INTERNATIONAL
HELLENIC
UNIVERSITY

Exploring Neuro-Symbolic Pipelines for Structured Knowledge Extraction

Nikolaos Laoutaris

SID: 3308240003

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Data Science

JANUARY 2026

THESSALONIKI – GREECE

INTERNATIONAL
HELLENIC
UNIVERSITY

Exploring Neuro-Symbolic Pipelines for Structured Knowledge Extraction

Nikolaos Laoutaris

SID: 3308240003

Supervisor:	Assoc. Prof. Vassilios Peristeras
Supervising Committee Members:	Prof. Christos Berberidis
	Prof. Ioannis Magnisalis

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Data Science

JANUARY 2026

THESSALONIKI – GREECE

Abstract

Public administration often seems like a maze of bureaucracy, which leaves citizens unaware of public services and benefits to which they are legally entitled. This phenomenon is rooted in the complex nature of legislative requirements and the fragmentation of information sources. As a result, the failure in the proactive delivery of administrative justice is perpetuated. This dissertation addresses this challenge by proposing a *Neuro-Symbolic pipeline* designed to power a *Public Service Recommender System* as an attempt to link unstructured knowledge locked away in legislative texts with personalized service delivery. What begins as an effort to empower citizen awareness, evolves into a generalized engineering framework that prioritizes certainty and interoperability.

The proposed architecture uses LLMs to extract eligibility preconditions from natural language documents and structure them into a symbolic layer of interoperable semantic graphs and executable SHACL/SPARQL validation logic. The system is evaluated through experimental runs, using a mutation testing framework that stress-tests the synthesized logic against heterogeneous citizen scenarios. The results show models achieving up to 80% syntactic validity, however, functional logic accuracy collapsed to ≈25% in complex use cases. To preserve auditability and compliance, this work argues for *local* (privacy-preserving) model deployments combined with deterministic validation, as results show current LLMs require them to reach acceptable operational trustworthiness.

This dissertation was written as part of the MSc in Data Science at the International Hellenic University. I would like to express my gratitude to my supervisor, Associate Professor Vassilios Peristeras, for his academic support, and also to PhD Candidate Ioannis Konstantinidis for our extensive discussions regarding the technical and conceptual dimensions of this study. Their foundational research and previous work served as inspiration for me to pursue this topic. Finally, I acknowledge the School of Science and Technology and the University Teaching Staff for providing the support and resources necessary for me to compose this dissertation.

Nikolaos Laoutaris

24 January 2026

Contents

List of Figures

List of Tables

Listings

1 Introduction

The digital transformation of the public sector is ongoing, and at the present time, it has begun to transition from its simple initial goal of mere digitization, towards the more ambitious vision of proactive governance [1]. According to this concept, often described in modern literature as the "No-Stop Government" [2], administrative services are to be delivered automatically. This would mean, for example, that the identification of eligible citizens and the facilitation of their rights would not require manual intervention neither repetitive data submission from them.

However, significant technological bottlenecks persist. Legislative rules, for instance, remain locked away in unstructured natural language documents, creating a bureaucratic maze that is often very troublesome to navigate [3]. As long as this phenomenon persists, the vision of a proactive government remains a theoretical ideal, leaving citizens struggling with being disconnected from the support systems and entitlements they are legally owed.

This dissertation is inspired by the need to dismantle this maze and empower citizens with effortless access to their rights. As we explore this path, it becomes evident that this is a high-stakes engineering challenge. Thinking of tools at our disposal, when it comes to document understanding and extracting knowledge from natural language, Generative AI naturally comes to mind. However, this challenge cannot rely on *approximate* AI assistance. A true attempt to tackle this issue will require an airtight, interoperable, explainable and rigorously tested solution. Thus, this work addresses the challenge by proposing a novel *Neuro-Symbolic pipeline* within a pilot study that establishes the technical requirements for automating the synthesis of machine-readable administrative logic derived from text.

Conceptually, the system follows the following workflow. First, it uses the interpretive capability of LLMs to extract meaning from text. Then, it transforms that meaning into intermediate representations like structured data objects and Knowledge Graphs. Finally, it generates executable logic in the form of a SHACL (Shapes Constraint Language) graph. The result is a deterministic mapping between natural language legislation and executable SHACL shapes.

To evaluate the feasibility of this approach and its potential for real-world application, the system is implemented and stress-tested within the context of a *Public Service Recommender System*, designed to provide citizens with legally-grounded eligibility recommendations. Through this implementation, we explore the boundaries of current generative AI in the high-stakes domain of public administration and its potential to be integrated with semantic technologies.

1.1 Background and Motivation

Public administration systems worldwide are characterized by an ever-increasing volume of regulatory documents [4]. These texts define the parameters of social welfare, tax obligations, civic entitlements and more. Currently, the primary burden of interpretation falls on the citizen. In the context of public services, determining whether one's attributes satisfy a set of legal preconditions requires significant time, effort and legal literacy. As reported by many citizens, this task is far from trivial [5]. The result is increased *administrative friction*, which is an issue for both the citizen and the government side. On the one hand, citizens (often the most vulnerable, or the most in need) have trouble navigating the maze of requirements. On the other hand, administrative staff are burdened with applying immutable rules to thousands of heterogeneous cases.

The emergence and continuous growth of Large Language Models has from early on been an opportunity for the field of digital governance. Unlike traditional hard-coded, rule-based algorithms, LLMs have the capability to process raw legal text and extract complex semantic relationships [6]. However, the use of generative AI in this context is stifled by the (well-documented) risk of hallucinations and lack of determinism. In a legal environment, an *approximate* answer is insufficient. Administrative acts are bound to require high precision and, perhaps more importantly, an auditable "paper trail" that can be followed back to the respective source.

The importance of this work lies primarily in its attempt to relieve this tension. By developing a methodology that prioritizes separation between the *interpretation* phase of Generative AI and the *execution* phase of logic, this research provides a blueprint for safe and accountable legal automation. Technology here acts as an equalizer, ensuring that the "logic of the law" is transparent, machine-readable and grounded in interoperable cross-border standards. This is undoubtedly a technical upgrade, but furthermore, it is a requirement for *administrative justice*, ensuring that every citizen can effortlessly discover and claim the opportunities they are entitled to, without being hindered by the (necessary) complexity of the state's bureaucracy.

1.2 Problem Statement

The primary challenge addressed in this research is the *gap* that exists between the unstructured, often ambiguous by nature legislative text, and the strict deterministic requirements of administrative execution logic. We have established that the simple usage of AI, generative or otherwise, is insufficient for a domain where errors potentially have legal consequences. Current systems struggle due to three main obstacles:

- **Reliability:** Large Language Models, while capable of interpreting text, are inherently non-deterministic and prone to hallucinations. This makes them unsuitable for direct, unsupervised generation of legal code [7].
- **Complexity:** Formal constraint languages such as SHACL and SPARQL require exact syntactic precision. Even minor "token-Level" hallucinations or cross-contamination from other programming languages render the resulting logic non-executable.
- **Interoperability:** Most existing automation attempts result in isolated, proprietary logic structures [8]. In the European digital landscape, however, failure to make automated rules interoperable with cross-border standards (such as the formal EU vocabularies) prevents the technical scalability required for a unified digital administration.

Without a structured methodology to addres these roadblocks, the vision of a proactive government remains hindered by the manual error-prone labour of human-encoded rules and leaves the system vulnerable both to the limitations of traditional software development and the trust barrier of black-box AI.

1.3 Objectives

The primary aim of this dissertation is to design and evaluate a Neuro-Symbolic Pipeline capable of transforming unstructured public service regulations into executable logic, leaving an auditable trail of interoperable artifacts. To achieve this, the following objectives were defined:

1. **Establish Theoretical Foundation:** To conduct a systematic literature review of existing Neuro-Symbolic technologies, identifying recurring patterns and research gaps. It is expected that this research will base this dissertation in established

academic methods while avoiding known logical and architectural pitfalls that have hindered previous attempts.

2. **Develop the Pipeline:** To construct a multi-stage "Text-to-Graph-to-Logic" pipeline that utilizes Large Language Models for semantic extraction and intermediate structured representations (JSON/RDF) for architectural grounding.
3. **Prioritize Interoperability:** To see if there is a way to ground the synthesized logic in established European semantic standards, specifically the *Core Public Service Vocabulary Application Profile* (CPSV-AP) and the *Core Criterion and Core Evidence Vocabulary* (CCCEV). If this attempt succeeds, it will offer future compatibility with established e-government frameworks that function cross-border.
4. **Test and Verify:** To develop a deterministic *Mutation Testing* framework, its goal being to stress-test the generated SHACL shapes against a collection of scenarios. This would provide a quantifiable metric of functional logic accuracy.
5. **Analyze Performance:** To conduct an experiment, of scale as large as possible for the scope of this dissertation. Benchmarking of different model architectures and prompting strategies will be required, to identify any limitations of current generative AI in this context.
6. **Assess Feasibility:** To evaluate the reliability of this system as an end-to-end Public Service Recommender System. This will be done by taking a particular interest in analyzing how accurately it can guide a citizen, minimizing the (trust-eroding) risks of False Positives and increasing the incentives of developing such a system by multiplying True Positives.

1.4 The Recommender Context

The symbolic logic generated by the proposed pipeline is intended to function as the decision engine of a proactive Public Service Recommender System, following the conceptual framework proposed by Konstantinidis et al. [9], summarized below.

In this vision, the recommender is embedded within a unified citizen portal that adheres to the Once-Only Principle (OOP), whereby public administrations reuse citizen data already held in authoritative registries rather than repeatedly requesting it from users. Under appropriate legal and technical safeguards, the portal has access to a citizen's existing data (e.g., income, family status, employment), represented as an RDF graph.

Rather than requiring citizens to actively search for applicable services, the recommender periodically executes the synthesized SHACL constraint shapes against the citizen data graphs. For each public service, the validation process produces a SHACL conformance report. If the report indicates eligibility, the system may proactively notify the citizen that they appear to satisfy the eligibility conditions for that service, inviting them to verify the information and initiate a formal application.

Within this broader system architecture, the scope of this dissertation is deliberately limited. It does not aim to design or implement a full recommender system or user interface. Instead, it addresses a critical reliability bottleneck identified in prior work: ensuring that the executable eligibility logic used by such systems is a faithful, logically correct and auditable representation of the underlying legislation. The Neuro-Symbolic pipeline and evaluation framework proposed here are motivated precisely by this requirement.

1.5 Dissertation Structure

The rest of the dissertation is organized in the following way:

Chapter 2 contains a Systematic Literature Review of the union between Large Language Models and the Semantic Web, as it is necessary to establish theoretical foundations before implementing the proposed architecture.

Chapter 3 details the methodology and technical architecture of the Neuro-Symbolic pipeline, describing the multi-stage extraction process and the mutation-testing framework used for validation.

Chapter 4 provides the quantitative findings from the experimental phase, analyzing the system's performance across syntactic and logical performance indicators.

Chapter 5 synthesizes these results and gives a subjective opinion and analysis of them, while attempting to tie those in with broader implications.

Chapter 6 summarizes the research findings, addresses the identified limitations and outlines a strategic roadmap for future work.

The source code for the Pilot Study implementation can be found in the dissertation's GitHub repository [10], along with the `TeX` source of the present document, under the "Thesis" directory.

2 Systematic Literature Review

This chapter details a Systematic Literature Review (SLR) with the goal to establish the necessary theoretical foundations of Neuro-Symbolic AI, narrowing the scope where necessary to the context defined as this dissertation's motivations in the previous chapter.

2.1 Introduction

We approach the current research in Neuro-Symbolic AI specifically focusing on how Large Language Models (LLMs) and Knowledge Graphs (KGs) are combined. We aim to find existing approaches for extracting rules from text and generating formal logic (with a focus on SPARQL and SHACL particularly), as well as methods of evaluating the results of such a process.

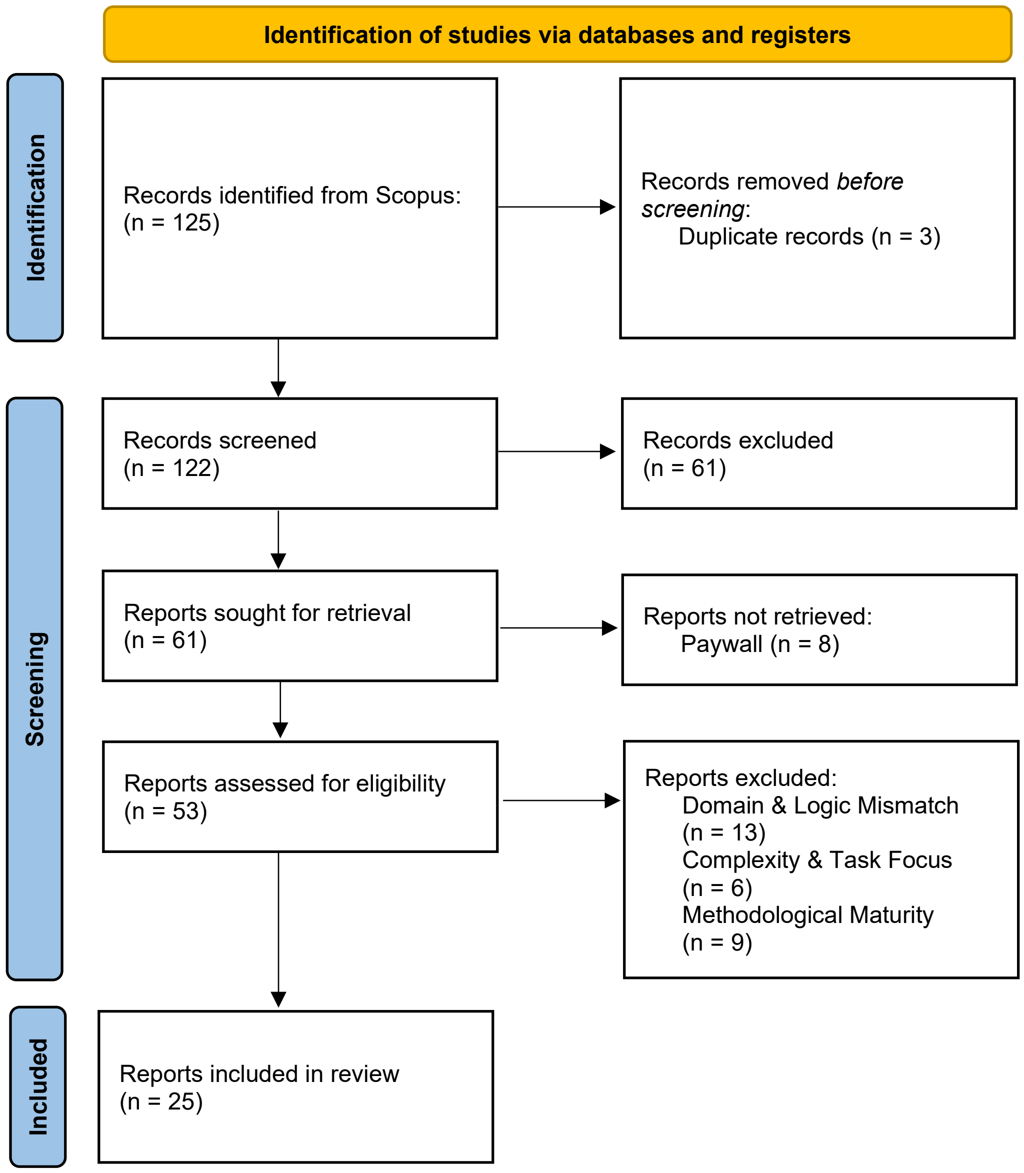
2.2 Methodology

To ensure scientific strictness and reproducibility, the review adheres to the PRISMA (Preferred Reporting Items for Systematic reviews and Meta-Analyses) guidelines [11]. The process was structured into four phases, which themselves define the structure of the rest of this chapter.

1. Defined the Research Questions (RQs).
2. Formulated a query and executed a search on the *Scopus* database.
3. Applied a two-stage screening process: An initial *practical* screening of only titles and abstracts, and then a more thorough *quality assessment* screening of the full texts. These screenings utilized specific inclusion/exclusion criteria and quality assessment criteria.
4. Extracted data from the selected primary studies into a standardized matrix to synthesize key themes and composed them into a thematic analysis.

2.2.1 PRISMA Flow Diagram

The above search and screening process can be summarized in the PRISMA flow diagram (Figure 2.1).



Identification of studies via databases and registers
Identification
Screening
Included
Records identified from Scopus: (n = 125)
Records removed *before screening*: Duplicate records (n = 3)
Records screened (n = 122)
Records excluded (n = 61)
Reports sought for retrieval (n = 61)
Reports not retrieved: Paywall (n = 8)
Reports assessed for eligibility (n = 53)
Reports excluded: Domain & Logic Mismatch (n = 13) Complexity & Task Focus (n = 6) Methodological Maturity (n = 9)
Reports included in review (n = 25)

Figure 2.1: PRISMA Flow Diagram of the selection process

2.2.2 Research Questions

To achieve our objective, we defined three Research Questions that guided the data extraction and synthesis process:

- **RQ1:** How are Large Language Models (LLMs) currently utilized to extract structured knowledge and conditional rules from unstructured text?
- **RQ2:** What are the state-of-the-art approaches for translating natural language re-

quirements into executable constraint languages (specifically SHACL and SPARQL)?

- **RQ3:** What methodologies exist for evaluating the functional correctness and operational stability of LLM-generated logic?

RQ1 explores the initial phases of the proposed pipeline (Text-to-Graph). **RQ2** deals with logic generation, being the core challenge of the proposed system. **RQ3** allows to examine how existing studies evaluate trust and correctness.

