Generating Test-Based Domain Models via Large Language Models

Andresa Almeida da Silva (aas10@cin.ufpe.br)

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

Recife
2025

Andresa Almeida da Silva (aas10@cin.ufpe.br)

Generating Test-Based Domain Models via Large Language Models

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)*
Co-Advisor: *Filipe Arruda (fmca@cin.ufpe.br)*

Recife
2025

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

Silva, Andresa Almeida da.
Generating Test-Based Domain Models via Large Language Models / Andresa Almeida da Silva. - Recife, 2025.
50 p. : il., tab.

Orientador(a): Augusto Cezar Alvez Sampaio
Cooorientador(a): Filipe Marques Chaves de Arruda
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Engenharia da Computação - Bacharelado, 2025.
Inclui referências.

1. Domain Model. 2. LLM. 3. Software Testing. I. Sampaio, Augusto Cezar Alvez. (Orientação). II. Arruda, Filipe Marques Chaves de. (Coorientação). IV. Título.

000 CDD (22.ed.)

ANDRESA ALMEIDA DA SILVA

GENERATING TEST-BASED DOMAIN MODELS VIA LARGE LANGUAGE MODELS

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: 11/04/2025

ACKNOWLEDGEMENTS

Esses agradecimentos são para todas as pessoas que me permitiram ou me ajudaram a ser mais de uma coisa ao mesmo tempo.

Quando tinha dezessete anos, minha música favorita – *Sleep On The Floor* de *The Lumineers* – falava algo sobre empacotar uma escova de dentes e uma blusa favorita e sair da cidade se quisesse conseguir algo. Então para minha família, eu quero agradecer pelo esforço imensurável em ter permitido e me ajudado a sair de nossa pequena cidade com pouca idade e muitos sonhos em busca de algo maior. Eu tenho muito orgulho de cada um de vocês, lhes devo tudo que sou hoje e serei eternamente grata.

Aos meus amigos do ensino médio, vocês me deram esperança e conforto em tempos difíceis e me encorajaram a ser extraordiária sendo comum. Obrigada por terem me mostrado que ser sensível é uma das qualidades mais incríveis e únicas que uma pessoa pode ter. Para os meus amigos Bia, Sophia, Mari, Beza, Malu, Mavi, Clara, João, Jonga, PV, Matheus e Uanderson, obrigada por cada conversa e risada ao longo dos últimos anos que tornaram semestres complicados em grandes histórias.

Ao meu parceiro José Victor, obrigada por ser meu melhor amigo e meu porto seguro pelos últimos sete anos e por acreditar em mim todas as vezes que eu não consegui.

Ademais, quero expressar minha gratidão ao CIn por todas as oportunidades e a infraestrutura que permitiram minha educação. Eu não poderia estar mais satisfeita e orgulhosa de ter completado minha graduação em uma instituição de tamanha excelência. Sou profundamente agradecida a todos os professores que me guiaram por essa jornada e que foram essenciais para o meu crescimento profissional, acadêmico e pessoal.

Especialmente para o meu orientador Augusto Sampaio, obrigada por todo o conhecimento, paciência, apoio e oportunidades. Eu tenho um orgulho imenso de dizer que tive a honra de trabalhar e, principalmente, aprender contigo. Para Breno Miranda, obrigada por toda a confiança quando tudo que eu sabia era um "Hello, World" em C, sem seu apoio no início eu provavelmente seria metade da profissional que sou hoje. Agradeço também ao Projeto CIn/Motorola e ao professor Alexandre Mota por todo o suporte e investimento não somente nesse projeto, mas em todos que tive a chance de participar. Por fim, agradeço Rafaela Gonçalves e Filipe Arruda pelo enorme apoio ao longo do desenvolvimento desta pesquisa.

Para todos vocês, muito obrigada.

ABSTRACT

The generative artificial intelligence, particularly Large Language Models, has revolutionized various stages of the software life cycle, such as requirements elicitation, code generation, formal specification, and model creation. In software modelling, domain models are abstract representations of concepts, entities, and relationships within a specific problem domain, serving as a guide for software solutions and design. Considered a critical aspect of software development, it requires specialists with domain expertise and significant time resources, as it is typically a manual task that demands considerable effort. This work proposes a systematic approach to generate domain models for the software testing activity (test-based domain models) from natural language test cases using Gemini. We incorporate a set of rules with an ASP solver to ensure structural consistency and apply SBERT to validate semantic aspects of the generated models. We compare prompt engineering techniques and evaluate Gemini's performance in generating domain models with and without interaction cycles of feedback, and under the guidance of ground-truth domain models. The results indicate that Gemini's effectiveness in producing consistent, test-based domain models, is influenced by domain complexity and feedback, achieving 80-90% satisfiability in simpler domains, with feedback significantly improving model quality. However, our analysis shows that regardless of the approach, the LLM still encounters limitations when inferring associations such as dependency, instantiation, and cancellation.

Keywords: Domain Model. LLM. Software Testing.

RESUMO

A inteligência artificial generativa, particularmente os Modelos de Linguagem de Grande Escala (LLMs), revolucionou várias etapas do ciclo de vida do software, como elicitação de requisitos, geração de código, especificação formal e criação de modelos. Na modelagem de software, os modelos de domínio são representações abstratas de conceitos, entidades e relações dentro de um domínio específico, servindo como guia para soluções e design de software. Considerada uma parte crítica do desenvolvimento de software, essa tarefa exige especialistas com conhecimento no domínio e recursos significativos de tempo, já que geralmente é uma tarefa manual que demanda considerável esforço. Este trabalho propõe uma abordagem sistemática para gerar modelos de domínio para a atividade de testes de software (modelos de domínio baseados em testes) a partir de casos de teste em linguagem natural usando o Gemini. Incorporamos um conjunto de regras com um solucionador ASP para garantir consistência estrutural e aplicamos o SBERT para validar os aspectos semânticos dos modelos gerados. Comparamos técnicas de engenharia de prompts e avaliamos o desempenho do Gemini na geração de modelos de domínio com e sem ciclos de interação de feedback, e sob a orientação de modelos de domínio de referência. Os resultados indicam que a eficácia do Gemini na produção de modelos de domínio consistentes é influenciada pela complexidade do domínio e pelo feedback, alcançando 80-90% de satisfatibilidade em domínios mais simples, com o feedback melhorando significativamente a qualidade do modelo. No entanto, a análise mostra que, independentemente da abordagem, a LLM ainda enfrenta limitações na inferência de associações, como dependência, instanciação e cancelamento.

Palavras-chave: Modelo de Domínio. LLM. Testes de Software.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

The growing technological advances in artificial intelligence (AI) have allowed innovation in software life cycles through process optimization based on generative AI (GAI). Unlike other AIs, such as traditional machine learning, GAI focuses on generating new data from pattern identification in its training set rather than making predictions about a dataset [25].

Requirements elicitation [27, 17], formal specification generation [39, 24], code generation [31, 13], model generation [10, 9], and test generation [15, 4] are examples of development stages that have been significantly improved with the use of GAI, more specifically, with Large Language Models (LLMs). LLMs are generative AI models based on deep learning techniques designed to understand/generate human language and process large amounts of text data efficiently – reducing effort, complexity, and resources – in different domains [22].

One specific context where LLMs allow optimizations is in software modelling activity [12]. A model captures the important aspects of a domain based on a certain point of view while abstracting the remaining elements [33]. Often presented in the early stages of software development, the modelling activity involves the conceptual definition and representation of the entities and relationships about a specific problem. Within a shared domain, it operates as a domain model.

In the context of software testing, domain models arise as an association between actions that provide test engineers with concrete implementation details when creating test cases. Unlike requirements, that outline what should be implemented and tested, domain models focus on describing the specifics of how the system behaves [6].

A test-based domain model includes several relations between test actions. One such relation is dependency: the occurrence of an action A in a test case requires that all actions on which A depends must have previously occurred in this test case. For example, sending a message in a mobile phone domain requires some form of internet connection. Another relation is cancellation: an action might cancel the effect of a previous action. For instance, airplane mode cancels a Wi-Fi connection. More details are given in the next section.

These models allow testers to understand the interactions between various components and ensure that all relevant scenarios are covered. Clearly defining these relations enhances test case organization, reduces redundancy, and ensures comprehensive test coverage. Furthermore, these domain models can be employed to ensure the consistency of both individual test cases and test suites – a group of selected test cases targeting a specific domain or feature – by analysing and identifying dependencies, edge cases, and relationships within a test case, across test cases, and between test cases [5]. From a well-defined domain model, techniques such as creating sound test cases for concurrent features [2] or Model-Based Testing (MBT) can also be applied to increase test coverage by adhering to a better representation of complex software behaviours.

Nevertheless, software modelling is not trivial. It requires extensive domain knowledge and time resources since it is commonly acquired through requirements analysis and by specialists without an automated process. Besides, manual domain models can sometimes lead to uncovered edge cases or introduce potential errors due to the high cognitive level required to carry out the modelling without creating inconsistencies.

Despite some initial attempts previously cited [10, 9], the use of LLMs to support the automation of the modelling process still lacks further investigation, indicating a need for additional research and analysis. A significant limitation tied to this is the absence of large datasets necessary for pre-training or fine-tuning these models [10], requiring much more effort to achieve satisfying results. When particularly considering domain models for software testing, the availability of relevant information sources is even more scarce.

This study aims to propose and evaluate a systematic approach to generate domain models from test cases written in natural language supported by LLMs and, more specifically, Gemini [36]. The particular context is that of a long-term cooperation with Motorola Mobility, a Lenovo Company, on techniques for automatic test case generation for mobile devices. The reason for using test cases (rather than requirements) as input to the process is that our particular focus is on generating new, and more elaborate, test scenarios for combined mobile features from test cases for individual features. There is already some evidence that this technique enhances both code coverage and bug detection [2, 3].

The adoption of LLMs to generate test-based domain models is particularly relevant because this is the only manual step in the test generation process, which ends up discouraging test engineering from adopting the process in practice. Besides the use of LLM, we propose a set of rules to verify the test-based domain model's consistency using the ASP (Answer Set Programming) solver, identifying the potential inconsistencies. We develop some case studies to evaluate the proposed approach and provide some metrics on its capacity to automatically infer relations between test actions. This is achieved by comparing the results against ground-truth test-based domain models.

The next section provides some background on software testing, domain models, and LLMs. Section 3 presents the proposed framework for generating domain models via the Gemini LLM using test cases, with a rationale for the adopted prompt engineering process. Section 4 presents the research questions, assesses the findings, and briefly examines potential threats to validity and limitations. Section 5 provides an overview of this study's field and related work. Section 6 concludes this study with a summary of the contributions and future work.

2 BACKGROUND

2.1 TEST-BASED DOMAIN MODELS

A domain model is a conceptual representation of the entities, their attributes, and the relationships that exist within a specific domain. Although the modelling activity usually refers to the model creation of a specific system, this study is based on a concept of domain models defined specifically for software testing activity [6].

Software testing is one discipline of the software development life cycle that focuses on validating whether a system correctly fulfils its requirements by executing manual and automated test cases. A test case is defined as a set of ordered steps with specific results and initial setups, where each step describes an action that must be executed before proceeding to the next one. In the context of this work, a test-based domain model defines the associations that connect all actions inside a specific tested domain by formalizing the requirements hidden within the execution of the tests.

