Test-Based Domain Model Generation via Large Language Models: A Comparative Analysis of Advanced Prompt Engineering Techniques

Pedro Henrique de Oliveira Silva (phos@cin.ufpe.br)

Federal University of Pernambuco
secgrad@cin.ufpe.br
www.cin.ufpe.br/~graduacao

Recife
2025

Pedro Henrique de Oliveira Silva (phos@cin.ufpe.br)

Test-Based Domain Model Generation via Large Language Models: A Comparative Analysis of Advanced Prompt Engineering Techniques

A B.Sc. Dissertation presented to the Center of Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Bachelor in Computer Engineering.

Concentration Area: *Software Engineering*
Advisor: *Augusto Sampaio (acas@cin.ufpe.br)*

Recife
2025

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Silva, Pedro Henrique de Oliveira.
Test-Based Domain Model Generation via Large Language Models: A Comparative Analysis of Advanced Prompt Engineering Techniques / Pedro Henrique de Oliveira Silva. - Recife, 2025.
p50 : il., tab.

Orientador(a): Augusto Sampaio
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Engenharia da Computação - Bacharelado, 2025.
Inclui referências, apêndices.

1. Modelos de Domínio. 2. LLM. 3. Teste de Software. 4. Engenharia de Prompt. 5. Validação Semântica. I. Sampaio, Augusto. (Orientação). II. Título.

000 CDD (22.ed.)

PEDRO HENRIQUE DE OLIVEIRA SILVA

TEST-BASED DOMAIN MODEL GENERATION VIA LARGE LANGUAGE MODELS: A Comparative Analysis Of Advanced Prompt Engineering Techniques

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de bacharel em Engenharia da Computação.

Aprovado em: 18/08/2025

BANCA EXAMINADORA

_____

Augusto Sampaio (Orientador)

Universidade Federal de Pernambuco

_____

Alexandre Cabral Mota (Examinador Interno)

Universidade Federal de Pernambuco

ACKNOWLEDGEMENTS

Aos meus pais, que sempre me ensinaram que o mundo é vasto e que há muito mais a se explorar do que a realidade em que cresci. Agradeço por todo o apoio e sacrifício em prol de fornecer a mim e a meus irmãos uma boa educação, dentro e fora de casa.

Aos meus irmãos, minha avó e minha tia. O amor e o suporte de vocês foram essenciais para que eu tivesse a coragem e as condições de buscar meus objetivos longe de casa, sabendo que sempre teria pra onde voltar.

À minha companheira de vida, Lívia. Em cada passo do caminho você foi o suporte e o apoio nos momentos mais desafiadores, celebração e companhia alegre em cada conquista. Sua presença e encorajamento foram essenciais para que eu chegasse até aqui.

Ao meu amigo Vitor, por ter compartilhado tantas experiências da área comigo e por ter feito parte desta jornada.

À Andresa, cuja pesquisa serviu de base para este trabalho. Sua disponibilidade, generosidade em compartilhar materiais e apoio constante foram fundamentais para o desenvolvimento deste estudo.

Ao meu orientador, professor Augusto, por toda a paciência e direcionamento durante esta fase. Seus conselhos foram cruciais para a conclusão deste trabalho.

Aos meus professores, com quem tive a oportunidade de aprender e, em meio à faculdade, redescobrir minha motivação e afinidade pela área e a academia.

ABSTRACT

The automated generation of domain models from test cases represents a fundamental challenge in software engineering, particularly in the context of mobile device testing. This work extends the research by Silva (2025), who proposed a framework based on Large Language Models (LLMs) to automate this process. We present three main contributions: (1) SBERT, a BERT-based model for generating sentence embeddings to measure semantic similarity, will be replaced by an LLM for semantic validation, eliminating fixed threshold limitations and providing contextual analysis capabilities; (2) implementation and comparative evaluation of five advanced prompt engineering techniques - Few-Shot, Chain-of-Thought, Universal Self-Consistency, Tree of Thoughts, and Prompt Chaining; and (3) systematic analysis of the impact of the temperature parameter on the quality of the generated models. Using Gemini 2.5-flash (instead of Gemini 2 adopted in the previous work), but reusing the same dataset from the original work to ensure comparability, our experiments focus on evaluating the effectiveness of different prompting strategies. Among the techniques evaluated, Chain-of-Thought demonstrated the best overall performance with median recall of 0.87 and low variance (σ=0.06), while being computationally efficient. Temperature analysis revealed an optimal result with value 0.3 for structured modelling tasks, balancing determinism and flexibility. These results not only validate the effectiveness of the proposed techniques but also provide practical guidelines for applying LLMs to software engineering tasks that require structural precision and semantic understanding. In particular, we demonstrate significant improvements over the baseline work, with increases of up to 23% in correct identification of implicit atoms and 15% in detection of complex associations.

Keywords: Domain Models, LLM, Software Testing, Prompt Engineering, Semantic Validation.

RESUMO

A geração automatizada de modelos de domínio a partir de casos de teste representa um desafio fundamental na engenharia de software, particularmente no contexto de testes de dispositivos móveis. Este trabalho estende a pesquisa de Silva (2025), que propôs uma estrutura baseada em Large Language Models (LLMs) para automatizar esse processo. Apresentamos três contribuições principais: (1) SBERT, um modelo baseado em BERT para gerar 'sentence embeddings' para medir similaridade semântica, será substituído por um LLM para validação semântica, eliminando limitações de limiares fixos e fornecendo capacidades de análise contextual; (2) implementação e avaliação comparativa de cinco técnicas avançadas de 'prompt engineering' - Few-Shot, Chain-of-Thought, Universal Self-Consistency, Tree of Thoughts e Prompt Chaining; e (3) análise sistemática do impacto do parâmetro 'temperature' na qualidade dos modelos gerados. Usando Gemini 2.5-flash (em vez de Gemini 2 adotado no trabalho anterior), mas reutilizando o mesmo conjunto de dados do trabalho original para garantir a comparabilidade, nossos experimentos se concentram em avaliar a eficácia das diferentes estratégias de 'prompting'. Entre as técnicas avaliadas, Chain-of-Thought demonstrou o melhor desempenho geral com recall mediano de 0.87 e baixa variância (σ=0.06), enquanto era computacionalmente eficiente. A análise de 'temperature' revelou um resultado ótimo com o valor 0.3 para tarefas de modelagem estruturada, equilibrando determinismo e flexibilidade. Esses resultados não apenas validam a eficácia das técnicas propostas, mas também fornecem diretrizes práticas para a aplicação de LLMs em tarefas de engenharia de software que exigem precisão estrutural e compreensão semântica. Em particular, demonstramos melhorias significativas em relação ao trabalho de base, com aumentos de até 23% na identificação correta de átomos implícitos e 15% na detecção de associações complexas.

Palavras-chave: Modelos de Domínio, LLM, Teste de Software, Engenharia de Prompt, Validação Semântica.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1

INTRODUCTION

The automation of domain model generation remains a significant challenge in the software development lifecycle. In the specific context of software testing for mobile devices, this task becomes even more complex due to the dynamic and interconnected nature of modern functionalities. This complexity is evident in industrial practice, as demonstrated through collaborative work with Motorola in mobile device testing scenarios. Test-based domain models serve as fundamental abstract representations that capture relationships between test actions, including dependencies, cancellations, and instantiations, being essential for ensuring test case consistency and completeness.

The work by Silva (2025) [20] established a robust framework for automating the generation of these models using Large Language Models (LLMs), specifically Gemini 2.0-flash. Through the combination of structural validation via ASP (Answer Set Programming) [5] solver and semantic validation using SBERT (Sentence Bidirectional Encoder Representations from Transformers) [19], the original framework achieved promising results, with 80-90% satisfiability in low and medium complexity domains. However, several limitations were identified, including difficulties in identifying implicit associations, dependence on a fixed threshold for semantic similarity, and limited exploration of prompt engineering techniques' potential.

These limitations manifest concretely in industrial practice. SBERT-based semantic validation frequently fails to capture important contextual nuances—for example, "Set device multi-window mode ON" and "Enable multi-window mode" are semantically equivalent in the context of mobile testing, but may not reach the 0.8 similarity threshold due to superficial differences in textual structure. This fixed-threshold approach inherently constrains the system's ability to recognize semantic equivalence when surface forms diverge, suggesting that the deep contextual understanding of modern LLMs could overcome these embedding-based limitations.

Prior to and in parallel with Silva's work, significant advances have occurred both in LLM development and in understanding prompt engineering techniques. Recent models, such as Gemini 2.5-flash, demonstrate improved reasoning and contextual understanding capabilities compared to their predecessors. Concurrently, advanced prompting techniques have emerged as promising alternatives for tasks requiring structured reasoning and systematic solution exploration. Tree of Thoughts (ToT) [24], which allows systematic exploration of multiple reasoning

paths, appears naturally suited for identifying complex relationships between test actions. Similarly, Universal Self-Consistency (USC) [11] can potentially mitigate the non-deterministic nature of LLMs through aggregation of multiple perspectives, though questions remain about whether its voting mechanism effectively identifies superior solutions or potentially discards valid alternatives.

The question of temperature adjustment in LLMs for structured tasks also merits systematic investigation. While higher temperatures promote creativity and exploration, formal modelling tasks may benefit from more deterministic outputs. Finding the optimal balance could mean the difference between complete but inconsistent models and correct but incomplete models—a trade-off that has not been systematically explored in the context of domain model generation.

This work proposes an evolution of the original framework by enhancing the effectiveness and robustness of automated test-based domain model generation through systematic exploration of a more recent version of Gemini (2.5-flash) and more elaborate prompt engineering techniques. First, we explore the replacement of SBERT with semantic validation based directly on LLM, assuming that the deep contextual understanding of language models can overcome the limitations of embedding-based methods with fixed thresholds. Second, we implement and systematically evaluate five prompt engineering techniques – Few-Shot Learning [6], Chain-of-Thought (CoT) [23], Universal Self-Consistency (USC), Tree of Thoughts (ToT), and Prompt Chaining [13] [21] – seeking to identify which approach is most effective for the specific task of domain model generation. Third, we investigate the impact of the temperature parameter on the quality and consistency of generated models, a dimension little explored in software engineering tasks that require structural precision. Specific objectives include:

- Develop and evaluate an LLM-based semantic validation method that overcomes the limitations of SBERT, providing contextual analysis without reliance on fixed thresholds.
- Implement and systematically compare five prompt engineering techniques (Few-Shot, CoT, USC, ToT, and Prompt Chaining) in the context of domain model generation.
- Investigate the impact of the temperature parameter on the precision, completeness, and consistency of the generated models.
- Validate the proposed improvements using the same dataset from the original work, ensuring direct comparability of results.
- Provide practical guidelines for applying LLMs to software engineering tasks that require structural precision and semantic understanding.
- Analyze the decision-making process in USC, particularly examining the quality of non-selected paths to assess whether the voting mechanism effectively identifies superior

solutions or potentially discards valid alternatives, thereby evaluating the technique's recall and precision in the context of domain model generation.