2.2.3 Search Strategy

To identify relevant records, an automated search was performed on the *Scopus* database. The search was executed on the 1st of December 2025. The search query was formed in a way able to find the intersection of Generative AI and Semantic Web technologies. We employed Boolean logic to combine three conceptual blocks:

1. **Generative AI Terms:** ("Large Language Model" OR "LLM")
2. **Target Logic:** ("SHACL" OR "SPARQL")
3. **Symbolic Terms:** ("Semantic Web" OR "Knowledge Graph")

These blocks were combined using the `AND` operator. Thus, the final search string applied to the Title, Abstract, and Keywords fields was:

```
( "Large Language Model" OR "LLM" ) AND
( "SHACL" OR "SPARQL" ) AND
( "Semantic Web" OR "Knowledge Graph" )
```

We also applied some necessary metadata filters during this phase:

- **Language:** Only papers written in English were considered.
- **Document Type:** We restricted the search to Articles and Conference Papers, excluding trade journals and errata.

Interestingly, despite the Date Range not being restricted, all results fell in the range of years 2023–2026. This could be an indication that the application of Large Language Models to formal constraint languages like SHACL is a newly emerging field, that appeared primarily after the widespread adoption of "GPT-4 class" models.

The described search strategy yielded an initial set of 125 candidates which, after removing 3 duplicates, were then subjected to the screening process described next.

2.2.4 Inclusion/Exclusion Criteria

We established a set of inclusion and exclusion criteria that reflect the focus of this review. These were applied to Titles and Abstracts during the initial "Practical Screening" phase of the 122 records. Table 2.1 summarizes the criteria used.

Table 2.1: Inclusion and Exclusion Criteria

Category	**Inclusion Criteria**	**Exclusion Criteria**
Task Focus	Text-to-Graph extraction, Text-to-SPARQL, SHACL shapes generation, GraphRAG architectures.	Pure NLP (summarization), low-level graph mechanics (Entity Alignment, Link Prediction, Subgraph Extraction), Dataset creation.
Methodology	Neuro-Symbolic architectures, Prompt Engineering for logic generation, Fine-tuning, Evaluation Frameworks for Semantic Accuracy.	Traditional Machine Learning (non-generative), Reinforcement Learning without LLMs.
Data Flow	*Forward:* Transforming unstructured text into formal logic or structured data (Text $\rightarrow$ Logic).	*Reverse:* Transforming structured data into natural language (Verbalization/Explanation).
Mode	Textual inputs with or without preprocessing.	Multimodal studies (Speech/Image), Computer Vision, Temporal Data.
Type	Peer-reviewed Articles and Conference Papers.	Conference Proceedings (Meta-entries), Posters, Editorials, Preliminary Results.

This screening process excluded 61 papers from the review. The rest were sought for retrieval from official channels. Of the 61 papers sought, 8 could not be retrieved due to access restrictions (paywall). The remaining 53 were downloaded and assessed for eligibility by reading the full text. In this "Quality Screening" phase, we applied a second set of quality exclusion criteria (QE), with the goal to further focus our scope and increase relevance to this study.

- **QE1 (Domain & Logic Mismatch):** From articles situated in descriptive scientific domains (e.g., bioinformatics, chemistry), exclude those where the knowledge structure is purely factual or relational rather than normative or rule-based. These are suspected of offering low transferability to eligibility logic.
- **QE2 (Complexity & Task Focus):** From studies focusing on simple factoid Question Answering (KGQA), exclude those that do not analyze the extraction or generation of complex conditional constraints required for compliance or eligibility. Otherwise, the core challenge and objective shifts significantly.

- **QE3 (Methodological Maturity):** From studies focusing on model-vs-model benchmarking or evaluation of existing datasets, exclude those that do not propose some novel approaches, architectures or logic validation frameworks. Benchmarking generally falls outside the scope of this research.

Following this quality assessment, 28 papers were excluded from the review (13 due to QE1, 6 due to QE2 and 9 due to QE3), leaving 25 papers to be included.

2.3 Results - Data Extraction Matrix

Table 2.2 presents the data extraction summary for the 25 included studies. The studies are categorized thematically, to reflect the research trajectory: from domain-specific applications in public administration and normative compliance, through the technical mechanisms of logic synthesis, to the frameworks of validation and trust.

Table 2.2: Data Extraction Matrix of Included Studies (n = 25)

Study	**Core Task / Domain**	**Logic**	**Neuro-Symbolic Integration**	**Validation Method**
Category 1: Public Administration & Normative Compliance				
Konstantinidis (2025) [9]	Recommendation (Public Services)	RDF, SHACL	LLM extraction, SHACL validation	Human Expert Assessment
Oranekwu (2026) [12]	Cybersecurity Compliance (IoT)	OWL, SPARQL	Ontology-driven RAG	Similarity over ground truth
Spyropoulos (2025) [13]	Entity Mining (Police Reports)	RDF, OWL	LLM Entity Extraction & linking	Visual and SPARQL Verification
Hanuragav (2025) [14]	CSR Validation (Medical)	SHACL, SPARQL	RTF to JSON to RDF with YAML mapper	SHACL (structure), SPARQL (content)
Category 2: Automated Logic Synthesis & Semantic Parsing				
Agarwal (2024) [15]	Complex QA (KGQA)	KoPL	SymKGQA: Symbolic Program Generation	Hits@1 and F1 on Benchmarks
Avila (2025) [16]	Scientific QA (KGQA)	SPARQL	Text-to-SPARQL with RAG + Few-shot ICL	F1 on Benchmarks
Jiang (2025) [17]	Multi-KG Generalization (KGQA)	SPARQL	Semantic sketch + KG population	Hits@1 and F1 on Benchmarks

Continued on next page...

Table 2.2 – continued from previous page

Study	**Core Task / Domain**	**Logic**	**Neuro-Symbolic Integration**	**Validation Method**
Shah (2024) [18]	Multi-hop QA (KGQA)	Cypher, SPARQL	Text-to-Logic with Few-shot + CoT	Match Accuracy on Benchmarks
Walter (2026) [19]	Reasoning / QA (Multi-domain)	SPARQL	Zero-shot Iterative Agent KG exploration	F1 on Benchmarks
Soularidis (2024) [20]	Rule Generation (Search & Rescue)	SWRL	Ontology-driven Text-to-SWRL	F1 on Human Expert
Lehmann (2023) [21]	Semantic Parsing (Wikidata)	CNL, SPARQL	Controlled Natural Language to Logic	Hits@1 on Benchmarks
Kovriguina (2023) [22]	SPARQL Generation (Fantasy)	SPARQL	Augmenting prompts with RDF subgraphs	F1-macro on Benchmarks
Mountantonakis [23] (2025)	Cultural Heritage (Art)	SPARQL	Path Pattern prediction + query generation	Accuracy on Benchmark
Ongris (2024) [24]	Wikidata QA (KGQA)	SPARQL	Sequential LLM Chaining + GraphRAG	Jaccard Similarity on Ground Truth
Vieira da Silva (2024) [25]	Capability Modeling (IoT)	OWL	TBox-contextualized prompting	Pellet (OWL reasoning) + SHACL
Emonet (2025) [26]	Federated QA (Bioinformatics)	SPARQL	ShEx/VoID-driven RAG query generation	Execution Success Rate and F1
Mashhaditafreshi (2025) [27]	Digital Twins (IoT)	RDF, SHACL	JSON to Aspect Models via bootstrapping	Human evaluation, Jena (RDF syntax)
Category 3: Evaluation, Stability & Trustworthiness				
Sequeda (2025) [28]	SQL Databases (Enterprise)	SPARQL	LLM query correction	Comparison with SQL ground truth
Allemang (2024) [29]	SQL Databases (Enterprise)	SPARQL	Ontology-based Error Detection + Repair	Execution Accuracy on Benchmark
Gashkov (2025) [30]	Query Filtering (Multilingual)	SPARQL	LLM-as-a-Judge binary classifier	Answer Trustworthiness on Benchmark
Adam (2025) [31]	Statement Verification (Bio-sci)	RDF	RAG using External Snippets	Precision / Recall on fixed dataset
Meyer (2025) [32]	KGE Benchmarking (Web)	RDF, SPARQL	LLM-KG-Bench 3.0 Framework	Parseable Syntax and F1

Continued on next page...

Table 2.2 – continued from previous page

Study	Core Task / Domain	Logic	Neuro-Symbolic Integration	Validation Method
Kosten (2024) [33]	Complex QA (KGQA)	SPARQL	Ontology-based prompt engineering	Execution Accuracy on Benchmark
Schmidt (2026) [34]	Systematicity Testing (Wiki)	SPARQL	CompoST: Compositional Testing	Compositionality F1 on ground truth
Tufek (2025) [35]	Artifact Validation (Industrial)	SPARQL	Zero-shot Instruction Prompting	Domain-specific Precision, Recall, F1

2.4 Thematic Analysis

After the described systematic selection process, all included studies were synthesized into a thematic analysis. The idea of this section is to escape the obsolete summarization of papers, but instead to construct a coherent narrative.

Three are the primary themes around which the this narrative is structured. First, an exploration of how high-stakes regulatory texts are being formalized in the present. Second, a technical dive into the mechanics of translating natural language into structured executable logic. Finally, a critical assessment of how the correctness and stability of these systems is being verified.

2.4.1 Neuro-Symbolic Pipelines in Public Administration

Neuro-Symbolic AI is frequently characterized as the integration of Large Language Models (LLMs) and Knowledge Graphs (KGs). It seeks to combine the flexibility of neural networks with the rigor of formal ontologies [29]. One of the many goals of this intersection is to address the complexities of public sector data management [9]. In this domain, authors are increasingly moving away from unstructured legislative texts and narrative reports and towards formal logic, to enable proactive and data-centric governance [9][13].

Knowledge Extraction and Formalization

The transition from unstructured text to formal logic typically follows a multi-stage pipeline. Ideally, such a pipeline is designed to preserve the semantic intricacies that are inherently present in legal rules, while trying to reduce hallucinations or at least control them as much as possible. [9][12].

Konstantinidis et al. [9] utilize LLMs to interpret complex legislative documents (in raw PDF format) which describe public services. First, they extract preconditions for eligibility and then, they translate them into SHACL (SHApes Constraint Language) rules. Their approach touches on Retrieval-Augmented Generation (RAG) and uses prompt chaining, with which they transform the raw input text into RDF-based evidence models. Notably, they make a point ensuring that the extracted rules are grounded in established EU standard vocabularies like *CPSV-AP* and *CCCEV*.

Similarly, Oranekwu et al. [12] employ a RAG pipeline to ingest regulatory texts and manufacturer privacy policies, using LLMs to extract subject-predicate-object triples that are then mapped into a compliance knowledge graph.

Spyropoulos and Tsiantos [13] focus on law-enforcement archives, using instruction-tuned models like OpenAI o3 on narrative police reports to extract entities and their interrelationships, subsequently converting this knowledge into OWL-compliant triples for ingestion into a triplestore.

The Role of Intermediate Models

Intermediate representations serve as *blueprints* or *mappers* that connect unstructured narratives with executable logic [13][14].

Hanuragav and Gopinath [14] demonstrate the utility of intermediate representations through a multi-stage pipeline designed for regulatory validation. In their framework, the transition from unstructured rich-text documents into formal RDF is facilitated by a JSON-to-YAML mapper. By using LLMs to draft YAML mapper files rather than direct triples, their architecture decouples the semantic extraction of data from the technical generation of the knowledge graph (thus essentially decoupling logic and reasoning from syntax and formality). This reliance on non-executable intermediate schemas suggests a shift toward modularity in public sector pipelines, where the LLM's role is confined to architectural drafting, a failsafe to ensure that the resulting logic is structurally "anchored" before final conversion.

Konstantinidis et al. [9] also utilize intermediate steps to formulate natural language rules into a template format before final SHACL generation, allowing for the hierarchical structuring of evidence data.

Spyropoulos and Tsiantos [13] employ intermediate tabular forms to organize recognized entities before they are formally mapped to the OWL ontology. As a side-effect, they report this makes human-in-the-loop validation much easier.

Current Limitations

Despite results showing promise, the afformentioned systems are currently characterized as conceptual or pilot-scale prototypes [9][12].

Konstantinidis et al. [9] emphasize that their pipeline is not yet end-to-end operational and faces significant hurdles regarding data fragmentation across administrative silos.

A primary critique of current methods is the lack of automated testing at scale. For instance, Oranekwu et al. [12] note that their ground truth dataset remains limited in statistical generalizability and has not yet undergone testing with end-users, in real-world conditions.

Furthermore, Spyropoulos and Tsiantos [13] admit to the use of simulated reports rather than authentic documents due to confidentiality, which may raise concerns about not fully capturing the complexity of real-world law-enforcement data.

Proposing future work in this sector, authors focus on overcoming legal and policy complexities, as continuous updates are required to accommodate rapidly evolving regulations [9][12]. Authors also suggest that federated Knowledge Graphs and decentralized technologies (like blockchain) may be necessary to address issues of data ownership and privacy compliance (such as GDPR requirements) [9]. Additionally, there is an identified need for more robust benchmarking methods to validate AI-driven interpretations against human-expert evaluations in high-stakes public environments [9][12].

2.4.2 State-of-the-art in Logic Synthesis

The challenge of logic synthesis appears to have evolved rapidly. An effort is being made to move toward modular, Neuro-Symbolic pipelines that break down the task of semantically parsing natural language into manageable logical components [15][17]. These state-of-the-art approaches take advantage of the linguistic fluency of Large Language Models while enforcing the structural constraints of Knowledge Graphs, through various technical mechanisms and intermediate representations [19][22][26].

Technical Mechanisms for Translation

Notable Neuro-Symbolic techniques for translating natural language to structured logic include Few-shot In-Context Learning (ICL) [16], Retrieval-Augmented Generation (RAG) [26] and Iterative Agentic Exploration [19].

Frameworks such as *SymKGQA* [15] combine few-shot ICL with function definitions, to generate symbolic programs in *KoPL* (Knowledge Oriented Programming Language), allowing step-by-step reasoning that is independent of the model's pre-trained knowledge of language grammars. Shah et al. [18] further enhance this via what they refer to as "Planned Query Guidance", where few-shot examples demonstrate a code-style reasoning process that handles multi-hop transitions line-by-line.

To ground logic in specific KG schemas, authors utilize RAG variations that inject minimal subgraphs [22], *VoID* (Vocabulary of Interlinked Datasets) descriptions or *ShEx* (Shape Expression) schemas into the prompt [26]. For example, *SPARQLGEN* [22] enriches prompts with a minimal RDF subgraph, sufficient to answer the query, reducing the need for models to memorize the entire graph. Emonet et al. [26] utilize *ShEx* to define available predicates for specific classes, which proved to significantly improve the model's ability to generate valid federated queries.

The use of RAG is further extended beyond simple fact retrieval to the generation of *ABox* (Assertional Box, storing factual statements) instances for complex domain models. Vieira da Silva et al. [25] demonstrate that providing the full *TBox* (Terminological Box, the schema-level knowledge of an ontology), within the prompt as context is essential. Their findings show that this approach reduced hallucinations when modeling industrial capabilities. By explicitly injecting the *TBox*, the LLM is forced to conform with predefined class hierarchies and relationship constraints, such as domain - range axioms. With this contextualization, the authors claim to ensure that generated instances are both linguistically plausible and logically consistent with the underlying ontology. The result is the successful reduction of of model contradictions and invented properties.

Most recent research introduces iterative frameworks like *GRASP* [19], which treats the LLM as an agent. The agent is tasked with exploring a graph through sequential function calls (search, list, execute). Authors claim that this approach allows the model to refine its understanding of the graph's topology iteratively, without being constrained by a fixed context window. Similarly, *SAMM Copilot* [27] employs iterative prompting and feedback loops to generate semantic Aspect Models from JSON data.

Beyond the choice of underlying model, the selection of the formal target language itself has become a point of contention. A significant shift in logic synthesis came with the proposal by Lehmann et al. [21] to relinquish formal languages like SPARQL as the target logical form and instead use Controlled Natural Languages (CNLs), such as *SQUALL* or *Sparklis*. The authors argue that CNLs will require significantly less fine-tuning to achieve high accuracy, since they are linguistically closer to natural language text, huge amount of which is used in LLM pre-training data, and also linguistically closer to the input question. Their findings indicate that despite LLMs struggling with the strict syntax of SPARQL, they show a "deeper understanding" of CNLs, allowing them to generate valid syntactic variations that are semantically equivalent to ground truth. For especially complex queries (comparatives, ordinals, differences), switching the parsing target from SPARQL to a natural and compact language like *SQUALL* seems to as much as double the semantic parsing accuracy.

Managing Structural Complexity

Handling complex schema mapping, particularly in event-based or multi-hop scenarios, requires more advanced strategies than simple triple-matching [17][23].

Jiang et al. [17] propose *OntoSCPrompt*, a two-stage architecture that separates Query Structure Prediction (dubbed Stage-S) from KG Content Population (dubbed Stage-C). In the first stage, the model predicts a sketch or "skeleton" of the SPARQL query, using special placeholders (e.g., [ent], [rel], [cct]), which is then populated with graph-specific identifiers in the second stage.

For event-based models like *CIDOC-CRM*, where answering a single question often requires traversing long complex paths, Mountantonakis and Tzitzikas [23] introduce a two-stage process using *Ontology Path Patterns*. Their method first predicts the required properties and classes to identify relevant paths of a specific radius before synthesizing the final query, effectively reducing the search space for the LLM.