Figure 1 illustrates the workflow of a test to send a message, for instance, using a mobile device. In this scenario, all relations represent a dependency since executing the second and third actions can only be performed once the preceding action has been successfully completed. From the initial state, the action of opening a message allows the action of typing a message to be successfully executed. Consequently, the final action of sending a message can be executed after the typing is done, achieving the final state of the workflow.

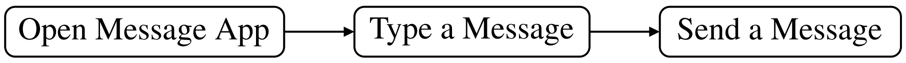

Figure 1: Workflow representation of a test to send a message. Source: The author.

This representation can expand to other types of associations like cancellations and instantiations. The cancellation association usually describes opposite actions, such as `Open Camera` *cancels* `Close Camera`. The definition implies that if *A cancels B*, the action performed by *B* will be annulled if *A* is executed afterward. Thus, *B* must be re-executed if a subsequent action depends on it. Along with the dependency association, cancellations are extremely helpful when verifying test case consistency by checking whether or not an action needs to be (re)included to execute the test case successfully.

Instantiation is an association of inheritance. From root action A, it defines possible actions whose execution will lead to the same purpose as executing action A. For instance, if an action requires inserting a SIM card, it can be completed by inserting a SIM card into either slot one or slot two. In this case, any node below the root action can properly fulfil the dependency of the root itself. However, if the action specifically requires the SIM card to be inserted into slot one, then the dependency cannot be established with the root action. This approach enables the use of more generalized actions and the specialization of those actions.

A test-based domain model can be represented as a directed graph where each node describes an atomic action called "atom" [3], and each edge is an association between the atoms (nodes). Figure 2 offers a summarized example of a domain model for an email application on a mobile device. Note that sending the email requires executing the ordered steps of opening the email app and creating the email. Therefore, *depends on* is transitive, and so are the other relations. However, to successfully send an email, an internet connection must be established to execute the action over the network properly, and can this either be satisfied by turning on the Wi-Fi or or by turning the mobile data on, since they are instances of the `Turn On Internet Connection` action on which `Send Email` depends. As turning off mobile data cancels the action of turning it on, the email cannot be sent if there is no other previous occurrence of an instance of `Turn On Internet Connection` (in this case, `Turn On Wi-Fi`) in the test action sequence. In general, sending an email with any type of internet connection is possible.

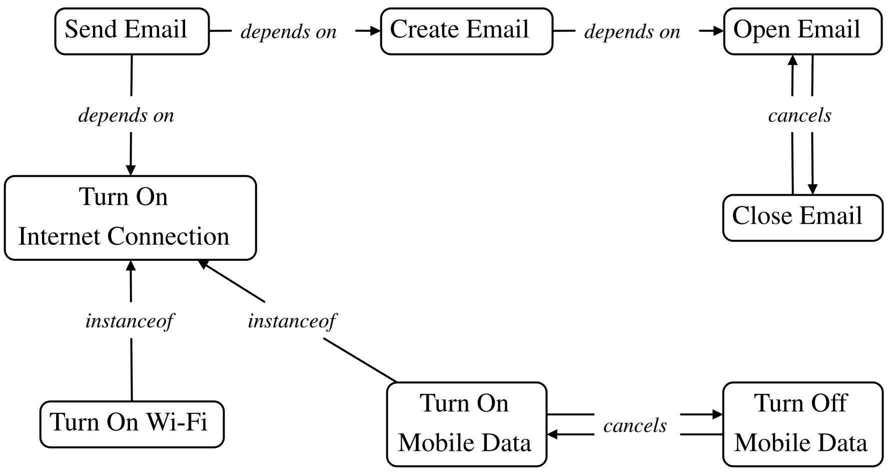

Figure 2: Graph representation of a domain model for an email application. Source: The author.

Such representations allow teams responsible for ensuring product and system quality to rely on precise relationship definitions of associations between the tested components, assisting the creation and execution of consistent test cases. By exposing the possible triggers that can lead to a failure in the test execution – such as the cancellation of an action – the domain model can reduce the effort in creating execution reports and test case analyses.

Additionally, it can be associated with a Model-Based Testing strategy, where consistent by construction test cases are automatically generated from a model representing the system's behaviour [23]. Since the domain model defines all possible execution flows, it helps explore uncovered paths inside the tested domain and increases test coverage.

At last, it can also be used to infer the consistency of user-defined test cases automatically. By using a tool such as Kaki [5], the definition of associations between the atoms within a test case enables the inference of satisfiability in a given domain model and allows for predictions about missing atoms essential for ensuring the test case's consistency [2].

2.2 LARGE LANGUAGE MODELS

A Large Language Model is an AI trained on large volumes of data (parameters) to understand, generate, and translate natural language. The model predicts the next word or token by calculating the probability of various options and selecting the most likely one. Its base is a type of neural network architecture that revolutionized the field of natural language processing called Transformer and was introduced in 2017 by Vaswani *et al.* [37].

Transformers are based on the attention mechanism, an encoder-decoder approach inspired by human attention that allows models to softly search the tokens and focus on a specific subset when generating the target word [7]. This dynamic attention process enables the model to identify and emphasize the most relevant parts of the input, improving accuracy in generating words or phrases in the output.

Strongly disseminated with the rise of the OpenAI model called GPT (Generative Pre-trained Transformer)[1] in 2018, the LLMs have been introduced and deeply studied in all kinds of usages, from law applications [35] to generation of formal rules specification in blockchains [24]. More recently, Deepseek emerged with a new approach by incentivizing the LLMs' reasoning with reinforcement learning (RL) without supervised fine-tuning (SFT) – the most common approach for training in other models [21].

Regardless of structural differences, all LLMs are creative and non-deterministic models whose performance effectivity depends on prompt engineering techniques and optimized parameter definitions to create specific responses.

2.2.1 Prompt Engineering

The prompt engineering focuses on creating and refining prompts to maximize the competence of LLMs in performing a specific task without modifying the model parameters [34].

A prompt is an instruction or query that guides the LLM behaviour by properly taking advantage of four main elements: instruction, context, input data, and output indicator [20]. The instruction describes the main tasks that will guide the output. The context contains specific and

[1]Available at https://openai.com/index/chatgpt

necessary knowledge about the domain addressed in the task. The input data is the essence of the question that directly impacts the output. The output indicator is an additional field that can be used to specify the desired output format. When accessed through APIs, this field can be hard-coded and included in the request to compel the model to follow the exact pattern.

These elements generally describe one of the most basic prompt formats called zero-shot, where all interpretation depends upon the natural language description given to the model with no further exemplifications. However, a new property called *in-context learning* emerged with the idea that the LLMs are few-shot learners, meaning that sometimes the interactions with demonstrations of a few examples to improve reasoning can nearly match the performance of fine-tuned approaches [8].

Moreover, other prompt engineering techniques emerged to cover different bottlenecks in using LLMs, such as performing tasks without intensive training, reducing hallucination, and improving reasoning [34]. In this context, a new strategy called Chain-of-Thought (CoT) prompting was proposed by Wei *et al.* [38] as a mechanism to optimize the reasoning capabilities of LLMs by incorporating intermediate reasoning steps as demonstrations in the exemplars for few-shot prompting.

However, since the CoT technique gives the LLM the step-by-step thinking, it is a time-consuming and error-prone activity. Recent approaches such as Automatic Chain-of-Thought (Auto-CoT) prompting [44] propose optimizations to enhance the creation and correctness of reasoning. Aligned with a stepwise approach, the CoT can be explored with single prompts that cover the entire task or with a chain of prompts. The chaining concept suggests that breaking the tasks into smaller ones can improve the LLM's effectiveness by allowing reasoning to be made instruction-by-instruction [40].

Although prompt engineering is guided by the concept of a prompt-based learning paradigm and has significantly exposed the potential of LLMs, it still has vulnerabilities. Inserting certain triggers into the text can generate incorrect, fabricated, or misleading information that sounds plausible but is actually false or not based on real data (LLMs' hallucinations) [41], needing external interference in the LLM reasoning when performing complex tasks.

3 FRAMEWORK

The proposed framework is illustrated in Figure 3. It conceives a strategy to generate test-based domain models from a test suite input within a specific domain using LLMs. This process involves evaluation and refinement in the LLMs' responses through an iterative cycle. In this work, we used Gemini gemini-2.0-flash as the reference LLM.

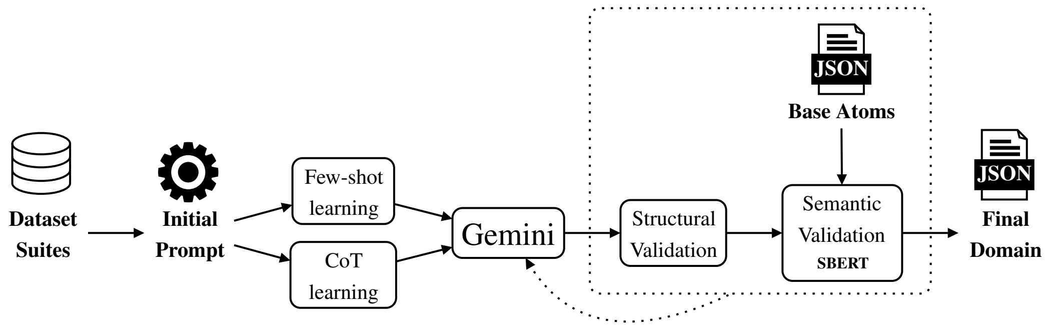

Figure 3: Proposed framework for generating test-based domain models using LLMs. Source: The author.

The prompt preparation used an ad-hoc prompt-engineering process[1] and is based on the prompt engineering techniques exposed earlier in Section 2.2.1, mainly the most promising ones – *few-shot* and *chain of thoughts learning* –, aiming to assess the impact of each technique's characteristics on the final response outcome.

The framework relies on recent evidence that the effectiveness of LLM responses is widely increased with feedback interaction [24, 26]. By addressing structural and semantic concerns, inconsistencies in the domain model are given as feedback to the LLM in an automatic cycle until a valid domain model is created or a pre-defined limit of cycles is reached.

The approach uses a specific domain model format to evaluate the responses in execution time. Since the LLM is susceptible to hallucinations, directly asking the format in the prompt is not reliable. To avoid this, the parameters of *schema* exposed in Listing 3.1 and *type mime* as *application/json* were defined in the request configurations, forcing all responses to follow a default template and avoiding potential crashes. The *schema* is represented by a `DomainModel`

[1]Available at https://cloud.google.com/vertex-ai/generative-ai/docs/learn/prompts/prompt-design-strategies

class that inherits from Python `BaseModel` class[2]. The atoms are stored as a list of strings, while the associations are represented by a list of `Association` instances (another `BaseModel` inheritance). Each `Association` class has two attributes: `source` and `target`, both of type string. The `source` indicates the atom from whom the association originates, and the `target` indicates the entity with whom the association is made.

Listing 3.1: Domain model template for LLM responses.

```
class Association(BaseModel):
    source: str
    target: str

class DomainModel(BaseModel):
    atoms: List[str]
    cancellations: List[Association]
    instances: List[Association]
    dependencies: List[Association]
```

3.1 PROMPT LEARNING

The prompt preparation is the initial step in providing the LLM with the learning context related to the framework objectives. Following a preliminary analysis of the LLM responses to the main task, it became evident that the complexity and the reasoning necessary for the precise/accurate definition of a domain model were significant factors. Consequently, we dismissed the zero-shot technique and focused primarily on few-shot learning and CoT methods, as they proved to offer more effective approaches. More details on each prompt technique are provided later.

