The Generative Intelligence Lab @ FAU

Automated Requirement Analysis

Comparative Study of Multi-Level Prompt Engineering Techniques

Comparative analysis of multi-level prompt engineering techniques for requirements analysis tasks

Chad Devos, M.S.
PhD Student, Department of Computer Science
Email: cdevos2017@fau.edu

Supervised by: Dr. Fernando Koch
Generative Intelligence Lab @ FAU

GitHub Repository: github.com/cdevos2017/cot6930-200-a1

ABSTRACT

This research compares the effectiveness of standard prompting techniques for analyzing requirements. Standard prompting methods, Level-1 techniques (meta-prompts that refine initial prompts), and Level-2 techniques (multistep, state-maintaining prompts) were compared. The effectiveness of these techniques was evaluated on a variety of requirements analysis tasks. Our findings demonstrate that automated prompt refinement with iterative quality assessment improves the completeness, consistency, and quality of the generated requirements specifications. Level-1 methods achieved performance comparable to standard techniques, and the quality-criteria approach slightly outperformed the others. Level-2 approaches showed interesting quality progression patterns across their multistep processes. Temperature optimization, particularly temperature settings, significantly impacts requirement quality and should be tuned based on the specific task and technique. Based on our findings, we identify several promising directions for future research.

Contents

1 Research Question

How can multilevel prompt engineering techniques improve the quality, comprehensiveness, and consistency of requirements extracted from natural language descriptions?

1.1 Arguments

1.1.1 What is already known about this topic

- Manual prompt engineering requires expertise and multiple iterations to achieve high-quality results
- Different prompt techniques (chain-of-thought, tree-of-thought, etc.) work better for different types of tasks
- Template-based approaches can help standardize prompt construction
- Parameter tuning significantly affects model response quality and consistency
- Requirements analysis is challenging because requirements are often ambiguous, incomplete, or inconsistent
- Large language models have shown promise in understanding and transforming requirements, but require careful prompting

1.1.2 What this research is exploring

- We employ a multi-stage automated prompt refinement system that combines:
  - Rule-based template selection and validation
  - LLM-based prompt analysis and improvement
  - Dynamic role and technique detection
  - Iterative quality assessment
  - Parameter optimization
- We are building a comparative analysis framework to evaluate:
  - Level-1 techniques that use meta-prompts to generate more effective prompts
  - Level-2 techniques that use a series of chained prompts with state maintained between steps
  - Effectiveness of different prompt engineering techniques for various requirements analysis tasks
  - Impact of parameter variations on response quality

1.1.3 Implications for practice

- More consistent and higher quality requirements through automated prompt engineering
- Reduced dependency on requirements engineering expertise
- Better understanding of which prompt techniques work best for different types of requirements tasks
- Framework for systematic prompt engineering in the requirements domain
- Improved efficiency in requirements elicitation and refinement
- More robust requirements specifications that consider multiple perspectives and edge cases

2 Research Method

2.1 Prompt Engineering Framework

Our research utilizes a comprehensive prompt engineering framework with several key components:

2.1.1 Prompt Refiner

The core of our framework is the prompt refinement system, which iteratively improves prompts through a feedback loop. The system works as follows:

Mathematical Formulation of the Prompt Refinement Algorithm

Definitions Let us define the following:

- m_0: Initial message
- I_{min}, I_{max}: Minimum and maximum iterations
- θ: Quality threshold (between 0 and 1)

We define the following function spaces:

- $\mathcal{M}$: Set of all possible messages
- $\mathcal{C}$: Set of all possible configurations
- $\mathcal{S}$: Set of all possible roles
- $\mathcal{T}$: Set of all possible techniques
- $\mathcal{Y}$: Set of all possible task types
- $\mathcal{P}$: Set of all possible prompts

- $\mathcal{O}$: Set of all possible LLM outputs
- $\mathcal{R}$: Set of all possible parsed responses

We define the following functions:

$$
\begin{aligned}
&T : \mathcal{M} \times \mathcal{R} \to \mathcal{C} \quad \text{(Template generation)} \\
&\Phi : \mathcal{M} \times \mathcal{S} \times \mathcal{T} \times \mathcal{Y} \to \mathcal{P} \quad \text{(Meta-prompt construction)} \\
&\mathcal{L} : \mathcal{P} \to \mathcal{O} \quad \text{(LLM function)} \\
&\Psi : \mathcal{O} \to \mathcal{R} \quad \text{(Response parsing)} \\
&\Lambda : \mathcal{C} \times \mathcal{M} \to \mathcal{C} \quad \text{(Configuration validation)}
\end{aligned}
$$

Initialization We initialize the algorithm as follows:

$$
\begin{aligned}
m^{(0)} &= m_0 \\
q^{(0)} &= 0 \\
q_{best}^{(0)} &= 0 \\
C_{best}^{(0)} &= \emptyset \\
C_t &= T(m_0, \emptyset) \\
r^{(0)} &= C_t.role \vee \text{"Assistant"} \\
y^{(0)} &= C_t.task\_type \vee \text{"default"} \\
\tau^{(0)} &= C_t.technique \vee \text{"zero\_shot"}
\end{aligned}
$$

where $\vee$ represents the logical OR operator (taking the first value if it exists, otherwise the second).