Descriptive vs. Prescriptive Logic Synthesis

At this point of the analysis, it is important to make a distinction. While many (the majority, in fact) technical advancements focus on *Descriptive* Question Answering (QA), which is retrieving factual data such as birthplaces or award winners [15][19][24], a distinct and smaller subset of research addresses the synthesis of *Prescriptive* or Normative Rules [9][20].

Systems like *GraphRAG* [24] and *UniKGQA* [17] primarily focus on factual retrieval, done by translating natural language into executable queries (SPARQL, Cypher) to fetch stored static values [15][19].

In contrast, Soularidis et al. [20] and Konstantinidis et al. [9] attempt to synthesize formal logic that encodes rules for behavior or eligibility. Soularidis et al. utilize template-driven prompts and RAG to translate natural language rules from the Search and Rescue (SAR) domain into *SWRL* (Semantic Web Rule Language). Similarly, Konstantinidis et al.

use LLMs to extract regulatory preconditions from legislative texts and formalize them as SHACL shapes, which serve as machine-readable rules for public service eligibility checks and recommendations. Oranekwu et al. [12] create a union between the two, by extracting triples from manufacturer privacy policies to verify compliance against structured *NIST* standards.

2.4.3 Methodologies for Logic Validation and Trust

If Neuro-Symbolic systems are to transition from conceptual pilots to operational environments, the methodologies we use in order to to validate their outputs need to become a priority for research. This idea is already shifting the emphasis from simple performance metrics to the establishment of trust and traceability [21]. Knowledge Graphs serve as the fundamental *source of trust* in these architectures, providing a formal framework to evaluate the validity of queries generated by Large Language Models and acting as a foundation for explaining results [28].

Knowledge Graphs as a Source of Grounding and Repair

The integration of KGs into the validation pipeline provides the necessary grounding to remedy the risks that stem from the probabilistic nature of LLMs [21].

Allemang and Sequeda [29] introduce *Ontology-based Query Check (OBQC)*, a deterministic approach that utilizes the semantic constraints of an ontology (such as domain - range rules) to identify errors in generated SPARQL queries without relying on an LLM. When an error is detected, an LLM-powered repair mechanism utilizes the previously output deterministic error explanations, to iteratively prompt the model for correction, a cycle that continues until the query passes the ontological rules or reaches a predefined iteration limit.

This focus on traceability is further advanced by Adam and Kliegr [31], who propose an inherently traceable approach to verification. They instruct LLMs to avoid (internal) factual knowledge and instead find justification for RDF statements within retrieved (external) text snippets. Part of the LLM instructions becomes to directly generate references for every claim they make in the process and the end result.

Probabilistic vs. Deterministic

Current literature reveals a quite clear separation between probabilistic and deterministic validation techniques.

Probabilistic approaches often rely on benchmarking and "LLM-as-a-Judge" frameworks. Gashkov et al. [30] employ instruction-tuned LLMs as binary classifiers to act as post-processing filters, removing potentially incorrect SPARQL queries to enhance their *Answer Trustworthiness Score (ATS)*. Similarly, Kosten et al. [33] and their *Spider4SPARQL* benchmark [36] evaluate model performances across queries of varying levels of difficulty. They then find that even state-of-the-art models struggle to surpass 51% accuracy on complex multi-hop tasks. Meyer et al. [32] provide an extensible framework, *LLM-KG-Bench 3.0*, which automates the evaluation of LLM answers using metrics such as normalized triple F1 and string similarity.

In contrast, high-stakes domains prioritize deterministic approaches and rigid constraints. Tufek et al. [35] focus on validating semantic artifacts against industrial standards (e.g., *OPC UA*), where natural language requirements are translated into SPARQL queries to ensure compliance. The requirement for absolute precision in high-stakes domains has led to the adoption of the *"draft only, never execute"* paradigm As proposed by Hanuragav and Gopinath [14], this methodology explicitly prohibits the Large Language Model from direct interaction with the triple store. Instead, the model's output is restricted to intermediate mappers that are subsequently processed by a deterministic Python translator. This creates a symbolic "circuit breaker", ensuring that the final RDF output is idempotent, auditable and strictly compliant with regulatory requirements. This is a level of reliability that probabilistic filtering methods struggle to guarantee. Furthermore, SHACL shapes are frequently used as *logical gates* to enforce structural integrity and detect hallucinations in digital twins and public administration service models [9][14][25].

Stability and Systematicity

A critical shortcoming in current validation research is the fact that models fail to maintain logical consistency across novel tasks. Schmidt et al. [34] investigate *systematicity*, defined as the ability of an agent to understand complex expressions built from known components, through the *CompoST* benchmark. Their findings indicate that LLMs have trouble interpreting questions when the complexity of their components deviates from their training samples. In fact, performance scores rarely exceeded 0.57 even under optimal self-contained conditions. This highlights that most current validation is static, that is, comparing an LLM's output against a single "golden standard" answer in a single-pass execution [18][21][28].

2.5 Discussion and Research Gap

The systematic review of the current literature paints a certain picture. The trajectory of Neuro-Symbolic research appears to be moving away from monolithic sequence-to-sequence translation and towards modular pipelines. These pipelines tend to distinguish semantic parsing from logical execution. However, the synthesis of the included studies identifies three fundamental gaps that remain unaddressed, as explained next.

2.5.1 The Prescriptive Synthesis Gap: From Facts to Rules

While the broader field of logic synthesis has achieved high accuracy in descriptive tasks, there is a marked lack of research into the synthesis of prescriptive governance logic. As established in the thematic analysis, dominant frameworks such as *UniKGQA* [17] and *GRASP* [19] focus almost exclusively on Knowledge Graph Question Answering (KGQA), that is, translating natural language into queries to retrieve stored facts. In contrast, the requirements of digital governance necessitate the synthesis of *rules* rather than just *questions*.

If we revisit Table 2.2, we can see that even within Category 1 (Public Administration), where we can find studies like Konstantinidis et al. [9] that attempt to extract service preconditions, the methodology remains largely conceptual. There is a noticable absence of a means to ensure that synthesized prescriptive logic (SHACL shapes) can survive the edge cases of diverse citizen profiles.

2.5.2 The Stability Gap: Correctness vs. Resilience

A significant paradox emerges in how current literature defines "trust" and "correctness." Current state-of-the-art mechanisms, such as *Ontology-based Query Check (OBQC)* and *LLM-Repair* [28][29], are designed as one-off error-correction loops. These systems simply ensure that a specific generated query is semantically valid according to an ontology schema for *a single* execution pass.

However, in the context of public service eligibility, the concept of a synthesized rule being similar to a one-off query does not respond well to reality. On the contrary, a rule is more of a permanent logic structure, and one intended for repeated application across potentially heterogeneous datasets. The literature focuses on making the LLM a more accurate translator momentarily, but fails to validate if the resulting logic is *stable* across a spectrum of different inputs. There is no evidence of research into whether a synthesized

SHACL shape, once generated, remains logically consistent when tested attributes are systematically altered, a requirement for the proactive "No-Stop Government" vision [9].

2.5.3 The Validation Gap: The Need for Mutation Testing

The most critical shortage identified by this research is the reliance on static or otherwise single-pass validation methodologies. Benchmarks like *CompoST* [34] and *LLM-KG-Bench 3.0* [32] measure accuracy by comparing model outputs against a "golden standard" answer. These metrics are definitely useful for measuring retrieval precision, but they appear to be insufficient for verifying the functional correctness of eligibility rules.

As identified in Table 2.2, current validation is predominantly probabilistic. Even deterministic approaches, for example those by Hanuragav and Gopinath [14] or Tufek et al. [35], focus on structural syntax or version-control compliance. To date, no study has implemented a comprehensive *Mutation Testing Framework* to strictly test the structural and logical stability of LLM-generated SHACL shapes. For high-stakes digital governance, an 88% precision rate (as reported in descriptive tasks [31]) is not a useful nor representative result. Future frameworks must address the variability of non-deterministic systems by establishing testing methodologies that guarantee logical stability across constantly evolving regulatory landscapes.

While Sequeda et al. [28] acknowledge the importance of regression testing to ensure that accuracy does not decrease as ontologies are extended, there is no evidence of widespread use to ensure logic remains resilient under variable conditions. For high-stakes governance and public administration, measuring accuracy on a single pass is simply insufficient.

2.6 Contribution of this Study

Based on the gaps identified above and the established methodologies described in modern literature, this dissertation proposes a Neuro-Symbolic framework that utilizes a *JSON Information Model* as an intermediate architectural stepping stone to generate SHACL-based eligibility rules. SHACL is chosen specifically for its ability to act as a *logical gate*, providing the unified graph structure and explainability required for public administration.

To address the topic of validation, this dissertation introduces a custom deterministic *Mutation Testing* framework. By systematically altering (mutating) citizen attributes to evaluate the rejection rate of synthesized SPARQL logic, we try to move beyond probabilistic retrieval to provide a functional guarantee of stability.

This approach, explained in granular detail in the following Pilot Study chapter, represents a transition from checking if a model is correct *once*, to verifying that the synthesized law is functionally infallible across variable scenarios.

3 Pilot Study

This chapter details the design, implementation and experimental testing of the pipeline that acts as the decision logic component of the proactive recommender architecture outlined in Section 1.4. The full implementation is made publicly available on GitHub [10].

3.1 Overview

The proposed architecture attempts to address the limitations of the "black-box" nature of Large Language Models, and the uncertainly that nature entails [37], by enforcing a strict separation between neural interpretation (to extract meaning from text) and symbolic execution (to validate logic against data).

The methodology is structured around a *Text-to-Graph-to-Logic* workflow. The system transforms unstructured administrative documents into formal Knowledge Graphs and executable SHACL shapes through a chain of intermediate structured representations. This design prioritizes explainability and determinism, ensuring that the final eligibility decision is derived from explicit, auditable rules rather than probabilistic approaches.

This chapter is organized as follows: Section 3.2.2 defines the semantic schemas that ground the system. Section 3.2.3 details the four-stage extraction and generation pipeline. Section 3.2.4 describes the validation engine, and Section 3.3 outlines the experimental framework used to stress-test the system's capabilities through automated mutation testing.

3.2 Methodology and System Architecture

3.2.1 Setup Environment

The pipeline was implemented using Python 3.12.9, utilizing a modular architecture to separate core processing logic from experimental orchestration. The system relies on local processing for semantic graph operations and cloud-based APIs for interfacing with Large Language Models.

System Architecture

The codebase follows a functional separation of concerns, organized into three distinct layers:

1. **The Core Logic Layer:** A modular custom Python library encapsulating the functional logic of the system. It contains the reusable logic, such as API communication, graph operations, parsing and testing utilities. It also contains the end-to-end extraction-generation workflow, dubbed the *pipeline core*.
2. **The Orchestration Layer:** An interactive Jupyter Notebook serves as the control interface. This layer manages the experimental loop, injects configuration variables into the core modules and handles exceptions without interrupting the iterative execution of experiments.
3. **The Persistence Layer:** As explained already, auditability and reproducibility are crucial. For this reason, the system employs a meticulous artifact preservation strategy. Every experimental run generates a dedicated directory (locally), containing all intermediate outputs of the core pipeline. Furthermore, during testing, metrics and metadata are saved in a Master CSV file for post-hoc analysis.

Technologies and Libraries

The system combines standard Semantic Web technologies with modern Data Science tools:

- **RDFLib:** Used for parsing, manipulating and serializing RDF graphs (in Turtle format), as well as executing local SPARQL queries.
- **PySHACL:** The standard Python implementation of the SHACL validation engine. It is used to validate the LLM-generated shapes against citizen data.
- **Pydantic:** Used for schema enforcement of the intermediate JSON representations, acting as a structural "gatekeeper". Its objects are sent to the LLM API, which in turn makes sure that the responses from the LLM adhere to strict data types and structural constraints.
- **Pandas:** Used for the post-hoc data analysis of the testing logs.
- **Protégé:** An open-source ontology editor used for designing and consistency checking the domain-specific Schemas.
- **VOWL:** Utilized for the standardized visualization of the RDF Schemas, both during the conceptual design phase and the presentation of the work.

3.2.2 Semantic Data Modeling

This pipeline was designed specifically with public service documents in mind. To make the transition from unstructured administrative text to deterministic validation logic, two distinct semantic layers were defined. These schemas serve as the symbolic grounding for the Large Language Model.

The Public Service Meta-Model

The modeling of the public service draws components from European formal vocabularies, specifically the *Core Public Service Vocabulary Application Profile (CPSV-AP)* [38] and the *Core Criterion and Core Evidence Vocabulary (CCCEV)* [39]. The schema follows the following hierarchical structure, simplified in Figure 3.1:

- **cpsv:PublicService**: The root node representing the public service itself.
- **cccev:Constraint**: Connected to the root node via `cpsv:holdsRequirement`, these nodes represent individual preconditions extracted from the text.
- **cccev:InformationConcept**: These nodes are connected to Constraint nodes via `cccev:constrains` and represent the abstract information required to evaluate a constraint.

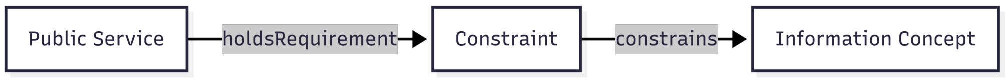


Figure 3.1: Simplified view of the Public Service Meta-Model

The adoption of established EU standards is a deliberate architectural choice, made to allow for cross-border interoperability and extensibility [40]. By anchoring the pipeline's output in the CPSV-AP and CCCEV ecosystems, the generated graphs are compatible with the broader European e-Government infrastructure (such as the Single Digital Gateway). Furthermore, it is hoped that this modular design will allow for future expansion where the pipeline could automatically ingest the entirety of these ontologies (complex Evidence mappings, Agent definitions, Output representations), without requiring any restructuring of the core logic.

Citizen Schema

If the Public Service Meta-Model describes the *rules*, then the Citizen Schema describes the *applicant*. This work utilizes a domain-specific *Resource Description Framework Schema (RDFS)* [41] tailored to the requirements of each document and generated in a separate workflow (not presented here) by the same LLM used in the implementation of the rest of the pipeline. The model is instructed to use granular instead of aggregate data as nodes (e.g. prefer "Date of Birth" rather than "Age") and is encourgaed to use abstract and reusable classes.

It has been demonstrated that the generation of such schemas can be automated as part of the pipeline [9]. However, for the scope of this pilot study, the Citizen Schema is treated as fixed input context. This choice serves two purposes:

1. **Experimental Control:** By fixing the target schema, we isolate the performance of the LLM in *logic generation* (SHACL/SPARQL) and *extraction*, without the confounding variable of schema generation errors.
2. **Prerequisite for Testing:** The automated testing framework that will be presented relies on injecting specific faults into the citizen graph (e.g., modifying property values to trigger violations). This requires a schema structure that is known in advance. Indeed, had the schema been generated dynamically during each run, it would be impossible to define a library of test scenarios targeting specific graph nodes.

Visualizations of the Schemas created for each document use case can be found in Section 3.3.2, while the full `ttl` files of the schemas created can be found in Appendix C.

3.2.3 The Extraction & Generation Pipeline

The core contribution of this work is the following multi-stage, Neuro-Symbolic pipeline. The process follows a sequential data flow, depicted in Figure 3.2, consisting of four primary stages. The code for this core part of the pipeline can be found in Appendix A.

Stage 1: Document Summarization and Preconditions Extraction

The pipeline begins with the ingestion of the raw public service document (PDF). Using a Large Language Model, the unstructured text is processed to extract a summary of eligibility preconditions. The prompt is designed to filter out administrative noise and standardize the format of the rules. Since this is the first and most vital piece of information to be passed downstream, summarization is necessary as it reduces the cognitive load required for the subsequent logic generation steps.

