CSC 510 SE Proj1d1

Section 001 - Group 29
Aditya Deshpande, Shreeya Ranwadkar, Yash Dhavale, Ravi Goparaju

TiffinTrail — Smarter routes, Warmer meals.

Reflection on Using LLMs for Requirements Engineering

1) Pain Points in Using LLMs

Working with LLMs during the food delivery app project highlighted several pain points. First, **inconsistency of responses** was an issue—similar prompts sometimes led to very different outputs, requiring repeated iterations to stabilize. Second, **verbosity and lack of focus** in responses created extra work, as outputs often needed to be filtered and condensed before they could be used in our design documents. Third, **hallucination** was an obstacle, as the LLM sometimes invented features, workflows, or requirements that were not part of our intended scope. Finally, **prompt sensitivity** was a major pain point; minor rewordings of the same query could dramatically shift the output quality, making experimentation costly in terms of time.

2) Surprises

One surprise was that **different prompting strategies produced drastically different conclusions**. For example, zero-shot prompting often generated overly generic requirements, while few-shot prompting (with examples of how requirements should be phrased) produced far more structured and relevant outputs. Another surprise was how well the LLM could **bridge ambiguity**—when given underspecified features (e.g., "add payment system"), it suggested specific use cases, actors, and constraints that we had not considered. This was both helpful and occasionally misleading, as it risked drifting from our actual scope.

3) What Worked Best

The most effective strategies included:

- **Few-shot prompting** with tailored examples to guide the LLM toward producing requirements in a consistent style.
- **Pre-processing of prompts** by structuring them as templates (e.g., actor → goal → rationale) instead of free-text questions.
- **Using the LLM as a brainstorming partner** to amplify requirements (adding alternative flows, non-functional requirements, and edge cases we missed).

4) What Worked Worst

The least effective approaches included:

- **Zero-shot prompting**, which often returned vague, boilerplate outputs.
- **Asking broad, open-ended questions** ("What requirements should a food delivery app have?") that led to lists of features without context or prioritization.
- **Blind reliance on LLM-generated text**—when we skipped validation, the results included redundancies and contradictions.

5) Useful Pre- and Post-Processing

Pre-processing prompts by **framing them in RE structures** (e.g., “List functional requirements in the form: The system shall…”) was very helpful. On the post-processing side, **summarization and filtering** were necessary to reduce the noise in the LLM’s verbose responses. We often combined outputs from multiple prompts and then distilled them into a clean, human-readable requirements list. In some cases, **graphical visualization** of LLM outputs (e.g., actor–goal diagrams) helped us verify completeness and consistency.

6) Best/Worst Prompting Strategies

The best prompting strategies involved **explicit role assignment** (“You are a requirements engineer helping refine requirements…”) combined with **structured instructions** (“Generate 5 functional and 5 non-functional requirements, avoid repetition”). Iterative prompting—feeding the LLM’s output back in with refinement instructions—also improved results. Worst strategies were **overly vague prompts** and **attempts to cover too many tasks in one query**, which led to incoherent outputs.

7) Looking Ahead: Tools Needed for RE with LLMs

Based on these lessons, a startup aiming to build LLM-based tools for Requirements Engineering should provide:

1. **Prompt templates and refinement pipelines** that enforce structured input and output formats for consistency.

2. **Validation and deduplication tools** to filter out contradictions, redundancies, and hallucinations in LLM-generated requirements.

3. **Visualization modules** (e.g., automatically generating UML diagrams, use-case maps) from textual outputs.

4. **Amplification support**, where LLMs propose edge cases, non-functional requirements, and “what-if” scenarios.

5. **Condensation support**, where verbose outputs are distilled into concise, stakeholder-friendly summaries.

6. **Traceability tools** to connect LLM-generated requirements back to source prompts and decisions, ensuring accountability.

7. **Collaboration features** that let human stakeholders interact with the LLM iteratively rather than treating it as a one-shot generator.

Conclusion

From our proj1 experience, LLMs are valuable assistants but cannot yet replace human judgment in requirements engineering. They amplify creativity and help uncover hidden requirements, but they also introduce noise, inconsistency, and scope drift. With proper tools for structuring, validating, and refining outputs, however, LLMs can become powerful subroutines for requirement amplification and condensation. Our proj2 and proj3 explorations should focus on designing these surrounding tools to bridge the gap between raw LLM capability and the rigor needed in software engineering practice.