When designing a prompt, a comparison metric can be the effectiveness based on the number of used tokens since it directly reflects the LLM processing time and cost concerns, but it is not within our scope to verify these issues. Regardless of the technique used, the first two sections of the prompt focus on contextualizing the definition of the test-based domain model and the nature of the associations within domain models.

A few tests were conducted to enhance the understanding of the LLM regarding the context by adopting various representations, such as Unified Modelling Language (UML) activity view [33], direct graphs as adjacency lists, and domain ontology [19]. The most promising one was the direct graph representation, where nodes represent the atoms and the edges their associations.

Figure 4 illustrates the contextualization prompt, where an overview of the main task is presented before adding more details in the next prompt sections. It is part of a reaffirmation

[2]Available at https://docs.pydantic.dev/latest/api/base_model/

premise that ratifies the main task to maximize the outcome. A special highlight is added to the idea of implicit atoms by emphasizing that a test step encompasses one or more atomic actions. The test step `Verify battery percentage with CLI closed` comprises two atomic actions: closing the CLI and verifying the battery percentage, with a relation of dependency established between them.

Note that *composition* also constitutes a type of association that could be included in the domain model. For example, the atom `Verify battery percentage with CLI closed` is the composition of the two ordered actions `"Close CLI"` and `"Verify battery percentage"`, respectively. However, as an initial proof of concept, we did not include this association requirement. This means that all examples used for the validation of this framework only contain atoms and relations that can be easily inferred from the idea of implicit and explicit atoms. Additional discussion regarding this concern is included in future work (Section 6).

> **A domain model in software testing** is a domain model where the entities are test actions and the relations are **dependency, cancellation, and instantiation**, for example.
>
> It can be represented as a **directed graph** where the **nodes** are the **atomic actions** within the tests and the **edges** are the **relations** between these atomic actions.
>
> A test action can contain one or more atomic actions. For example, the test action `"Open camera by clicking in the menu"` has two atomic actions: `"Open camera"` and `"Click in the menu"`.
>
> Thus, an atomic action is an **atom** that is **necessary for the execution** of one or more test steps and **may not be explicit**.
>
> I will provide you **test cases formatted as JSON** containing multiple test actions.
>
> Your task is to **generate a domain model as a directed graph** by extracting **ALL atoms** and defining **ALL relations** between these atoms.

Figure 4: Prompt for contextualization of the domain model section.

The second section of the default prompt provides a general overview of each association's meaning and the general rules for describing these associations, as depicted in Figure 5. It aims to enhance the LLM ability to generate consistent domain models correctly from the start, reducing the amount of feedback provided by the structural validation stage of the framework. The set of rules introduced in this section to guide LLM responses will be further formalized in the structural validation process (Section 3.2).

The **edges** are defined as follows:

- **Dependency**: An atom depends on another atom when its execution can only occur **after** the execution of the atom it depends on.
  - *Example*: The atom "`send message`" depends on "`write message`".
- **Cancellation**: An atom cancels another when its execution **reverses or negates** a previously executed action.
  - *Example*: Atoms like "`open something`" and "`close something`" are opposites, meaning that "`open`" cancels "`close`" and vice versa.
- **Instantiation**: An atom is an instance of another atom when they **share a common atomic action**, even if this action is not explicitly defined in the test cases.
  - *Example*: The atoms "`Receive a voice call`" and "`Make a voice call`" are instances of "`Participate in a voice call`".

Some **rules** about the edges:

- An atom A cannot depend on atom B while atom B depends on atom A, directly or indirectly.
- An atom A cannot cancel atom B while atom B cancels atom A, directly or indirectly.
- An atom A cannot depend on or cancel atom B while being instantiating atom B.

Figure 5: Prompt for contextualization of domain models associations (edges) section.

After attempts, a key insight was uncovered: incorporating courtesy phrases such as "please," "think carefully," and "take your time," as well as using bold and uppercase formatting, substantially enhances the task's clarity and intent [43], leading to the markdown-based format exposed in the prompts in Figures 4 and 5, as well as in those that follow. However, it became evident that the LLM struggled to complete the task successfully, regardless of prompt variations. To minimize this, the system instruction shown in Figure 6 was introduced. This instruction provides the LLM with a comprehensive understanding of its expected role and how it should respond to any given prompt.

Whenever the user requests the creation or analysis of a domain model for software testing, carefully consider all explicit and implicit atoms and their relationships. Perform a detailed semantic analysis of each provided test case, ensuring that no relevant relationship or atom is omitted. Focus on ensuring that the models presented are complete and coherent and that all dependencies, cancellations, and instances are accurately represented.

Figure 6: System instruction.

3.1.1 Few-shot learning

The few-shot learning prompt is divided into two more sections: general instructions with examples and main task reaffirmation.

The first section highlights all instructions as a guideline that must be followed to properly identify all atoms and associations within the provided domain, as shown in Figure 7.

You **MUST** follow the **instructions**:

1. **Define the explicit nodes**: Analyze the test step **DESCRIPTIONS** and **SETUPS** to identify the atoms.
2. **Define the implicit nodes**: Check if any necessary actions required to execute the explicit atoms are missing (they can be within the atoms themselves, e.g., "`Verify the message with the camera closed`" has an implicit node "`Close camera`").
3. **Create the instantiation edges**: Analyze the tests and atoms to identify atoms that **share a common atomic action**, regardless of whether it is explicit.
4. **Create the cancellation edges**: Analyze the tests and atoms to identify atoms that have an **opposite** atom (e.g., "`open`" vs "`close`", "`enable`" vs "`disable`", "`remove`" vs "`insert`", etc.).
5. **Create the dependency edges**: Analyze the tests and atoms to define atoms that **depend** on the execution of other atoms.

Figure 7: Prompt for general instructions stage in few-shot learning technique.

Meanwhile, the second and final section presented in Figure 8 adds the few-shot examples (represented by the placeholder `$example$`) and concludes the learning context by emphasizing the main task addressed in the first section of the prompt. It also provides the specific input data (test suite), represented by the placeholder `$test_suite$` regarding the target domain to be considered in the test-based domain model generation. Additionally, it employs the previously mentioned courtesy expressions to prevent "impulsive" answers. A final key finding was the inclusion of a sentence to account for the semantics of the tests. We observed that, as a result, more deliberate elements began to be generated, breaking away from the usual identification – noticeable with an increase in the generation of implicit atoms and instantiations, for example.

```
$example$
```

Take your time and **think carefully** before answering. Consider the **SEMANTICS** behind the tests.

Follow the instructions step by step.
Do NOT forget ANY explicit or IMPLICIT atom (node).
Do NOT forget ANY relation (edge).

Once you feel confident, generate the **domain model as a directed graph** based on the test cases below about a **mobile application**.

Before answering, review all test cases and check if any atom (explicit or implicit) or relation is missing!!!

```
$test_suite$
```

Figure 8: Prompt for exemplification and main task request with input data stage.

3.1.2 CoT learning

The CoT learning prompt is based on the approach proposed in [38], but also follows the prompt chaining technique by splitting the generation of the domain models into five stages: generation of (1) atoms, (2) instantiations, (3) cancellations, (4) dependencies, and (5) domain model. It focuses on guiding the LLM through the reasoning necessary to properly identify all the elements presented in the domain model instruction-by-instruction. The entire context is maintained within the chaining of prompts.

After the initial sections of prompting, the first CoT learning section consists of an initial setup represented in Figure 9. It introduces the model with the "chain-of-thoughts" keyword and recaps the need to follow the rules and instructions step-by-step.

I will guide you through your reasoning process with chain-of-thought examples, alright?

Follow the rules and instructions step by step.

Figure 9: Prompt for initial CoT setup.

The second section is used across all stages and is illustrated by Figure 10. It aims to explain the specific task, such as generating the atoms or the dependencies, while showing a chain of thoughts example about the task.

The `$specific_task$` placeholder specifies the stage of the chaining. For example, in the fourth stage, which involves identifying the instantiations, it can be replaced by "**generate the INSTANTIATION edges**". The `$rule_explanation$` recaps the rules for performing the specific task as an instruction. For the last stage, its replacement can be "join all your responses and give me the **domain model as a directed graph**". Meanwhile, the `$test_suite_example$`, `$chain_of_thoughts$`, and `$response$` consist of the placeholders that compose an exemplification when in a CoT prompting technique. It specifies an example of input data in `$test_suite_example$` that will guide all chain-of-thoughts, the CoT for solving the specific task based on the input data in `$chain_of_thoughts$`, and the correct outcome based on the reasoning process in `$response$`. Finally, the `$TARGET_NAME$` placeholder represents the specific target for the task based on the stage: atoms, instantiations, cancellations, dependencies, or domain models (the last stage).

The last prompt section (Figure 11) relates specifically to the final stage of the CoT learning. As in Figure 8, it concludes the learning context by emphasizing the main task presented in the first section of the prompt. Unlike the few-shot approach, the instruction to account for semantics is repeated throughout the chain of thought exemplification, rather than in the final section.

Identify the `$specific_task$`, please.

To do this, `$rule_explanation$`.

For example, given the following test cases:
`$test_suite_example$`

The chain of thoughts is:
`$chain_of_thoughts$`

The response is:
`$response$`

Now, **apply the same reasoning and chain of thought** to **identify the `$TARGET_NAME$`** based on the same tests below about a mobile application and the atoms you have already created.

Figure 10: Prompt with CoT template used across all CoT prompting stages.

Do NOT forget ANY explicit or implicit atom (node).
Do NOT forget ANY relation (edge).

Before answering, review all test cases and check if any atoms (explicit or implicit) or relations are missing!

Figure 11: Prompt for final stage of CoT learning.

3.2 STRUCTURAL VALIDATION

With the learning context established by the main prompt, the framework employs a satisfiability analysis to address structural concerns. With the integration of an ASP Solver named Clingo[3], the domain model validation is reduced to a logic verification problem of satisfiability based on well-defined rules.

Clingo is an ASP solver for modelling complex problems such as planning, automated reasoning, and combinatorial optimization. It combines a grounder responsible for translating high-level rules into a processable format and a solver responsible for finding solutions for

[3]Available at https://potassco.org/

the given rules. Besides, it allows the creation of generic models that can be instantiated with different values or configurations without modifying the ASP code directly through a Logic Program (LP) Template that uses dynamic parameters – typically through directives or preprocessing – to generate more flexible and reusable rules.

The structural validation module relies on an LP template with a set of rules and clauses that infer new associations between atoms – those that may not be explicitly defined in the domain model – based on the defined associations of cancellation, dependency, and instantiation established earlier. After taking the domain model generated by the LLM as input, the Clingo ASP solver defines all possible relations and the consistency of the given domain model. Consequently, it validates the rules defined in the section of the domain model's edges contextualization (Figure 5), which guide the LLM responses. This process involves creating a set of literals based on the given logical rules and verifying whether the domain model aligns with those rules. If no assignment of values satisfies all the conditions or constraints simultaneously, the model is identified as unsatisfiable and the set of unsatisfiable conditions and constraints is given to the LLM as feedback. More detailed and formalized information about the rules is presented subsequently.