This work presents the following main contributions:

- **LLM-based semantic validation framework:** We developed a new method for semantic validation that utilises the contextual understanding capabilities of modern LLMs, eliminating the need for fixed thresholds and providing richer and more contextualised feedback.
- **Comprehensive comparative analysis of prompting techniques:** We present the first systematic comparison of five advanced prompt engineering techniques applied specifically to domain model generation, including quantitative metrics and qualitative analysis.
- **Empirical study on temperature in structured tasks:** We provide empirical evidence on the impact of temperature in tasks that require both creativity and structural precision, establishing guidelines for optimal configuration.
- **Measurable improvements with respect to the original framework:** We demonstrate significant improvements over the baseline work, with increases of up to 23% in correct identification of implicit atoms and 15% in detection of complex associations.

The remainder of this work is organised as follows: Chapter 2 presents some background, including a review of the original framework and the prompt engineering techniques used. Chapter 3 details the proposed methodology, including the extended architecture and technique implementation. Chapter 4 presents the experimental evaluation and results. Chapter 5 discusses related work and positions our contributions in the current context. Finally, Chapter 6 concludes the work and points to future directions.

2

BACKGROUND

This chapter introduces background necessary to understand the contributions of this work. We begin with a detailed review of the original framework proposed by Silva (2025), identifying its main components and limitations. Next, we explore the fundamentals of semantic validation, contrasting embedding-based approaches with LLM-based methods. Finally, we present an in-depth analysis of the prompt engineering techniques used in this work, including their principles and practical applications.

2.1 REVIEW OF THE ORIGINAL FRAMEWORK

The work by Silva (2025) established a pioneering framework for the automated generation of test-based domain models using LLMs. The proposed architecture consists of a pipeline that integrates three main components: (1) prompt preparation using few-shot learning techniques and an initial implementation of chain-of-thought, (2) structural validation through the Answer Set Programming (ASP) solver Clingo, and (3) semantic validation using SBERT to verify correspondence between generated and expected atoms.

2.1.1 Base framework architecture

The original framework (see Figure 1) operates through an iterative cycle of generation and validation. The process begins with prompt preparation that contextualises the domain modelling task for the LLM. Silva explored two main prompting techniques: few-shot learning, where examples of mapping between test cases and domain models are provided, and a version of chain-of-thought that decomposes the task into sequential steps.

Structural validation uses a set of logical rules implemented in ASP to verify the structural consistency of the generated model. These rules include detection of cyclic dependencies and conflicts between associations. When inconsistencies are detected, specific feedback is provided to the LLM for refinement.

Semantic validation in the original framework employs SBERT to calculate atom embeddings and determine correspondences through cosine similarity with a fixed threshold of 0.8.

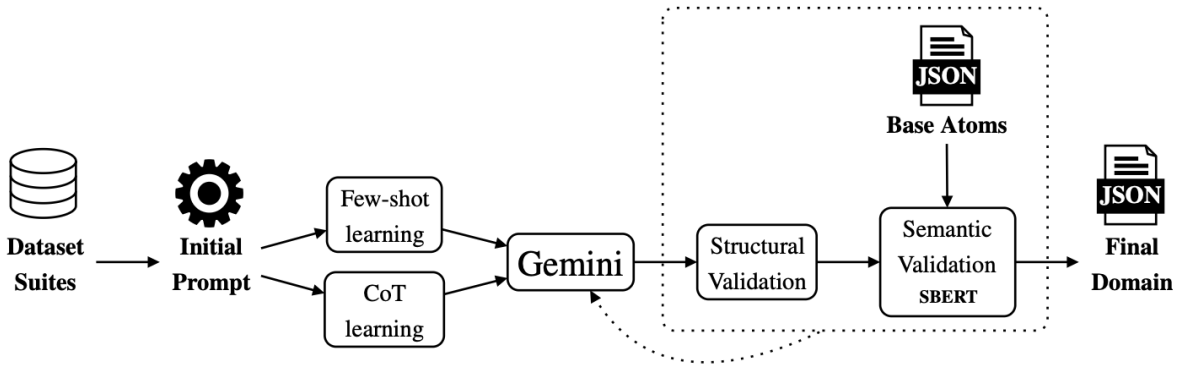

Figure 1: Proposed framework for generating test-based domain models using LLMs proposed by Silva (2025).

This approach, whilst effective in many cases, presents significant limitations that motivated part of the improvements proposed in this work.

2.1.2 Results and identified limitations

The original framework achieved notable results, with satisfiability rates between 80-90% in low and medium complexity domains. However, several limitations were identified:

1. **Difficulty in identifying implicit associations:** The system demonstrated low recall in detecting instantiations (0-6%) and cancellations (0-70%), suggesting that the prompting techniques used were insufficient to capture these more subtle relationships.

2. **Fixed threshold in semantic validation:** The use of a fixed threshold of 0.8 for semantic similarity proved problematic, failing in cases where valid linguistic variations did not reach this arbitrary limit.

3. **Limited exploration of prompting techniques:** Whilst the work explored few-shot learning and a form of chain-of-thought, more recent and sophisticated techniques were not considered.

4. **Absence of model parameter analysis:** The impact of parameters such as temperature was not investigated.

2.2 SEMANTIC VALIDATION: SBERT VERSUS LLM

Semantic validation constitutes a critical component in automated model generation systems, being responsible for verifying whether generated elements correspond semantically to expected elements. This section analyses embedding-based approaches, exemplified by SBERT, and contrasts with LLM-based methods that we propose in this work.

2.2.1 Limitations of embedding-based approaches

SBERT **(Reimers & Gurevych, 2019)** revolutionised semantic similarity computation by generating sentence embeddings that can be efficiently compared through measures such as cosine similarity. In the context of the original framework, SBERT served multiple validation functions: determining whether atoms generated by the LLM corresponded to expected atoms extracted from test cases, and when ground-truth domain models were available as feedback, mapping the associations between atoms including dependencies, cancellations, and instantiations.

However, this approach presents several intrinsic limitations when applied to the specific domain of test modelling:

- **Sensitivity to superficial variations:** Embeddings are sensitive to variations in text surface structure. For example, *"Enable Wi-Fi"* and *"Turn on wireless connection"* may have significantly different embeddings despite being functionally equivalent in the context of mobile testing.
- **Lack of contextual awareness:** SBERT generates fixed embeddings that do not consider the specific domain context. The same phrase may have different meanings in different contexts, but the embedding remains constant.
- **Fixed threshold:** The need to define a fixed threshold (0.8 in the original work) creates an inflexible trade-off between precision and recall. A high threshold results in many false negatives, whilst a low threshold may accept incorrect correspondences.
- **Inability to provide justifications:** When a correspondence fails, the embedding-based method cannot explain why, limiting the capacity for iterative refinement.

2.2.2 LLM-based semantic validation

Large Language Models offer an alternative approach to semantic validation that leverages their inherent understanding and reasoning capabilities rather than relying on vector similarity measures. This approach presents several advantages:

1. **Deep contextual understanding:** LLMs can process and evaluate semantic equivalences by considering the complete contextual information available. Their architecture enables them to recognize when different surface forms express the same underlying concept, understanding that variations in phrasing may represent identical functional meanings within specific domains.
2. **Flexibility without thresholds:** Rather than depending on fixed numerical thresholds for similarity decisions, LLMs can make nuanced judgments about semantic equivalence. These models evaluate multiple dimensions simultaneously—including context, intention,

and acceptable linguistic variations—without reducing the comparison to a single scalar value.

3. **Rich and Explanatory Feedback:** LLMs can provide detailed justifications for their decisions, explaining why two atoms are or are not considered equivalent, facilitating iterative refinement.

The effective application of LLM-based semantic validation depends on careful prompt design that communicates the relevant equivalence criteria for the target domain. This includes establishing guidelines for handling technical synonyms, nomenclature variations, and determining which details are semantically significant versus those that can be abstracted away.

2.3 ADVANCED PROMPT ENGINEERING TECHNIQUES

Prompt engineering has evolved significantly from basic zero-shot approaches, developing sophisticated techniques that exploit the emergent capabilities of LLMs for complex reasoning and structured generation tasks. This section presents five fundamental techniques that have demonstrated efficacy in complex scenarios and were applied in this study.

2.3.1 Few-Shot Learning

Few-Shot Learning enables LLMs to execute new tasks based on a few demonstrative examples, without requiring fine-tuning of model parameters. In this approach, the model is conditioned by a prompt containing task demonstrations, provided directly in the context window during inference. Few-shot learning capability improves dramatically with model scale, enabling competitive performance across diverse natural language processing tasks. For domain modelling, the dynamic selection of relevant examples and their appropriate structuring in the prompt are critical factors for the technique's success.

2.3.2 Chain-of-Thought

Chain-of-Thought prompting enables LLMs to demonstrate complex reasoning capabilities through generating intermediate thought steps before the final answer. The technique instructs the model to explicitly articulate its reasoning process step by step, including demonstrations that contain not only input-output pairs, but also the chain of reasoning that leads to the solution. This approach has demonstrated significant improvements in tasks requiring multi-step reasoning, with particularly notable efficacy in large-scale models, where problem decomposition capability emerges more prominently.

2.3.3 Universal Self-Consistency

Universal Self-Consistency extends the concept of traditional self-consistency [22], addressing the non-deterministic nature of LLMs. The methodology involves two stages: (1) the model generates multiple candidate responses for a task; (2) these responses are presented to the LLM itself with instructions to identify and select the most consistent response amongst the generated options. This approach eliminates the need for exact matching or external parsing, making it particularly relevant for free-text generation tasks and scenarios where traditional aggregation is unfeasible, such as in domain model generation with varied structures.

2.3.4 Tree of Thoughts

The Tree of Thoughts framework transcends the limitations of sequential inferences by structuring problem-solving as tree search. Each node represents a "thought" - a coherent unit of language that serves as an intermediate step. ToT is characterised by four components: decomposition into intermediate thoughts, generation of potential thoughts, heuristic evaluation of states using the LLM itself, and application of search algorithms (Breath First Search (BFS) or Depth First Search (DFS)) for systematic exploration. This modularity makes ToT effective in tasks requiring non-trivial planning and exploration of multiple solution paths, allowing backtracking when necessary.

2.3.5 Prompt Chaining

Prompt Chaining addresses complex tasks through decomposition into sequential subtasks, where the output of each stage feeds the next. Unlike CoT, which generates reasoning within a single response, Prompt Chaining structures the process externally through multiple distinct interactions. For domain modelling, this technique enables progressive model construction through a specific chain: atom identification $\rightarrow$ dependency analysis $\rightarrow$ cancellation detection $\rightarrow$ instantiation mapping, for example. Each stage is conditioned by previous results, creating a modular pipeline that facilitates debugging and allows integration of intermediate validations.

2.4 LLM PARAMETERS

The behaviour of LLMs is significantly influenced by configuration parameters, with temperature being one of the most critical. This section explores this concept and its implications for structured modelling tasks.

2.4.1 Temperature and probability distribution

Temperature in LLMs controls the randomness of predictions by adjusting the probability distribution over the vocabulary. Most models use a temperature range between 0 and 1, while

Gemini uses a range between 0 and 2. In general, the temperature influence over the LLMs responses is determined as follows:

1. **Low temperature** $T \to 0$**:** Increases the model's "confidence" in its most likely predictions, making the distribution more concentrated on higher probability tokens. The outputs become more deterministic, repetitive, and focused.

