llmtestgen – test specification generation from code and code specification

Cédric DERACHE, Arthur RICHELET – dec 9, 2025

Introduction

1. Problem overview
2. Project approach
3. Experiments & Results
4. Key findings & Conclusion

Introduction

- **Problem**: "Manual test specification is time consuming and error prone"
- **Solution**: "Automation using Large Language Models that can process large amount of data"
- **Objective**: "Build a toolchain capable of leveraging the capabilities of LLMs to reduce human effort and improve reliability and consistency"

Problem overview – Manual test writing

- **Manual test specifying:**
  - Time consuming & error prone
  - Hard to ensure complete and accurate coverage
  - Consistency problems between specs from multiple software engineers

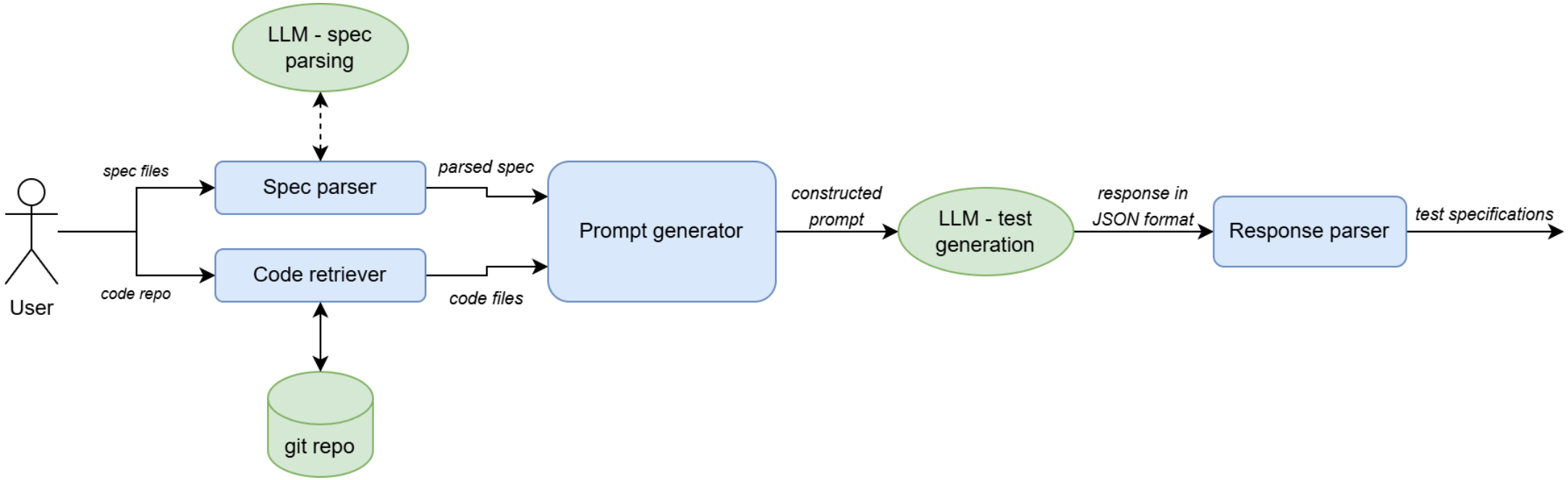
Problem overview – Why use LLMs

- **Why use Large Language Models**:
  - Able to handle large amounts of text data quickly
  - Very versatile, not bound to specific format
  - Automatable

Solution approach - Objectives

- **Objectives of the toolchain:**
  - Extract specs and put them in standard format for better comprehension by the LLM
  - Get relevant code files
  - Construct a prompt using elements
  - Parse the response into a file of test requirements

