
🔧

PromptBench

: Minimal Viable Spec (MVS) for LLMKit

✅ Purpose

To systematize and evaluate prompt performance on real-world dev use cases — like writing features, debugging code, or generating documentation — across multiple models, in a reusable, testable, reproducible pipeline.

📁 Module Path

```
llmkit/
├── bench/              # NEW: The core module
│   ├── profiles/       # Task definitions as YAML/JSON
│   ├── runs/           # Model responses from prompt tests
│   ├── evals/          # Human or model evaluations
│   ├── lib/            # Reusable prompt templates & fragments
│   └── __init__.py     # For internal Python module usage (optional)
```

🔨 Core Concepts

1.

PromptProfile

Defines a single prompt evaluation unit for a task.

```
id: bug_fix_001
title: Fix sorting bug in Python
task_type: bug_fix
description: A basic Python sort function returns incorrect results on duplicate elements.
input_code: |
  def sort(arr): ...
context: >
  This is part of a larger application used to sort user-generated data.
system_prompt: You are a senior Python engineer...
user_prompt: Please fix the sorting logic.
tags: [python, bug, dev, test]
```

Saved as: bench/profiles/bug_fix_001.yaml

2.

PromptRun

Captures metadata about a prompt run through a specific model.

```
{
  "run_id": "bug_fix_001_gpt-4",
  "prompt_id": "bug_fix_001",
  "model": "gpt-4",
  "response": "...LLM output...",
  "token_usage": {
    "prompt_tokens": 110,
    "completion_tokens": 89,
    "total_tokens": 199
  },
  "latency_seconds": 1.7,
  "timestamp": "2025-06-11T14:33:02"
}
```

Saved as: bench/runs/bug_fix_001_gpt-4.json

3.

PromptEval

Allows subjective evaluation of the LLM response — can be done manually or with an LLM.


```
{
  "run_id": "bug_fix_001_gpt-4",
  "correctness": 5,
  "clarity": 4,
  "coherence": 4,
  "usefulness": 5,
  "notes": "Correctly identified logic bug, added edge case handling."
}
```


Saved as: bench/evals/bug_fix_001_gpt-4_eval.json

🛠️ Core Commands (CLI MVP)

You can extend the current CLI or expose subcommands:

```
# Run a single prompt profile against a model
llmkit bench run bug_fix_001 --model gpt-4

# Evaluate the run (interactive or file input)
llmkit bench eval bug_fix_001_gpt-4

# Compare same task across models
llmkit bench compare bug_fix_001 --models gpt-4,claude-3,command-r

# List all available prompt profiles
llmkit bench list

# Create a new profile (interactive)
llmkit bench create
```

🧩 Prompt Template Library

Each task type (e.g. bug_fix, write_doc, feature_add) gets:

- A default system prompt template
- A user prompt scaffolding
- Optional “hints” or completions

This lives in:

```
bench/lib/
├── system/
│   └── bug_fix.txt
├── user/
│   └── bug_fix.jinja
```

💥 MVP Feature Set

Feature	Status
Task profiles	✅ Spec’d
Prompt run + response dump	✅ Spec’d
Evaluation schema	✅ Spec’d
CLI runners	MVP Defined
Prompt templating	Optional
Web view/export	Later

🔁 Dev Cycle (How to Work It)

- Start locally: Build and run a few profiles manually.
- Then: Drop in a PR draft once you hit stable CLI + one or two working flows.
- Label PR as feat: prompt bench module [draft] and include:
    - docs/bench.md explaining usage
    - bench/profiles/ with a few examples
    - bench/run.py or CLI wiring
    - bench/utils.py for loading profiles, sending prompts

🧠 Strategic Payoff

- Makes llmkit 10× more useful for any real developer using LLMs
- Creates reusable evaluation data for others in OSS
- Starts building a community-driven prompt eval dataset (future export: JSONL to HuggingFace or Weights & Biases)