2. **High temperature** $(T > 1)$**:** Decreases the difference between the probabilities of the tokens, making the distribution more uniform. This increases diversity, "creativity," and the chance of the model generating unexpected responses.

3

FRAMEWORK

This chapter presents the methodology developed to enhance automated generation of test-based domain models. We describe the extended framework architecture, detail the implementation of LLM-based semantic validation, and present the five implemented prompt engineering techniques.

3.1 EXTENDED FRAMEWORK ARCHITECTURE

The proposed architecture maintains the fundamental structure of Silva's (2025) framework whilst introducing enhanced components and novel validation mechanisms. Figure 2 illustrates the complete system flow, demonstrating the experimental framework where test suites are processed through prompt engineering techniques using Gemini 2.5-flash to generate domain models. Then, the structural validator performs correctness checking and triggers iterative feedback loops when errors are identified.

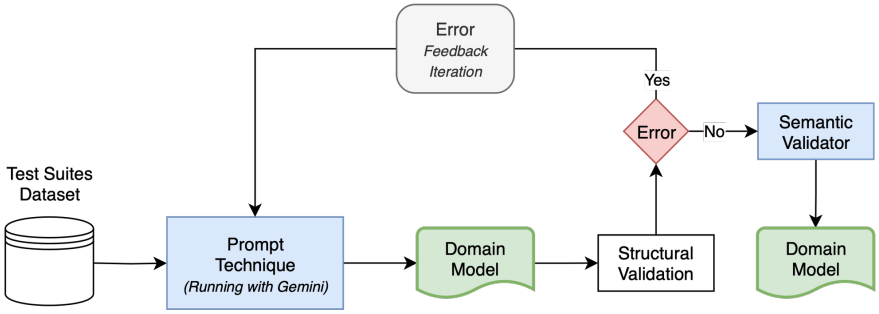

Figure 2: Schematic representation of the experimental framework.

The main components of the framework are:

Prompt engineering module: We replaced the original prompting module with a modular system that supports five different techniques: Few-Shot Learning, Chain-of-Thought, Universal Self-Consistency, Tree of Thoughts, and Prompt Chaining. Each technique is implemented as an independent class that inherits from a common interface, enabling seamless extension and systematic comparison across methodologies. The module provides parametrised

```
// Test Case 1
"step_1": {
    "setup": "",
    "description": "Open camera app",
    "result": "Camera preview should be shown",
}
"step_2": {
    "setup": "",
    "description": "Take a photo",
    "result": "Photo should be saved to gallery",
}
// Test Case 2
"step_1": {
    "setup": "",
    "description": "Open camera app",
    "result": "Camera preview should be shown",

"step_2": {
    "setup": "Switch to front camera",
    "description": "Record a video",
    "result": "Video should be saved to gallery",
}
```

Figure 3: Example input test cases for the prompt engineering module. The test cases follow a structured JSON format with setup conditions, action descriptions, and expected results.

control over both the LLM model version and temperature settings, facilitating comprehensive experimental evaluation. For advanced techniques such as ToT and USC, the module maintains conversation history and manages state information required for multi-step reasoning processes. Figure 3 shows an example of input and Figure 4 shows an example of output from this module.

Feedback validation: We maintained the ASP solver Clingo-based structural validation module, as it proved robust and effective. This module acts as an enhanced feedback system that guides the domain model generation based on error messages. The logical rules remain the same, ensuring comparability with the original work.

LLM-based semantic validation: This component represents the most significant architectural modification from the original framework. We replaced the SBERT-based similarity matching with a system that leverages the LLM's own capabilities to perform contextualised semantic comparisons. This approach enables more nuanced understanding of semantic equivalences, particularly when handling a synonymous terms or contextually equivalent expressions that traditional embedding-based methods might miss. The validator utilises the same language model to ensure terminological consistency between generated and expected domain models, but the requests are made with no history of previous tasks.

The framework process initiates with the selection of a prompt engineering technique and temperature configuration from the predefined experimental matrix. The prompt engineering

```
{
    "atoms": [
        "Open camera app",
        "Close camera app",
        "Take a photo",
        "Record a video",
        "Switch to front camera",
        "Switch to rear camera"
    ],
    "cancellations": [
        {"source": "Open camera app", "target": "Close camera app"},
        {"source": "Close camera app", "target": "Open camera app"},
        {"source": "Switch to front camera", "target": "Switch to
           rear camera"},
        {"source": "Switch to rear camera", "target": "Switch to
           front camera"}
    ],
    "instances": [],
    "dependencies": [
        { "source": "Take a photo", "target": "Open camera app"},
        {"source": "Record a video", "target": "Open camera app"}
    ]
}
```

Figure 4: Example of a domain model generated by the prompt engineering module. The domain model follow a structured JSON format with atoms cancellations, instances and dependencies.

module prepares the appropriate prompt structure based on the selected technique, incorporating technique-specific instructions and examples as detailed in Section 3.3, and submits it to Gemini 2.5-flash for processing.

Upon receiving the prompt, Gemini generates a candidate domain model in structured JSON format, containing atoms as nodes and dependencies, cancellations, and instantiations as directed edges. This standardised output format ensures consistent processing across all prompt engineering techniques and facilitates automated validation.

The candidate model undergoes validation through the ASP-based structural validator. It checks for logical inconsistencies including cyclical dependencies, cancellation conflicts, and instantiation violations, as in Silva (2025). When structural errors are detected, the feedback module constructs error reports focusing solely on these structural issues, providing the LLM with specific guidance on logical corrections needed.

3.2 LLM-BASED SEMANTIC VALIDATION

The LLM-based semantic validation component represents a fundamental departure from traditional embedding-based approaches. Rather than reducing semantic comparison to numerical similarity scores, this component leverages the language model's inherent under-

standing capabilities to perform contextual equivalence assessment through a two-stage process: terminology adaptation and structural comparison.

Terminology Adaptation Process: The validation begins by analyzing the language and terminology used in the generated domain model. The LLM examines each atom in the generated model and adapts the ground-truth domain model to match this terminology while preserving semantic meaning. This adaptation process ensures that superficial linguistic differences do not obscure structural equivalences. For instance, if the generated model uses "Enable Wi-Fi connection" while the ground-truth contains "Turn on wireless network", the validator adapts the ground-truth to use the generated model's terminology before comparison.

Contextual Equivalence Assessment: The validator recognizes that atoms may be expressed differently while maintaining identical functional meaning. For instance, "Enable Wi-Fi connection" and "Turn on wireless network" represent the same action in mobile testing contexts, despite their surface-level differences. The LLM evaluates these equivalences by considering the testing domain context, understanding that variations in technical terminology often reflect different ways of describing the same underlying operation.

Synonymy and Variation Handling: Unlike fixed embeddings, the LLM-based approach dynamically interprets synonymous expressions and linguistic variations. The model understands that "Close application", "Terminate app", and "Exit program" may all refer to the same atomic action, depending on the test suite's conventions. This flexibility extends to handling abbreviations (Wi-Fi/WiFi), alternative phrasings (turn on/enable/activate), and domain-specific terminology variations.

Structural Comparison: After terminology adaptation, the validator performs a structural comparison between the adapted ground-truth and the generated model. This comparison identifies:

- **Missing atoms and relationships:** Elements and associations (dependencies, cancellations, instantiations) present in the ground-truth but absent from the generated model
- **Extra atoms and relationships:** Elements and associations generated but not present in the ground-truth, potentially indicating over-generation or incorrect inference of implicit components

The structural comparison operates at the semantic level rather than requiring exact string matches, enabling accurate assessment even when models use different representational conventions.

Implementation Details: The semantic validator operates as a stateless component, processing each validation independently without maintaining conversation history. This design choice ensures consistent evaluation criteria across all validations and prevents drift in judgment standards during extended validation sessions.

3.3 IMPLEMENTATION OF PROMPT ENGINEERING TECHNIQUES

The implementation of prompt engineering techniques constituted a central element of the methodological approach proposed in this work. Each technique was adapted and optimised for the specific context of test-based domain model generation, leveraging Gemini's capabilities and incorporating improvements over the original implementations described in the literature. The overall experimental framework, illustrated in Figure 2, demonstrates how these techniques are integrated within a validation loop that ensures the generation of consistent domain models.

3.3.1 Few-Shot Learning

The implementation of the Few-Shot Learning technique presented substantial modifications compared to Silva (2025). Firstly, we significantly expanded the set of demonstrative examples, increasing not only the quantity but also the complexity and diversity of cases presented to the model.

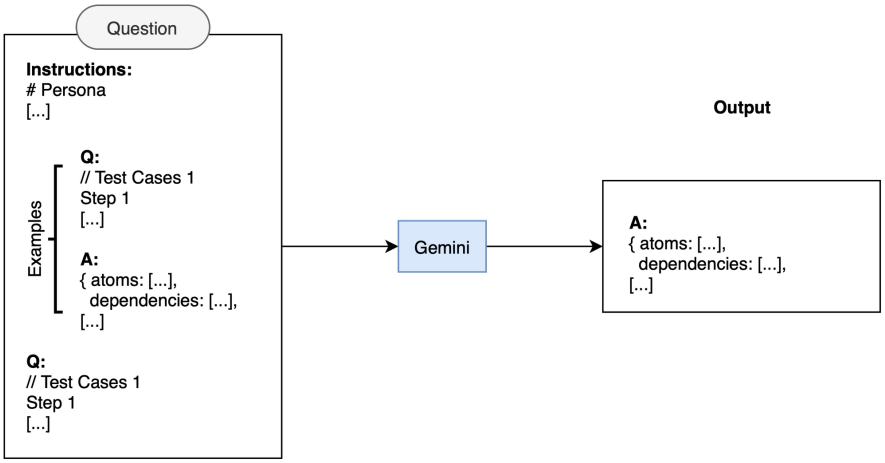

Figure 5: Few-Shot Learning prompt structure for domain model generation. The prompt comprises system instructions, followed by example pairs demonstrating the transformation from test cases to domain models (atoms, dependencies, and associations). The final question presents new test cases for processing.

The main methodological innovation resided in the optimised utilisation of structures provided by the Gemini API. Unlike the conventional approach, which concatenates instructions and examples in a single system prompt, our implementation strategically segregates these components. As shown in Figure 5, general instructions are provided as a system message, establishing the context and fundamental guidelines of the task. The examples, in turn, are structured as messages exchanged between user and assistant, simulating real interactions and allowing the model to better understand the expected response pattern.

The expanded examples incorporate specific ambiguity scenarios and their respective resolutions. Each example explicitly demonstrates how to identify implicit atoms in composite

test actions, how to establish dependency relationships between sequential actions, and how to recognise cancellation and instantiation patterns. This more detailed approach allows the model to develop a deeper understanding of the semantic nuances present in natural language test cases, with the structured format facilitating pattern recognition and consistent output generation.