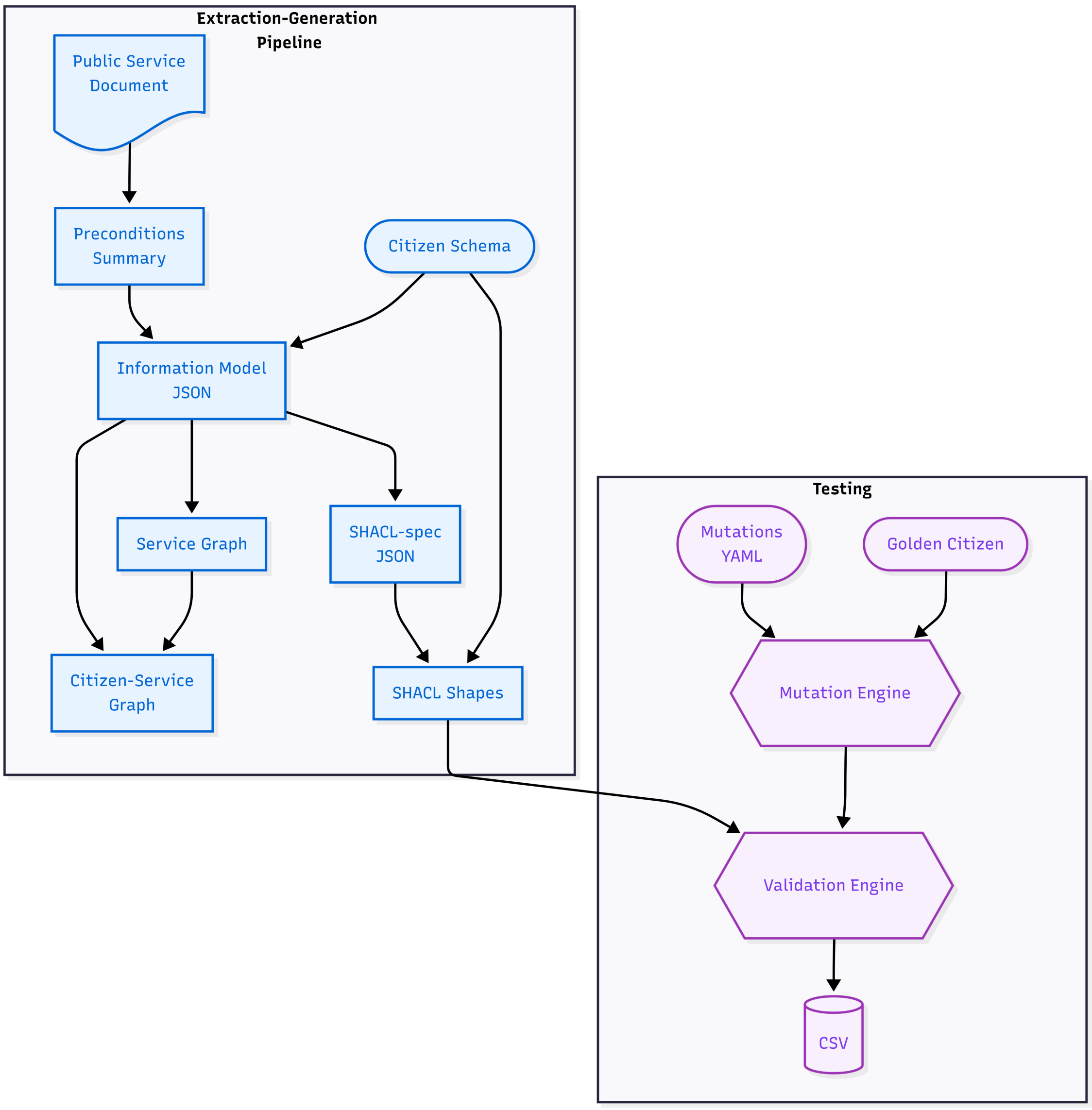


Figure 3.2: Flow Chart of the core pipeline and testing framework

Stage 2: Information Model Generation

This is a critical step, and the first one that makes this pipeline "Neuro-Symbolic". Here, the extracted preconditions are transformed into a structured JSON representation coined the *Information Model*. The Information Model organizes the unstructured list of rules into a strict hierarchy that mirrors the Meta-Model structure:

- **Constraints:** Each eligibility rule is encapsulated as a Constraint object, containing the natural language description of the rule.
- **Information Concepts:** Nested within each Constraint are the abstract Information Concepts, representing the specific pieces of evidence or data required to evaluate that rule.

Inferring these concepts from the list of preconditions is the main reasoning task of the LLM at this stage. However, a second task it is prompted with is to act as a semantic mapper. The LLM is provided with the Citizen Schema (defined in Section 3.2.2) as a strict vocabulary constraint to prevent the hallucination of non-existent properties. With it, it is instructed to connect each Information Concept with a number of Citizen nodes, by constructing specific traversal paths through the ontology (e.g., mapping the concept of "Applicant Age" to the path `:Applicant` $\rightarrow$ `:birthDate`). This method has been shown to potentially increase the consistency and reliability of the produced results [42].

The resulting artifact effectively creates a *blueprint* for downstream tasks. It contains all the necessary semantic links to be deterministically serialized into valid RDF triples in the subsequent stage, while ensuring that all data references are grounded in the controlled vocabulary of the Citizen Schema.

Listing 3.1 provides an example snippet of the JSON Information Model, showing one constraint with its associated information concepts inside the `constrains` field, and the Citizen Schema nodes it maps to in the `related paths` field. Note the `desc` field for the natural language description of the precondition, as it was interpreted from the document by the LLM during Stage 1.

Listing 3.1: JSON Information Model snippet

```
{
    "name": "family_income_condition",
    "desc": "Annual family income must not exceed 30,000,
        increased by 3,000 for each dependent child after the
        first.",
    "constrains": [
      {
        "name": "total_family_income",
        "related_paths": [
          {
            "path": ["hasIncome","amount"],
            "datatype": "xsd:decimal"
          },
          {
            "path": ["hasParent","hasIncome","amount"],
            "datatype": "xsd:decimal"
          }
        ]
      },
      {
        "name": "is_dependent_child_of_parents",
        "related_paths": [
          {
            "path": ["hasParent","hasChild","isDependent"],
            "datatype": "xsd:boolean"
          }
        ]
      }
    ]
}
```

The structure of the output is strictly enforced using a *Pydantic* schema definition. Pydantic is a data validation library for Python that enforces type hints at runtime. The Pydantic models are explicitly coded and passed to the LLM API as a structural constraint. The LLM is forced to restrict its token generation process to the exact structure defined by the Pydantic model. By enforcing a strict schema at the output, the system ensures that any structural hallucinations (malformed nested objects, incorrect data types) are *intercepted* before they reach the RDF serialization logic. This choice provides a safety net that is often missing in monolithic LLM implementations. Listing 3.2 is the Pydantic representation of the Information Model schema in the pipeline's core Python environment.

Listing 3.2: Pydantic Model for Information Model Generation

```python
class Paths(BaseModel):
    path: List[str]
    datatype: str

class InformationConcept(BaseModel):
    name: str
    related_paths: List[Paths]

class Constraint(BaseModel):
    name: str
    desc: str
    constrains: List[InformationConcept]

schema = list[Constraint]
```

Stage 3: Semantic Graph Construction

Once the Information Model is established, the system deterministically (via Python code) constructs two RDF artifacts without further LLM inference:

1. **The Service Graph:** A formal representation of the public service using the *CPSV-AP* and *CCCEV* vocabularies and following the Meta-Model schema defined in Section 3.2.2.
2. **The Citizen-Service Graph:** By loading an *Example Citizen* (a valid applicant instance), the system uses the Information Model to link the abstract Information Concepts from the Service Graph directly to the actual data nodes in the Citizen Graph via `ex:mapsTo` edges, completing the bridge. This unified graph serves as a visual audit trail, allowing human inspectors or automated agents to trace exactly which specific data points are being used to evaluate a specific constraint.

Figure 3.3 expands upon the simplified view of the meta-model introduced in Section 3.2.2 to present the completed bridge.

Both Graphs are serialized using Turtle syntax [43] and saved to file as artifacts. Interactive visualizations of them are generated using the *PyVis* library and also saved to file as `html` files. Figure 3.4 shows a simplified view of a citizen-service graph as it was automatically produced by the pipeline. The yellow node is the `cpsv:PublicService` node, the red

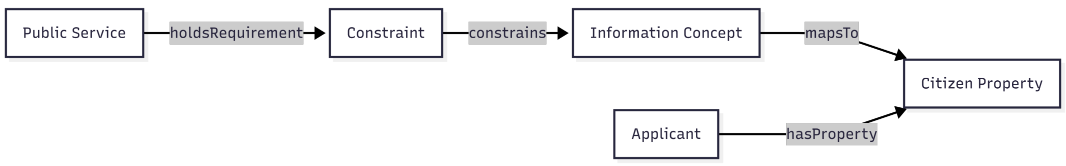
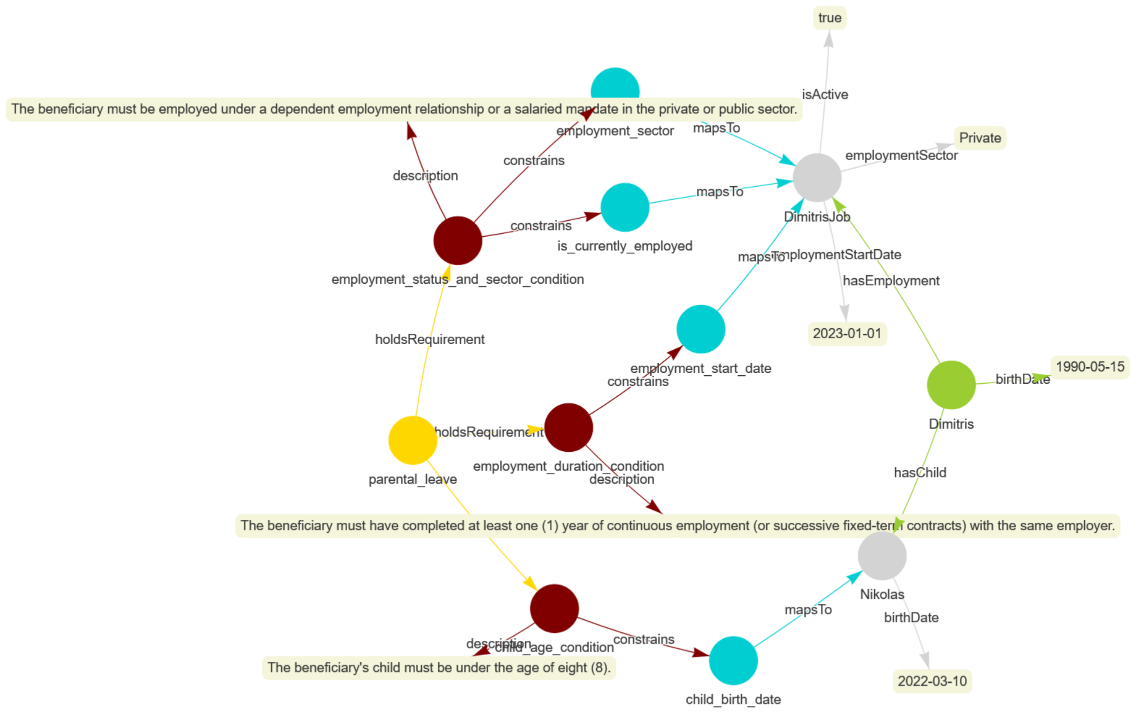


Figure 3.3: Simplified view of the Citizen-Service Graph blueprint

nodes are `cccev:Constraint` nodes, the blue nodes are `cccev:InformationConcept` nodes. These connect to the Citizen nodes (Green, the root, and grey, the properties) completing the Citizen-Service Graph.



true
isActive
The beneficiary must be employed under a dependent employment relationship or a salaried mandate in the private or public sector.
employment_sector
mapsTo
Private
employmentSector
description
constrains
mapsTo
constrains
is_currently_employed
DimitrisJob
employment_status_and_sector_condition
mapsTo
employmentStartDate
hasEmployment
2023-01-01
holdsRequirement
employment_start_date
constrains
1990-05-15
birthDate
Dimitris
holdsRequirement
parental_leave
employment_duration_condition
description
hasChild
The beneficiary must have completed at least one (1) year of continuous employment (or successive fixed-term contracts) with the same employer.
Nikolas
mapsTo
birthDate
description
child_age_condition
constrains
The beneficiary's child must be under the age of eight (8).
child_birth_date
2022-03-10

Figure 3.4: An example citizen-service graph according to the Meta-Model

The full `ttl` serializastion of the visualized example graph can be found in Appendix B.

Stage 4: SHACL Shapes Generation

The final and arguably most crucial stage of the pipeline is the one responsible for synthesizing the executable validation logic. This stage transforms the abstract requirements from the Information Model into a *Shapes Constraint Language (SHACL)* [44] graph. It should be noted that this is also (theoretically) the most demanding task the LLM performs throughout this pipeline, from a semantic understanding perspective. The generation happens in two steps.

Firstly, the system deterministically distills the rich Information Model into a simplified, noise-free JSON structure termed the *SHACL-Spec JSON*. This intermediate representation reorganizes the structure and retains only the logical primitives required for validation (e.g. rules, target paths and data types). This step acts as a context cleaner, helping the LLM focus exclusively on code synthesis.

Secondly, the LLM is invoked to translate this specification into RDF triples (Turtle format). The model is once again restricted to using the fixed Citizen Schema, which is once again given as context to act as a failsafe, in case earlier path generation failed to include crucial nodes. The prompt enforces a Dual-Strategy Protocol for logic synthesis, as explained below.

For atomic constraints involving single-hop properties and literal comparisons (e.g., `Citizenship = 'GR'`), the model is instructed to prioritize the use of *SHACL-Core*, with standard `sh:property` shapes. SHACL-Core provides a rich set of structural and logical components (e.g., `sh:and`, `sh:or`, `sh:not`) and can encode fairly complex conditional logic and exception patterns [45]. Using the built-in components of the SHACL vocabulary, the system exploits the most stable and least error-prone usage of the language.

For requirements involving arithmetic, aggregations, date comparisons or crossreferenced data (e.g., `now() - birthDate > 18`), the model is instructed to use *SPARQL-based constraints* as per the standard and recommended approach in the literature [46][47], encapsulating the logic within a `SPARQL Constraint` node. This allows for the expression of complex conditional logic that exceeds the expressivity of the SHACL-Core vocabulary [48].

This hybrid approach prevents *over-engineering* simple rules. By avoiding unnecessary SPARQL generation for straightforward checks, we significantly reduce the probability of avoidable syntax and logical errors, ensuring that the heavy reasoning capabilities of SPARQL are reserved for when the core vocabulary is insufficient.

As a last addition, the LLM generates an error message for every shape, which is intended to be displayed as part of the Validation Engine report in case of a violation (e.g., "Income exceeds threshold"). Remembering the wider real-world context of the Recommender System, for which this pipeline is intended, this message would be useful for Citizen feedback in cases of marginal conformance or non-conformance.

The output is a fully serialized `ttl` file containing the `sh:NodeShape` definitions. This file serves as the executable input for the Validation Engine, the mechanics of which are detailed in the next section.

Listing 3.3 shows a SHACL Shape and its embedded SPARQL query generated by this process. Specifically, it is the shape that corresponds to the constraint we showed in Listing 3.1 in its Information Model form.

Listing 3.3: An example SHACL Shape generated by the pipeline

```
:family_income_shape
    a sh:NodeShape ;
    sh:targetClass :Applicant ;
    sh:sparql [
        a sh:SPARQLConstraint ;
        sh:message "Annual family income exceeds the limit." ;
        sh:select """
            SELECT ?this
            WHERE {
                {
                    SELECT ?this (SUM(?val) AS ?totalIncome)
                       WHERE {
                        { ?this :hasIncome ?inc .
                          ?inc :amount ?val . }
                        UNION
                        { ?this :hasParent ?p .
                          ?p :hasIncome ?p_inc .
                          ?p_inc :amount ?val . }
                    } GROUP BY ?this
                }
                OPTIONAL {
                    {
                        SELECT ?this (COUNT(DISTINCT ?child)
                           AS ?depChildCountResult) WHERE {
                            ?this :hasParent ?parent .
                            ?parent :hasChild ?child .
                            ?child :isDependent true .
                        } GROUP BY ?this
                    }
                }
                BIND(COALESCE(?depChildCountResult, 0) AS ?
                   depChildCount)
```

```
                    BIND(30000 + (3000 * (IF(?depChildCount > 1, ?
                        depChildCount - 1, 0))) AS ?limit)
                    FILTER(?totalIncome > ?limit)
                }
            """ ;
    ] .
```

Some complete, correct examples of artifacts produced by the pipeline can be found in the dissertation's GitHub repository [10] under the "Good Results" directory.

3.2.4 The Validation Engine

The final component of the architecture is the Validation Engine. While the afformentioned pipeline stages focused on structuring and grounding the data, this engine is responsible for applying the generated constraints against specific citizen data to render a final eligibility decision.

The engine requires two distinct RDF graphs to operate:

- **The Shapes Graph:** The `ttl` file generated by Stage 4 of the pipeline, containing the `sh:NodeShape` definitions and SPARQL constraints within.
- **The Data Graph (Citizen Instance):** An RDF graph representing a specific applicant (a concrete instantiation of the Citizen Schema). It contains the factual assertions about an individual, structured strictly according to the domain ontology.

For the execution and reasoning part, the system utilizes *PySHACL*, a Python-based implementation of the W3C SHACL standard, to perform the validation. The execution follows a standard protocol:

- **Targeting:** The engine identifies the *Focus Node* in the Data Graph (defined during Stage 4 as the sole instance of the class `:Applicant`).
- **Constraint Evaluation:** The engine evaluates the logic that corresponds to every Shape mapped to the Applicant. Simple property shapes are validated via graph traversal. Complex conditions trigger the execution of the embedded SPARQL queries against the Data Graph.
- **Entailment:** The engine operates under the RDFS entailment regime, allowing it to infer class hierarchies (e.g., understanding that a `:Child` is also a `:Person`) during validation.

The output of the engine is a formal *Validation Report Graph* adhering to the SHACL standard. This report provides as output:

1. **Boolean Conformance:** A `sh:conforms` value (can be either True or False). This serves as the system's final decision on eligibility.
2. **Violation Details:** A set of `sh:ValidationResult` nodes in cases of non conformance. Each one links to the specific Shape that failed and includes the generated error message, providing explanation for the rejection.

3.3 Experimental Design

To evaluate the reliability, functional correctness and operational stability of the proposed architecture, a custom experimental framework was developed. The design moves beyond simple anecdotal testing, implementing a means to quantify the performance of the Neuro-Symbolic pipeline under varying conditions.

The core unit of the experiment is defined as a "run". A run represents a single end-to-end execution of the pipeline, characterized and configured by a specific combination of variables, dubbed the *Configuration Tuple*:

```
(Document, Model, Prompting Strategy)
```

Even when generating executable code, Large Language Models are inherently non-deterministic when operating at non-zero temperature settings and can generate vastly different results given the exact same configuration (inputs, prompt, context) [49]. For some model architectures, even setting temperature to *zero* reduces but *does not eliminate* non-determinism [50]. Consequently, a single generation is insufficient to prove or disprove their reliability.