The first rule (Listing 3.2) states that cancellations are inherited through instantiations, meaning that when an instantiation occurs, it propagates the cancellation through a chain of related instances. For example, let *A* represent `Turn ON Wi-Fi`, *B* represent `Turn OFF Wi-Fi`, and *C* represent `Turn OFF Wi-Fi from Settings`. In this case, *C inherits B* once turning off the Wi-Fi using the settings app is a specific mode of turning off the Wi-Fi. Since turning on the Wi-Fi (*A*) would override any action related to turning off the Wi-Fi (*B*) regardless of the level of specialization, it is possible to infer that *A cancels C*.

In contrast, if *C* represents `Turn ON Wi-Fi from Settings`, then it is a specific mode of achieving *A*, meaning that *C inherits A*. As any action that turns on the Wi-Fi (*A*) – any specialization of *A* – cancels the action of turning off the Wi-Fi (*B*), then turning on the Wi-Fi using the settings app (*C*) cancels the action of turning off the Wi-Fi, meaning that *C cancels B*.

Listing 3.2: Logical clauses for cancellation-instantiation rule.

```
cancels(A, C) :- cancels(A, B), instanceof(C, B).
cancels(C, B) :- cancels(A, B), instanceof(C, A).
```

As explained earlier, if action *A* depends on action *B*, but action *B* has specializations, then any action inheriting from *B* can fulfil the dependency required by *A*. For example, if executing *A* requires an internet connection, it can be satisfied by enabling mobile data or Wi-Fi. In this case, the actions `Turn ON Wi-Fi` and `Turn ON mobile data` would be instances of `Turn ON internet`. The second rule (Listing 3.3) is based on this definition.

The rule at Listing 3.3 states that the dependency association is inherited from an instantiation, meaning that dependencies are passed down from the base entity to its instances. For example, let *A* represent `Send message`, *B* represent `Create message`, and *C* represent

`Send message using SIM Card 1`. Since sending a message using a specific SIM card is a specialization of the action of sending a message, it inherits all the dependencies of sending a message. In this case, because sending a message requires creating one first, sending a message using the default SIM card also inherits this dependency.

Listing 3.3: Logical clause for dependency-instantiation rule.

```
depends(C, B) :- depends(A, B), instanceof(C, A).
```

Note that the opposite affirmation is false. If action *A* depends on action *B*, then regardless of any inheritance chain preceding *B*, action *A* can only be executed if action *B* or one of its specific specializations is executed before *A*. We observe that a domain model establishing a dependency of an action *A* with both a root action *B* and its specialization *C* would be redundant, as the dependency of *A* with *B* implies that of *A* with *C*.

This reasoning follows the basic principle of inheritance: if a concrete instance is created with a child and not with the parent, then its parent instance cannot be used to fulfill such a dependence. For example, if *A* as `Send an email` depends on *B* as `Create an email`, and *B inherits C* as `Create a message`, it is invalid to assume that sending an email depends on creating a message. In this case, the dependency specifically relates to the specialization where the message is an email. In general, sending an email may depend on more specific actions, such as sending an email with emojis or with a PDF, but not on higher-level actions like simply creating a message.

The third rule (Listing 3.4) states that the cancellation association is inherited from an instantiation, meaning that, as in the cancellation-instantiation rule, a transitive relation is established between atoms of the same instances. This rule can be more easily understood by seeing the inheritances as a tree, where any node reachable by traversing the tree from the root is considered an instance of the root itself. If *A* is a child of *B* and *B* is a child of *C*, then the main root of *A* is also the main root of *C*, meaning that *A inherits C*.

Listing 3.4: Logical clauses for instantiation-instantiation rule.

```
instanceof(A, C) :- instanceof(A, B), instanceof(B, C).
```

Based on the defined clauses, this validation stage analyses the structural consistency of the domain models by identifying conflicts, including circular dependencies, cancellation loops, and inconsistencies between different relation types, such as dependency, cancellation, and instantiation.

Cyclical dependency (Listing 3.5) occurs when an atom depends on itself with a self-referential cycle or when an arbitrary number of elements generates a cycle of dependency – *e.g. A depends B*, *B depends C*, and *C depends A*) – so that the dependency can never be resolved. This verification avoids situations where sending a message relies on sending a message, or where sending a message depends on creating a message, which likewise depends on sending a message, for example. In this study, we only covered the self-referential cycle and cyclical dependencies regarding two elements that mutually depend on each other; more complex scenarios will be further addressed in future work.

Listing 3.5: Logical clauses for identification of cyclical dependency unsatisfiability.

```
cyclical_dependency(A, A) :- depends(A, A).
cyclical_dependency(A, B) :- depends(A, B), depends(B, A).
```

Similarly, cyclical cancellation (Listing 3.6) means that an atom cancels itself. In this case, if atom A depends on atom B, but B cancels itself, A can never be executed because the dependency can never be resolved. Note that in this case, a cyclical dependency between different elements is not an issue, since opposite actions mutually cancel each other. For example, disabling the internet cancels the action of enabling it, just as enabling the internet cancels the action of disabling it.

Listing 3.6: Logical clause for identification of cyclical cancellation unsatisfiability.

```
cyclical_cancellation(A, A) :- cancels(A, A).
```

A conflict between dependency and cancellation (Listing 3.7) arises when an atom A both depends on and cancels another atom B simultaneously or when A depends on B, but B cancels A. In the first case, the action defined by A can never be executed, as it cancels its own dependency. In the second case, A relies on an action that ultimately cancels it, leading to an inconsistency between execution and cancellation.

Listing 3.7: Logical clauses for identification of dependency-cancellation unsatisfiability.

```
dependency_cancellation_conflict(A, B)
    :- depends(A, B), cancels(A, B).
dependency_cancellation_conflict(A, B)
    :- depends(A, B), cancels(B, A).
```

Analogously, a conflict between dependency and instantiation (Listing 3.8) emerges when an atom both depends and inherits from another atom B, creating a dependency of its own base entity, or when atom A depends on atom B, but B is an instance of A, creating a contradictory and circular dependency that leads to inconsistency.

Listing 3.8: Logical clauses for identification of dependency-instantiation unsatisfiability.

```
dependency_instance_conflit(A, B)
    :- depends(A, B), instanceof(A, B).
dependency_instance_conflit(A, B)
    :- depends(A, B), instanceof(B, A).
```

The structural validation models also verify if $\forall x \in A' : x \in A$ where A is the set of atoms defined in the domain model and A' is the set of atoms appearing in the associations. This ensures that a domain model cannot have an atomic action that appears in an association unless the action is an atom of the domain model.

The ASP verifies the domain model consistency by checking any dependency cycles, cancellation cycles, and inconsistencies between dependency-cancellation and dependency-instantiation relations. A parsing is made to extract the core assumptions returned by the ASP that led to the structural inconsistency based on the rules. The LLM receives all the identified conflicts and inconsistencies, along with the feedback about atoms from the associations not being included in the list of mapped atoms.

3.3 SEMANTIC VALIDATION

Semantic correctness refers to the model's ability to ensure that the interpretations or results it generates about a domain are accurate with the intended meaning [18]. Although the domain model can be structurally sound, no guarantee is given that it is semantically correct since the atoms and associations may not capture the semantics of the correct relationships (associations) inside the tests.

Starting from the premise that each atom a represents an explicit unique step in a test suite, let T be the set of atomic actions in a test case and A the set of atoms in the domain model. Then, it must be the case that $\forall a \in T: \ a \in A$, which ensures that all atoms defined by the steps are correctly mapped onto an atom in the domain model. Additionally, to be semantically correct, the domain model should define all the dependency, cancellation, and instantiation associations needed to execute any valid sequence of steps inside the domain.

However, addressing *semantic correctness check* [18] is not trivial without a proper conformance notion between a domain model and a test suite, as well as the availability of explicit semantic information (or an inference mechanism of such semantic information) about the test suite to generate the domain model. Due to such constraints, we verify only whether the generated domain model includes the minimum set of atoms that appear in the set of atomic actions in the tests ($\forall a \in T: \ a \in A$). However, another semantic analysis approach would be to assume that all tests provided in the test suite as input data are consistent and then verify whether they are indeed consistent based on the generated domain model. While this does not fully ensure semantic correctness, it would provide feedback about wrong dependencies and cancellations, for example. Verifications about the associations and implicit atoms were addressed as part of the evaluation process and are further discussed as limitations of this work.

An internal solution from our industrial partner involved in this study was used to extract a list of atomic actions associated with each test step in the suite, based on the approach proposed in [3]. However, the atom extraction process is highly context-sensitive. Aligned with the level of creativity and non-determinism in the LLMs' responses, the generated atoms and those mapped by the tool may not match exactly in terms of writing, despite conveying the same semantic meaning.

To address this issue, a cosine similarity over word embedding technique was applied to evaluate the semantic similarity of the atomic actions and the generated atoms. Both cosine similarity and word embedding are strongly related to natural language processing (NLP). The first metric measures the similarity between two vectors in a vector space by assessing their angle. The second refers to dense, distributed, fixed-length word vectors created using word co-occurrence statistics based on the hypothesis that words with similar contexts (other words) share the same meaning [1].

The techniques for word embedding are usually divided into those that provide non-contextualized word embeddings, such as GloVe (Global Vectors for Word Representation) [30], and those that generate contextualized word embeddings, such as BERT (Bidirectional Encoder Representations from Transformers) [16]. Unlike traditional embeddings, which assign a single fixed vector to each word, contextual embeddings adjust the representation according to the word's meaning within a given sentence or paragraph. However, achieving better accuracy through contextualization often involves expensive computations, representing a common trade-off when selecting the optimal embedding technique for an application.

Due to the characteristic of context sensitivity in the domain models, we adhere to the contextualized advanced word embedding SBERT (Siamese-BERT), a modification of the pre-trained BERT model that uses Siamese and triplet network architectures. It generates semantically meaningful sentence embeddings, and reduces the effort for finding the most similar pair from 65 hours with BERT to about 5 seconds, while maintaining the accuracy [32].

A cosine similarity matrix is created from the embedding generated by SBERT for the list of base atoms and the generated atoms with a threshold of 0.8. The value was reached after refinements based on experimental analysis, and it assures that any match with a minimum similarity of 80% is considered valid. Then, a matching algorithm is employed to map the base atoms using the maximum semantic similarity, ensuring each generated atom is only mapped to one specific base atom. If a base atom has no match, feedback is provided to the LLM with a list of missing atoms, and a new iteration cycle is started.

4 EVALUATION

This chapter presents the evaluation process and metrics for analysing the generated domain models. It highlights the research questions that guided the evaluation and their corresponding answers. Finally, it discusses the limitations and threats to this study's validity.

The evaluation is based on a case study that includes six test suites from the industrial partner in a representative domain of mobile applications. Each test suite represents a specific domain, with various test cases, atoms, and associations, as illustrated in Table 1.

Table 1: Distribution of test cases, atoms, and associations of each selected test suite.

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

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.

A set of ground-truth domain models was created to properly validate the generated domain models. This approach adheres to the earlier embedding technique by ensuring a similarity check between all atoms and associations in the reference and generated domains. To keep consistency with the feedback given to the LLM on missing base atoms introduced in the semantic validation discussion, the same threshold of 0.8 minimal similarity was adopted.

After creating each ground-truth domain model, an analysis was conducted to support the evaluation process. The goal was to establish a baseline for analysing the results, considering the complexity of each test suite addressed in the evaluation. The analysis was based on (1) the difficulty of generating atoms (both implicit and explicit) and (2) the difficulty of generating each type of association (instantiations, cancellations, and dependencies). Each metric was ranked as low, medium, or high. Table 2 presents this baseline.