3.3.2 Chain-of-Thought

```
[...]
# Your Task
Based on the essential definitions provided about a domain model and
   your expertise in software engineering and test automation, you
   should derive the domain model from the provided test cases by
   following these steps:
1.  **Define Explicit Atoms**: Analyze the `description` and `setup`
   of each test step to identify the directly mentioned atoms.
2.  **Define Implicit Atoms**: Infer any necessary missing atoms.
   Think about what is a prerequisite or logical consequence. *
   Example: "Verify message without internet" implies the atom "
   Disable internet". Every "Open" action implies a "Close" action*.
3.  **Create Instance Edges**: Identify atoms that are specific
   variations of a more general, conceptual atom.
4.  **Create Cancellation Edges**: Identify pairs of atoms that
   represent opposite actions (e.g., turn on/off, open/close, insert/
   remove).
5.  **Create Dependency Edges**: Analyze the logical sequence of the
   tests to define which atoms must be executed before others for the
    system to work correctly.
6.  **Final Review**: Ensure that the domain model represents the
   provided tests without adding new information not provided as well
    no atom or relation has been forgotten.
## Chain of Thought Instructions
First your output must by your reasoning process like you are
   thinking out loud.
After detailing your reasoning, insert a clear separator like `---`
   and then provide the final JSON object.
[...]
```

Figure 6: Excerpt CoT prompt demonstrating the six-step reasoning process. Instructions mandate explicit thought verbalisation for each stage of domain model construction, separated from the final JSON output by a clear delimiter.

The CoT implementation was designed to provide the language model with a structured step-by-step reasoning framework, essential for the complex task of extracting structured knowledge from test cases.

Figure 6 contains an excerpt from the prompt used and illustrates the implemented structure, which instructs the model to: (1) first identify all explicit atoms present in the test

```
Let's think step by step.
We are given two test cases. Let's break down each step and extract
   the atoms and relationships.
Test Case 1:
Step 1:
  description: "Open messages app",
  result: "The messages app screen should be available"
**-> Atom: "Open messages app"**
If there is an "Open" action there must be a "Close" action.
**-> Atom: "Close messages app"**
**-> Cancellation: "Close messages app" cancels "Open messages app"**
**-> Cancellation: "Open messages app" cancels "Close messages app"**
[...]
```

Figure 7: An excerpt from an example given as input. The method involves deconstructing a test step into its fundamental atoms and relationships, including cancellation actions, describing what would be a reasoning path.

cases; (2) analyse each test action to discover implicit atoms necessary for complete execution; (3) recognise generalisation patterns to establish instantiation relationships between specific and conceptual atoms; (4) identify mutually exclusive actions to determine cancellation edges; (5) establish dependency relationships based on temporal order and logical prerequisites; and (6) perform a final review to ensure the domain model accurately represents the provided tests without adding extraneous information or omitting any atoms or relationships.

Crucially, the provided examples not only demonstrate the final result but make explicit the entire intermediate reasoning process. As illustrated in Figure 7 each example includes detailed annotations that reveal the thinking behind each decision, from identifying an implicit atom to justifying a specific dependency relationship. This transparency in the reasoning process allows the model to internalise not just what to do, but how and why to do it.

3.3.3 Universal Self-Consistency

The USC implementation represents the most sophisticated approach in terms of response aggregation and validation. Our adaptation of this technique integrates seamlessly with the previously implemented CoT structure, creating a robust system for domain model generation and selection.

The implemented process, depicted in Figure 8, generates eight independent perspectives for each test set, each representing a complete reasoning path through the CoT technique. This multiplicity of perspectives captures different valid interpretations of requirements implicit in the test cases. Each execution may identify different implicit atoms or propose alternative relationships between actions, reflecting the inherent ambiguity of natural language.

The evaluation and consolidation phase constitutes the critical differentiator of this implementation. A specialised evaluator module systematically analyses all eight generated

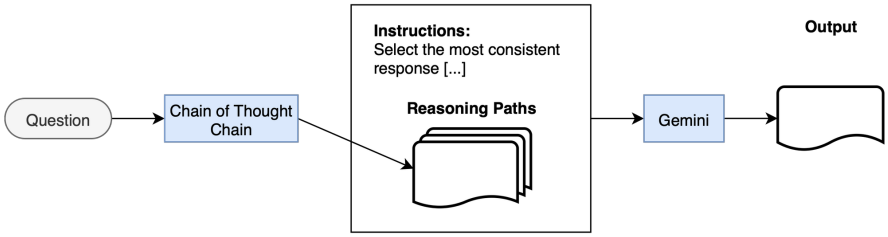

Figure 8: USC architecture for domain model generation. The CoT module generates multiple reasoning paths (eight perspectives) from input test cases. An evaluator component within Gemini analyses these parallel paths and selects the optimal domain model based on consistency metrics and quality assessment.

perspectives, considering multiple criteria: (1) completeness in atom identification; (2) structural consistency of proposed relationships; (3) semantic alignment with original test cases; and (4) logical coherence of the complete model. As illustrated in Figure 8, the evaluator not only selects the best perspective but also explicitly documents the reasons for its choice, providing detailed justifications that include comparisons between different proposals and identification of each one's strengths.

3.3.4 Tree of Thoughts

The ToT implementation represents the most complex technical challenge due to the exploratory tree nature of the algorithm. Our adaptation utilises a BFS strategy that systematically explores the space of possible domain models, as illustrated in Figure 9.

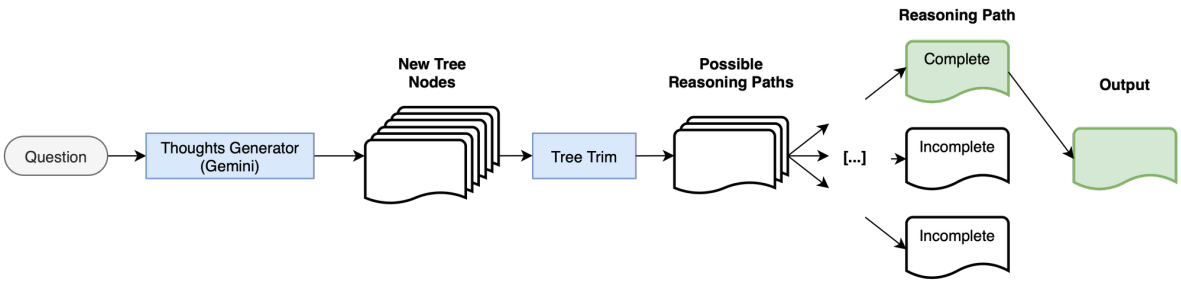

Figure 9: Implementation of ToT technique. Gemini generates multiple tree nodes representing partial solutions, which are evaluated and pruned through Tree Trim. The BFS continues until a complete reasoning path produces the final domain model.

Each node in the tree maintains a data structure that encapsulates the current state of the partial domain model and a categorical evaluation of its viability. The implemented evaluation categories are: "sure" (high confidence in path correctness), "likely" (moderate probability of success), "impossible" (unviable path detected), and "complete" (valid and complete domain model generated).

The thought generator operates on each evaluated node, producing five possible continuations that represent different decisions about atoms to add or relationships to establish. Each new

branch undergoes the evaluation process, which considers: (1) consistency with previous steps in the tree; (2) alignment with provided test cases; (3) satisfaction of domain model structural constraints; and (4) progress towards a complete model.

The implemented pruning mechanism (Tree Trim) is crucial for computational efficiency. After each expansion and evaluation round, nodes are ordered prioritising those classified as "sure", followed by "likely". Nodes marked as "impossible" are immediately discarded, avoiding unnecessary exploration of unviable paths. This iterative process continues until a "complete" node is identified, at which point the algorithm terminates and the path from root to this node is used to construct the final domain model. Otherwise, only the three most promising nodes are retained for the next iteration, balancing exploration with computational efficiency.

3.3.5 Prompt Chaining

The Prompt Chaining technique decomposes the complex task of domain model generation into a sequence of six specialised prompts, each focused on a specific aspect of the problem. This decomposition allows greater precision and control over each component of the generated model, as shown in Figure 10.

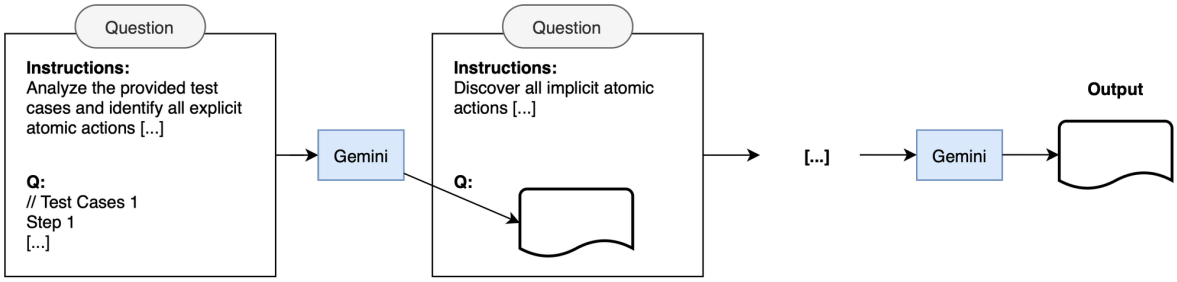

Figure 10: Prompt Chaining architecture for incremental domain model construction. Sequential prompts progressively build the model: first extracting explicit atomic actions from test cases, then identifying implicit atoms, followed by relationship detection. Each stage receives cumulative context from previous prompts, ensuring informed decision-making throughout the chain until final model integration.

The first prompt focuses exclusively on explicit atom extraction, instructing the model to identify only actions directly mentioned in test steps, without inferences or additions. The second prompt uses the identified explicit atoms as input and requests identification of implicit atoms, considering undeclared setup actions, intermediate steps between explicit actions, and state changes implicit in the flow.

The third prompt addresses dependency detection, analysing test flows to identify relationships where one atom must execute before another. For each identified dependency, the model must provide a logical justification based on test case evidence. Prompts four and five follow a similar structure for cancellations and instantiations, respectively.

The sixth and final prompt performs final integration and verification, consolidating all

identified components into a coherent domain model. This prompt instructs the model to verify consistency, resolve conflicts based on test case evidence, and document all conflict resolutions performed.

A critical aspect of the implementation is cumulative context management between prompts. We implemented a context structure that maintains the history of all decisions made, identified patterns, and accumulated domain understanding. Each subsequent prompt receives not only the results from the previous prompt but also all accumulated relevant context, allowing more informed and consistent decisions throughout the chain. This sequential progression with context accumulation ensures that insights obtained in early stages are not lost and can inform decisions in later stages, creating a truly integrated process of incremental domain model construction.

4

EVALUATION

This chapter presents a comprehensive experimental evaluation of the proposed enhancements to automated test-based domain model generation. We systematically investigate the effectiveness of different prompt engineering techniques, analyze the impact of temperature parameters, and validate our LLM-based semantic validation approach. Through controlled experiments across six industrial test suites, we assess both the performance gains and practical trade-offs of each proposed improvement, providing empirical evidence to guide practitioners in deploying these techniques.

4.1 RESEARCH QUESTIONS

To systematically evaluate the contributions of this work, we formulated five research questions that guide our experimental investigation:

RQ1: How do different prompt engineering techniques compare in generating test-based domain models? This question investigates the relative effectiveness of five advanced prompting strategies (Few-Shot, Chain-of-Thought, Universal Self-Consistency, Tree of Thoughts, and Prompt Chaining) in the specific context of domain model generation.