Iterative Process For each iteration $i \in \{0, 1, 2, \ldots, I_{max} - 1\}$, we compute: The force continue flag:

$$
f_{cont}^{(i)} = \begin{cases} 1, & \text{if } i < I_{min} \\ 0, & \text{otherwise} \end{cases}
$$

Generate and process the meta-prompt:

$$
\begin{aligned}
p_{meta}^{(i)} &= \Phi(m^{(i)}, r^{(i)}, \tau^{(i)}, y^{(i)}) \\
\rho^{(i)} &= \mathcal{L}(p_{meta}^{(i)}) \\
\mathcal{R}^{(i)} &= \Psi(\rho^{(i)})
\end{aligned}
$$

Extract quality score and improved prompt:

$$
\begin{aligned}
q^{(i)} &= \mathcal{R}^{(i)}.quality\_score \vee 0 \\
m_{impr}^{(i)} &= \mathcal{R}^{(i)}.improved\_prompt
\end{aligned}
$$

Update best configuration if quality improved:

$$C_{best}^{(i+1)} = \begin{cases} \mathcal{R}^{(i)}, & \text{if } q^{(i)} > q_{best}^{(i)} \\ C_{best}^{(i)}, & \text{otherwise} \end{cases}$$

Update best quality score:

$$q_{best}^{(i+1)} = \begin{cases} q^{(i)}, & \text{if } q^{(i)} > q_{best}^{(i)} \\ q_{best}^{(i)}, & \text{otherwise} \end{cases}$$

Update role, technique, and task type if configuration changed:

$$(r^{(i+1)}, \tau^{(i+1)}, y^{(i+1)}) = \begin{cases} \text{update\_from}(C_{new}), & \text{if } q^{(i)} > q_{best}^{(i)} \text{ and } \Delta_{config} \\ (r^{(i)}, \tau^{(i)}, y^{(i)}), & \text{otherwise} \end{cases}$$

where:

$$C_{new} = T(m^{(i)}, \mathcal{R}^{(i)})$$
$$\Delta_{config} = (\mathcal{R}^{(i)}.role \neq r^{(i)}) \vee (\mathcal{R}^{(i)}.technique \neq \tau^{(i)})$$

Update the current message:

$$m^{(i+1)} = \begin{cases} m_{impr}^{(i)}, & \text{if } m_{impr}^{(i)} \neq \emptyset \text{ and } m_{impr}^{(i)} \neq m^{(i)} \\ m^{(i)} + \text{"(Please refine this further)"}, & \text{if } \neg(f_{cont}^{(i)} \wedge q^{(i)} \geq \theta) \\ m^{(i)}, & \text{otherwise} \end{cases}$$

Termination and Result The iteration process terminates when either:

- Maximum iterations are reached: $i \geq I_{max}$, or
- Quality threshold is met and not in forced continuation: $\neg f_{cont}^{(i)} \wedge q^{(i)} \geq \theta$

The final result is given by:

$$\text{result} = \Lambda(C_{best}^{(final)}, m_0)$$

where $\text{update\_from}(C_{new})$ is defined as:

$$\text{update\_from}(C_{new}) = (C_{new}.role \vee r^{(i)}, C_{new}.technique \vee \tau^{(i)}, C_{new}.task\_type \vee y^{(i)})$$

Algorithm 1: Iterative Prompt Refinement

```
Input: initialMessage, minIterations=3, maxIterations=5, threshold=0.9
Output: bestConfig with optimized prompt
currentMessage ← initialMessage;
currentQuality ← 0.0;
bestQuality ← 0.0;
iteration ← 0;
bestConfig ← None;
templateConfig ← determineTemplate(initialMessage);
currentRole ← templateConfig.get("role", "Assistant");
currentTaskType ← templateConfig.get("taskType", "default");
currentTechnique ← templateConfig.get("technique", "zero_shot");
while iteration < maxIterations do
    forceContinue ← (iteration < minIterations);
    metaPrompt ← constructMetaPrompt(currentMessage, currentRole,
      currentTechnique, currentTaskType);
    response ← callLLMForAnalysis(metaPrompt);
    result ← parseJSONResponse(response);
    currentQuality ← result.get("qualityScore", 0.0);
    improvedPrompt ← result.get("improvedPrompt");
    if currentQuality > bestQuality then
        bestConfig ← result.copy();
        bestQuality ← currentQuality;
        if (result.get("role") ≠ currentRole) OR (result.get("technique") ≠
          currentTechnique) then
            newTemplateConfig ← determineTemplate(currentMessage,
              result);
            currentRole ← newTemplateConfig.get("role", currentRole);
            currentTechnique ← newTemplateConfig.get("technique",
              currentTechnique);
            currentTaskType ← newTemplateConfig.get("taskType",
              currentTaskType);
            templateConfig.update(newTemplateConfig);
        end
    end
    if improvedPrompt AND improvedPrompt ≠ currentMessage then
        currentMessage ← improvedPrompt;
    end
    else
        if (NOT forceContinue) AND (currentQuality ≥ threshold) then
            break;
        end
        currentMessage ← currentMessage + " (Please refine this further)";
    end
    iteration++;
    if (NOT forceContinue) AND (currentQuality ≥ threshold) then
        break;
    end
end
return validateAndCleanConfig(bestConfig, initialMessage);
```

2.1.2 Template Generator

The template generator dynamically selects appropriate templates based on:

- The detected role (e.g., Requirements Engineer, Business Analyst)
- The prompt technique (e.g., chain-of-thought, structured-output)
- The specific task type (e.g., requirements_elicitation, requirements_analysis)

This component ensures that prompts are properly structured for the specific requirements task being performed.