Table 2: Qualitative baseline of test suites difficulty in generating the ground-truth domain models.

Test Suite	Atoms	Instantiations	Cancellations	Dependencies
FWUI	Low	Low	Low	Low
HostpotTimeout	Medium	Low	Medium	Low
PreloadContacts	Low	Low	Low	Low
MobileData	High	Medium	Medium	High
InternationalRoamingMenu	Medium	Medium	Medium	Medium
VoiceServicesSupport	High	High	Medium	High

To handle the variability in responses caused by LLMs' non-determinism – where different interactions (even with the same prompt) may lead to responses with varying degrees of correctness – the experiments were conducted by generating the domain model for each test suite 10 times, ensuring no history was preserved between interactions. All LLM parameters remained with the default values.

4.1 METRICS

The pass rate and recall were utilized as quantitative metrics to assess the strengths and limitations in generating the domain models. All metrics were applied to each prompt technique defined in the framework, with the quantitative metrics representing the average value of the 10 runs performed for each test suite. We use Δ to represent the pass rate difference between the CoT and the few-shot (FS) approaches.

The pass rate is defined by Equation 4.1. It evaluates Gemini's capacity to create consistent domain models by analysing the ratio between the number of passed tests and the total number of tests. The latter equals all validations defined in the structural validation process (identification of missing atoms in the atoms list mapped into one of the associations, cyclical dependency, cyclical cancellation, dependency-cancellation, and dependency-instantiation conflicts). It also includes the test of correctly identifying all explicit base atoms presented in the test actions during the semantic validation process. Thus, the total number of tests equals six.

Given the creativity of LLMs, the pass rate focuses on addressing the LLM capability in maintaining consistency between associations and atoms, especially since the responses may contain elements not presented in the ground-truth domain models. Consequently, this particular metric was adopted to assess the effectiveness of Gemini not only in generating complete domain models but also in generating valid ones. Additionally, since the feedback is made by dealing with the structural and semantic errors, it guides the discussion about improvements across interactions.

$$\text{Pass Rate} = \frac{\text{Number of passed test cases}}{\text{Total number of tests}} \tag{4.1}$$

The recall is defined by Equation 4.2. It measures the Gemini's ability to correctly identify all the explicit and implicit atoms and associations that should have been generated (positive instances) compared to the ground-truth domain models used as baselines.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4.2}$$

Where:

- **TP (True Positives)** represents the number of base atoms or associations correctly generated by the model.
- **FN (False Negatives)** represents the number of base atoms or associations that should have been generated but were not.

4.2 RESEARCH QUESTIONS

To properly evaluate the proposed framework, we formulated three research questions (RQs). RQ1 and RQ2 focus on assessing Gemini's effectiveness when no reference model is provided, directly applying the proposed framework with structural validation and explicit atom identifications. Meanwhile, the final RQ explores the impact of incorporating base atoms and associations from the ground-truth domain models, aiming to guide semantic consistency throughout the interaction cycles.

The research questions were designed as follows:

1. How effective is Gemini for test-based domain model generation?
2. How do the iterative cycles enhance the framework's effectiveness?
3. How does a ground-truth domain model contribute to the framework's effectiveness?

4.2.1 RQ1: Evaluation of domain models generation

The RQ1 assesses Gemini's effectiveness in generating domain models by comparing the results obtained with both prompting techniques. It evaluates the generated results based solely on the LLM's interpretation of the given prompts, meaning no structural and semantic feedback validation was provided to enhance the effectiveness of the outcome. The evaluation results focus on pass rate (Table 3), structural and semantic validation (Table 4), and recall (Table 5).

Table 3 illustrates the average pass rate of Gemini for each test suite. In general, the few-shot performed better in addressing all validation requirements, with CoT having an average decrease of 0.16 when accounting for all test domains.

Table 3: Average pass rate without interaction cycles.

Test Suite	FS	CoT	Δ
FWUI	0.83	0.83	0.00
HostpotTimeout	0.95	0.97	0.02
PreloadContacts	0.95	0.83	-0.12
MobileData	0.65	0.65	0.00
InternationalRoamingMenu	0.83	0.82	-0.01
VoiceServicesSupport	0.62	0.57	-0.05
Δ Total Gain	-	-	-0.16

The VoiceServicesSupport had the minimal pass rate in both scenarios (0.62 and 0.57 for FS and CoT, respectively), matching the effort baseline addressed previously, where this domain was considered the most complex one. Generally, the most promising pass rate values occurred in the test suites where no higher effort was reported, with the average minimal value of 0.83 in FS and 0.82 in CoT. This demonstrates that for domains with low or medium complexity, the LLM shows better effectiveness in performing the task properly.

Table 4 highlights the LLM's bottlenecks in generating domain models based on the structural and semantic validations presented in Sections 3.2 and 3.3, respectively. It accounts for the average error occurrences in each test domain distinguished by the learning context. As each sample was run 10 times, a bottleneck of 0.1 means that from 10 runs, only one has the error's occurrence.

Table 4: Average value of errors occurrence in each test suite without interaction cycles, where AA = missing atoms in the atom list included in at least one association, CD = cyclical-dependency, CC = cyclical-cancellation, DC = dependency-cancellation conflict, DI = dependency-instance conflict, and MA = missing atoms.

Test Suite	AA			CD			CC			DC			DI			MA		
	FS	CoT	Δ	FS	CoT	Δ	FS	CoT	Δ	FS	CoT	Δ	FS	CoT	Δ	FS	CoT	Δ
FWUI	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0
HotspotTimeout	0.3	0.0	-0.3	0.0	0.0	0.0	0.0	0.1	0.1	0.0	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0
PreloadContacts	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.3	0.0	0.0	0.0	0.3	0.7	0.4
MobileData	0.6	0.3	-0.3	0.1	0.1	0.0	0.0	0.1	0.1	0.4	0.6	0.2	0.0	0.0	0.0	1.0	1.0	0.0
InternationalRoamingMenu	0.8	0.5	-0.3	0.0	0.1	0.1	0.0	0.0	0.0	0.0	0.1	0.1	0.0	0.0	0.0	0.2	0.4	0.2
VoiceServicesSupport	0.9	0.8	-0.1	0.2	0.4	0.2	0.0	0.0	0.0	0.2	0.4	0.2	0.0	0.0	0.0	1.0	1.0	0.0
Δ Total Gain	-	-	-1.0	-	-	0.3	-	-	0.2	-	-	0.9	-	-	0.0	-	-	0.6

Notably, the only test in which the CoT performed better (less errors occurred compared to the FS approach) was in verifying that all atoms used in the associations were also listed in the atoms' list. Although this is a consistency issue based on the default template adopted as the domain model, it is not as relevant as the remaining structural validations that generate more severe inconsistencies, where the FS technique demonstrated better performance.

Table 5 presents the results based on the recall metric. It highlights that Gemini struggles to define associations, one limitation already reported by other studies regarding the domain modelling field [9, 12]. Cancellations and instantiations are the most impacted, especially when dealing with medium and high complexity domains. Although the recall for dependencies did not reach the ideal values, some dependencies were identified, and no zero recall values occurred, as seen in situations involving instantiations and cancellations.

The most significant discrepancy between the prompt techniques is observed on the dependencies identification, where CoT recall decreases by 0.77. However, this value is primarily influenced by the FWUI test suite, where FS has a recall of 0.50 compared to 0.22 for CoT. This trend is consistent across other associations as well. CoT appears to perform better in test suites with medium or high effort baselines, as the significant discrepancies that guide the overall gain are seen in the FWUI and PreloadContacts test domains, which are ranked as low.

Table 5: Average recall without interaction cycles.

Test Suite	Atoms			Instantiations			Cancellations			Dependencies		
	FS	CoT	Δ	FS	CoT	Δ	FS	CoT	Δ	FS	CoT	Δ
FWUI	0.68	0.62	-0.06	0.00	0.00	0.00	0.70	0.40	-0.30	0.50	0.22	-0.28
HotspotTimeout	0.70	0.88	0.18	NA	NA	NA	0.13	0.47	0.33	0.75	0.77	0.02
PreloadContacts	0.88	0.62	-0.26	NA	NA	NA	NA	NA	NA	0.77	0.36	-0.41
MobileData	0.62	0.64	0.02	0.00	0.00	0.00	0.31	0.41	0.10	0.19	0.18	-0.01
InternationalRoamingMenu	0.82	0.80	-0.02	0.00	0.00	0.00	0.40	0.07	-0.33	0.49	0.37	-0.12
VoiceServicesSupport	0.82	0.83	0.01	0.06	0.00	-0.06	0.00	0.09	0.09	0.20	0.24	0.04
Δ Total Gain	-	-	-0.14	-	-	-0.06	-	-	-0.11	-	-	-0.77

Analysing the generated domain models revealed difficulties, especially in defining the relationships between the atoms. For example, correctly adding the atoms but failing to assign them to the defined relations, or applying the same initial setup to all the first test steps, were observed. Challenges appeared when attempting to define associations, such as adding the dependencies between implicit and explicit atoms or defining the inheritance between atoms.

Answer to RQ1: The Gemini effectiveness in generating consistent test-based domain models without any provided feedback is increased when considering the context of medium and low domains, achieving from 80-90% of satisfiability under the structural and semantic validations proposed in this work (pass rate). When analysing the effectiveness of generating atoms and associations (recall), it faces a challenge for instantiations and cancellations, unable to identify any association in instantiation, for example. In dependencies identification, the recall achieved 77% maximum regardless of the prompting technique. Additionally, the FS overcame the CoT approach by improving overall domain models completeness and consistency. However, the results were highly influenced by Δ within test domains ranked as low. The CoT technique presented better recall values in some cases of medium or high complexity. Further study is required to reveal the CoT benefits when properly generating domain models.

4.2.2 RQ2: Evaluation of feedback influence in domain models generation

RQ2 examines the impact of feedback on Gemini's effectiveness by enhancing the generation of domain models. This evaluation is based on the predefined interaction cycles required to produce a domain model that successfully passes all validation stages. In this study, we set the number of cycles to 10, each incorporating all possible feedback from the validation phases.

Table 6 illustrates how many times out of 10 runs the LLM generated a valid (but possibly incomplete) domain model and the interval (in brackets) of interactions necessary to achieve the values for both prompting techniques. A value 9 [0;3] means Gemini created nine test-based domain models that passed all structural and semantic validation tests by using an interval of interaction cycles between zero – no feedback was needed – and a maximum of three. It highlights that the error feedback indeed guides the LLM through the process of achieving a valid domain model, since most experiments have a minimal interaction cycle of at least one.

Table 6: Frequency out of 10 runs that a valid domain model was created and the interval of iterated feedback necessary to achieve the result.

Test Suite	FS	CoT
FWUI	10 [1;1]	10 [1;1]
HostpotTimeout	10 [0;1]	10 [0;1]
PreloadContacts	10 [0;1]	10 [0;4]
MobileData	10 [1;8]	9 [1;4]
InternationalRoamingMenu	9 [0;2]	9 [0;2]
VoiceServicesSupport	10 [1;6]	8 [2;5]

Figure 12 shows the pass rate for both prompting techniques concerning the different test domains. When no value is reported for a subsequent interaction, it indicates that the LLM successfully addressed all errors and no additional feedback was required. For instance, as shown above, all FWUI samples passed with just one interaction cycle, so the pass rate was not measured in subsequent interactions, as they did not occur. The dashed line indicates that, from the point where the dashes start, no other sample could pass all tests regardless of the increase in feedback.