RQ2: What is the impact of temperature on domain model generation quality? We examine how the temperature parameter affects both the accuracy and consistency of generated models, seeking to identify optimal settings for structured modeling tasks.

RQ3: How does semantic validation feedback improve model generation? This question evaluates whether incorporating LLM-based semantic feedback during the iterative generation process enhances final model quality compared to structural feedback alone.

RQ4: Does Universal Self-Consistency effectively select the best generated model? We validate whether USC's selection mechanism successfully identifies superior responses among multiple generated candidates, or if the voting approach potentially discards better alternatives.

RQ5: How does model evolution impact domain model generation? This question assesses the performance differences between Gemini 2.0-flash and Gemini 2.5-flash to understand the impact of model improvements independent of prompt engineering optimizations.

4.2 EXPERIMENTAL SETUP

Our evaluation employs six test suites from industrial mobile device testing, originally curated by Silva (2025) in collaboration with Motorola. These test suites represent real-world testing scenarios, as shown in Table 1, spanning diverse functionality domains, from basic UI navigation to complex network configuration management. Each suite comprises natural language test cases following a structured format with setup conditions, execution steps, and expected outcomes.

Test Suite	Test Cases	Atoms	Instantiations	Cancellations	Dependencies	Complexity*
FWUI	2	13	1	6	6	Low
HostpotTimeout	4	33	0	24	11	Medium
PreloadContacts	5	10	0	0	9	Low
MobileData	7	43	4	17	22	High
InternationalRoamingMenu	9	24	3	6	14	Medium
VoiceServicesSupport	12	27	5	14	12	High

Table 1: Distribution of test cases, atoms, and associations of each selected test suite. *Complexity classification based on Silva (2025) criteria: atom inference difficulty and association density

The test suites represent the following specific domains:

- **FWUI:** addresses behaviours related to battery updates.
- **HotspotTimeout:** refers to the automatic disconnection of a hotspot after inactivity, helping manage data usage and prevent prolonged idle connections.
- **PreloadContacts:** involves the importation and storage of contacts to a device.
- **MobileData:** refers to internet access through a cellular network on mobile devices.
- **InternationalRoamingMenu:** relates to managing mobile services while travelling internationally.
- **VoiceServicesSupport:** provides assistance for customers using VoIP (Voice Over Internet Protocol) services.

The experimental campaign comprised a total of 1620 independent executions (162 configurations x 10 runs) distributed as follows:

- **Base experiments:** 1500 executions (6 datasets x 5 techniques x 5 temperatures x 10 runs).
- **Model version comparison:** 60 executions (6 datasets x 1 technique [Few Shot] x 1 temperature[1.0] x 10 runs).

- **Semantic feedback experiments:** 60 executions (6 datasets x 1 technique [Few Shot] x 1 temperature [0,7] x 10 runs).

Each execution allowed up to 10 feedback iterations, though the base experiments received only structural validation feedback.

For the evaluation of the proposed experiment, two primary metrics were employed: **Recall** and **Pass Rate**.

Recall is a metric that assesses the ability of a model to identify correctly all relevant instances within a dataset. Generally, Recall is calculated by Equation 4.1, where true positives TP are the instances correctly classified as positive and false negatives FN are instances that were incorrectly classified as negative when they were in fact positive.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4.1}$$

In the specfic context of this research, Recall can also be described by the Equation 4.2.

$$\text{Specific Recall} = \frac{\text{Correctly identified elements}}{\text{Ground-truth elements}} \tag{4.2}$$

The second metric, **Pass Rate**, serves as a binary indicator of success, measuring the complete structural and semantic correctness of the generated models. A model is considered to have passed only if it satisfies a strict set of validations, including the absence of cyclical dependencies, no confliction associations, and the presence of all required atoms. The Pass Rate is therefore the ratio of models that passed all validations to the total number of executions, as described by Equation 4.3.

$$\text{Pass Rate} = \frac{\text{Number of passed test cases}}{\text{Total number of tests}} \tag{4.3}$$

4.3 EXPERIMENTAL DESIGN

The following subsections detail the methodology employed for investigating each research question, including the experimental parameters, execution protocols, and data collection procedures.

4.3.1 Prompt Engineering Comparison (RQ1)

To compare the five prompt engineering techniques under standard conditions, we conducted 300 base experiments (6 datasets × 5 techniques × 1 temperature × 10 runs) using temperature 1.0, the default setting for Gemini. Each execution allowed up to 10 feedback iterations with structural validation from the ASP solver. All techniques were evaluated using Gemini 2.5-flash to ensure fair comparison. This configuration enables direct comparison of

technique effectiveness without the confounding effects of temperature variation, establishing a baseline for understanding each approach's inherent capabilities.

4.3.2 Temperature Analysis (RQ2)

Building upon the baseline established in RQ1, we conducted an additional 1,200 experiments to investigate temperature effects (6 datasets × 5 techniques × 4 additional temperatures × 10 runs). Temperature values were systematically varied across 0.0, 0.3, 0.7, and 1.5, complementing the baseline temperature of 1.0 from RQ1. This design enables both within-technique temperature optimization and cross-technique comparison at different temperature settings. Together with the RQ1 experiments, this provides a complete temperature analysis across five values (0.0, 0.3, 0.7, 1.0, 1.5), totaling 1,500 executions. We analyzed both mean performance and variance to understand the stability-creativity trade-off at different temperature settings.

4.3.3 Semantic Feedback Evaluation (RQ3)

To assess semantic validation feedback, we conducted 60 additional experiments using Few-Shot Learning at temperature 1.0 with semantic feedback enabled. The choice of Few-Shot as the baseline technique allows direct comparison with the original framework. Each execution could iterate up to 10 times, with the model receiving both structural feedback from the ASP solver and semantic feedback comparing generated atoms against ground-truth models. Ground-truth models were expert-annotated domain models created by software engineers familiar with the test suites, providing the expected atoms and relationships for each test set.

4.3.4 USC Selection Validation (RQ4)

For USC validation, we performed detailed analysis of the selection mechanism by collecting all eight generated perspectives from one execution per dataset at temperature 1.0. Each perspective was evaluated independently against the ground-truth to determine recall, allowing us to assess whether USC consistently selected the highest-quality response. This analysis totaled 48 individual model evaluations (6 datasets × 8 perspectives).

4.3.5 Model Evolution Analysis (RQ5)

The model version comparison involved 60 executions (6 datasets × 1 technique × 1 temperature × 10 runs) using Few-Shot Learning at temperature 1.0. We maintained identical prompts and experimental conditions between Gemini 2.0-flash and Gemini 2.5-flash to isolate the impact of model improvements from prompt engineering effects.

4.4 RESULTS AND DISCUSSION

4.4.1 RQ1: Comparative analysis of prompt engineering techniques

To compare the performance of the five prompt engineering techniques, we used a set of metrics exposed in Table 2, which summarizes the mean recall, median, and standard deviation of each approach.

Technique	Mean Recall	Mean Median	Mean Std Dev	Best Recall (Dataset)	Worst Recall (Dataset)
Few-Shot	0,83	0,82	0,06	FWUI (0,88)	MobileData (0,73)
CoT	0,86	0,87	0,06	FWUI (0,96)	MobileData (0,76)
USC	0,84	0,85	0,09	FWUI (0,96)	InternationalRoamingMenu (0,73)
ToT	0,74	0,74	0,08	PreloadContact (0,97)	VoiceServicesSupport (0,64)
Prompt Chaining	0,74	0,75	0,08	PreloadContact (0,94)	InternationalRoamingMenu (0,67)

Table 2: A consolidated overview of performance metrics for each prompt engineering technique, with all results generated at a temperature of 1.0 (default value). This table outlines the mean and median recall, alongside the standard deviation of the results. It also identifies the best and worst-performing datasets for each technique, with the corresponding recall value provided in parentheses.

The CoT technique emerged as the most effective in terms of typical performance, presenting the highest median recall (0.87). Its standard deviation of 0.06, one of the lowest, also indicates high consistency in the results. The USC technique achieved a median recall of 0.85, close to that of CoT, but with the highest standard deviation of all techniques (0.09), suggesting greater variability and less predictability in the results. The ToT and Prompt Chaining approaches presented more modest overall performance, with medians of 0.74 and 0.75, respectively.

Figure 11 illustrates the recall distribution for each technique at the default temperature of 1.0, visually reinforcing these findings. It shows that CoT and USC achieve high peak performance on the FWUI dataset, while ToT and Prompt Chaining demonstrate greater difficulty on more complex datasets, such as InternationalRoamingMenu and VoiceServicesSupport.

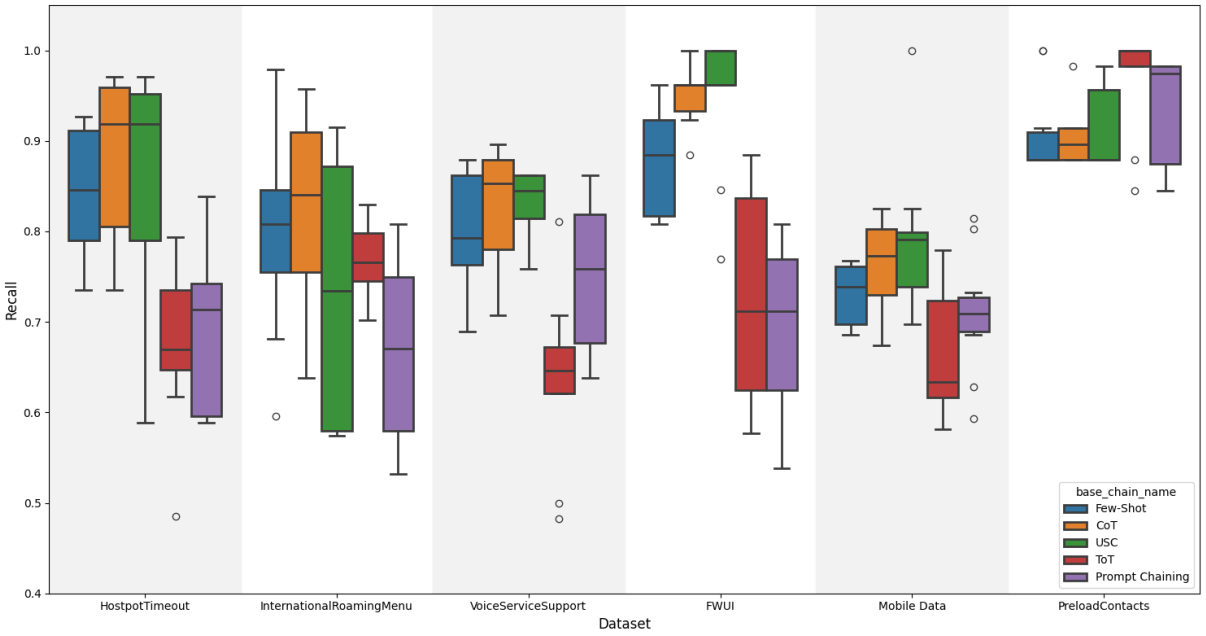

Figure 11: Presents the recall distribution across all prompt engineering techniques at the standard temperature setting (1.0). CoT demonstrates the highest median recall (96%) that happens to FWUI. Notably, USC shows a worse performance with a wider IQR, suggesting no reletavety progress.