To address this, the framework executes a loop of multiple iterations for each unique configuration. Indeed, empirical recommendations emphasize multiple generations to draw reliable conclusions [51]. This repetition allows for "drowning out" stochasticity and for the results metrics to converge to values that describe the actual stability of the pipeline with more fidelity.

The execution of these runs is done in The Orchestration Layer (see Section 3.2.1), which oversees the following lifecycle for every iteration:

1. **Context Initialization:** At the start of a run, a dictionary is initialized. This volatile data structure acts as a "flight recorder", accumulating outputs and metadata.
2. **Pipeline Execution:** The extraction and generation pipeline is triggered. If the pipeline encounters a critical failure, the failure mode is logged and the run is marked as incomplete before moving to the next.

3. **Scenario Validation:** Upon successful generation of a valid SHACL graph, the system proceeds to the Mutation Testing phase (detailed in the following subsection), where the generated logic is stress-tested against a battery of specific, pre-made scenarios.
4. **Persistence:** Finally, the accumulated metrics are flushed to a CSV file. Results are persisted immediately to prevent data loss during long-running batch experiments.

3.3.1 The Mutation Testing Framework

The functional correctness of the generated SHACL shapes certainly needs to be evaluated in a quantifiable manner. For this task, the system implements a Mutation Testing Framework. Mutation testing is a methodology that has matured and gained popularity in evaluating test suites and supporting experimentation [52]. Unlike traditional unit tests that might check for static string matches, this framework dynamically generates RDF graph instances to test whether the generated logic correctly distinguishes between eligible and ineligible applicants. The framework operates on a *Baseline and Perturbation* model, consisting of the components analyzed below.

The Golden Citizen Baseline

For each public service document, a single, syntactically perfect RDF graph termed the *Golden Citizen* is manually constructed. This data instance represents an applicant who satisfies *all* eligibility preconditions. This baseline graph is constructed to adhere strictly to the corresponding Citizen Schema of each document. The data values are calibrated to demonstrate marginal eligibility (e.g., if an income upper limit is €12,000, the Golden Citizen might have €11,999). This ensures that the testing framework evaluates the precision of the logic, not just its general functionality.

The full `ttl` files of these hand-crafted instance graphs are found along with their RDFS files in Appendix C.

Scenarios

The test cases, dubbed *Scenarios*, represent slight but critical alterations to the baseline citizen. They are defined in a declarative YAML configuration file. The file contains YAML descriptions of distinct Scenarios, each designed to test a specific logical constraint found in the document, if possible, in isolation. A Scenario definition includes:

1. **Expected Violation Count:** The ground truth for the test. A compliant scenario

expects 0 violations, a failure scenario typically expects 1.

2. **Mutation Actions:** A set of instructions to alter (mutate) the Golden Citizen.

Listing 3.4 provides an example of one mutation scenario. The full list of scenarios used in this framework can be found in Appendix D.

Listing 3.4: YAML description of a scenario

```yaml
- id: SCN_03
  description: "Base income is too high."
  expected_violation_count: 1
  actions:
    - type: patch_node
      turtle: ex:DadIncome :amount 26001.0 .
```

Crucially, mutations are designed to be atomic. Each scenario targets a *single fact* in the graph (e.g., changing a Literal value or a URI reference) to nudge the applicant from an "Eligible" state to a "Non-Eligible" state. This isolation allows the Validation Engine to pinpoint exactly which specific rule the LLM failed to generate correctly, if any.

For this work, all mutation scenarios were hand-crafted to reinforce confidence in the results. However, recent work has used LLMs to generate mutants, showing that mutation scores align with detection of real errors, extending classic mutation assumptions into LLM settings [53].

The Mutation Engine

For every iteration (run):

1. The system loads the Golden Citizen graph into memory.
2. It creates a deep copy of the graph to ensure test isolation.
3. Once per scenario, it applies the Patch Logic. The engine parses the Turtle snippets defined in the YAML actions (e.g., `ex:Income :amount 12,000.1`) and updates the graph triples accordingly. This allows for complex graph transformations, such as replacing nodes or updating relationships, without manual RDF manipulation.

The resulting Mutated Citizen Graph and the generated Shapes Graph (from Stage 4) are then passed into the afformentioned Validation Engine (section 3.2.4). The boolean outcome (`conforms`) and the number of violations with their associates messages are captured and logged to later be compared against the Expected Violation Count defined in each scenario.

3.3.2 Experimental Configurations

Recall the configuration tuple around which the experiment was designed:

```
(Document, Model, Prompting Strategy)
```

For this work we chose 2 documents, 2 models and 3 prompting strategies, for a total of 12 different experimental configurations. This combinatorial approach allows for the isolation of specific failure modes, distinguishing between errors caused by document complexity, model reasoning capacity, or prompting sufficiency. Below we analyze each component of the tuple and the configurations explored in the scope of this work.

Document Corpora (Use Cases)

This selection tests the pipeline's ability to generalize across different domains and logical structures. This discrepancy was deemed necessary, as research indicates that LLMs often experience a non-linear degradation in performance, as reasoning depth and contextual density increase [54].

Two public service documents were selected to represent two different levels of beurocratic complexity. The full documents are also available in the dissertation's GitHub repository [10].

Student Housing Allowance (High Complexity)

The more difficult and complex document of the two, selected as the stress test for the system. This document is characterized by:

- **Deep Graph Traversal:** Verification requires traversing multiple hops (Applicant → Parents → Properties → Location).
- **Recursive Arithmetic:** It involves dynamic income thresholds, calculated based on the count of dependent children (e.g., $Limit = Base + (N \times Bonus)$). Finding the count will require navigation of bi-directional relationships (Parent ↔ Child).
- **Referential Integrity Constraints:** Verification requires comparing the identity of URI nodes rather than literal values (e.g., validating that the `:UniversityCity` node is distinct from the `:FamilyResidenceCity` node).

Figure 3.5 shows a visualization of the Citizen Schema created for this use case. Its corresponding `ttl` file can be found in Appendix C.

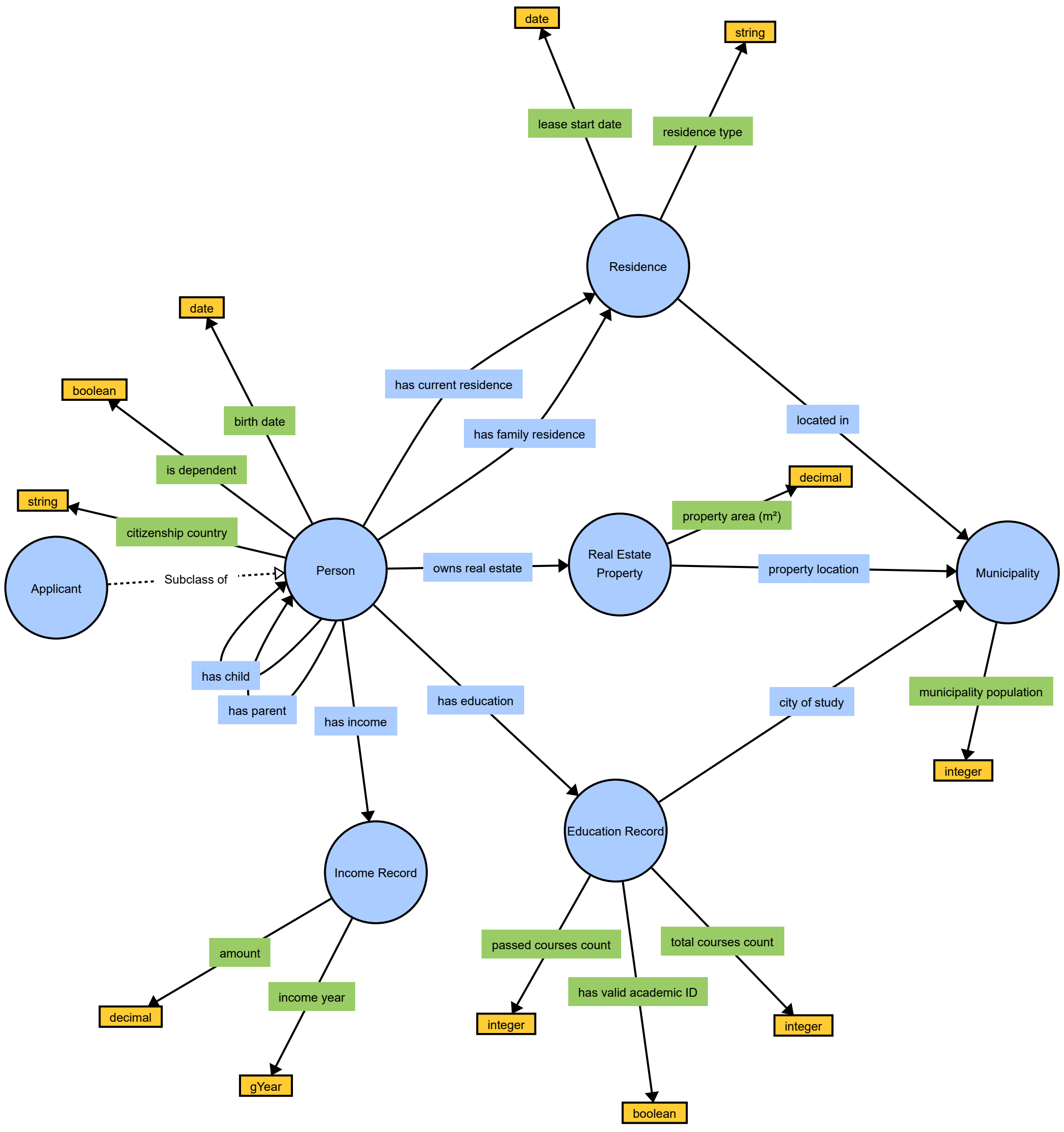


Figure 3.5: The Citizen Schema for the Student Housing document

Special Parental Leave Allowance (Intermediate Complexity)

Selected to evaluate standard administrative processing. This document focuses on:

- **Categorical Classification:** Eligibility relies on specific enumerated values (e.g., Employment Sector must be "Private" or "Public").
- **Temporal Logic:** Involves duration calculations (e.g., "1 year of continuous employment") rather than complex arithmetic aggregations.

Figure 3.6 shows a visualization of the Citizen Schema created for this use case. Its corresponding `ttl` file can be found in Appendix C.

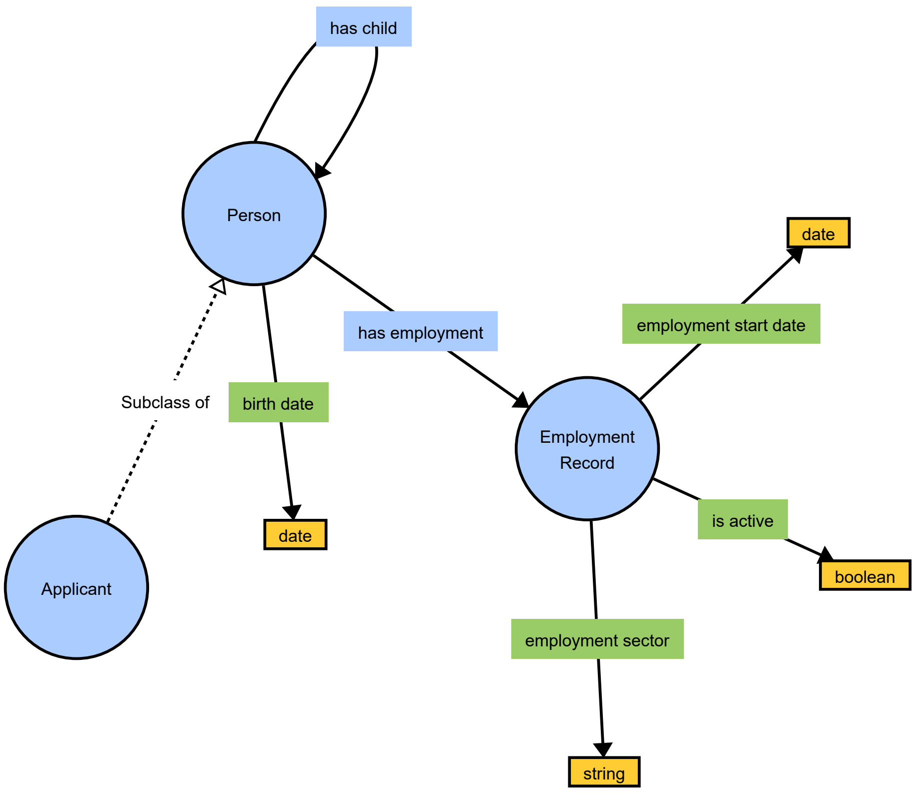


Figure 3.6: The Citizen Schema for the Parental Leave document

Large Language Models

The experiment uses two of the models in the Google Gemini 2.5 family, to evaluate the trade-off between reasoning capability and computational efficiency.

- **Gemini 2.5 Pro:** The high-parameter "reasoning" model. It is hypothesized to excel at complex SPARQL generation and abstracting vague requirements into formal logic, potentially at the cost of higher latency.
- **Gemini 2.5 Flash:** The lightweight, low-latency model. It was chosen to test the feasibility of a *high-throughput* pipeline. Another question is whether this smaller

model might be able to adhere to the strict SPARQL syntax requirements without the (theoretically) broader capabilities of its Pro sibling.

Prompting Strategies

Three distinct prompting strategies were implemented to evaluate the impact of *In-Context Learning* and *Self-Correction* on code quality. The full plain text of the prompts can be found in the dissertation's GitHub repository [10].

Default Strategy (Few-Shot with Guardrails)

This strategy represents the baseline optimized approach. The system prompt instructs the model to "act as an expert", a *role-playing* strategy that can potentially improve answer quality for some models and tasks [55]. The prompt also provides:

- **Proposed Strategy:** Explicit instructions to choose between, a simplification of the *Plan-and-Solve* strategy [56].
- **Syntactic Guardrails:** A set of negative constraints, derived from pilot testing frequent errors, as an attempt to improve alignment with syntactic expectations [57].
- **Few-Shot Examples:** Concrete examples demonstrating correct and desired outputs, a method that has shown to consistently improve results with harder extraction tasks, especially when examples clarify desired structure [58].

Zero-Shot Strategy (Ablation Study)

To quantify the value of the engineering effort put into the Default prompt, the Zero-Shot strategy removes all Few-Shot Examples. Thus, the model is given the instructions but no reference implementations. This tests the model's innate reasoning powers and knowledge of syntax versus its reliance on pattern matching from examples. This was also an attempt to see if diminishing returns in simple tasks and small context, observed in other studies [59], will affect our use cases.

Reflexion Strategy (Iterative Self-Correction)

This strategy implements a *Prompt Chaining* loop that has the following steps:

1. The model generates a draft response using the Default strategy.
2. The output is passed back to the model with a new persona: *"Senior Data Quality Assurance Auditor."* This agent is instructed to critique the quality of the draft with regards to criteria such as completeness, logical contradictions and syntactic validity.

3. If errors are found, the model rewrites the response based on its own critique.

Listing 3.5 shows an excerpt of the reflexion prompt.

Listing 3.5: Excerpt of the reflexion prompt

```
You are a Senior Data Quality Assurance Auditor.
Your task is to review the DRAFT RESPONSE provided below,
   which was generated based on the ORIGINAL INPUT DATA.
1. Analyze the Draft Response for:
   - Completeness: Did it miss any details from the input?
   - Logic: Are there contradictions or hallucinations?
   - Syntax: If the output is code (JSON/SPARQL/Turtle), is it
       syntactically valid?
...
```

This configuration evaluates the efficacy of self-correction mechanisms in code generation, specifically testing whether the computational overhead of iterative refinement yields a statistically significant reduction in errors [60].

3.3.3 Evaluation Metrics

To move beyond qualitative observation, the experimental framework was designed in such a way to capture a granular dataset for every execution cycle. This data collection strategy was designed to separate structural failures (code that does not compile) from logical failures (code that compiles but yields incorrect decisions), enabling a multi-dimensional analysis of the pipeline's performance.

Data Collection

Each run generates and saves a dataset that captures the complete state of the pipeline at the moment of execution, categorized into five dimensions:

- **Configuration Metadata:** Contextual fields containing the unique Run ID, a times-tamp, the document used as inout, the LLM employed and the active prompting strategy.
- **Artifact Fingerprinting:** An initial effort to track the stability and uniqueness of the LLM's output. The system computes the cryptographic hashes (MD5) of the generated graphs. This allows for the detection of potentially identical artifacts generated across different runs.
- **Syntactic Verification:** Before execution, the system first verifies if the generated

text is a valid RDF/Turtle graph (valid here means parsable by RDFLib). Then, it performs a deep compile check on every SPARQL query embedded in SHACL shapes, to ensure the syntax adheres to the SPARQL standard. Those errors, should they occur, are flagged differently to be distinguishable.

- **Validation Outcome Metrics:** The raw output of the validation engine is captured in detail. This includes the Actual Violation Count, the Expected Violation Count (derived from the scenario definition) and a serialized list of the specific Violated Shapes and their associated error messages. These fields enable the calculation of granular error metrics beyond simple binary accuracy.
- **Operational Diagnostics:** To monitor system health, metrics such as end-to-end Execution Time (latency) and Runtime Error Messages (e.g., Python exceptions) are logged. These fields are critical for quantifying the operational stability of components such as the external API.

At this point it is important to note that while the Information Model JSON generated in Stage 2 is structurally enforced by Pydantic, its logical accuracy is not measured independently. Instead, in the spirit of this study's end-to-end validation philosophy, any logical errors or semantic misalignments occurring during the information extraction phase are allowed to propagate downstream, where they are eventually surfaced by the mutation testing of the SHACL shapes.