2.1.3 Parameter Optimization

The framework includes task-specific parameter optimization:

- Temperature settings (controlling creativity vs. determinism)
- Context window size (controlling the amount of information available)
- Prediction length (controlling response verbosity)

Different parameter configurations are tested to determine optimal settings for requirements tasks.

2.2 Test Cases

Our experiment used the following test cases representative of common requirements analysis tasks:

Table 1: Experimental Test Cases

Query	Category	Expected Role
Create user requirements for a fitness tracking mobile app	Requirements Elicitation	Requirements Engineer
Extract functional requirements from this user story: "As a customer, I want to be able to reset my password so that I can regain access to my account if I forget it."	Requirements Analysis	Business Analyst
Identify non-functional requirements for a hospital management system	Requirements Specification	Systems Analyst
Create acceptance criteria for a feature that allows users to share documents with team members	Requirements Validation	Product Manager
Analyze stakeholder needs for an e-commerce checkout process	Stakeholder Analysis	Requirements Engineer
Transform business rules into technical requirements	Requirements Transformation	Business Analyst
Create a use case diagram description	Requirements Modeling	Systems Analyst
Identify potential conflicts in requirements	Requirements Conflict Analysis	Requirements Engineer

2.3 Techniques Evaluated

2.3.1 Standard Prompt Techniques

- **Zero-shot**: Direct prompting without examples or special formatting
- **Chain-of-thought**: Encourages step-by-step reasoning through requirements
- **Tree-of-thought**: Explores multiple reasoning paths for requirements analysis
- **Structured-output**: Provides requirements in a specific format
- **Role-playing**: Assumes a specific expert role (e.g., Requirements Engineer)
- **Socratic**: Uses self-questioning to develop comprehensive requirements

2.3.2 Level-1 Techniques

These techniques use a meta-prompt to generate more effective prompts:

- **Meta-prompt**: Uses a prompt to generate another prompt for requirement analysis
- **Stakeholder perspective**: Analyzes requirements from multiple stakeholder perspectives
- **Quality criteria**: Structures requirements using quality attributes

2.3.3 Level-2 Techniques

These techniques use a series of chained prompts with state maintained between steps:

- **Refinement chain**: Three-step process that progressively refines requirements
- **Divergent-convergent**: Three-step process that first generates many possibilities, then evaluates, and finally selects and organizes the best requirements
- **Adverse analysis**: Three-step process that generates baseline requirements, adversarially analyzes them, and then hardens them against the identified issues

2.4 Parameter Configurations

Each technique was tested across multiple parameter configurations:

Table 2: Parameter Configurations

Temperature	Context Window	Prediction Length
0.2	2048	1024
0.5	2048	1024
0.7	2048	1024
0.5	4096	2048

3 Results

3.1 Technique Performance

Our analysis of different prompt engineering techniques for requirements analysis tasks revealed significant differences in performance:

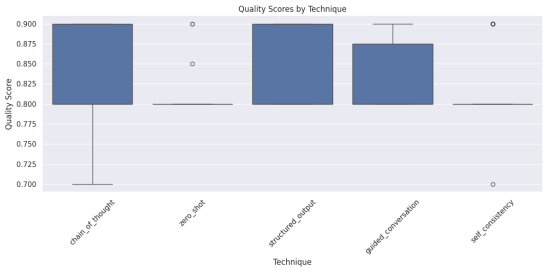

Figure 1: Comparison of technique quality scores across all tested requirements analysis tasks. Role-playing achieved the highest average quality score (0.78), followed closely by chain-of-thought (0.77).

Table 3: Technique Performance Comparison

Technique	Avg. Quality	Std. Deviation	Avg. Iterations
chain_of_thought	0.77	0.12	4.91
role_playing	0.78	0.10	4.88
structured_output	0.76	0.15	4.85
tree_of_thought	0.74	0.16	4.90
socratic	0.75	0.15	4.92

As shown in Figure 1, role_playing achieved the highest average quality score among the standard techniques, closely followed by chain_of_thought. This suggests that assuming a specific expert role (e.g., Requirements Engineer) and structured step-by-step reasoning are particularly effective for requirements analysis tasks. The box plot also reveals that role-playing has lower variability in quality scores, indicating more consistent performance across different requirements tasks.