From this, we can conclude that the feedback is highly significant to acquiring consistent domain models, since the ones generated after the last interaction have a pass rate equal to or close to 1.0 (where all structural and semantic validations are covered). Moreover, we can infer that even with feedback, the Gemini effectiveness is still remarkable in the FS approach compared to CoT, as it consistently maintained the highest values throughout all cycles.

The advantages of the feedback are especially highlighted in the MobileData and VoiceServicesSupport, both domains catalogued with the highest level of effort. Their first interactions have the minimum pass rate global values, but their performance has significantly improved after the second cycle. It re-emphasizes that though prompt engineering is a significant guide, in complex scenarios it has its vulnerabilities [41] and requires integrated and robust approaches with validation stages to achieve better results.

Table 7 presents the recall for the atoms and associations for both prompt techniques, when the interaction cycles reach the maximum, or a valid domain model is created. When the recall reaches its maximum value, the domains are considered complete, but they may not be structurally valid, as this metrics is based on all samples rather than only those that passed the structural tests. Compared to the similar analysis in the previous research question, it is noticeable that the feedback increases the LLM's capability to identify the atoms and associations. However, the limitations for associations, especially for instantiations, remain significant.

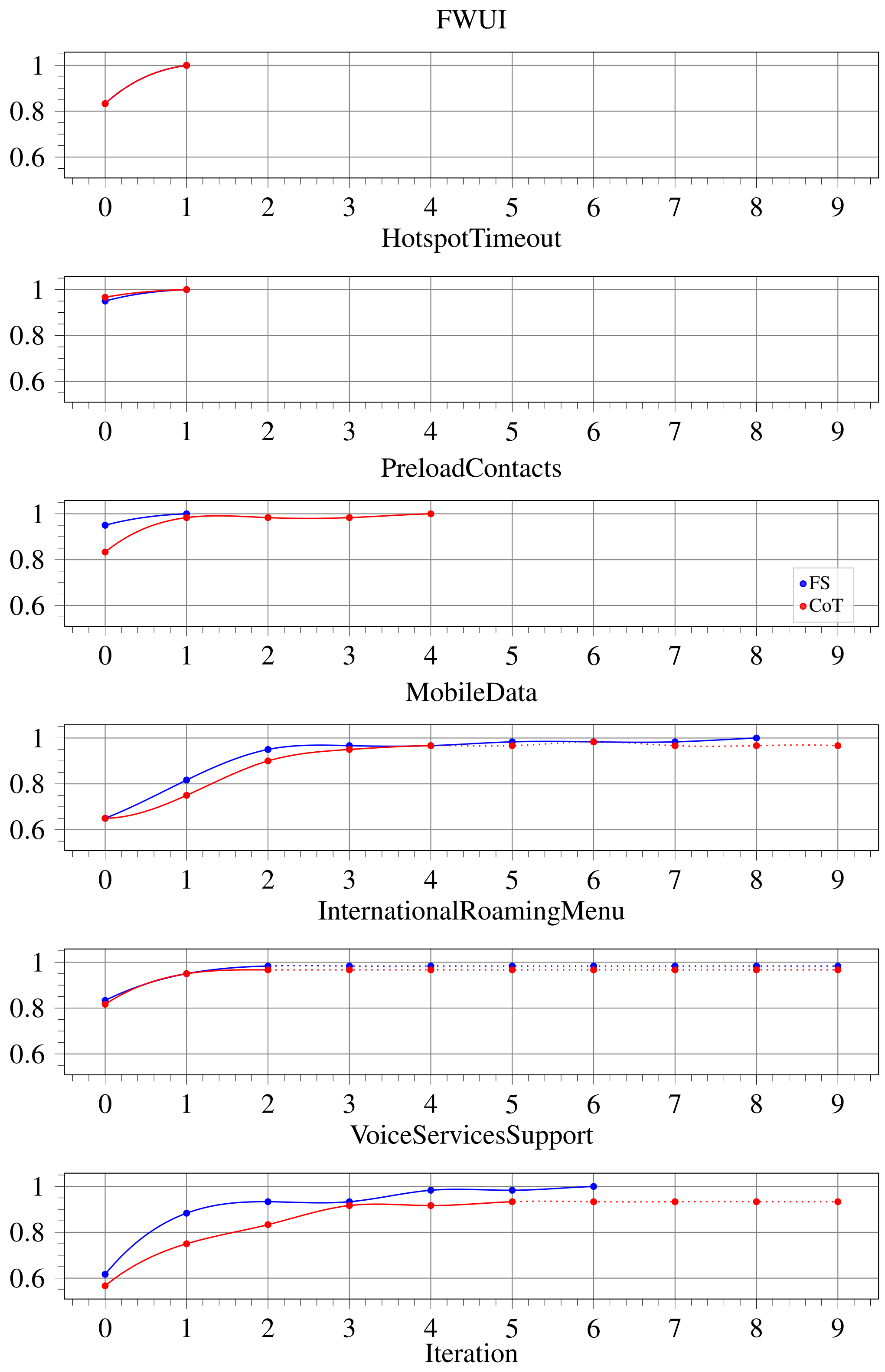

Figure 12: Average pass rate after all possible feedback.

Table 7: Average recall after all possible feedback.

Test Suite	Atoms			Instantiations			Cancellations			Dependencies		
	FS	CoT	Δ	FS	CoT	Δ	FS	CoT	Δ	FS	CoT	Δ
FWUI	0.81	0.78	-0.03	0.00	0.20	0.20	0.70	0.40	-0.30	0.53	0.22	-0.32
HotspotTimeout	0.70	0.88	0.17	NA	NA	NA	0.13	0.47	0.33	0.72	0.77	0.05
PreloadContacts	0.95	0.90	-0.05	NA	NA	NA	NA	NA	NA	0.79	0.53	-0.26
MobileData	0.80	0.82	0.02	0.00	0.00	0.00	0.27	0.38	0.11	0.41	0.36	-0.05
InternationalRoamingMenu	0.82	0.82	-0.00	0.00	0.00	0.00	0.37	0.08	-0.28	0.44	0.39	-0.06
VoiceServicesSupport	1.00	1.00	0.00	0.00	0.02	0.02	0.27	0.18	-0.09	0.21	0.22	0.02
Δ Total Gain	-	-	0.11	-	-	0.22	-	-	-0.24	-	-	-0.61

Unlike the recall calculated without interaction cycles, when adding feedback, the CoT achieved better results in identifying atoms and dependencies, even though the cancellations and instantiations remained with lower values. However, as reported by Zhang *et al.* [44], the creation of CoT is prone to errors and inaccuracies since it is a manual activity. This suggests that enhancing the reasoning process and providing more examples related to cancellation and instantiations could help improve the recall values for those associations.

Answer to RQ2: The Gemini effectiveness in generating consistent test-based results, when including an oracle providing iterated feedback on the problems in the generated model, is highly increased. Out of 120 domain models – 10 domains for each of the 6 test suites, ranging between 2 prompting techniques – only 5 were considered unsatisfiable based on the structural and semantic validations proposed in this work. Additionally, the semantic validation improved the overall recall based on atom identification regardless of the prompt approach. As in RQ1, the few-shot consistently maintained better results than the CoT technique when accounting for the Gemini effectiveness with pass rate. However, when comparing the final recall result, the identification of atoms and instantiations increased with CoT, though the general identification of associations remains challenging.

4.2.3 RQ3: Evaluation on ground-truth influence in domain models generation

As previously discussed, one central aspect and limitation of the semantics validation stage is the correct identification of all atoms and edges since there is no default parameter to address this concern automatically. The RQ3 addresses the impact on Gemini effectiveness if a ground-truth domain model containing a base for explicit and implicit atoms and associations were provided as part of the framework (Figure 13), other than just the base atoms extracted from the test cases' actions.

The main change lies in the increased feedback provided to the LLM, as associations and implicit atoms will now be included as part of the requirements when verifying the semantics of the domain model. Although it can reduce the number of domain models considered valid, it possibly guides the LLM in generating more precise associations and generalized atoms, as will be analysed when evaluating this question response.

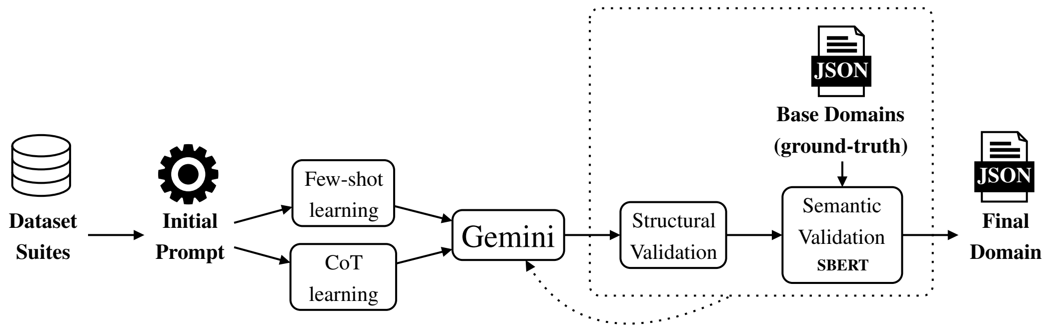

Figure 13: Framework for evaluating RQ3 by generating test-based domain models using LLMs leveraging ground-truth domain models. Source: The author.

Since this approach only affects the test-based domain models generated with interaction cycles, only the analysis regarding the presence of feedback was applied. Additionally, even though it adds the comparison of associations and implicit atoms in each cycle, the metrics remained unchanged. The total number of test cases addressed in the pass rate was unmodified to maintain the cross-evaluation of the research questions' consistency. The missing associations identified during the interaction cycles were not accounted for in the pass rate values. Its influence was only reflected when analysing this experiment recall results.

Table 8 illustrates how many times out of 10 runs the LLM generated a valid domain model and the interval of interactions necessary to achieve the values for both prompting techniques when using a ground-truth domain model.

As expected, the number of valid domain models that meet all requirements decreases compared to the approach without ground-truth. However, as previously mentioned, the validation of the atoms is determined by the embedding technique with a threshold of 0.8. This suggests that a lower semantic similarity might still consider the mapping between generated and base atoms complete, without compromising semantic integrity. Further discussion on this analysis will be provided in the limitations section of this work (Section 4.3).

When comparing with the frequency presented in Figure 6, we noticed that as the amount of the feedback interval range increases, the identification of implicit atoms poses a challenge for the LLM.

Figure 14 shows a comparison of the pass rate value changes through the interaction cycles with and without the presence of the ground-truth domain models.

Table 8: Frequency out of 10 runs that a valid domain model was created and the interval of iterated feedback necessary to achieve the result when using ground-truth domain models.

Test Suite	FS	CoT
FWUI	10 [1;1]	6 [1;3]
HostpotTimeout	8 [1;2]	3 [3;9]
PreloadContacts	10 [0;3]	8 [1;4]
MobileData	2 [3;3]	0
InternationalRoamingMenu	0	4 [1;5]
VoiceServicesSupport	5 [3;9]	2 [2;5]

It recaps the previous finding: although the associations were not included when calculating the pass rate, the overall results were still affected due to the LLM's difficulty in finding the implicit atoms. The worst values occurred essentially for the test domain with the highest number of atoms – MobileData (43 atoms), and HotsportTimeout (33 atoms). However, the positive impact of the feedback is still noticeable since, regardless of the number of implicit atoms or the effort baselines, the pass rate still improved when comparing the first and last interactions.

