, pair them with real execution feedback and corrected calls, and fine-tune with these errorfeedbackfix triples. We also contribute realistic evaluation and data fixes across three popular benchmarks. *ToolReflection* yields substantial gains where execution feedback matters most: **+25.4 points (SR) on GPT4Tools-OOD, +56.2 on GPT4Tools-OOD-Hard, +14.0 accuracy on ToolAlpaca (Simulated), +10.5 on ToolAlpaca (Real), and +4 points SR on Multistep-100.**

Motivation

Tool-augmented LLMs often hallucinate tool names/params and fail to recover after API errors.

Real APIs return heterogeneous errors; docs are incomplete; success can require iterative fixing.

Existing benchmarks under-emphasize realistic, callable tools and multi-step dependencies.

Contributions

1. ***ToolReflection***: Instruction-tune with self-generated errorfeedbackfix examples to learn recovery from real API feedback.
2. **Benchmarking improvements**:
   GPT4Tools prompt reformatting to executable Python calls; **GPT4FakeTools** training add-on; new callable **GPT4Tools-OOD** and **GPT4Tools-OOD-Hard** test sets.
   ToolAlpaca evaluation fix: enable format checker; unify errors into HTTP-like messages.
   ToolBench cleaning with LLM-based error detection; new **Multistep-100** dataset.
3. **Consistent gains across settings** with the largest boosts where error recovery is essential.

Datasets at a Glance

Dataset	APIs	Train	Eval	Callable Eval
GPT4Tools	23	71.4K	1.8K	Limited (ours adds GPT4Tools-OOD/GPT4Tools-OOD-Hard)
ToolAlpaca	426	3.9K	200	Yes (small curated set)
ToolBench	16.5K	120K	6 splits	Mixed; LLM-eval originally

GPT4Tools: Python-call formatting to avoid parsing issues; **GPT4FakeTools** (+141 tools, 3,180 samples); **GPT4Tools-OOD/GPT4Tools-OOD-Hard** with five callable APIs and strict verification.

ToolAlpaca: Enable and standardize format checking at eval; translate checker messages to HTTP-style errors for training robustness.

ToolBench: LLM-based error filtering (LLaMA-2-7B-chat: Acc 0.89 ZS); **Multistep-100** for realistic multi-step dependencies and exact final-call checking.

Method: ToolReflection

Goal: Teach the model to *use real error messages* to fix its own API calls.

Core idea:

1. Run a tool-usage model on tool-requiring queries.
2. Collect cases with failed API calls and *real execution feedback* (Python/HTTP).
3. Build triples: (incorrect call, feedback, corrected call), with a short reflection step.
4. Fine-tune with these triples:
   Post-finetune (*PFT*): start from an instruction-tuned tool-usage model and continue on error-correction data (+ a small mix of original).
   From-scratch finetune (*FT*): augment original tool data with corrections and train anew.

Why it works:

Real APIs reveal *actionable* details (missing/typed params, formats, valid choices).

Learning to map feedback to fixes improves recovery across diverse APIs and formats.

Complementary to planners/trees: strengthens the *base* models single-call robustness.

Key Results

Out-of-Domain tool use (GPT4Tools)

Model / Training	GPT4Tools-OOD (SR_{last})	GPT4Tools-OOD-Hard (SR_{last})
GPT4Tools (orig.)	47.5	6.7
+ GPT4FakeTools	**72.9**	40.4
ToolReflection (PFT)	70.5	57.3
ToolReflection (FT)	**72.9**	**62.9**

Gains vs. GPT4Tools: +25.4 (GPT4Tools-OOD), +56.2 (GPT4Tools-OOD-Hard). Larger gap between first/last attempts indicates learned *recovery*.