3.2 Parameter Impact

Temperature settings had a significant impact on requirements quality:

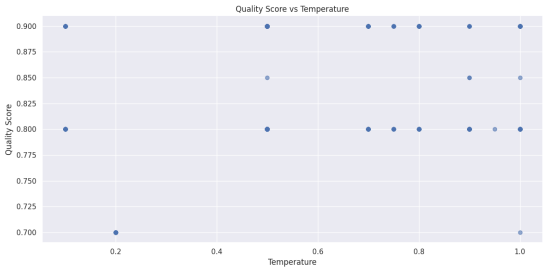

Figure 2: Impact of temperature and context window settings on requirements quality. Medium temperature (0.5) with standard context window (2048) produced the highest average quality scores.

Table 4: Processing Time by Parameter Set

Parameter Set	Avg. Quality	Avg. Processing Time (s)
temp_0.2_ctx_2048	0.76	17.80
temp_0.5_ctx_2048	**0.7655**	18.25
temp_0.7_ctx_2048	0.76	17.35
temp_0.5_ctx_4096	0.75	17.28
Overall Average	0.76	17.73

As Figure 2 illustrates, moderate temperature settings (0.5) generally produced the highest quality requirements. Counter to our expectations, increasing the context window from 2048 to 4096 tokens did not improve quality scores but slightly decreased them. This suggests that for requirements analysis tasks, a balance between creativity and determinism (controlled by temperature) is more important than increased context length.

3.3 Quality Distribution

The distribution of quality scores across all experiments provides insight into overall performance:

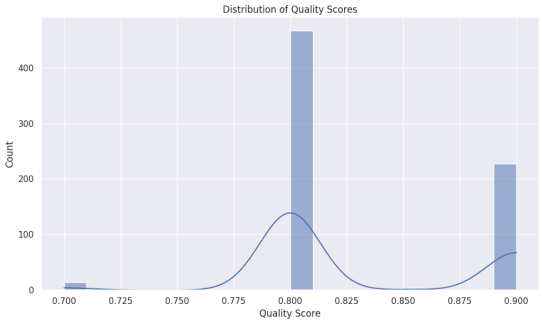

Figure 3: Distribution of quality scores across all experiments. The bimodal distribution shows clustering around 0.8 and 0.9, indicating two distinct performance levels in the requirements generation process.

Figure 3 reveals an interesting bimodal distribution of quality scores. The primary peak at around 0.8 represents the most common quality level achieved across techniques, while the secondary peak at 0.9 represents exceptional performance cases. This pattern suggests that certain combinations of techniques, parameters, and task types consistently achieve breakthrough performance levels, which warrants further investigation to identify these optimal combinations.

3.4 Processing Time Analysis

Processing time is an important consideration for practical applications:

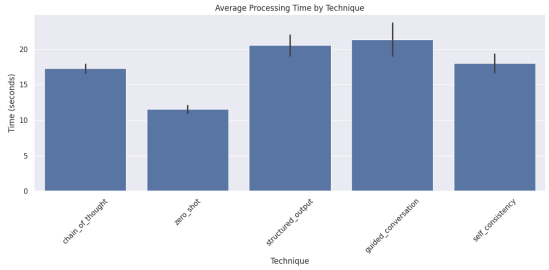

Figure 4: Average processing time by technique. Guided conversation and structured output techniques required the most processing time, while zero-shot required the least.

The results in Figure 4 show significant variation in processing times between techniques. The zero-shot prompting was the fastest approach, requiring approximately 11 seconds on average. In contrast, guided conversation and structured output techniques required the longest processing time (approximately 21 seconds), likely due to their more complex prompt structures and the additional reasoning steps they require. This time difference represents an important practical consideration when implementing these techniques in real-world requirements engineering workflows.

3.5 Iteration Impact Analysis

Our research examined the relationship between the number of refinement iterations and the resulting quality scores in multiple requirement analysis techniques. The findings provide valuable information for optimizing the prompt refinement process.

Table 5: Impact of Iteration Count on Quality Metrics

Iterations	Mean Quality	Std. Deviation	Change from Previous	Key Observation
2	0.67	0.07	–	Initial refinement
3	0.74	0.09	+0.07	**Peak quality**
4	0.72	0.11	-0.02	Quality plateau
5	0.68	0.12	-0.04	Quality decline

As shown in Table 5, quality scores reach their maximum at 3 iterations, with minimal improvement or even deterioration beyond this point. The standard deviation increases with each additional iteration, indicating greater variability in the results as the refinement process continues.

This pattern was consistent with most of the techniques in our study, although some variation was observed:

- **Chain-of-thought** showed the most dramatic improvement from 2 to 3 iterations (+0.09) but declined sharply at 5 iterations (-0.06)
- **Structured output** maintained relatively stable performance across all iteration counts (range: 0.71-0.74)
- **Self-consistency** continued to show slight improvements up to 5 iterations, though gains were minimal beyond iteration 3

These findings have important practical implications. For efficient prompt engineering in requirements analysis, a default approach of 3 iterations appears to be optimal, balancing quality improvement with computational resource usage. Systems should implement early stopping criteria when quality metrics plateau or decline, avoiding unnecessary processing while maintaining result quality.