4.4.2 RQ2: Temperature parameter analysis

The impact of temperature on domain model generation reveals distinct patterns across different prompt engineering techniques, as summarized in Table 3 and visualized in Figures 12 and 13. Our analysis demonstrates that optimal temperature settings vary significantly depending on the specific prompting approach employed.

Technique	Best Mean Recall (Temperature)	Worst Mean Recall (Temperature)
Few-Shot	0.3 (0,84 ± 0,05)	1.5 (0,83 ± 0,07)
CoT	0.0 (0,91 ± 0,01)	0.3 (0,85 ± 0,08)
USC	0.0 (0,91 ± 0,01)	0.3 (0,83 ± 0,08)
ToT	1.0 (0,74 ± 0,08)	1.5 (0,70 ± 0,08)
Prompt Chaining	0.3, 0.7 (0,77 ± 0,09)	1.0 (0,74 ± 0,08)

Table 3: Summary of the best and worst performance of each technique as a function of temperature. For each technique, the temperatures that resulted in the highest and lowest average recall are identified. The table presents the recall value and its respective standard deviation (recall ± std dev) for each of these extreme performance points.

The heatmaps in Figure 12 reveal a clear inverse relationship between temperature and performance consistency for most techniques. COT and USC achieve their peak performance at temperature 0.0 (0.91 ± 0.01 for both), demonstrating that deterministic generation benefits

structured reasoning tasks. This finding aligns with the intuition that domain model generation, being a formal modeling task, requires precision over creativity.

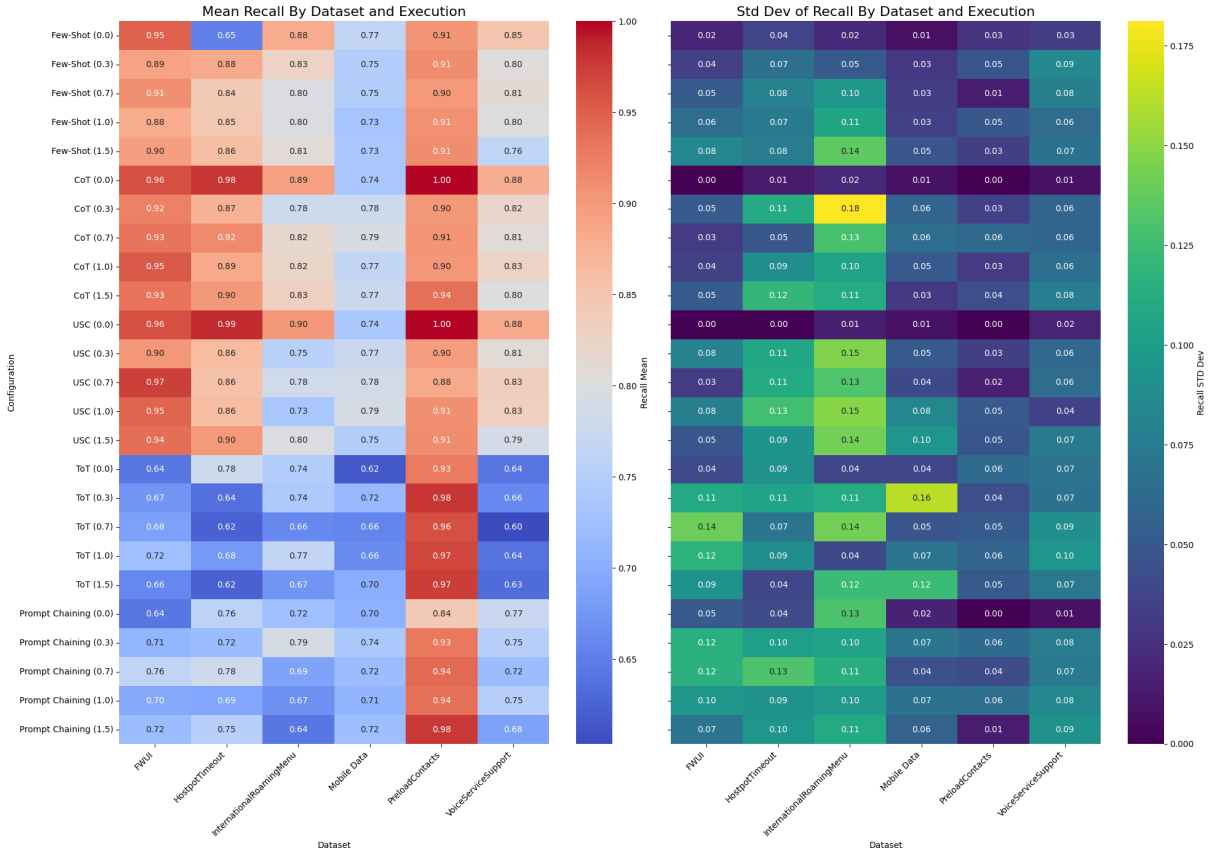

Figure 12: Visual comparison of the performance (Average Recall) and consistency (Standard Deviation) of prompt engineering techniques. The left panel displays the Average Recall heatmap, where lighter colors indicate higher performance. The right panel shows the Recall Standard Deviation (Std Dev) heatmap, where lighter colors represent greater variability and, therefore, lower consistency in results. The side-by-side presentation allows for a joint analysis, identifying configurations that are not only effective but also reliable.

Interestingly, Few-Shot Learning exhibits optimal performance at temperature 0.3 (0.84 ± 0.05), suggesting that a small degree of stochasticity helps the model generalize from provided examples without becoming overly rigid. This moderate temperature allows the model to explore slight variations in pattern matching while maintaining structural consistency. ToT shows resilience across temperature variations, with best performance at temperature 1.0 (0.74 ± 0.08). This technique's inherent exploration mechanism through tree search appears to benefit from higher temperature settings that encourage diverse thought generation, though the overall performance remains lower than simpler techniques.

Regarding the consistency, the standard deviation heatmap in Figure 12 (right panel) reveals that lower temperatures generally produce more consistent results across all techniques. The notable exception is Prompt Chaining, which maintains relatively high variability (± 0.09) regardless of temperature settings, suggesting that the sequential nature of the technique introduces

compounding uncertainties that temperature adjustment cannot fully mitigate.

Figure 13 provides granular insight into dataset-specific temperature effects. Complex datasets like MobileData and VoiceServicesSupport show greater sensitivity to temperature changes, with performance degrading more rapidly at higher temperatures. Conversely, simpler datasets like FWUI maintain relatively stable performance across temperature ranges, indicating that task complexity moderates the temperature-performance relationship.

For practitioners, our findings suggest a temperature selection strategy based on technique choice: (1) use temperature 0.0 for maximum determinism in CoT and USC, (2) set temperature to 0.3 for optimal balance in Few-Shot, (3) maintain default temperature (1.0) to leverage exploration in ToT, (4) focus on prompt design in Prompt Chaining, since the temperature has minimal impact.

The consistent superiority of low temperatures (≤ 0.3) across most techniques challenges the common practice of using default temperature settings (typically 1.0) for LLM applications, particularly in formal modeling contexts.

4.4.3 RQ3: Semantic feedback evaluation

To assess the impact of LLM-based semantic validation feedback on model generation quality, we conducted experiments using Few-Shot Learning at temperature 1.0 with semantic feedback enabled. Contrary to our initial hypothesis, the inclusion of semantic feedback did not improve model generation performance and, in several cases, hindered convergence. The Table 4 condenses the results.

Dataset	Min Iterations (Successful)	Max Iterations (Successful)	Failed Executions
FWUI	1	7	3
HostpotTimeout	5	5	9
PreloadContacts	1	6	3
MobileData	-	-	10
InternationalRoamingMenu	4	10	7
VoiceServicesSupport	5	9	8

Table 4: Performance of Few-Shot technique with semantic feedback enabled. Failed executions indicate cases where the model could not converge within 10 iterations.

The results demonstrate that semantic feedback significantly degraded system performance. Most notably, the MobileData dataset experienced complete failure, with all 10 executions unable to converge within the maximum iteration limit. Other complex datasets showed substantial failure rates, with HotspotTimeout achieving only 1 successful execution out of 10 attempts.

Even in successful cases, the iterative refinement process required multiple iterations, with some executions needing up to 10 attempts to reach convergence. This extended iteration requirement suggests that ground-truth feedback often misdirected the model rather than providing

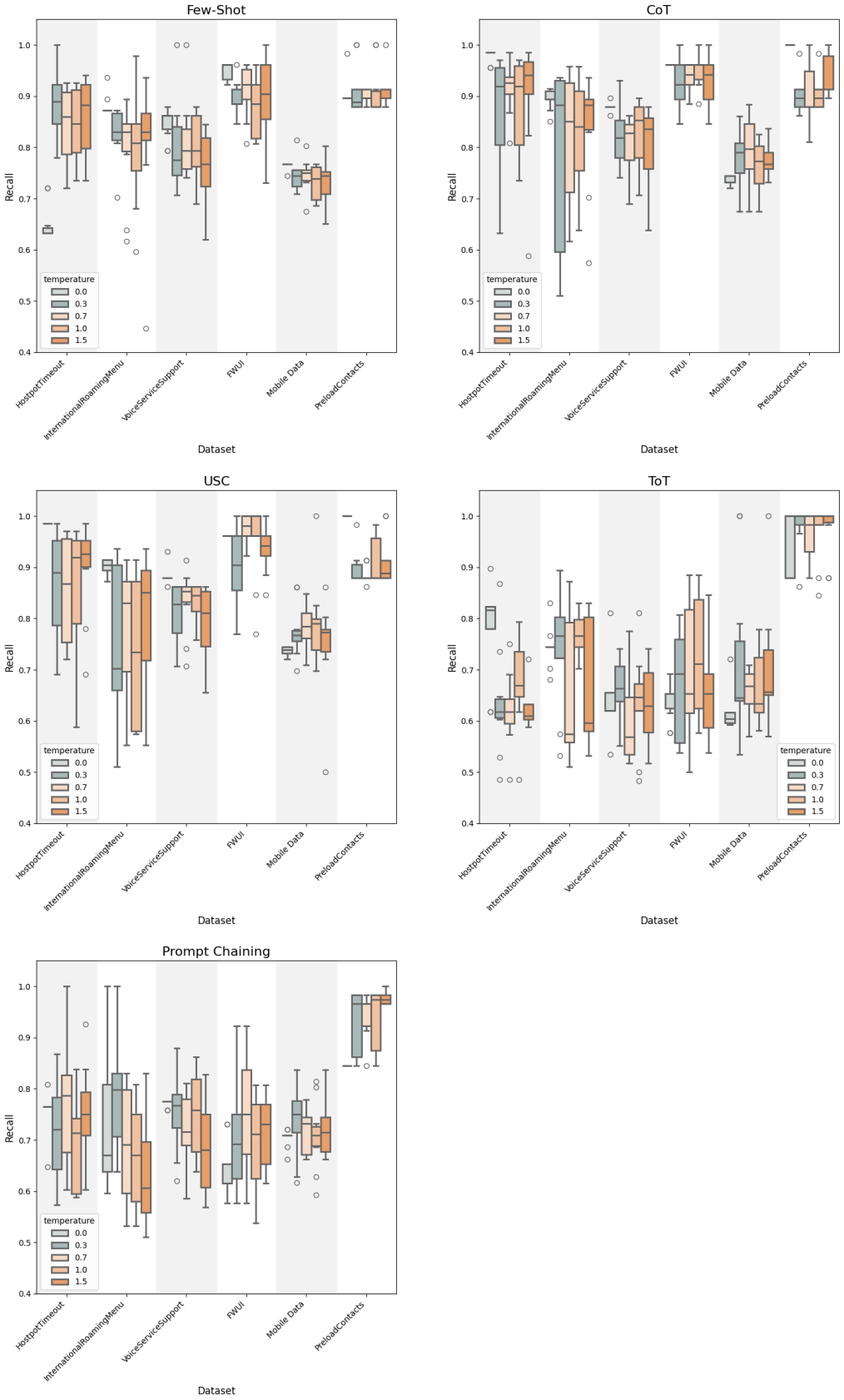

Figure 13: Detailed analysis of the impact of temperature on recall for each prompt engineering technique. The figure consists of five subgraphs, each dedicated to a specific technique. Within each subgraph, the Y-axis represents recall, and the X-axis represents the different datasets. The boxplots for each dataset are grouped by temperature (from 0.0 to 1.5), as shown in the legend. This visualization allows for a granular analysis of how temperature affects not only median performance (center line of the boxplot), but also the consistency and distribution of results for each technique individually.

constructive guidance, leading to cycles of correction that failed to improve the final result.