Meanwhile, the pass rate Δ between the prompt techniques with and without ground-truth domain models only showed a significant value when within a test domain ranked at some medium or high level. The graphs were the same or closely for the easiest ones – FWUI and PreloadContacts. As before (Figure 12), the few-shot technique surpassed the CoT, reiterating that the few-shot learning increased the LLM's knowledge about the specificities of the task. Experimental analysis shows that the LLM hallucination increased with the amount of information in the CoT prompting and higher cycles of interaction. Further study on the prompting and LLM parameters is needed to understand how to reduce this behaviour.

Table 9 presents the recall for the atoms and associations when the interaction cycles reach the maximum, or a valid domain model is created using ground-truth domain models. It highlights the Gemini limitation in correctly identifying mostly cancellations and dependencies, even when feedback about these elements is given. Regardless of the prompt technique, the LLM could generate all base atoms and instantiations. This last association is the most improved one, since it had the minimal values in the two previous approaches.

The minimal values lie mainly in the test domains whose associations are ranked with medium or high complexity. However, when compared to previous results, the inclusion of a ground-truth domain model improved the recall from a minimum value of 0.00 to 0.85. This means that the worst-case scenario involved identifying dependencies in the InternationalRoamingMenu, achieving an 85% match. Additionally, the Gemini identification can be considered satisfactory even for the domains ranked with the highest complexities (VoiceServicesSupport and MobileData).

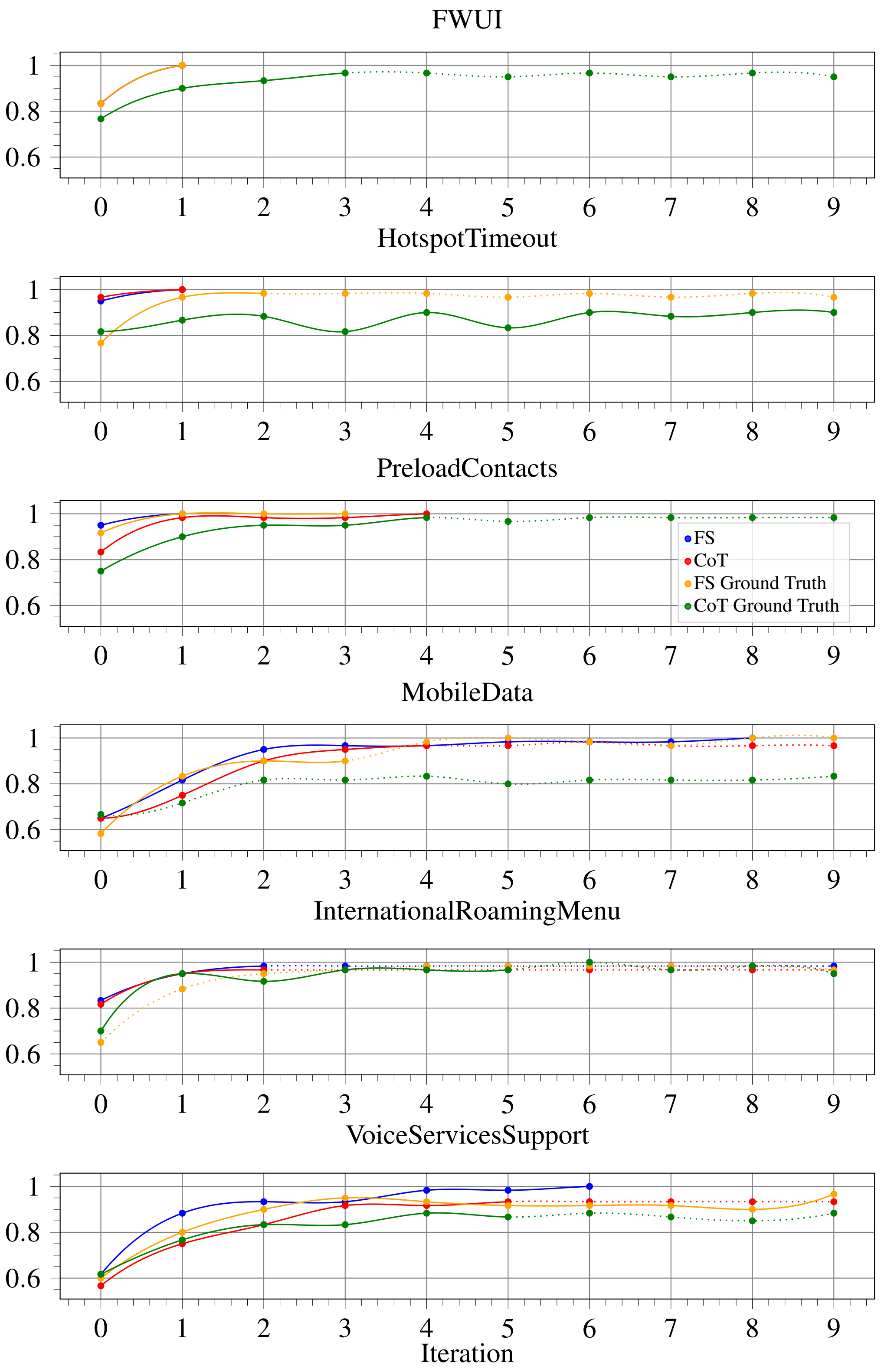

Figure 14: Average pass rate after all possible feedback when using ground-truth domain models.

Table 9: Average recall after all possible feedback when using ground-truth domain models.

Test Suite	Atoms			Instantiations			Cancellations			Dependencies		
	FS	CoT	Δ	FS	CoT	Δ	FS	CoT	Δ	FS	CoT	Δ
FWUI	1.00	1.00	0.00	1.00	1.00	0.00	1.00	0.93	-0.07	1.00	1.00	0.00
HotspotTimeout	1.00	1.00	0.00	NA	NA	NA	0.99	0.93	-0.05	1.00	1.00	0.00
PreloadContacts	1.00	1.00	0.00	NA	NA	NA	NA	NA	NA	1.00	0.98	-0.02
MobileData	1.00	1.00	0.00	1.00	1.00	0.00	1.00	0.98	-0.02	0.89	0.95	0.06
InternationalRoamingMenu	1.00	1.00	0.00	1.00	1.00	0.00	0.70	0.80	0.10	0.85	0.91	0.06
VoiceServicesSupport	1.00	1.00	0.00	1.00	1.00	0.00	0.92	0.95	0.03	0.91	0.97	0.06
Δ Total Gain	-	-	0.00	-	-	0.00	-	-	-0.01	-	-	0.15

A key difference between the results regarding prompt techniques is that the dependencies were better identified in the CoT approach, especially compared to the recall results without ground-truth. The previous reasoning helps clarify this effect: the CoT prompt likely influenced the LLM understanding of the dependencies and cancellation associations. An improved CoT prompt could have led to better results, directly enhancing the framework's performance in more complex scenarios like MobileData. With the highest number of dependencies (22), this domain outperforms the other test domains. Since the ground-truth approach provides feedback on the missing dependencies, it enhances the LLM knowledge, which explains the observed differences. For example, the recall for MobileData dependencies without ground-truth was 0.41 and 0.36, while with ground-truth, it increased to 0.89 and 0.95 for FS and CoT, respectively.

Answer to RQ3: The Gemini effectiveness in generating consistent test-based results is affected by feedback insights based on ground-truth domain models. Out of 120 domain models – 10 domains for each of the 6 test suites, ranging between 2 prompting techniques – 68 were considered unsatisfiable based on the structural and semantic validations proposed in this work. However, considering the recall results, the decrease only means that more consistent, correct, and complete domain models are created, as the number of positive instances (atoms and associations) identification increases. Additionally, regardless of the decrease, the positive impact of adding an interaction cycle remains, with effects especially highlighted in the pass rate results. Unlike other analyses, the CoT technique improves the identification of dependencies. Still, hallucinations were detected during the experimental evaluation of this specific prompt technique, requiring more careful consideration in future work.

4.3 LIMITATIONS

The proposed framework's limitations are rooted in the challenge of validating the domain model's semantics. It lacks automatic analysis for new atoms generated by the domain model that were absent in the base atoms and the associations' definitions. Without a specialist analysis, there is no certainty to guarantee that all atoms and associations are complete and valid. However, as with most AI-based applications, we can enhance the reliability of the framework by ensuring that the responses are sound. To achieve this, more robust approaches, such as fine-tuning the model, can be applied.

Additionally, the SBERT threshold in the embedding stage highly influences the framework's output. Analysis of the experiments revealed that, in some cases, the LLM generated correct atoms, but the threshold was high enough to prevent them from being mapped to base atoms. For example, `Set device multi-window mode ON` and `Enable multi-window mode` were not matched, leading to an extra cycle of interaction and, consequently, to the accounting of error in the results analysis. Future improvements in this module are necessary to enhance the inference of semantic similarity, either by decreasing the threshold or by applying alternative embedding techniques that better suit the application.

Moreover, due to the need for input-provided domain models as ground-truths, the framework's effectiveness was not validated in test suites with large test cases or running experiments with more samples. Besides, all the test suites were manually selected to ensure a controlled environment where atoms and associations could be correctly inferred, avoiding the need for specialized training before the experiments. Further analysis is needed to correctly infer the Gemini's effectiveness in higher complex scenarios, such as domains covering tests with the interaction of multiple devices, several test cases, and composition associations.

4.4 THREATS TO VALIDITY

The main threat to the validity of this study revolves around the test suites used to develop the domain models. In formal approaches, domain models are generated from requirements or by specialists following the traditional software engineering cycle. Since the test suites are written in natural language, the clarity of each test step may affect the LLM comprehension, leading to inefficient responses or even hallucinations. Furthermore, unlike a proper requirements elicitation, a test suite might not include all the relevant atoms and associations of a domain. So, completeness analysis is an issue.

Another aspect concerns the correctness of the test cases. If a test case creates an invalid context by describing incorrect execution steps within the domain being tested, the generated domain models will reflect these errors and, as a result, will be incorrect. Additionally, for tests where atomic actions are implicit and the general context does not help to identify those actions correctly, the identification of atoms may be affected without fine-tuning in the specific domain. For example, for non-public features – the LLM was not trained with the information –, identifying specific dependencies that are not in the initial setup and are not common sense would be hard to achieve.

As we propose generating domain models based on test suites, the outcome depends on the clarity and correctness of the tests used as input and the accessibility of information about the domain. It introduces a bias in all subsequent applications that depend on the generated domain model. For example, in Kaki [5], the domain model is the ground-truth used to infer the satisfiability of a given test step order within the domain. If the domain model is incorrect, the errors will affect all inferences. Similarly, when domain models are used to verify the soundness of test cases [2], invalid or incomplete models can result in test cases that are themselves invalid.

Lastly, another threat concerns the creation of ground-truths. Since the established baseline effort was defined by a single individual and used consistently across all evaluations conducted by the same person, the conclusions are heavily influenced by a single perspective. Although another researcher reviewed the base domain models, the qualitative analysis of the LLM-generated domain models could achieve different results if more researchers' opinions were considered.

5 RELATED WORK

The definition of domain models is one of the most complex and time-consuming tasks in the software life cycle. Its integration with LLMs has increasingly been explored to enhance processing and optimize development. However, they are still receiving reduced attention when compared with that regarding code generation [9].