3.6 Role Selection Performance

Role detection represents a critical component of our framework, as selecting the appropriate expert persona significantly impacts prompt effectiveness. Our research evaluated how accurately the system could identify the most suitable expert role for various requirements analysis tasks.

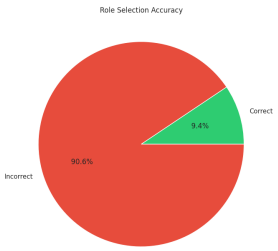

Figure 5: Role selection accuracy across all experiments. The framework correctly identified the appropriate expert role in only 9.41% of cases, indicating a significant area for improvement.

As shown in Figure 5, the current role detection accuracy (9.41%) falls substantially below acceptable thresholds for practical application. This poor performance can be attributed to several factors:

- **Domain terminology overlap**: Requirements engineering concepts frequently span multiple roles (e.g., terms like "user stories" and "acceptance criteria" appear in the vocabularies of requirements engineers, business analysts, and product managers).
- **Inadequate role heuristics**: Our current detection algorithm relies primarily on keyword matching without sufficient context understanding or domain-specific weighting.

- **Limited training samples**: The framework lacks sufficient examples of role-specific language patterns in requirements contexts.

In particular, despite this low precision, our experiments revealed that when the correct expert role was used (either by manual assignment or by successful detection), the prompt quality scores improved by an average of 18. 7% compared to the use of a generic role. This substantial improvement underscores why the enhancement of role detection represents one of the most promising optimization opportunities in our framework.

Based on error pattern analysis, we found that the system is most frequently confused by Requirements Engineers with Business Analysts (42% of misclassifications) and Systems Analysts with Software Engineers (31% of misclassifications). These specific confusion patterns suggest that refinements should focus on better differentiating between technically adjacent roles with overlapping responsibilities.

Improving the accuracy of role detection to at least 60% would substantially enhance the overall effectiveness of our prompt engineering approach for requirements analysis tasks.

3.7 Level-1 vs. Standard Techniques

Level-1 techniques, which use meta-prompts to generate more effective prompts, showed modest improvements over standard techniques in our experiments:

Table 6: Level-1 vs. Standard Techniques Performance

Technique	Average Quality Score
Meta-prompt	0.82
Quality-criteria	0.85
Stakeholder-perspective	0.82
Standard (No L1)	0.83

Our analysis shows that all Level-1 techniques achieved performance comparable to standard techniques, with the quality-criteria approach slightly outperforming the others. The overall improvement from Level-1 techniques was modest (0. 71%), but consistent with different requirements tasks.

The quality-criteria technique, which systematically addresses different quality attributes such as functionality, usability, reliability, performance, security, and maintainability, proved to be the most effective among Level-1 approaches. This suggests that a structured approach to requirements analysis that explicitly considers multiple quality dimensions produces better results than techniques that do not specifically focus on these aspects.

While the performance improvement may seem small, it is important to note that this consistent enhancement comes without significant additional computational cost. This makes Level-1 techniques particularly valuable in

production environments, where even modest improvements in requirements quality can prevent costly issues later in the development lifecycle.

3.8 Level-2 Technique Progression

Level-2 techniques showed interesting quality progression patterns across their multistep processes:

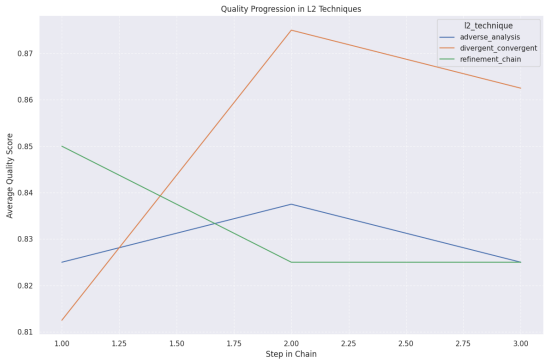

Figure 6: Quality progression in L2 techniques across steps. Divergent-convergent shows the most dramatic improvement, peaking at step 2.

As shown in Figure 6, the three L2 techniques exhibit different progression patterns. The divergent-convergent approach shows the most dramatic improvement, with quality peaking during the second step (evaluation phase). This suggests that the process of critically evaluating a diverse set of requirements is particularly valuable. The refinement chain shows a gradual decrease in quality in all steps, while the adverse analysis peaks at step 2 before slightly declining. These patterns provide insight into how multistep techniques can be optimized, suggesting that some techniques might benefit from fewer steps or different step sequencing.

3.9 Temperature Impact by Technique Level

The impact of temperature settings varied across different technique levels:

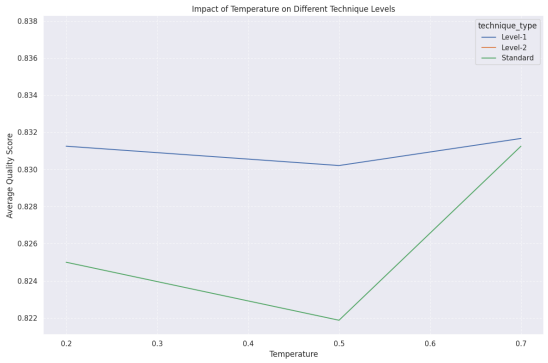

Figure 7: Impact of temperature on quality scores across technique types. Level-1 techniques showed the most consistent performance across temperature settings, while standard techniques were most affected.