Performance Indicators

The analysis of the persisted dataset focuses on two definitions of success.

Syntactic Validity

The first hurdle for any code-generating system is the production of executable syntax [61]. This metric quantifies the percentage of runs where the LLM produced a `ttl` file that could be successfully parsed by the RDFLib graph library *and* whose embedded SPARQL queries could be compiled without error. A run that fails this check is distinguished from runs that simply produce incorrect logic.

Functional Logic Accuracy

For runs that pass the syntax check, the focus shifts to logical fidelity. Indeed, it would hold no meaning to check for logic inside a code that does not even compile or in a structure that is unparseable. Accuracy is measured by comparing the final boolean eligibility decision made by the system, against the known truth hardcoded inside the mutation scenarios. We are basically treating the validation outcome as a binary classification task. For our context, if we theorize that a "Conformance" is the Positive class and "Violation" is the Negative class, standard machine learning metrics can be calculated.

3.4 Conclusion

This chapter has detailed the architectural and experimental foundations of the Neuro-Symbolic pipeline. By combining a schema-grounded generation process with a deterministic mutation testing framework, the system is designed to provide a quantifiable evaluation of LLM capabilities in the context of this task.

The following chapter presents the results of these experiments, analyzing the pipeline's performance across the afformentioned dimensions.

4 Results

This chapter presents the quantitative findings of the experiments detailed in Section 3.3. The analysis follows the main performance metrics described in section 3.3.3, assessing the system in terms of syntactic validity of generated code and functional logic accuracy. To further the analysis, another dimension is added, that of overall operational reliability (end-to-end feasibility in real-world context). A broad, subjective interpretation of the observed patterns and their implications for public administration systems is discussed in Chapter 5. The complete CSV file with all the logged data that produced the results presented in this chapter can be found in the dissertation's GitHub repository [10].

4.1 Experimental Dataset

The experiments consisted of a total of 170 end-to-end pipeline executions (runs). The distribution of these runs across the varying configurations is detailed in Table 4.1. Due to operational constraints that will be discussed in Section 6.1, the dataset is unbalanced, with the Flash model variant accounting for a larger proportion of the total runs.

Table 4.1: Distribution of Experimental Runs per Configuration

Document	**Model**	**Prompt Strategy**	**Runs (*N*)**
Parental Leave	gemini-2.5-flash	Default	20
		Reflexion	20
		ZeroShot	20
	gemini-2.5-pro	Default	10
		ZeroShot	10
Student Housing	gemini-2.5-flash	Default	20
		Reflexion	20
		ZeroShot	20
	gemini-2.5-pro	Default	10
		Reflexion	10
		ZeroShot	10

4.2 Syntactic Validity

As previously established, the first criterion for the pipeline's utility is the generation of syntactically valid code. A run is characterized as being *Syntactically Valid* only if the LLM produces a Turtle file that can be parsed by RDFLib *and* that contains SPARQL constraints that successfully compile without syntax errors.

4.2.1 Success Rate

Figure 4.1 illustrates the success rates across all configurations.

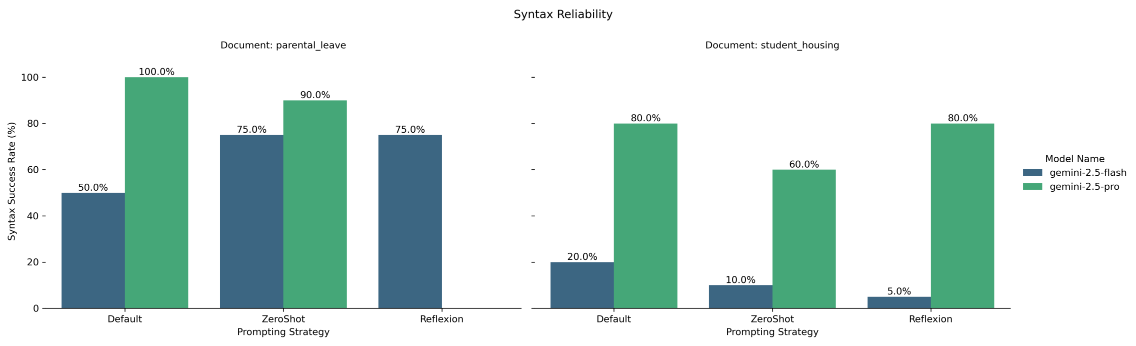


Figure 4.1: Syntactic Validity Rates by Configuration

Impact of Document Complexity

As was expected, the complexity of the source document strongly influenced the results.

In the case of the Parental Leave document (intermediate complexity), both models performed adequately. Even the smaller Flash model achieved a 75% validity rate using the Reflexion and ZeroShot strategies.

On the contrary, the Student Housing document (high complexity) acted as a stricter filter. Flash failed to produce valid code in the vast majority of attempts (36 out of 60 runs failed syntax checks), while Pro proved strong enough to handle the increased logical depth, even though it still suffered a 20% degradation compared to the simpler use case.

Impact of Model Class

Data reveals a disparity in syntactic capabilities between the two model variants.

The Pro model demonstrated high reliability, achieving a 100% success rate on the Parental Leave document and maintaining an 80% success rate on the complex Student Housing document (under Default prompting).

In contrast, the Flash model struggled significantly with maintaining syntactic precision.

While it achieved moderate success on the simpler Parental Leave document (ranging from 50% to 75%), its performance collapsed on the complex Student Housing document, with success rates dropping as low as 5% (Reflexion) to 20% (Default).

Impact of Prompting Strategy

The impact of prompting strategies varied by model architecture.

For Flash, the *Reflexion* strategy provided a significant boost on the simpler document (improving validity from 50% to 75%). However, this benefit vanished on the complex document, where Reflexion actually performed worse (5%) than the Default prompt (20%).

For Pro, the *Default* and *Reflexion* strategies performed identically (80% on Housing), while the *ZeroShot* strategy resulted in a notable drop in stability (falling to 60% on Housing).

4.2.2 Failure Mode Analysis

To better understand the reasons behind failure, the invalid runs were categorized by error type: *RDF Syntax Errors* (invalid Turtle file structure) and *SPARQL Syntax Errors* (malformed queries within valid Turtle). Figure 4.2 presents the error rates normalized by the total number of runs for each model-document pair.

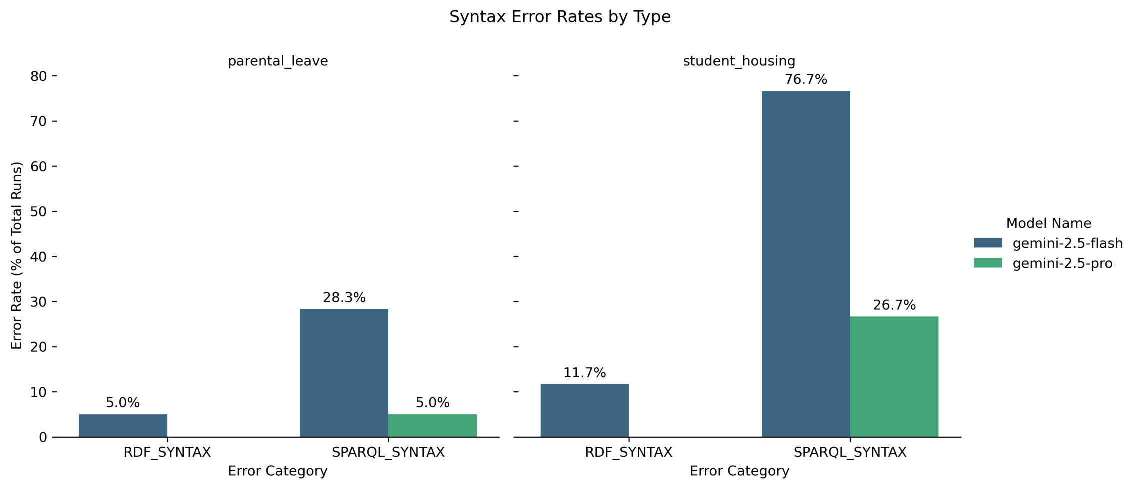


Figure 4.2: Distribution of Syntax Error Types by Model and Document

SPARQL Syntax Errors seem to be the dominant failure mode across all configurations.

Gemini 2.5 Flash exhibited a high frequency of SPARQL errors, particularly on the complex Student Housing document, where 76.7% of all runs failed due to query syntax. Notably, Flash also produced a non-negligible rate of RDF Syntax errors (5.0% on Parental Leave, 11.7% on Student Housing), indicating occasional failures in generating even fundamental structure.

Gemini 2.5 Pro demonstrated significantly higher syntax reliability. It produced zero RDF syntax errors across all 50 experiments using it. Its failures were exclusively confined to SPARQL syntax, with error rates of 5.0% on the simpler document and 26.7% on the complex document.

4.3 Logic Validity

As explained in the previous chapter, for the subset of runs that successfully produced syntactically valid code, the focus shifts to *Functional Logic Accuracy*. Runs that crashed during execution were similarily excluded from this analysis.

It was proven useful at this point to differentiate between partially correct runs, where some scenarios returned positive results and some negative, and a *Perfect Logic* run. A run is classified as having Perfect Logic if and only if the generated SHACL shapes correctly identify the expected number of violations for *every single scenario* in the test suite (both the baseline Golden Citizen and all mutated edge cases).

It is also important to note here that Logic Accuracy is evaluated as a *holistic performance metric*. This is important because a failure to correctly validate a citizen scenario may very well stem from errors at any stage of the Neuro-Symbolic pipeline. Examples include a missed precondition during the initial summarization (Stage 1), a malformed mapping in the Information Model (Stage 2), or an incorrect SHACL generation (Stage 4). Consequently, a Logic Failure indicates that the system, as a whole, failed to enforce the regulation, regardless of which specific component was the root cause.

4.3.1 Success Rate

Figure 4.3 illustrates the rate of flawless logical execution across configurations.

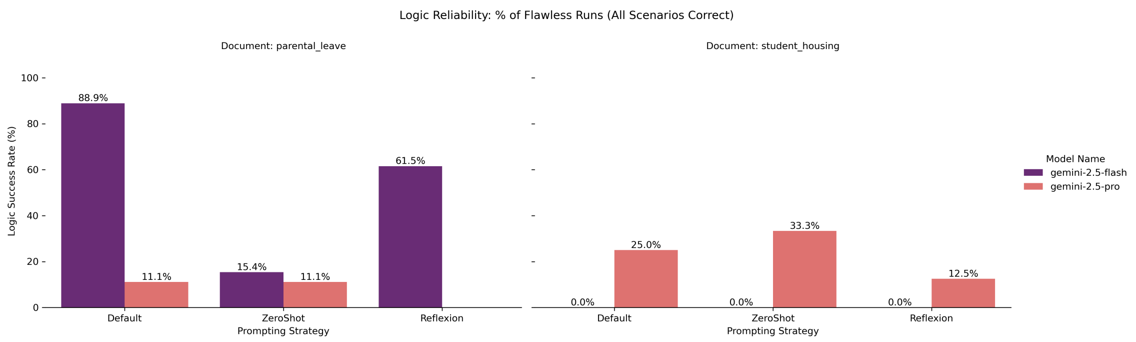


Figure 4.3: Logic Validity: Percentage of Perfect Logic Runs (All Scenarios Correct)

Impact of Document Complexity

Consistent with the syntax results, the complexity of the document was the primary determinant of success.

Parental Leave, having simpler logic, allowed for high performance, with the best-performing configuration (Flash with Default prompts) achieving an 88.9% perfect run rate.

Student Housing and its complex logic requirements caused a near-total collapse in functional accuracy. Across all models and prompts, the highest achieved success rate was only 33.3%, with many configurations failing to produce even a single logically correct run.

Impact of Model Class

The performance relationship between models *inverted* depending on the task.

On the simple document, Flash significantly outperformed Pro, achieving 88.9% accuracy (Default) compared to Pro's 11.1%. However, on the complex document, Flash failed completely, with a 0.0% success rate across all 60 attempts.

While Pro underperformed on the simple document, it was the only model capable of solving the complex Student Housing logic, achieving success rates between 12.5% and 33.3%.

Impact of Prompting Strategy

Removing examples (ZeroShot) caused a sharp drop in accuracy from 88.9% to 15.4% for Flash on the simple document. Conversely, for Pro on the complex document, ZeroShot unexpectedly yielded the highest accuracy (33.3%).

The self-correction strategy (Reflexion) did not yield consistent improvements. For Flash, it reduced accuracy from 88.9% to 61.5% on the simple document. For Pro, it performed roughly equivalent to the Default strategy.

4.3.2 The Syntax-Logic Gap

A comparison between the syntax validity rates (Figure 4.1) and logic accuracy rates (Figure 4.3) reveals a characteristic degradation of performance as the evaluation becomes harder. This becomes especially apparent if we isolate the (more complex) Student Housing use case. While the Pro model generated valid syntax in a satisfyingly high $\approx$80% of runs, only $\approx$25% of those valid runs contained correct logic. The Flash model, on the other hand, struggled at both levels, with low syntax validity (20%) and zero functional correctness (0%). Further analysis of this phenomenon will be discussed in Chapter 5.

4.4 Overall Pipeline Reliability

Beyond specific syntax and logic metrics, this section evaluates the system's viability as an end-to-end automated service. The analysis considers two perspectives: the operational stability of the pipeline and its reliability inside the context of public service recommendations, the borader context this work.

4.4.1 Pipeline Feasibility and Attrition

Figure 4.4 presents the distribution of final outcomes for all 170 experimental runs. This analysis categorizes every attempt into a single mutually exclusive outcome, revealing the attrition rate of the system.

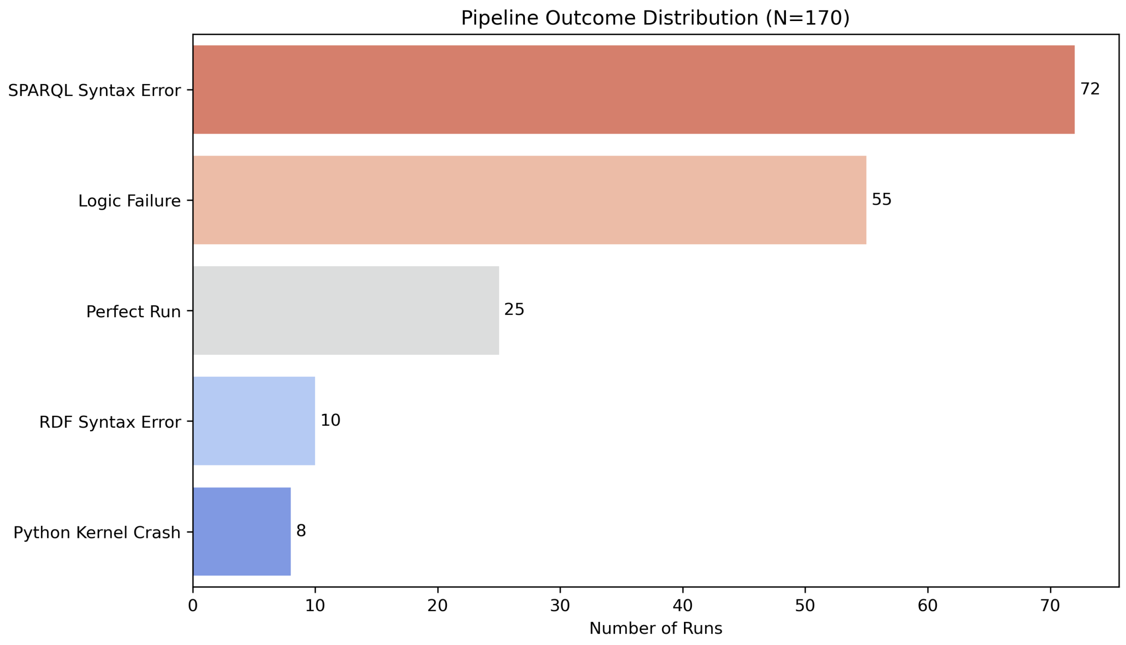


Figure 4.4: Distribution of Final Pipeline Outcomes

The data indicates a high attrition rate:

- **Syntax Failures:** The majority of runs failed early. SPARQL Syntax Errors accounted for the largest share of failures (N=72), followed by RDF Syntax Errors (N=10).
- **Logic Failures:** Of the runs that compiled, a significant portion (N=55) produced code that executed but failed to correctly validate all test scenarios.

- **Success:** Only 25 runs (14.7% of the total) achieved the status of a Perfect Run, generating both valid syntax and flawless logic across all edge cases.
- **System Stability:** Operational crashes (Python/API errors) were rare (N=8), indicating that the underlying infrastructure, retry mechanisms and exception handling were largely sufficient.

4.4.2 In-context (Recommender System) reliability

To evaluate the system's utility as a public service recommender, the validation outcomes were aggregated into a Confusion Matrix (Figure 4.5). In this context, the classes are defined based on the goal of recommending eligible services:

- **Positive Class (Recommendation):** The system validates the citizen as eligible for the public service and makes the recommendation.
- **Negative Class (Rejection):** The system flags at least one violation of a precondition and does not make the recommendation.

To interpret the confusion matrix in this specific context, the standard machine learning classifications were mapped to domain-specific service outcomes, as defined in Table 4.2.