Chaaben *et al.* [11] proposed and evaluated a novel approach using LLM and few-shot prompt learning for domain modelling assistance. This includes a tool, MAGDA, to address time constraints and incomplete domain understanding, reducing the need for extensive training on scarce datasets while providing versatile support for modelling activities. Oswald *et al.* [28] investigated the use of LLMs for automating the generation of planning domain models from simple textual descriptions, with the introduction of a framework for evaluating LLM-generated domains by comparing the sets of plans for domain instances. An empirical analysis of 7 LLMs across 9 planning domains was performed, and the results indicated that LLMs – especially those with high parameter counts – show moderate proficiency in generating correct planning domains.

Chaaben *et al.* [10] focused on the ability of LLMs to enhance model completion in domain modelling without requiring extensive training, utilizing few-shot prompt learning. They proposed that LLMs can identify and add missing elements in diagrams by leveraging their pattern recognition capabilities, effectively modelling static and dynamic diagrams. However, additional calibration studies are needed to optimize their suggestions. Meanwhile, Cámara *et al.* [9] defended that modelling methods should adapt to include GAI tools and assess ChatGPT's capabilities in generating syntactically and semantically correct UML models. The models produced contain high variability and inconsistency, with limitations in generating basic modelling concepts like association classes, and performance varying significantly by the problem domain.

The limitation in creating associations has been reported in other studies. Chen *et al.* [12] addressed the automation of domain modelling using GPT-3.5 and GPT-4 through zero-shot, few-shot, and CoT techniques. They found that while LLMs demonstrate strong domain understanding, they face significant challenges in generating relationships and applying advanced modelling best practices. The research emphasizes the need for enhanced prompting methods and the incorporation of explicit modelling knowledge into LLMs to improve results.

More robust approaches have been designed to address the performance and automation limitations. Yang [42] proposes a framework to create domain models without human interaction or supervised training by adopting a multi-step automated domain modelling approach that extracts model elements (e.g., class, attributes, and relationships) from problem descriptions. It also adhered to a self-reflection mechanism that assesses each generated model element, offering internal feedback for necessary modifications or removals, and integrates the domain model with the generated feedback. As with most LLM-based applications, one limitation lies in the coupling with the prompt when adopting different LLMs, since it might need to be adjusted to ensure the task and format description are followed. Concurrently, Chen *et al.* [14] proposed an LLM-based domain modelling approach that utilizes question decomposition to generate object models from complex system descriptions. It significantly improves model quality and recall by creating manageable sub-tasks based on human reasoning when modelling, focusing first on classes, second on associations, and third on inheritances.

The reviewed studies highlight the promising potential of LLMs for automating domain modelling through different prompt engineering techniques. While LLMs demonstrate strong capabilities in recognizing patterns and creating general domain models [11], challenges persist in areas like relationship generation [9, 12], consistency, and adherence to advanced modelling practices. Approaches such as decomposition [14] and multi-step frameworks [42] have improved model quality and recall. In general, the most commonly used models are those in the GPT[1], specially the GPT-3.5, and LLama[2] families. However, this is largely influenced by the timing of the researches, as more robust models were not yet available. Consequently, some studies highlight that better results could be achieved by adopting more powerful models in the future.

Unlike our approach, which creates domain models from tests written in natural language, most studies focus on defining domain models for software system development, where entities are classes and associations represent the relationships between these classes. These studies are based on requirements or partial versions of domain models. In the last scenario, the emphasis is on completing models rather than fully creating them. In general, the use of LLMs to define test-based domain models remains unexplored, presenting opportunities for further advancements in both software testing and generative AI.

When comparing the limitations and results of our work with the ones previously mentioned, several common issues emerge. The studies also report significant challenges in creating associations and the difficulty of fully automating the process of domain model generation. Additionally, as discussed in the limitations, incorporating more domain-specific knowledge into LLMs is essential for achieving better results, as specific domains can reduce the effectiveness of general LLMs in accurately resolving the task. This highlights that further research is needed to refine prompting techniques, integrate explicit modelling knowledge, and optimize the calibration of LLMs for specific tasks. Despite these limitations, LLMs provide valuable assistance in automating and improving domain modelling processes.

[1]Available at https://platform.openai.com/docs/models
[2]Available at https://www.llama.com/docs/overview/

6 CONCLUSION

This study proposes an LLM-based approach for generating domain models from test cases written in natural language. Among all representations of domain models, it specifically focuses on describing domain models applied in the context of software testing, as defined by Arruda F. [6]. Additionally, a set of formal rules covering associations of instantiations, cancellations, and dependencies is proposed to verify the consistency of test-based domain models using the Clingo ASP Solver.

Experiments involving some case studies were conducted to compare the effectiveness of the LLM, specifically Gemini, in generating test-based domain models with and without feedback from structural and semantic validations. Additionally, research questions were designed to cover the LLM effectiveness in creating a domain model. To achieve consistent evaluations, we defined a set of ground-truth domain models as a comparison reference. Another research question also addressed further discussion about the effectiveness of the LLM when under these baselines, accounting for semantics issues.

The analyses show that Gemini's effectiveness in generating consistent test-based domain models varies significantly depending on domain complexity and the presence of feedback insights. In RQ1, Gemini achieved 80-90% satisfiability in medium and low-complexity domains, but faced challenges in identifying associations, with the FS model outperforming the CoT approach in terms of completeness and consistency. In RQ2, results improved significantly when Gemini received feedback, with 95% of the 120 evaluated instances being considered satisfiable. Semantic validation also improved atom identification, with the few-shot technique outperforming CoT in structural and semantic validation, but the limitations in identifying the associations remained. In RQ3, the impact of feedback based on ground-truth domain models led to an initial decrease in the number of satisfiable models based on pass rate values, but with an increase in recall. The CoT technique, while improving dependency identification, showed challenges such as hallucinations, requiring more careful consideration in its application.

A more thorough analysis will be conducted as future work to determine how Gemini can generate additional atoms and associations beyond those included in the ground-truth domain models, while considering relevance and coherence. Besides, our goal is also to explore the influence of the model's parameters on the effectiveness when creating test-based domain models.

For example, the common belief was that the temperature parameter in LLM interactions dictated response determinism, where lower values ensured more consistent answers. However, as exposed by Ouyang, S. *et al.* [29], the LLMs cannot be completely deterministic, meaning that the influence of this parameter must be properly addressed to improve the evaluation of the LLM-based applications.

We also aim to evaluate the framework effectiveness using more elaborate test suites by analysing the pass rate and recall across a larger set of test cases. Additionally, we seek to develop more sophisticated domain models that encompass more elaborate scenarios, such as multi-device interactions within the same domain, while also addressing composition associations and dependency cycles involving multiple atoms. However, since domains in software testing are not static, features and test cases are always being added and/or updated. To account for this more dynamic scenario, we intend to evaluate the LLM's capability of self-updating the domain models while maintaining the atoms and associations that are still relevant based on the older version.

REFERENCES

[1] Almeida, F. & Xexéo, G. (2019). Word embeddings: A survey. *arXiv preprint arXiv:1901.09069*.

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

[3] ALMEIDA, R. G. d. (2024). Sound test case generation for concurrent features combining test cases for individual features.

[4] 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.

[5] Arruda, F., Barros, F., & Sampaio, A. (2020). Automation and consistency analysis of test cases written in natural language: An industrial context. *Science of Computer Programming*, 189:102377.

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

[7] Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

[8] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., *et al.* (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

[9] 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.

[10] 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.

[11] Chaaben, M. B., Burgueño, L., David, I., & Sahraoui, H. (2024). On the utility of domain modeling assistance with large language models. *arXiv preprint arXiv:2410.12577*.

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

[13] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., *et al.* (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

[14] 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*.

[15] 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.

[16] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 4171–4186.

[17] Fazelnia, M., Mirakhorli, M., & Bagheri, H. (2024). Translation titans, reasoning challenges: Satisfiability-aided language models for detecting conflicting requirements. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2294–2298.

[18] Fellmann, M., Hogrebe, F., Thomas, O., & Nüttgens, M. (2011). Checking the semantic correctness of process models-an ontology-driven approach using domain knowledge and rules. *Enterprise Modelling and Information Systems Architectures (EMISAJ)*, 6(3):25–35.

[19] Freitas, F., Stuckenschmidt, H., & Noy, N. F. (2005). Ontology issues and applications guest editors' introduction. *Journal of the Brazilian Computer Society*, 11:5–16.

[20] Giray, L. (2023). Prompt engineering with chatgpt: a guide for academic writers. *Annals of biomedical engineering*, 51(12):2629–2633.

[21] Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., *et al.* (2025). Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948.*

[22] Hagos, D. H., Battle, R., & Rawat, D. B. (2024). Recent advances in generative ai and large language models: Current status, challenges, and perspectives.

[23] Kull, A. (2009). *Model-Based Testing of Reactive Systems.* PhD thesis.

[24] 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.

[25] Liao, W., Lu, X., Fei, Y., Gu, Y., & Huang, Y. (2024). Generative ai design for building structures. *Automation in Construction*, 157:105187.

[26] 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.*

[27] Marques, N., Silva, R. R., & Bernardino, J. (2024). Using chatgpt in software requirements engineering: A comprehensive review. *Future Internet*, 16(6).

[28] Oswald, J., Srinivas, K., Kokel, H., Lee, J., Katz, M., & Sohrabi, S. (2024). Large language models as planning domain generators. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 34:423–431.

[29] Ouyang, S., Zhang, J. M., Harman, M., & Wang, M. (2025). An empirical study of the non-determinism of chatgpt in code generation. *ACM Trans. Softw. Eng. Methodol.*, 34(2).

[30] Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. 14:1532–1543.

[31] Poesia, G., Polozov, O., Le, V., Tiwari, A., Soares, G., Meek, C., & Gulwani, S. (2022). Synchromesh: Reliable code generation from pre-trained language models. *arXiv preprint arXiv:2201.11227*.

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

[33] Rumbaugh, J., Jacobson, I., & Booch, G. (2004). *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education.

[34] Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., & Chadha, A. (2024). A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927*.

[35] Siino, M., Falco, M., Croce, D., & Rosso, P. (2025). Exploring llms applications in law: A literature review on current legal nlp approaches. *IEEE Access*, 13:18253–18276.

[36] Team, G., Anil, R., Borgeaud, S., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., *et al.* (2023). Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

[37] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[38] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., *et al.* (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

[39] Wu, G., Cao, W., Yao, Y., Wei, H., Chen, T., & Ma, X. (2024). Llm meets bounded model checking: Neuro-symbolic loop invariant inference. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 406–417.

[40] Wu, T., Jiang, E., Donsbach, A., Gray, J., Molina, A., Terry, M., & Cai, C. J. (2022). Promptchainer: Chaining large language model prompts through visual programming. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, 1–10.

[41] Xu, L., Chen, Y., Cui, G., Gao, H., & Liu, Z. (2022). Exploring the universal vulnerability of prompt-based learning paradigm. *arXiv preprint arXiv:2204.05239*.

[42] Yang, Y. (2024). *Multi-step Automated Domain Modeling with Large Language Models*. McGill University (Canada).

[43] Yin, Z., Wang, H., Horio, K., Kawahara, D., & Sekine, S. (2024). Should we respect llms? a cross-lingual study on the influence of prompt politeness on llm performance. In *Proceedings of the Second Workshop on Social Influence in Conversations (SICon 2024)*, 9–35.

[44] Zhang, Z., Zhang, A., Li, M., & Smola, A. (2022). Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*.