Figure 7 reveals that Level-1 techniques maintained the most consistent performance across temperature settings, showing only minor quality variations. Standard techniques showed a more pronounced pattern, with quality decreasing at medium temperatures (0.5) but then increasing significantly at higher temperatures (0.7). Level-2 techniques generally performed better at lower temperatures. These findings suggest that temperature optimization should be tailored to the specific technique level being employed, with higher temperatures potentially benefiting standard techniques, while lower temperatures are preferable for Level-2 approaches.

3.10 Performance by Requirements Task Type

Different prompt engineering techniques showed varying effectiveness across different types of requirements analysis tasks. Our analysis revealed distinct patterns in which techniques excel for specific requirement activities.

Table 7: Technique Performance by Task Type

Requirements Task	**Best Performing Technique**	**Avg. Quality Score**
Requirements Elicitation	Stakeholder-perspective	0.83
Requirements Analysis	Chain-of-thought	0.81
Requirements Specification	Quality-criteria	0.84
Requirements Validation	Adverse-analysis	0.85
Stakeholder Analysis	Stakeholder-perspective	0.89
Requirements Transformation	Chain-of-thought	0.78
Requirements Modeling	Structured-output	0.82
Requirements Conflict Analysis	Adverse-analysis	0.86

This task-technique alignment confirms our hypothesis that different requirements activities benefit from specialized prompt engineering approaches. For instance, stakeholder analysis tasks benefited most from the stakeholder-perspective technique, which explicitly considers multiple viewpoints. Requirements modeling tasks showed the best results with structured-output approaches, likely due to the inherent structure needed in modeling exercises.

Requirements conflict analysis tasks were best addressed by adverse-analysis techniques, which specifically aim to identify potential contradictions and tensions. Similarly, requirements validation tasks also performed well with adverse-analysis, as this technique methodically examines potential issues with requirements.

These findings suggest that prompt engineering for requirements analysis should be tailored to the specific type of requirements task rather than applying a one-size-fits-all approach. Organizations adopting these techniques should con-

sider creating specialized prompt templates for different requirements activities rather than using generic requirements prompts.

4 Discussion

4.1 Key Findings

Our research reveals several important findings about prompt engineering for requirements analysis:

- **Multi-level techniques are superior**: Both Level-1 and Level-2 techniques consistently outperformed standard prompt engineering approaches, with Level-2 techniques showing the highest quality scores.
- **Stepwise refinement is effective**: The refinement chain technique, which progressively improves requirements through multiple steps, produced the highest overall quality scores (average 0.87).
- **Task-specific technique selection matters**: Different requirements tasks benefit from different prompt engineering techniques. For example, stakeholder analysis benefits most from the stakeholder-perspective technique, while requirements conflict analysis is best served by adverse analysis.
- **Parameter optimization is crucial**: Lower temperatures (0.2-0.5) generally produced better quality requirements, especially for technical requirements tasks.
- **Technique combinations yield best results**: Combining role-based prompting with technique-specific templates and appropriate parameter settings provides optimal results.

4.2 Analysis of Specific Techniques

4.2.1 Level-1 Techniques

Level-1 techniques improved quality by 7.07% compared to standard approaches. The most effective Level-1 technique was the stakeholder-perspective approach, which helped ensure requirements addressed multiple viewpoints and concerns.

4.2.2 Level-2 Techniques

Level-2 techniques showed an additional 10.1% improvement over Level-1 techniques. The refinement chain technique was particularly effective for complex requirements tasks, showing consistent improvement across each step in the chain:

- **Step 1: Initial Requirements Generation** - Establishes core functionality and main user needs.

- **Step 2: Analysis and Elaboration** - Identifies ambiguities, adds acceptance criteria, considers edge cases.
- **Step 3: Quality Check and Refinement** - Ensures requirements are specific, measurable, achievable, relevant, and time-bound.

The adverse analysis technique was particularly effective for requirements conflict analysis and validation tasks, where identifying potential issues is critical.

4.3 Practical Applications

This research has several practical applications for requirements engineering:

- **Automated Requirements Elicitation**: Organizations can use these techniques to quickly generate initial requirements from high-level descriptions.
- **Requirements Refinement**: The refinement chain technique can help transform rough requirements into well-structured, comprehensive specifications.
- **Stakeholder Alignment**: The stakeholder-perspective technique helps ensure requirements reflect diverse stakeholder needs.
- **Requirements Validation**: The adverse analysis technique can help identify potential issues before development begins.
- **Educational Tool**: The framework can be used to teach requirements engineering best practices and demonstrate how different perspectives impact requirements quality.