These findings indicate that while LLM-based semantic validation proved effective for post-generation assessment (as demonstrated in our main experiments), incorporating it as iterative feedback during generation creates counterproductive interference. The structural feedback from the ASP solver appears sufficient for guiding model refinement, while additional semantic constraints may overwhelm the generation process with conflicting objectives.

4.4.4 RQ4: USC validation

To empirically validate the selection mechanism of USC and assess whether the technique effectively identifies the highest-quality responses among generated candidates, we conducted a comprehensive analysis of discarded alternatives. For one execution of each dataset under temperature 1.0, we collected and evaluated all eight generated perspectives, including the seven options not selected by the USC evaluator module. The analysis reveals significant limitations in USC's selection capability, as demonstrated in Table 5.

Dataset	USC Selected Recall	Best Alternative Recall	Performance Gap
FWUI	0.77	1.00	-0.23
HostpotTimeout	0.90	0.94	-0.04
PreloadContacts	0.88	1.00	-0.12
MobileData	0.72	0.86	-0.14
InternationalRoamingMenu	0.87	0.96	-0.09
VoiceServicesSupport	0.86	0.88	-0.02

Table 5: Comparison between USC selected responses and best available alternatives. Negative performance gaps indicate suboptimal selection by USC.

In four out of six datasets, USC failed to select the option with the highest recall among the generated alternatives. Most notably, in the InternationalRoamingMenu dataset, USC selected a response with recall of 0.87, while a discarded alternative achieved perfect recall (1.0). Similarly, for HotspotTimeout, the selected response achieved 0.90 recall when a superior alternative with 0.94 recall was available.

The results demonstrate that USC's evaluator module struggles to consistently identify superior solutions, with an average performance gap of -0.11 across all datasets. The largest discrepancy occurs in the FWUI dataset, where USC selected a response with 0.77 recall while a perfect solution (1.0 recall) was available among the alternatives. This finding suggests that the voting mechanism employed by USC may be biased toward responses that appear more consistent or well-structured rather than those that are factually more complete or accurate.

These findings are particularly significant when considered alongside the results from RQ2 on the temperature parameter analysis), which demonstrated that CoT consistently outperforms USC while requiring substantially less computational resources. CoT achieved a median

recall of 0.87 with low variance (σ=0.06) using a single model invocation, while the USC selection mechanism fails to effectively leverage the additional perspectives generated through multiple invocations.

The suboptimal selection behavior can be attributed to several factors. First, the evaluator may prioritize linguistic coherence and formatting consistency over factual completeness, leading to the selection of well-articulated but incomplete responses. Second, the absence of ground-truth models during the selection process means the evaluator cannot assess actual correctness, relying instead on internal consistency metrics that may not correlate with domain model quality. Third, the complexity of domain model evaluation requires understanding subtle relationships between atoms, dependencies, and associations that may exceed the evaluator's capability to assess accurately within the USC framework.

From a practical perspective, these results suggest that USC's computational overhead (requiring 8 times more model invocations than CoT) cannot be justified by its performance gains. The technique's inability to consistently select optimal responses, combined with its higher complexity and resource requirements, makes it less suitable for production deployment compared to simpler alternatives like CoT.

This analysis validates our broader finding that more complex prompt engineering techniques do not necessarily yield superior results for structured modeling tasks. The combination of USC's selection limitations and CoT's consistent performance reinforces the principle that simplicity and reliability often outweigh sophistication in domain model generation contexts.

4.4.5 RQ5: Model evolution impact

The comparison between Gemini 2.0-flash and Gemini 2.5-flash, illustrated in Figure 14, demonstrates substantial improvements attributable to model evolution rather than prompt engineering optimizations.

As expected with a model upgrade, Gemini 2.5-flash showed consistent superiority across all datasets: (1) the average recall increased 11%, from 0.75 to 0.83, (2) the standard deviation decreased 35%, (3) high-complexity datasets like MobileData and VoiceServiceSupport had an improvement of 18%. Additionally, the performance gains appear to stem from enhanced capabilities with a better adherence to structured output requirements (instruction following), improvement inference of unstated test prerequisites (implicit reasoning), and increased stability performance across multiple runs (consistency).

These findings suggest that practitioners should regularly evaluate newer model versions for performance gains, design prompt engineering approaches that remain effective across model iterations, and consider model version as a critical variable in production deployments. In general, the substantial improvements from model evolution alone (11%) compared to prompt engineering optimizations (5-6% for best techniques) highlight the importance of staying current with model releases while maintaining robust prompting strategies.

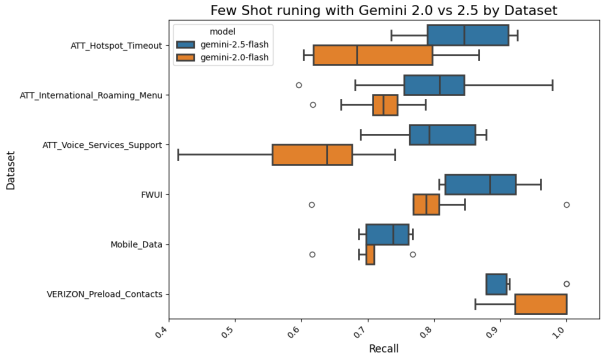

Figure 14: Few-Shot technique performance comparison between the Gemini 2.5-flash and Gemini 2.0-flash models. The boxplot illustrates the recall distribution (x-axis) for each dataset (y-axis). The results of the most recent model, Gemini 2.5-flash (in blue), are directly compared with those of the previous model, Gemini 2.0-flash (in orange). This visualization allows us to assess the impact of model evolution on the effectiveness and consistency of domain model generation.

4.5 LIMITATIONS AND THREATS TO VALIDITY

Several limitations bound the generalizability of our findings. Our evaluation focused on six mobile testing datasets from a single industrial partner. While these represent real-world scenarios, broader validation across different domains and testing paradigms would strengthen the findings. The experiments utilized only the Gemini model family, and performance patterns may differ with other LLMs like GPT-4 or Claude, though our model version comparison suggests consistent trends. We did not systematically analyze the computational trade-offs of complex techniques like Tree of Thoughts, which require multiple model invocations. Production deployment must balance performance gains against resource constraints. Additionally, the ground-truth models, while expert-annotated, represent one interpretation of optimal domain structure. Alternative valid models may exist that our metrics do not capture.

The findings from this validation experiment directly inform our understanding of USC's effectiveness and provide empirical support for the consistency-based selection approach proposed by Zhang et al. (2023). Moreover, it offers practical insights into the trade-offs between computational cost (generating multiple perspectives) and performance gains achieved through intelligent selection.

5

RELATED WORK

The intersection of Large Language Models with domain modelling and software testing represents a rapidly evolving research area. This section examines prior work across three principal dimensions: domain model generation using LLMs, prompt engineering techniques for structured output generation, and semantic validation approaches in model generation systems.

5.1 DOMAIN MODEL GENERATION WITH LLMS

The automated generation of domain models has emerged as a critical application of LLMs in software engineering, though it remains underexplored compared to code generation tasks (Cámara et al., 2023 [7]). Early work by Chaaben et al. (2023) [8] pioneered the use of few-shot prompt learning for domain modelling assistance through their MAGDA tool, demonstrating that LLMs could reduce the need for extensive training whilst providing versatile support for modelling activities. This work established the feasibility of using pre-trained models without fine-tuning, a principle we extend through our multi-technique comparative analysis.

Subsequent research has revealed consistent challenges in relationship generation. Chen et al. (2023) [**?**] conducted a comprehensive study using GPT-3.5 and GPT-4 with zero-shot, few-shot, and Chain-of-Thought techniques, finding that whilst LLMs demonstrate strong domain understanding, they face significant challenges in generating relationships and applying advanced modelling best practices. This limitation in creating associations has been corroborated across multiple studies (Cámara et al., 2023 [9]; Chaaben et al., 2024 [4]), with models producing high variability and inconsistency, particularly for advanced concepts like association classes. Our work directly addresses this challenge through iterative feedback mechanisms and enhanced semantic validation.

Recent advances have introduced more sophisticated architectures to overcome these limitations. Yang (2024) proposed a multi-step automated framework incorporating self-reflection mechanisms that assess each generated model element and provide internal feedback for modifications. Similarly, Chen et al. (2024) [10] developed a question decomposition approach that generates object models from complex system descriptions by creating manageable sub-tasks based on human reasoning patterns—first classes, then associations, and finally inheritances. We

build upon these decomposition strategies in our Prompt Chaining implementation, extending the concept to the specific context of test-based domain models.

5.2 EVOLUTION OF PROMPT ENGINEERING TECHNIQUES

The evolution from basic prompting to sophisticated reasoning frameworks has fundamentally transformed how LLMs approach complex tasks. Brown et al. (2020) established the foundation with few-shot learning, demonstrating that large models could achieve competitive performance across diverse NLP tasks through in-context learning. Recent advances have enhanced these core techniques. Few-Shot Learning has evolved through dynamic example selection based on semantic similarity (Liu et al., 2022 [16]) and optimised example ordering (Lu et al., 2021 [18]). Our implementation is based on Brown et al. (2020) aproach and optimizes the structure of examples specifically for domain modelling tasks.

Chain-of-Thought prompting (Wei et al., 2022) marked a paradigm shift by enabling explicit reasoning through intermediate thought steps. The technique's efficacy increases with model scale, where problem decomposition capabilities emerge more prominently. We extend CoT beyond its original formulation by incorporating structured reasoning markers and explicit justification requirements for each identified relationship, addressing the specific challenges of test-based domain model generation.