Table 4.2: Definition of Classification Outcomes in the Recommender Context

	System: "Violation" (No Recommendation) *Triggered Violations > 0*	**System: "Conforms"** (Recommendation) *Triggered Violations = 0*
Citizen is Ineligible *Expected Violations > 0*	**True Negative (TN)** *Correct Rejection* (System works)	**False Positive (FP)** *Bad Recommendation* (Trust Risk)
Citizen is Eligible *Expected Violations = 0*	**False Negative (FN)** *Missed Opportunity* (Service Failure)	**True Positive (TP)** *Correct Recommendation* (System works)

The confusion matrix is then created based on this terminology. Naturally, syntactically invalid runs or runs that crashed sue to a system error, did not participate in the confusion matrix counts since no logic was produced by them.

Note that in this context, one data point corresponds to one specific scenario, not one full run. This fits the metric better, since one scenario corresponds to one citizen instance.

The matrix reveals the system's risk profile:

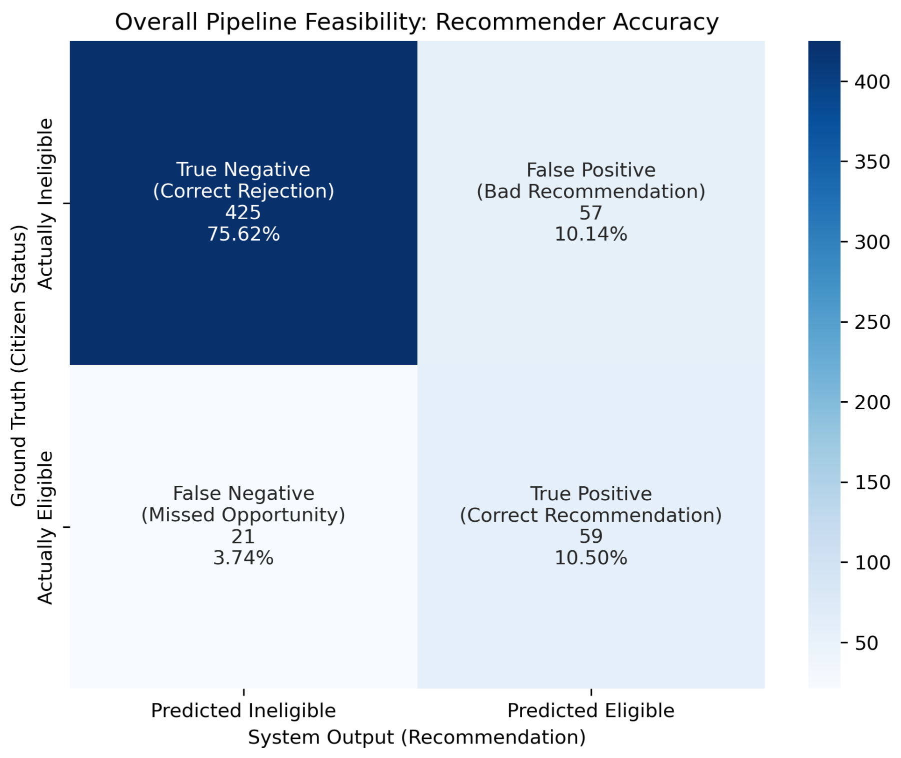


Figure 4.5: Confusion Matrix of Eligibility Recommendations

- **True Positives (10.5%):** In 59 cases, the system correctly identified and recommended the service to an eligible citizen ("Correct Recommendation"). This confirms the system's ability to successfully validate legitimate claims when the generated logic is sound.
- **False Positives (10.1%):** In 57 cases, the system erroneously recommended the service to an ineligible citizen ("Bad Recommendation"). This represents a "Trust Risk," where users might be guided to apply for benefits they cannot receive.
- **True Negatives (75.6%):** The system correctly rejected ineligible applicants in the majority of cases.
- **False Negatives (3.7%):** In 21 cases, the system incorrectly rejected an eligible applicant ("Missed Opportunity"). While this number is low, it represents a "Service Failure," denying access to entitled benefits.

5 Discussion

This chapter uses the quantitative results presented in Chapter 4 and attempts to provide some explanations and subjective opinions about them. By analyzing the patterns of failure, ranging from syntactic hallucinations to logical paradoxes, this discussion aims to characterize the main limitations of the proposed Neuro-Symbolic architecture. The analysis moves beyond simple performance metrics to address the challenges of semantic fidelity, algorithmic determinism and the operational sovereignty required for deployment in public administration.

5.1 Cognitive Dissonance in Code Generation

One of the arguably most disruptive patterns observed throughout the results of the experiments, was the documented significant disconnect between the Large Language Model's ability to generate valid *syntax* and its ability to construct valid *logic*. This phenomenon, termed here *Cognitive Dissonance*, confirms a known limitation of Transformer-based models, especially when they are applied to formal reasoning tasks: they operate as approximate pattern matchers [62] in a domain that requires exact symbolic execution.

5.1.1 The Illusion of Fluency

The results from the Student Housing use case serve as the primary evidence for this phenomenon. The Gemini 2.5 Pro model achieved an 80% syntactic validity rate, successfully producing well-formed Turtle files with structurally correct SPARQL queries. To a human reviewer, this code appeared indistinguishable from expert-written logic. However, the functional accuracy of this valid code was only ≈25%.

This difference reveals that the model has successfully memorized the *grammar* of SHACL (e.g., correct brackets, prefixes, keywords) but often failed to grasp the *semantics* of the query it was constructing. As current research has suggested (but not yet proved), the model *knows* how to write a query, but it does not *understand* well enough what the query actually calculates [63].

5.1.2 Structural Logic Failures

The model's inability to properly understand graph topology led to recurring failures in the generated SPARQL constraints. Below we analyze some of the most prominent mistakes discovered upon human inspection of the runs flagged by the system as not perfect.

The Double-Counting Trap

In scenarios involving family units (for example, two parents with two children), the model consistently failed to apply set-theoretic distinctness. By generating queries that traversed from `:Parent` to `:Child` without the `COUNT(DISTINCT ?child)` modifier, the model inadvertently created a multiplicative join. Since both parents are linked with the same two children, the query counted each child twice (once per parent path). The result was an artificial inflation of the child count. This subsequently distorted calculations, leading to false validations where ineligible families were approved due to miscalculated thresholds.

Figure 5.1 illustrates this. The traversal from Applicant to Mother (Blue) and Father (Red) creates two parallel branches. Because both branches terminate at the same sibling nodes, the resulting count is the sum of the branches (4) rather than the cardinality of the unique entities (2).

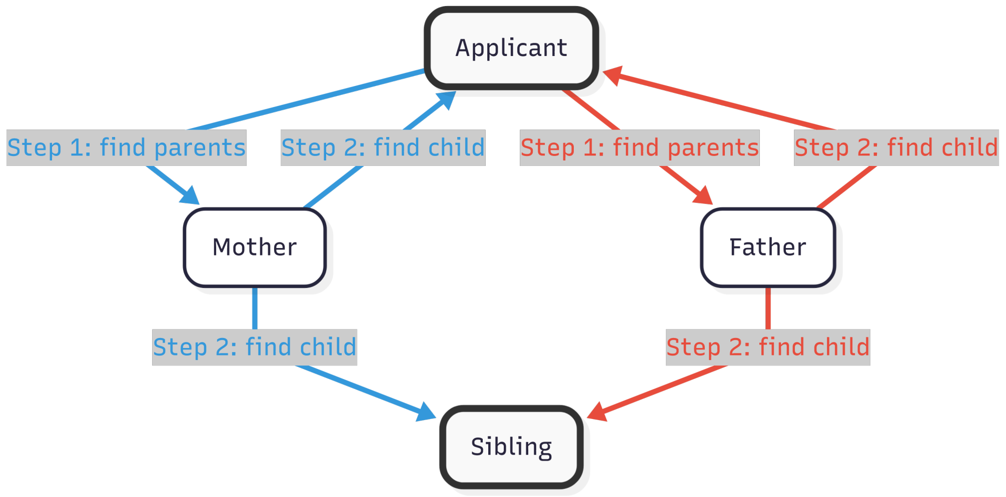


Figure 5.1: Illustration of the Double Counting trap

The Cartesian Product Trap

A very similar "bug" with the previous one appeared when aggregating income. The model was observed to frequently join Income patterns and Child patterns in a single `WHERE` clause, without sub-query separation. This caused the SPARQL engine to generate a Cartesian product of the two datasets, effectively multiplying every income record by every child record. This resulted in erroneous rejections where families were flagged with violations due to massive over-estimations of total family incomes. A partial example of a wrong query generated to trigger the Catresian Product Trap can be seen in Listing 5.1

Listing 5.1: Generated query that contains the Cartesian Product Trap

```
SELECT ?this (SUM(?income) as ?totalIncome)
(COUNT(DISTINCT ?child) as ?childCount)
WHERE {
    OPTIONAL {
        { ?this :hasIncome ?inc . ?inc :amount ?income . }
        UNION
        { ?this :hasParent ?p .
        ?p :hasIncome ?p_inc .
        ?p_inc :amount ?income . }
    }
    OPTIONAL {
        ?this :hasParent ?parent .
        ?parent :hasChild ?child .
        ?child :isDependent true .
    }
}
GROUP BY ?this
```

Recursive Loops and Infinite Regression

A more catastrophic failure mode was observed in the Flash model's handling of bidirectional relationships. The Student Housing Citizen Schema defines inverse relationships (e.g., a `:Parent` has a `:Child`, and that `:Child` has a `:Parent`). In several runs, the model generated SPARQL property paths that traversed these links cyclically (e.g., `:hasParent :hasChild :hasParent ...`) without a terminating condition. This created infinite recursion loops during execution, causing the PySHACL validation engine to crash entirely (logged as "Python Kernel Crash" in Section 4.4.1). This might suggest that the model treats graph traversal as a linguistic association task (Parents are related to Children) rather than a directed graph walk, failing to anticipate the computational consequences of such cycles.

5.2 Syntax Hallucination and Language Bleed

While the Pro variant's failures were predominantly logical, the Flash model struggled significantly to maintain the boundaries of the language itself. The experimental data reveals a phenomenon termed *Language Bleed*, where the model, optimized for high-throughput generalized text generation (perhaps not a coincidence), conflated the syntax of similar languages.

5.2.1 SQL Contamination

The most frequent syntax error was the appearance of illegitimate keywords, such as `FILTER NOT (...)`. This construct is reminiscent of valid SQL syntax (`WHERE NOT`) but it is invalid in SPARQL (which for example would require `FILTER (! ...)` or `FILTER NOT EXISTS`). This hints to what other research has previously suggested [63]: the model's training data contains significantly more SQL examples than SPARQL, leading it to default to the more dominant syntax when the probability distribution for the next token is ambiguous.

5.2.2 Token-Level Hallucinations

The model also exhibited errors that can be attributed to its nature as a token predictor rather than a parser [64]. It frequently attempted to use dot notation (e.g., `?s.hasChild`) or complex property path slashes (e.g., `?s /:hasChild`) in contexts where explicit triple patterns were required. While property paths exist in SPARQL 1.1, the specific context in which such syntax was generated often mimicked Object-Oriented programming accessors rather than valid RDF graph traversal.

5.2.3 Namespace Invention

A distinct class of errors involved the hallucination of ontology definitions. Despite being provided with a fixed set of prefixes, the model occasionally invented new namespaces (e.g., using the deprecated 2007 SHACL draft URI or inventing an `ex:Citizen` ontology). This behavior was the primary cause of all recorded RDF Syntax Errors (as distinct from SPARQL errors), suggesting that smaller models might struggle more to adhere to strict *Negative Constraints* (i.e., "Do not use any other prefix").

5.3 Abstraction vs. Fidelity Trade-off

An intuitive thought concerning Large Language Model size would be that model capability (size, reasoning power) correlates linearly with performance across all tasks. However, it is increasingly recognized in LLM research that model capabilities are multifactor, task-dependent, often nonlinear and sometimes involve trade-offs [65][66]. The experimental results from the Parental Leave use case confirm this critical inversion of the basic intuition.

5.3.1 The Smart Model Trap

The extraction of eligibility preconditions requires extreme fidelity to the source text. Legal constraints often rely on specific enumerations that define the scope of the law. Such a case was presented when models came accross the following precondition in the Parental Leave use case: *"The applicant must be employed under a dependent employment regime, in the Private or Public sector."*

In this task, the Gemini 2.5 Flash model (theoretically less capable model) significantly outperformed the Gemini 2.5 Pro model. Flash, lacking the capacity for deep abstraction, tended to "copy-paste" the precondition literally. When presented with the employment requirement, it preserved the disjunction ("Private OR Public").

Pro, optimized for high-level reasoning and helpfulness, attempted to *summarize* the requirement. It interpreted "Private or Public" as a generic concept ("Employed"), effectively deleting the exclusion of other sectors (e.g., Freelancers). This finding suggests that for compliance tasks, "Smart" models may be fundamentally misaligned with the goal. Their training bias towards summarization and abstraction leads to *semantic drift*, where the gist of the rule is preserved but the legal boundary is lost.

5.3.2 Superiority of Intermediate Representations

This trade-off extends to the architectural design of the pipeline itself. A contrast was observed between the error rates of the neural components and the symbolic components.

Notably, zero syntax errors were recorded in the generation of the Citizen-Service Graph (Stage 3). This is a direct result of the choice to use Pydantic for structural gatekeeping in the preceding stage. By resolving any and all ambiguities at the JSON level, the pipeline reinforces intelf and ensures that the Python-based serialization logic receives clean data.

The perfect stability of Stage 3, contrasted with the high failure rate of the LLM-generated SHACL shapes (Stage 4), empirically validates the architectural decision to offload structural tasks to deterministic code wherever possible.

This leads back to a key design principle for Neuro-Symbolic systems [67]: LLMs are necessary for interpretation (Extraction), but they are suboptimal for serialization (Code Generation). A robust pipeline must treat the LLM as a *translator* of natural language, but never as an *architect* of the final system structure [14].

5.3.3 Efficiency of the SHACL-SPARQL Hybrid

The results provide empirical evidence for the validity of the Dual-Strategy Protocol introduced in Section 3.2.3. It was observed that while Flash experienced a near-total collapse when generating complex SPARQL queries (76.7% error rate in Student Housing), it maintained much higher stability in the Parental Leave case where more constraints could be handled via SHACL-Core.

This confirms that a strategy of defaulting to simple SHACL shapes for the majority of checks, creates a more resilient sytem that is less susceptible to the Language Bleed and Token-Level Hallucinations identified in Section 5.2. These findings could also be indicative of yet another trade-off: the more a regulatory framework can be expressed through standard shapes rather than custom SPARQL queries, the higher the overall system reliability, as it minimizes area where the LLM's non-deterministic nature can introduce failure.

5.4 Complexity Ceiling

The divergence in performance between the Student Housing and Parental Leave use cases identifies a potential complexity ceiling for current LLM-based logic generation. While the pipeline demonstrated high viability for administrative tasks involving categorical classification (Parental Leave), it experienced a near-total collapse when tasked with recursive arithmetic (Student Housing).

This failure mode correlates strongly with the Dependency Depth of the required logic:

- **Shallow Dependencies (Success):** Constraints that rely on single-node checks (e.g., *Nationality* depends only on *Applicant*) or flat Boolean logic (e.g., *isValid* is True OR False) were handled with high accuracy (88.9% logic success).
- **Deep Dependencies (Failure):** Constraints that require multi-hop traversal (e.g., *Applicant* $\rightarrow$ *Parent* $\rightarrow$ *Residence*) or recursive variable modification (e.g., *Income Limit* changes based on *Dependent Child Count*) consistently triggered the Cartesian

Product bug or infinite recursion errors.

This suggests that while the tested LLMs can successfully act like semantic parsers for straightforward bureaucracy, they lack the internal working memory required to maintain the state of multi-variable equations throughout the code generation process, consistent with observed computational limits of transformer-based models [68].

5.5 Contribution of Prompt Engineering

The experimental results challenge the prevailing narrative that improving prompting is a universal solution to model limitations [69]. Instead, the data reveals complex trade-offs where techniques that improve syntactic stability may inadvertently degrade logical reasoning.

5.5.1 The Copy-Paste Bias

The Default (Few-Shot) strategy proved essential for stabilizing syntax in the Pro model, boosting syntactic validity from 60% to 80% on the complex document. However, this stability came at a cost to logical accuracy. Zero-Shot strategy, despite producing broken code more often, achieved the highest logical accuracy (33.3%) when it *did* compile.

This suggests a *Copy-Paste Bias*: when provided with examples, the model seems to sometimes overfit to the logic of the template, attempting to force the new problem into the old structure. Without examples (Zero-Shot), the model is forced to reason from first principles, leading to messier syntax but potentially more original (and correct) logical derivations. There are behavious also observed under different domains [70], but efforts are being proposed to mitigate them [71].

5.5.2 Failure to Self-Correct

The Reflexion strategy failed to deliver the expected performance gains. For the less capable Flash model on the complex document, Reflexion actually *degraded* performance, dropping syntax validity from 20% to 5%. This indicates that a model incapable of solving a problem in the first pass might very well be equally incapable of critiquing its own solution. In fact, asking a confused model to double-check its own work might merel introduce a second opportunity for hallucination, compounding errors rather than resolving them.

5.5.3 The Engineering Ceiling

These findings imply an *Engineering Ceiling*: one cannot prompt their way out of a fundamental reasoning deficit. While Prompt Engineering can guide a capable model (Pro) to follow syntactic rules, it cannot bestow reasoning capabilities upon a smaller model (Flash) that physically lacks them. For high-stakes logic generation, architectural scale remains the dominant variable.

5.6 Feasibility and Sovereignty

The final dimensions of analysis concern the operational viability of deploying such a system within a public administration context. The experimental campaign revealed critical vulnerabilities in the reliance on proprietary Model-as-a-Service (MaaS) infrastructure.