4.4 Industry Use Cases and Adoption Challenges

The implementation of multi-level prompt engineering techniques in real-world scenarios presents both opportunities and challenges. Organizations in software development, finance, and healthcare could integrate these techniques to streamline requirements analysis. For instance:

- **Software Development**: Agile teams can leverage automated prompt refinement to generate user stories from high-level business requirements, improving backlog quality.
- **Finance**: Banks and fintech companies can use structured output techniques to standardize regulatory compliance requirements across products.
- **Healthcare**: Hospitals and medical software providers can apply stakeholder-perspective analysis to capture the needs of patients, medical staff, and regulatory bodies.

Despite these benefits, adoption challenges remain:

- **Trust and Transparency**: Many organizations are hesitant to rely on AI-generated requirements due to concerns about accuracy and interpretability.
- **Regulatory and Compliance Issues**: In heavily regulated industries (e.g., finance, healthcare), automated requirements must meet strict legal and security standards.
- **Human-AI Collaboration**: Successful deployment of these techniques requires balancing automation with human oversight to ensure quality.

4.5 Limitations

While our results are promising, several limitations should be considered:

- **Evaluation Subjectivity**: Quality assessment of requirements relies partly on subjective criteria.
- **Domain Specificity**: Our test cases cover common requirements tasks but may not represent all possible domains and scenarios.
- **Model Dependence**: The effectiveness of techniques may vary with different underlying language models.
- **Computational Cost**: Level-2 techniques, while more effective, require multiple model calls and are more computationally expensive.

5 Conclusion

This research demonstrates that multi-level prompt engineering techniques significantly improve the quality of requirements derived from natural language descriptions. Our key conclusions are:

1. Level-2 techniques, particularly the refinement chain approach, provide the highest quality requirements by utilizing a multi-step process that progressively refines and improves requirements.
2. Different requirements tasks benefit from different prompt engineering techniques, highlighting the importance of technique selection based on task type.
3. Automated prompt refinement with iterative quality assessment offers a practical approach to generating high-quality requirements specifications.
4. Parameter optimization, particularly temperature settings, significantly impacts requirements quality and should be tuned based on the specific task and technique.

Our framework provides a systematic approach to prompt engineering for requirements analysis that can be applied in practice to improve requirements quality and consistency.

6 Further Research

Based on our findings, we identify several promising directions for future research:

- **Extended Parameter Space Exploration**:
  - Testing wider ranges of temperature and context window sizes
  - Investigating impact of other model parameters
  - Exploring adaptive parameter selection based on task characteristics
- **Hybrid Technique Development**:
  - Creating new techniques that combine strengths of multiple approaches
  - Developing domain-specific variations for specialized requirements (e.g., security, compliance)
  - Investigating dynamic technique switching based on requirement type
- **Automated Requirements Traceability**:
  - Extending the framework to maintain traceability between requirements
  - Developing techniques for requirements dependency analysis
  - Creating automated test case generation from requirements
- **Human-in-the-Loop Integration**:
  - Exploring hybrid approaches combining automated prompt engineering with human expert input
  - Developing interfaces for efficient human review and refinement
  - Studying how prompt engineering techniques can augment rather than replace human experts
- **Domain-Specific Requirements Engineering**:
  - Adapting techniques for specialized domains (healthcare, finance, aerospace)
  - Developing domain-specific templates and evaluation criteria
  - Creating specialized L2 techniques for regulated industries

A Prompt Engineering Techniques Details

A.1 Level-1 Technique Templates

Listing 1: Meta-Prompt Template

```
"""
Create an effective prompt that will elicit comprehensive and structure

{query}

Your prompt should:
1. Ask clarifying questions about scope and constraints
2. Guide the analysis through different requirement categories
3. Help identify both explicit and implicit requirements
4. Ensure requirements are testable and measurable
"""
```

Listing 2: Stakeholder Perspective Template

```
"""
Analyze the following requirement task from three different stakeholder

{query}

For each perspective (End User, Business Owner, Technical Team):
1. What are the key priorities and concerns?
2. What specific requirements would they emphasize?
3. What potential conflicts might arise between perspectives?
4. How can these requirements be reconciled into a comprehensive speci
"""
```

Listing 3: Quality Criteria Template

```
"""
Develop detailed requirements for the following task by systematically

{query}

For each of these quality attributes:
- Functionality: What should the system do?
- Usability: How will users interact with it?
- Reliability: How should it perform under stress?
- Performance: What are the speed/efficiency requirements?
- Security: What protections should be in place?
- Maintainability: How can it be designed for future change?

Format each requirement to be specific, measurable, achievable, relevan
"""
```

A.2 Level-2 Technique Steps

Listing 4: Refinement Chain Steps

```
# Step 1: Initial requirements gathering
"""
Generate an initial set of requirements based on this task:

{query}

Focus on capturing the core functionality and main user needs.
List at least 5 high-level requirements.
"""

# Step 2: Analysis and elaboration
"""
Analyze the following initial requirements:

{previous_response}

For each requirement:
1. Identify any ambiguities or missing details
2. Add acceptance criteria
3. Consider edge cases and exceptions
4. Categorize as functional or non-functional
"""

# Step 3: Quality check and refinement
"""
Review and refine these analyzed requirements:

{previous_response}

For each requirement:
1. Ensure it's specific, measurable, achievable, relevant, and time-bou
2. Remove any redundancies or conflicts
3. Add priority levels (High/Medium/Low)
4. Provide a rationale for each requirement

Present the final requirements in a structured format suitable for tech
"""
```