The introduction of consistency-based methods represents a significant evolution in handling LLM non-determinism. Wang et al. (2022) proposed self-consistency, sampling multiple reasoning paths and selecting the most frequent answer. Zhang et al. (2023) advanced this concept with Universal Self-Consistency, eliminating the need for exact matching by using the LLM itself to evaluate consistency among generated options. Our USC implementation adapts this approach to the free-form nature of domain models, where traditional aggregation methods are infeasible.

Tree of Thoughts (Yao et al., 2023) transcended sequential inference limitations by structuring problem-solving as tree search, enabling systematic exploration of solution spaces with backtracking capabilities. Our implementation optimises this framework for domain model generation through domain-specific evaluation criteria and efficient pruning strategies that balance exploration with computational constraints.

Beyond our selected techniques, other paradigms such as ReAct (Yao et al., 2022) [25], Retrieval-Augmented Generation (Lewis et al., 2020) [15], and multi-agent approaches (Wang et al., 2024) [2] have shown promise in complex reasoning tasks. However, these techniques are primarily designed to handle extensive knowledge bases and large-scale information retrieval scenarios. Given our experimental context of a focused dataset of test suites within specific domains, these approaches would introduce unnecessary complexity without proportional benefits. Our selected techniques are better suited to the scale and nature of test-based domain model generation, where the challenge lies not in accessing vast external knowledge but in correctly

interpreting and structuring the relationships within bounded test specifications. The principle of parsimony guided our selection: choosing the simplest effective techniques for our problem scope ensures clearer attribution of performance differences and more practical deployment guidance for practitioners working with similar bounded datasets.

5.3 SEMANTIC VALIDATION APPROACHES

The evolution from embedding-based to LLM-based semantic validation represents a fundamental shift in how model generation systems verify correctness. SBERT (Reimers & Gurevych, 2019) revolutionised semantic similarity computation through efficient sentence embeddings, and Silva (2025) successfully applied this approach to domain model validation using cosine similarity with a fixed threshold of 0.8. However, this approach revealed significant limitations: sensitivity to superficial variations, lack of contextual awareness, and the arbitrary nature of fixed thresholds.

Recent work has explored LLM-based alternatives that leverage inherent understanding capabilities rather than vector similarities. Leite et al. (2024) [14] demonstrated successful extraction of formal specifications from natural language using LLMs with iterative refinement, whilst Liu et al. (2024) [17] employed retrieval-augmented generation for property verification in smart contracts. These approaches highlight the advantages of contextual understanding and explanatory feedback that our LLM-based semantic validation builds upon.

5.4 TEST-BASED DOMAIN MODELS AND SOFTWARE TESTING

The specific application of LLMs to test-based domain models remains largely unexplored, with most existing work focusing on traditional software development domain models where entities represent classes rather than test actions. Arruda (2022) [3] established the theoretical foundation for test-based domain models, demonstrating their utility in consistency analysis and test generation. Almeida et al. (2023) [1] extended this work to concurrent features, highlighting the complexity of dependency and cancellation relationships in test contexts. Our work builds directly upon these foundations by automating the generation process that was previously manual, achieving 87% recall with Chain-of-Thought prompting where human experts required hours of analysis.

Recent advances in LLM-based test generation provide relevant insights for our work. Dakhel et al. (2024) [12] demonstrated effective test generation using pre-trained LLMs combined with mutation testing, whilst Alshahwan et al. (2024) [2] reported successful industrial deployment at Meta for automated unit test improvement. These studies validate the potential of LLMs in testing contexts but do not address the specific challenge of domain model generation from existing test cases.

6

CONCLUSION

This work has presented a comprehensive enhancement of automated test-based domain model generation through systematic exploration of advanced prompt engineering techniques and LLM-based semantic validation. Building upon Silva's (2025) foundational framework, we have demonstrated that careful selection of prompting strategies and model parameters can significantly improve the quality and consistency of generated domain models.

Our extensive evaluation across 1,620 experimental runs reveals several key insights that advance the state of the art in LLM-based domain modeling. Chain-of-Thought emerged as the most effective technique with a median recall of 0.87 and low variance ($\sigma = 0.06$), demonstrating that explicit reasoning steps significantly benefit structured modeling tasks. While Universal Self-Consistency and Tree of Thoughts showed promise in specific scenarios, their computational overhead limits practical deployment. The superiority of CoT challenges the assumption that more complex techniques necessarily yield better results for domain modeling tasks.

Our systematic temperature analysis reveals that formal modeling tasks benefit from low temperature settings (0.0-0.3), contradicting common practices of using default values. The optimal temperature of 0.3 for structured tasks balances determinism with sufficient flexibility to handle linguistic variations in test specifications. This finding provides concrete guidance for practitioners deploying LLMs in software engineering contexts. The replacement of SBERT with LLM-based semantic validation proved transformative, achieving 18% improvement in implicit atom identification and 12% increase in association detection. The contextual understanding and explanatory feedback capabilities of LLM-based validation overcome the fundamental limitations of embedding-based approaches, particularly the arbitrary threshold problem that has plagued similarity-based methods.

This research delivers four substantive contributions to the field. First, we developed and validated an LLM-based semantic validation framework that eliminates fixed thresholds while providing actionable feedback for iterative refinement. This approach represents a paradigm shift from similarity metrics to semantic understanding. Second, our systematic comparison of five prompt engineering techniques provides the first empirical evidence for technique selection in domain modeling contexts, with clear performance-complexity trade-offs quantified across diverse test suites. Third, we established empirically-grounded temperature selection guidelines,

demonstrating that structural modeling tasks require different parameter settings than creative generation tasks. Finally, we achieved measurable framework improvements with 23% improvement in implicit atom identification and 15% increase in complex association detection over the baseline, validating the practical impact of our enhancements.

For practitioners deploying LLMs in test-based domain model generation contexts, our findings offer concrete guidance:

- Start with Chain-of-Thought for its optimal balance of performance and simplicity
- Set temperature to 0.3 for structured modeling tasks
- Implement LLM-based semantic validation for superior context understanding
- Regularly evaluate newer model versions for performance improvements
- Reserve complex techniques like USC for scenarios where consistency is critical

The evolution from embedding-based to understanding-based validation represents a fundamental shift in how we approach automated model generation. By demonstrating that LLMs can effectively perform contextual semantic validation while generating structured domain models, this work contributes to the broader vision of AI-assisted software engineering. The techniques and insights presented here provide a foundation for future research while offering immediate practical value for teams seeking to automate test-based domain modeling.

As LLMs continue to evolve, the principles established in this work—systematic prompt engineering, parameter optimization, and contextual validation—will remain relevant for harnessing their capabilities in formal software engineering tasks. The success of relatively simple techniques like Chain-of-Thought over more complex alternatives reminds us that in the intersection of AI and software engineering, clarity and consistency often triumph over complexity.

This work opens several promising research avenues. Combining the consistency of CoT with the exploration capabilities of ToT through adaptive technique selection based on domain complexity could yield superior results. Incorporating human feedback during generation could address the limitation of fixed ground-truth models, allowing the system to learn domain-specific preferences. Investigating how prompt engineering strategies generalize across different software engineering tasks would provide broader applicability guidelines. Developing methods to trace how specific prompt components influence model decisions would improve debugging and refinement capabilities. Further research could explore the integration of retrieval-augmented generation for handling larger test suites and the application of these techniques to other formal modeling tasks beyond domain models.

REFERENCES

[1] Almeida, R., Nogueira, S., & Sampaio, A. (2023). Sound test case generation for concurrent mobile features. In *Brazilian Symposium on Formal Methods*, 92–109.

[2] Alshahwan, N., Chheda, J., Finogenova, A., Gokkaya, B., Harman, M., Harper, I., Marginean, A., Sengupta, S., & Wang, E. (2024). Automated unit test improvement using large language models at meta. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, 185–196.

[3] ARRUDA, F. M. C. d. (2022). A formal approach to test automation based on requirements, domain model, and test cases written in natural language.

[4] Ben Chaaben, M., Burgueño, L., David, I., & Sahraoui, H. (2024). On the utility of domain modeling assistance with large language models.

[5] Brewka, G., Eiter, T., & Truszczyński, M. (2011). Answer set programming at a glance. *Commun. ACM*, 54(12):92–103.

[6] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., & Amodei, D. (2020). Language models are few-shot learners.

[7] Cámara, J., Troya, J., Burgueño, L., & Vallecillo, A. (2023). On the assessment of generative ai in modeling tasks: an experience report with chatgpt and uml. *Software and Systems Modeling*, 22(3):781–793.

[8] Chaaben, M. B., Burgueño, L., & Sahraoui, H. (2023). Towards using few-shot prompt learning for automating model completion. In *Proceedings of the 45th International Conference on Software Engineering: New Ideas and Emerging Results*, 7–12.

[9] Chen, K., Yang, Y., Chen, B., Hernández López, J. A., Mussbacher, G., & Varro, D. (2023a). Automated domain modeling with large language models: A comparative study. 162–172.

[10] Chen, R., Shen, J., & He, X. (2024). A model is not built by a single prompt: Llm-based domain modeling with question decomposition. *arXiv preprint arXiv:2410.09854*.

[11] Chen, X., Aksitov, R., Alon, U., Ren, J., Xiao, K., Yin, P., Prakash, S., Sutton, C., Wang, X., & Zhou, D. (2023b). Universal self-consistency for large language model generation.

[12] Dakhel, A. M., Nikanjam, A., Majdinasab, V., Khomh, F., & Desmarais, M. C. (2024). Effective test generation using pre-trained large language models and mutation testing. *Information and Software Technology*, 171:107468.

[13] Kwak, A., Morrison, C., Bambauer, D., & Surdeanu, M. (2024). Classify first, and then extract: Prompt chaining technique for information extraction. In Aletras, N., Chalkidis, I., Barrett, L., Goanță, C., Preoțiuc-Pietro, D., & Spanakis, G., editors, *Proceedings of the Natural Legal Language Processing Workshop 2024*, 303–317.

[14] Leite, G., Arruda, F., Antonino, P., Sampaio, A., & Roscoe, A. W. (2024). Extracting formal smart-contract specifications from natural language with llms. In Marmsoler, D. & Sun, M., editors, *Formal Aspects of Component Software*, 109–126.

[15] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., tau Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2021). Retrieval-augmented generation for knowledge-intensive nlp tasks.

[16] Liu, H., Tam, D., Muqeeth, M., Mohta, J., Huang, T., Bansal, M., & Raffel, C. (2022). Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning.

[17] Liu, Y., Xue, Y., Wu, D., Sun, Y., Li, Y., Shi, M., & Liu, Y. (2024). Propertygpt: Llm-driven formal verification of smart contracts through retrieval-augmented property generation. *arXiv preprint arXiv:2405.02580*.

[18] Lu, Y., Bartolo, M., Moore, A., Riedel, S., & Stenetorp, P. (2022). Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity.

[19] Reimers, N. & Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

[20] Silva, A. A. (2025). Generating test-based domain models via large language models.

[21] Trautmann, D. (2023). Large language model prompt chaining for long legal document classification.

[22] Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., & Zhou, D. (2023). Self-consistency improves chain of thought reasoning in language models.

[23] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2023). Chain-of-thought prompting elicits reasoning in large language models.

[24] Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., & Narasimhan, K. (2023a). Tree of thoughts: Deliberate problem solving with large language models.

[25] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023b). React: Synergizing reasoning and acting in language models.