Reflection on Using LLMs for Requirements Engineering

Authors: Robert Kemp, Matthew Nguyen, Tiehang Zhang, Rishi Jeswani

Lessons Learned

Working with large language models (LLMs) for requirements engineering has been both eye-opening and challenging. The experience revealed clear strengths of these tools but also exposed pain points that limit their direct usability without additional support systems. One of the most important lessons learned is that while LLMs can generate content quickly, the reliability and scalability of their outputs remain inconsistent. For example, during the creation of 30 use cases, we discovered that the models could not reliably handle large outputs in a single request. Instead, we had to process them in batches of five, which created overhead and made the workflow more fragmented. Similarly, models would occasionally generate implausible use cases, such as an "emergency response coordination" scenario that was completely outside the project's scope. These experiences underscored the importance of human oversight and the necessity for tools that can standardize, filter, and validate LLM outputs. Another lesson was the variability in LLM behavior across different providers. ChatGPT proved particularly strong at formatting outputs and following detailed instructions, while Claude was better at inference tasks and capturing nuanced meaning. However, both had drawbacks: ChatGPT occasionally hallucinated details or gave outdated information about its own API, while Claude sometimes produced overly verbose or abstract content. These differences demonstrated that no single LLM could cover all requirements, and leveraging multiple models in tandem was the most effective strategy. Finally, prompt engineering emerged as a central skill. Zero-shot prompting worked well for brainstorming broad ideas but was ineffective when narrowing the scope to specific requirements. In contrast, structured prompts that provided context, templates, and examples consistently yielded higher-quality results. The need to iterate and refine prompts highlighted that using LLMs is less about asking a single "smart question" and more about designing a structured workflow of questions, checks, and refinements.

Pain Points

The pain points clustered around four themes:

1. **Scalability of Output** – LLMs struggled to generate large batches of structured content without either truncating or introducing errors.
2. **Verification Overhead** – Every output had to be checked carefully. The risk of errors or "confidently wrong" answers meant we could not trust the LLM without review.
3. **Inconsistent Standardization** – Different prompts, or even the same prompt asked at different times, often led to variations in structure. This made it difficult to align outputs into a consistent final document.
4. **Cost and Efficiency** – Finding a sustainable balance between model accuracy, prompt iterations, and API usage cost proved to be non-trivial.

These pain points illustrate why LLMs alone are insufficient for production-level requirements engineering.

What Worked Best and Worst

The most effective strategy was combining multiple LLMs and leveraging their complementary strengths. Having ChatGPT handle structured formatting while Claude worked on inference created better results than relying on one model. Another success factor was breaking tasks into smaller, structured prompts and gradually building toward the final deliverable.

Conversely, the least effective strategy was asking for large-scale tasks in a single step. Models either failed to complete the task or introduced errors that required even more rework. Similarly, when prompts lacked sufficient context, the LLMs would often hallucinate, make assumptions, or “double down” on incorrect interpretations.

Pre- and Post-Processing Value

Pre-processing data and carefully structuring inputs significantly improved results. For instance, arranging files in a consistent tree structure before importing them into a retrieval-augmented generation (RAG) pipeline made it easier for the LLM to parse and summarize them.
Post-processing, especially feeding outputs back into ChatGPT for formatting, was also highly beneficial. This layering allowed us to generate structured use cases that looked polished while maintaining human control over accuracy.

Surprises

One surprise was how differently models behaved despite appearing to solve similar tasks. Claude and ChatGPT reached different conclusions when interpreting requirements, showing that model design and training data create non-trivial divergences. Another surprise was discovering that ChatGPT-4 itself was not always aware of updates to its own API, which reinforced the importance of verifying information independently rather than relying solely on the LLM’s “knowledge.”

Designing Tools for the Future

Based on these experiences, if I were to design a startup focused on tools for requirements engineering with LLMs, I would target the pain points directly. Such a startup could provide:

1. **Output Standardization Layer** – A tool that enforces consistent formatting and structure across LLM outputs, regardless of the model used.
2. **Validation and Filtering Engine** – An AI-assisted system that checks generated requirements against project constraints to prevent “outrageous” or irrelevant suggestions.
3. **Batching and Chunking Manager** – Automated splitting of large tasks into smaller manageable batches and recombining them into cohesive final documents.
4. **Multi-Model Orchestration** – A framework that routes subtasks to the model best suited for them (e.g., Claude for reasoning, GPT for formatting).
5. **Prompt Engineering Toolkit** – Libraries of templates, context-builders, and best practices for RE-specific prompts.
6. **Cost-Aware API Optimizer** – A tool that balances model usage, price, and accuracy to help teams make informed tradeoffs.

By Project 2 and 3, these tools would mature into a requirements engineering platform where LLMs act as subroutines within a controlled ecosystem. Instead of relying on an LLM to “do everything,” they would become specialized assistants embedded within a larger workflow that ensures consistency, accuracy, and efficiency.

Conclusion

The experience with LLMs revealed both promise and limitations. They can accelerate brainstorming, formatting, and structuring tasks, but they cannot yet be trusted as autonomous requirement engineers. With appropriate supporting tools—particularly in output standardization, validation, orchestration, and cost management—LLMs could become powerful allies in requirements engineering. The next step is not to replace human judgment but to embed LLMs into carefully designed pipelines that harness their strengths while compensating for their weaknesses.