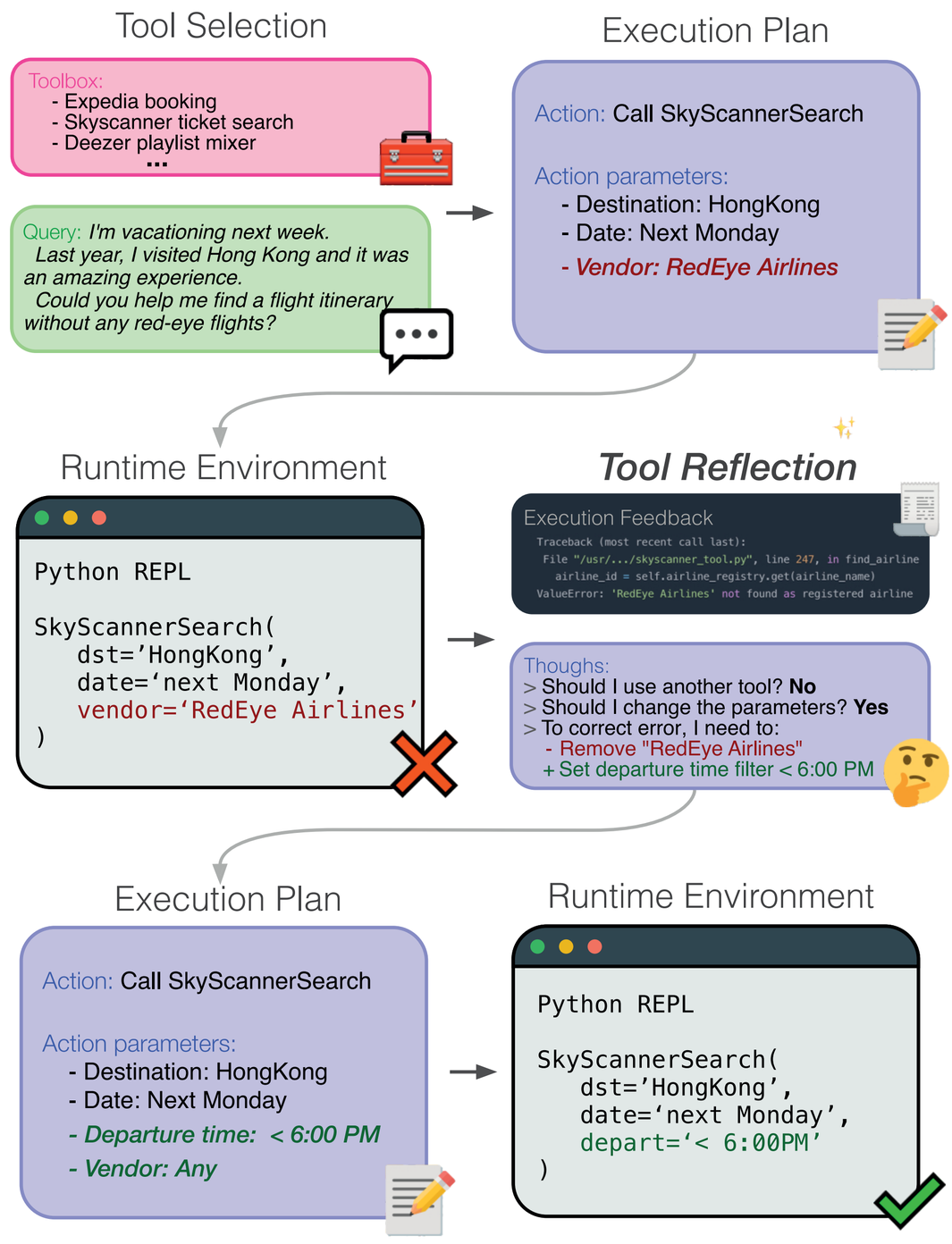

Figure 1: ToolReflection Overview: The model selects tools, generates API calls, processes feedback, and adjusts calls when errors occur.

Simulated and Real tools (ToolAlpaca)

Model / Training	Simulated Acc	Real Acc
Baseline (Train + Spec check)	42.0	35.1
ToolReflection (PFT, synth val)	49.0	43.0
ToolReflection (FT, synth train+val)	**56.0**	**45.6**

+14.0 (Sim), +10.5 (Real) absolute accuracy.

Diverse, multi-step APIs (ToolBench and Multistep-100)

Model / Training	Passes (ToolBench)	Wins (ToolBench)	SR (Multistep-100)
ToolLLaMA baseline	42.6	53.7	0.13
ToolReflection (PFT, synth val)	**44.3**	**53.8**	**0.17**

ToolReflection improves Pass rate (+1.7) and Multistep-100 (+0.04 SR) where error-driven recovery matters.

Error Correction in Action

Query: Convert 100 Pascals to atmospheres

Attempt 1: Incorrect Call

```
AI: pintapi_convert_units(
      from_value=100,
      from_unit="Pascals",
      to_unit="atmospheres"
    )
```

Observation: API Error

```
Pascals is not defined in the unit registry
```

Reflection: Model Thought

```
Thought: The tool returned an error because
Pascals is not a valid unit. The standard
abbreviation is likely Pa. I will try again.
```

Attempt 2: Correct Call

```
AI: pintapi_convert_units(
      from_value=100,
      from_unit="Pa",
      to_unit="atmospheres"
    )
```

Here, the model must convert pressure units. The API documentation is minimal, not specifying the exact unit format (e.g., Pa vs. Pascals). This example shows how ToolReflection enables the model to learn from API feedback, which is essential when documentation is incomplete.