Solution approach – Pipeline



Solution approach – Prompt engineering

System prompt for arbitrary specification "parsing"

```python
def _build_system_prompt(self) -> str:
    """Explain how the LLM should behave and what to extract."""
    return (
        "You are a strict specification parser. "
        "Your task is to read input text and extract structured information.\n\n"
        "Return ONLY a JSON object with these exact fields:\n"
        "  - title: string or null\n"
        "  - sections: object mapping section names to text\n"
        "  - requirements: array of requirement-like sentences\n"
        "  - acceptance_criteria: array of acceptance criteria\n"
        "  - examples: array of example statements\n"
        "  - confidence: integer 0-100 expressing how confident you are "
        "    that you extracted the spec correctly\n\n"
        "DO NOT return explanations. DO NOT wrap the JSON in code fences. "
        "Only output raw JSON."
    )

def _build_user_prompt(self, text: str) -> str:
    return (
        "Parse the following specification file. Use your best judgment to identify:\n"
        "- title or heading\n"
        "- sections and subsections\n"
        "- requirement-like sentences (must, shall, should, cannot, etc.)\n"
        "- acceptance criteria (Given/When/Then patterns, etc.)\n"
        "- examples or usage patterns\n\n"
        "Return ONLY the JSON object described above.\n\n"
        "Here is the specification content:\n"
        "------------------\n"
        f"{text}\n"
        "------------------\n"
    )
```

Solution approach – Prompt engineering

System prompt for test specification generation

```
def _build_system_prompt(self) -> str:
    """Explain to the LLM how to behave and what JSON to return."""
    return (
        "You are an expert software test engineer. "
        "Your task is to design high-quality test cases for a Python project.\n\n"
        "Given a project specification and an optional view of the codebase, "
        "you must produce a structured list of test cases as JSON.\n\n"
        "Return ONLY a JSON object with this exact structure:\n"
        "{\n"
        '  "test_cases": [\n'
        "    {\n"
        '      "id": string or null,\n'
        '      "requirement": string or null,\n'
        '      "description": string,\n'
        '      "preconditions": [string, ...],\n'
        '      "steps": [string, ...],\n'
        '      "expected_result": string,\n'
        '      "target_code_elements": [string, ...]\n'
        "    },\n"
        "    ...\n"
        "  ]\n"
        "}\n\n"
        "Do not include explanations, comments, or additional fields. "
        "Do not wrap the JSON in code fences."
    )
```

Solution approach – Generated test case

Example of generated test case

Test 009: Deleting an existing task removes it from the service

Requirement: Task deletion

Preconditions

- TaskService contains at least one task

Target code elements

- `TaskService.delete_task`
- `TaskService.list_tasks`

Steps

1. Store the task ID.
2. Call `delete_task` with a valid ID.
3. Call `list_tasks()`.

Expected Result

- The deleted task no longer appears in the task list.

Experiment setup

- **Comparison of test generation context level:**
  - **None**, High-level tests, low code alignment.
  - **File List**: High requirement coverage, moderate alignment.
  - **Code snippets**: Best results, high code alignment and coverage.
  - **Full Code**: Failed due to excessive prompt size, (can work with a better model)

Experiment setup

Context Level	Requirement coverage	Code alignment	Validity of output
None	High	Low	Yes
File list	High	Medium	Yes
Code snippets	Very high	High	Yes
Full	N/A	N/A	*Invalid JSON output*

Key findings

- **Most accurate tests were generated using limited context sent to the LLM:**
  - Too much files overwhelmed the model
  - Reduced quality of invalid response
  - Moderate context seemed to lead better results
  - These limitations can be due to a limited model (*deepseek-chimera*, free API key from OpenRouter)

Conclusion

- Large Language Models can improve efficiency, coverage and consistency of test generation:
  - Generates tests cases autonomously
  - Reduces time and effort spent by human developer
- Further directions:
  - Improving prompt engineering to avoid alignment errors on less powerful models
  - Support for more LLMs endpoints
  - Automatic test writing based on given programming language
  - Automatic test runs and verification by secondary LLM

Thank you !

Questions ?