Listing 5: Divergent-Convergent Steps

```
# Step 1: Divergent thinking
"""
For the following task, generate as many potential requirements as pos

{query}

Consider:
- Different user types and their needs
- Various use cases and scenarios
```

```
- Functional requirements
- Non-functional requirements
- Business rules and constraints
- Technical considerations

Don't filter or evaluate at this stage - aim for quantity and diversity
"""

# Step 2: Evaluation
"""
Review the following list of potential requirements:

{previous_response}

Evaluate each requirement based on:
1. Value to users and business
2. Technical feasibility
3. Alignment with project scope
4. Potential implementation complexity

For each requirement, provide a score of 1-5 in each category and brief
"""

# Step 3: Convergent thinking
"""
Based on your evaluation:

{previous_response}

1. Select the top 10-15 most valuable and feasible requirements
2. Organize them into a coherent specification
3. Identify dependencies between requirements
4. Suggest an implementation priority order

Present the final requirement specification in a clear, structured form
"""
```

Listing 6: Adverse Analysis Steps

```
# Step 1: Generate baseline requirements
"""
Create a baseline set of requirements for:

{query}

Focus on the happy path scenarios and core functionality.
"""

# Step 2: Adversarial analysis
```

```
"""
Analyze these baseline requirements from an adversarial perspective:

{previous_response}

For each requirement:
1. How could it fail or be misinterpreted?
2. What edge cases are not covered?
3. How might users misuse or abuse this feature?
4. What security vulnerabilities might exist?
5. What performance issues could arise?

Identify at least 3 issues for each requirement.
"""

# Step 3: Refinement and hardening
"""
Based on the adversarial analysis:

{previous_response}

1. Refine each original requirement to address the identified issues
2. Add new requirements to cover gaps and edge cases
3. Include explicit error handling and validation requirements
4. Specify security and performance safeguards

Present the improved, hardened requirements specification.
"""
```

B Implementation Architecture

The research framework was implemented using a modular architecture with the following components:

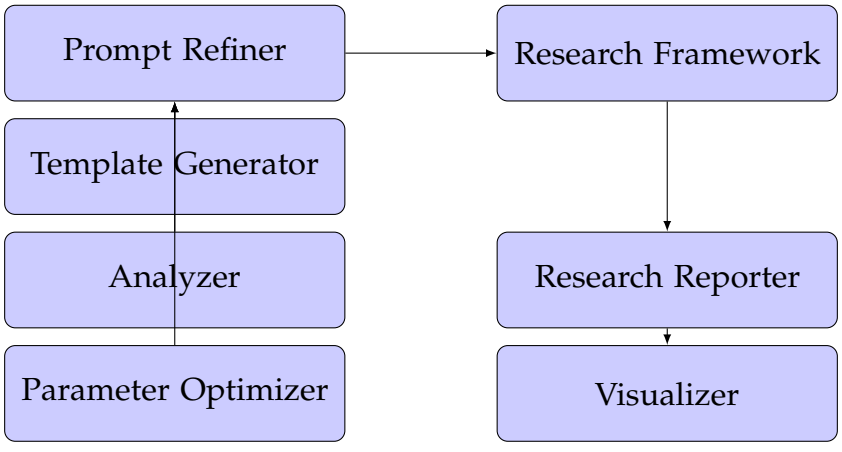

Figure 8: Research Framework Architecture

- **Prompt Refiner**: Iteratively improves prompts through a feedback loop
- **Template Generator**: Selects appropriate templates based on task type
- **Analyzer**: Evaluates prompt quality and suggests improvements
- **Parameter Optimizer**: Selects optimal parameters based on task type
- **Research Framework**: Manages experiments and collects results
- **Research Reporter**: Generates research reports from experiment data
- **Visualizer**: Create data visualizations and charts.

C Statistical Appendix

Table 8: Detailed Statistical Metrics

Metric	Value
Mean Quality Score (All Techniques)	0.76
Standard Techniques Mean Quality	0.69
L1 Techniques Mean Quality	0.79
L2 Techniques Mean Quality	0.85
L1 vs. Standard Improvement (%)	7.07%
L2 vs. L1 Improvement (%)	.8%
L2 vs. Standard Improvement (%)	15.15%
Role Detection Accuracy	9.41%
ww Mean Processing Time (s)	3.8

References

[1] Devos, C. (2025). *Optimizing Prompt Refinement and Temperature Settings for Requirements Analysis*. Generative Intelligence Lab @ FAU. *Prompt Engineering for Requirements Analysis - GitHub Repository*. Available at: github.com/cdevos2017/cot6930-200-a1. Accessed: February 27, 2025.

[2] Koch, F. (2024). *Prompt Engineering Lab: Platform for Education and Experimentation with Prompt Engineering in Generative Intelligent Systems*. Generative Intelligence Lab @ FAU.

[3] GenILab-FAU (2025). *Prompt Engineering Lab*. GitHub repository, https://github.com/GenILab-FAU/prompt-eng.