5.6.1 The Semantic Drift of "Eligibility"

Feasibility is first challenged at the point of ingestion. The extraction and summarization phase (Stage 1) demonstrated a persistent ambiguity in defining the fundamental term of "Eligibility". Despite explicit prompt instructions to ignore administrative steps, the models frequently conflated procedural requirements (e.g., "Log in to TaxisNet") with substantive facts (e.g., "Be employed"). This semantic drift creates a system that validates paperwork rather than reality. While acceptable for a pilot, a production system would require a stricter, legally-grounded ontology of evidence vs. conditions, to prevent the digitization of bureaucracy from becoming merely the automation of red tape.

5.6.2 Replication Crisis

The most severe threat to feasibility, however, emerged from the infrastructure itself. Throughout the implementation phase, unannounced updates to the underlying models and shifting rate-limiting policies resulted in regressions in throughput and occasionally also logic. More details about this hurdle will be discussed in Section 6.1. This lack of control over the versioned state of the model created a significant barrier to reproducibility as well. This event serves as a potent case study for Digital Sovereignty, an idea analyzed further in Section 6.1.4

5.7 Risk Asymmetry in Public Administration

The evaluation of the system's feasibility as a Recommender System (Section 4.4.2) gives an important revelation regarding the deployment readiness of these Neuro-Symbolic agents. While standard ML models optimise for balanced F1 scores, the cost of error imposed by the operational context of public administration is *asymmetrical*.

The experimental data showed a 10.1% False Positive Rate, which corresponds to instances where the system erroneously recommended a service to an ineligible citizen. In a commercial context (e.g., movie recommendations), such errors are trivial. However, in digital governance, a False Positive actively generates bureaucratic friction. Specifically, it compels a citizen to gather documents and submit an application that is destined to fail. This wastes public resources and also erodes trust in the automated system. Conversely, the 3.7% False Negative Rate (Missed Opportunities), while statistically undesirable, represents a safer failure mode that maintains the status quo.

Consequently, the current pipeline's bias towards *over-recommending* presents a significant barrier to unsupervised deployment in a real-world administrative setting.

These findings might characterize the current performance ceiling, but they also provide the technical requirements for the next generation of research and developement of this class of systems, as detailed in the following chapter.

6 Limitations & Future Work

The development and evaluation of the proposed Neuro-Symbolic pipeline has highlighted several boundaries that currently exist between generative AI capabilities and the requirements of administrative automation. This chapter synthesizes both the technical and the conceptual constraints encountered during the pilot study with suggestions for future development. The goal of this last part is to define the necessary trajectory for moving from pilot-scale experimentation toward production-grade digital governance.

6.1 Infrastructure Dependencies & Digital Sovereignty

The experiments conducted in this study were significantly shaped by their reliance on proprietary, cloud-based Model-as-a-Service (MaaS) infrastructure. While the use of the Google Gemini ecosystem provided reasoning capabilities adequate for a pilot-scale prototype, it introduced a significant *infrastructural dependency* that reveals an important limitation for public sector applications.

6.1.1 Model and Resource Constraints

A primary limitation of this work is the narrow diversity of the models used. Due to the reliance on free-tier access, the study was restricted to just two models within a single vendor's ecosystem. This prevents a broader comparative analysis of how different architectural families (such as GPT or Llama) handle the specific nuances of administrative logic.

Future research must expand to include a statistically significant variety of LLMs. This could be a way to determine if any of the inadequacies, such as the semantic drift observed in this study, are universal traits of generative AI or a specific characteristic of the utilized models.

6.1.2 Model Specialization

A sigificant limitation of relying on models from a foreign, usually general-purpose ecosystem (like Gemini or GPT), is their lack of alignment with specific legislative drafting styles

[72]. Future research should possibly prioritize *Domain-Specific Fine-Tuning*. By taking an open-weights local model and training it on a curated dataset of adminstrative texts, together with their formal corresponding logic, we can create a specialized LLM. This transition from few-shot (or zero-shot) prompting to a fine-tuned model is expected, according to the literature [73], to significantly reduce the occurrence of phenomena observed in this dissertation, such as the Language Bleed, and more importantly, to improve the model's understanding of concepts like administrative hierarchy.

6.1.3 Operational Fragility and the Replication Crisis

The reliance on external APIs introduced another roadblock. During the experimental time window, that lasted several days to months, unannounced changes to the Google Gemini API decisively hindered experimentation. Rate limits were lowered by as much as 90%, causing a catastrophic degradation in pipeline throughput, which in turn led to the severe imbalance of the experimental configuration dataset. Even worse, the models that were being used in the experiments were suddenly discontinued from the API, putting a de-facto swift end to the experiment. In a digital governance context, such operational instability is unacceptable [74]. A legal pipeline must be idempotent and resilient to the business decisions of third-party "Big Tech" providers [75].

6.1.4 A Roadmap Toward Sovereign AI

To address the dependencies discussed previously, the future roadmap for this research prioritizes the transition to *Digital Sovereignty*. By migrating the Neuro-Symbolic pipeline from cloud-based APIs to local, open-weights models hosted on private government infrastructure, several strategic goals can be achieved:

- **Data Privacy:** Full compliance with GDPR by ensuring citizen data never leaves a secure environmen.
- **Operational Stability:** Eliminating the risk of API-related model deprecations or downtime.
- **Fine-tuning:** The ability to perform Domain-Specific Fine-tuning on actual legislative corpuses, which is often restricted or cost-prohibitive on proprietary cloud platforms.

The move toward local infrastructure is definitely a technical upgrade but more importantly it is a stepping stone for the No-Stop Government vision, ensuring that the code remains under the exclusive control of the state.

6.2 Scalability

While the pilot study successfully demonstrated the technical feasibility of the Neuro-Symbolic pipeline, the scope of the experimental campaign was intentionally narrowed to prioritize analytical depth over statistical breadth. A clear path exists for scaling the framework in future research.

6.2.1 Document Corpus and Prompting Strategies

The primary limitation regarding generalizability is the limited sample size of the included document corpus. By focusing the evaluation on two administrative document sets, the study provided a high-resolution view of the pipeline's behavior in a complex scenario. However, to validate the framework's usefulness across the entire spectrum of public administration, future work must scale the experimental setup to include hundreds of heterogeneous documents across different domains (e.g., healthcare, transportation, business licensing etc.).

Furthermore, the prompt engineering strategies employed were primarily focused on instruction following. Future research should investigate more sophisticated strategies, such as deeper Chain-of-Thought (CoT) prompting or more complex and specific *Self-Refine* loops, to determine if these techniques can improve the success rates identified in the Pilot Study.

6.2.2 From Pilot to Interoperability

A fundamental strength of this dissertation's methodology is the decision to ground the symbolic components in established European standards, turning the synthesized graphs from isolated artifacts to being inherently interoperable with the broader European semantic ecosystem. Currently, the pipeline utilizes a subset of these vocabularies. The next phase of development should involve:

- **Full Vocabulary Integration:** Expanding to cover the entirety of the *CCCEV* and *CPSV-AP* classes, allowing for the representation of the whole spectrum of the Public Service and its preconditions.
- **Cross-Border Interoperability:** Testing the pipeline on legislative texts from different EU member states to evaluate if the Neuro-Symbolic pipeline can handle multi-lingual semantics.
- **Automated Ontology Mapping:** Implementing a dynamic mapping stage where the

LLM can identify and utilize newly added classes from the official EU vocabularies without manual updates.

By leaning heavily on these standards, the framework serves as a modular component of the Once-Only Principle, ensuring that once a rule is synthesized and validated, it can be shared and understood by any administrative system across the European Union.

6.3 Methodological Refinement

Beyond scaling the quantity of data, future iterations of this research must refine the qualitative evaluation of the pipeline's artifacts. The current study relied on functional execution success, but if we are to gain a deeper understanding of the failure modes, we require more granular metrics.

6.3.1 Root Cause Analysis

While our automated testing logs provide a baseline for failure rates, they cannot fully explain the linguistic nor the logical nuances that lead to a hallucinated constraint. A necessary next step is the introduction of a human expert at the role of auditor. According to this idea, domain experts (possibly legal and administrative professionals) will review the specific cases where the pipeline failed either in syntax or logic, to categorize the errors. This will provide valuable insights such as if failures occur primarily during the precondition extraction (interpreting the text) or text-to-logic (generating the code). Understanding this divide is very important for pinpointing which stage of the pipeline requires more intensive engineering.

6.3.2 Artifact Evaluation

Currently, the evaluation of the pipeline artifacts is primarily functional. Future work should implement a method to evaluate them in depth.

- **Topology vs. Content:** Instead of comparing node names or descriptions, the analyzer will use metrics such as Graph Edit Distance (GED) to evaluate how different the LLM-generated structures are between different iterations (runs) using the same configuration.
- **Logic Similarity Metrics:** Future frameworks should explore similarity metrics for SPARQL as well, such as Tree Edit Distance on Abstract Syntax Trees (AST), to assess how logically different the generated queries are.

- **Natural Language Similarity:** The stability of natural language summaries can be assessed through *embeddings* to test for consistency.
- **Artifact Benchmarking:** Another approach to facilitate the same goal is for human experts to draft perfect reference graphs, summaries and SPARQL queries for a control set of documents. By comparing against these gold standards, we can measure semantic and structural drifts with mathematical precision, identifying patterns where the LLM consistently over-simplifies or over-complicates.

It is worth noting that since this work employs a meticulous persistence approach, keeping all artifacts on file, many of these ideas can be implemented already using the stored artifacts without needing to re-run experiments.

6.4 Trust-Centric Optimization

As discussed previously, in the context of public service eligibility, the cost of a False Positive (incorrectly recommending a benefit) is often higher than the cost of a False Negative (missing an opportunity for plausible eligibility). Future iterations of the pipeline must be refined to prioritize *Precision* (Trustworthiness) over *Recall* (Coverage).

The prompts should be engineered and the intermediate representations should be calibrated to be *conservative by default*. If the model encounters an ambiguous clause, t should be instructed to be strict or even possibly to flag the constraint for human review rather than attempting a probabilistic best guess.

Through this lens, the JSON Information Model can be expanded to include fields like some type of *Confidence Attribute* where the model self-reports its own certainty for each extracted precondition, allowing the symbolic stage to automatically reject any logic that falls below a specific trust threshold.

Bibliography

[1] P. Dunleavy, H. Margetts, S. Bastow, and J. Tinkler, "New public management is dead. Long live digital-era governance," *Journal of Public Administration Research and Theory*, vol. 16, pp. 467–494, 2005. DOI: `10.1093/jopart/mui057`

[2] H. Scholta, W. Mertens, M. Kowalkiewicz, and J. Becker, "From one-stop shop to no-stop shop: An e-government stage model," *Government Information Quarterly*, vol. 36, no. 1, pp. 11–26, 2019. DOI: `10.1016/j.giq.2018.11.010`

[3] L. Manny, M. Duygan, M. Fischer, and J. Rieckermann, "Barriers to the digital transformation of infrastructure sectors," *Policy Sciences*, vol. 54, pp. 943–983, 2021. DOI: `10.1007/s11077-021-09438-y`

[4] C. Adam, C. Knill, and X. Fernandez-i-Marín, "Rule growth and government effectiveness: Why it takes the capacity to learn and coordinate to constrain rule growth," *Policy Sciences*, 2017. DOI: `10.1007/s11077-016-9265-x`

[5] M. Döring and J. Madsen, "Mitigating psychological costs - the role of citizens' administrative literacy and social capital," *Public Administration Review*, 2022. DOI: `10.1111/puar.13472`

[6] R. M. Bakker, A. J. Schoevers, R. A. N. van Drie, M. Schraagen, and M. H. T. de Boer, "Semantic role extraction in law texts: A comparative analysis of language models for legal information extraction," *Artificial Intelligence and Law*, pp. 1–35, 2025. DOI: `10.1007/s10506-025-09437-x`

[7] M. Siino, M. Falco, D. Croce, and P. Rosso, "Exploring LLMs applications in law: A literature review on current legal NLP approaches," *IEEE Access*, vol. 13, pp. 18 253–18 276, 2025. DOI: `10.1109/access.2025.3533217`

[8] M. A. Loutsaris and Y. Charalabidis, "Legal informatics from the aspect of interoperability: A review of systems, tools and ontologies," in *Proceedings of the 13th International Conference on Theory and Practice of Electronic Governance*, 2020, pp. 731–737. DOI: `10.1145/3428502.3428611`

[9] I. Konstantinidis, I. Magnisalis, and V. Peristeras, “A framework for a public service recommender system based on neuro-symbolic AI,” *Applied Sciences (Switzerland)*, vol. 15, no. 20, 2025. DOI: 10.3390/app152011235

[10] N. Laoutaris, “Exploring neuro-symbolic pipelines for structured knowledge extraction,” GitHub Repository, 2026. Accessed: Jan. 24, 2026. [Online]. Available: https://github.com/n-laoutaris/neuro-symbolic-dissertation

[11] M. J. Page et al., “The prisma 2020 statement: An updated guideline for reporting systematic reviews,” *BMJ*, vol. 372, 2021. DOI: 10.1136/bmj.n71

[12] I. Oranekwu, L. Elluri, R. Yus, and A. Kotal, “Scalable automation for IoT cybersecurity compliance: Ontology-driven reasoning for real-time assessment,” *Computers & Security*, vol. 161, 2026. DOI: 10.1016/j.cose.2025.104711

[13] A. Z. Spyropoulos and V. D. Tsiantos, “Interoperable semantic systems in public administration: AI-driven data mining from law-enforcement reports,” *Computers*, vol. 14, 2025. DOI: 10.3390/computers14090376

[14] M. G. Hanuragav and V. Gopinath, “Graph-driven validation of CSR (TFLs) using semantic technologies,” in *CEUR Workshop Proceedings*, vol. 4085, 2025. Accessed: Jan. 16, 2026. [Online]. Available: https://ceur-ws.org/Vol-4085/paper10.pdf

[15] P. Agarwal, N. Kumar, and S. Bedathur, “SymKGQA: Few-shot knowledge graph question answering via symbolic program generation and execution,” in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024, pp. 10 119–10 140. DOI: 10.18653/v1/2024.acl-long.545

[16] C. V. S. Avila, V. M. Vidal, W. Franco, and M. A. Casanova, “Few-shot learning or RAG in LLM-based text-to-SPARQL? Why not both?” In *2025 19th International Conference on Semantic Computing (ICSC)*, 2025, pp. 76–79. DOI: 10.1109/ICSC64641.2025.00016

[17] L. Jiang, J. Huang, C. Möller, and R. Usbeck, “Ontology-guided, hybrid prompt learning for generalization in knowledge graph question answering,” in *2025 19th International Conference on Semantic Computing (ICSC)*, 2025, pp. 28–35. DOI: 10.1109/ICSC64641.2025.00010

[18] M. Shah et al., "Improving LLM-based KGQA for multi-hop question answering with implicit reasoning in few-shot examples," in *Proceedings of the 1st Workshop on Knowledge Graphs and Large Language Models (KaLLM 2024)*, Association for Computational Linguistics, 2024, pp. 125–135. DOI: `10.18653/v1/2024.kallm-1.13`

[19] S. Walter and H. Bast, "GRASP: Generic Reasoning And SPARQL generation across knowledge graphs," *Lecture Notes in Computer Science*, vol. 16140, pp. 271–289, 2026. DOI: `10.1007/978-3-032-09527-5_15`

[20] A. Soularidis, K. Kotis, M. Lamolle, Z. Mejdoul, G. Lortal, and G. Vouros, "LLM-assisted generation of SWRL rules from natural language," in *2024 International Conference on AI x Data and Knowledge Engineering (AIxDKE)*, 2024, pp. 7–12. DOI: `10.1109/AIxDKE63520.2024.00008`

[21] J. F. Lehmann, P. Gattogi, D. Bhandiwad, S. Ferré, and S. Vahdati, "Language models as controlled natural language semantic parsers for knowledge graph question answering," *Frontiers in Artificial Intelligence and Applications*, vol. 372, pp. 1348–1356, 2023. DOI: `10.3233/FAIA230411`

[22] L. Kovriguina, R. Teucher, D. Radyush, and D. Mouromtsev, "SPARQLGEN: One-shot prompt-based approach for SPARQL query generation," in *International Conference on Semantic Systems*, 2023. Accessed: Jan. 16, 2026. [Online]. Available: `https://ceur-ws.org/Vol-3526/paper-08.pdf`

[23] M. Mountantonakis and Y. Tzitzikas, "Generating SPARQL queries over CIDOC-CRM using a two-stage ontology path patterns method in LLM prompts," *Journal on Computing and Cultural Heritage*, vol. 18, 2025. DOI: `10.1145/3708326`

[24] J. G. Ongris, E. Tjitrahardja, F. Darari, and F. J. Ekaputra, "Towards an open NLI LLM-based system for KGs: A case study of Wikidata," in *2024 7th International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, 2024, pp. 44–49. DOI: `10.1109/ISRITI64779.2024.10963661`

[25] L. M. Vieira da Silva, A. Kocher, F. Gehlhoff, and A. Fay, "On the use of large language models to generate capability ontologies," in *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2024, pp. 1–8. DOI: `10.1109/ETFA61755.2024.10710775`

[26] V. Emonet, J. Bolleman, S. Duvaud, T. M. de Farias, and A. C. Sima, "LLM-based SPARQL query generation from natural language over federated knowledge graphs," 2025. DOI: 10.48550/arXiv.2410.06062 arXiv: 2410.06062 [cs.DB].

[27] N. Mashhaditafreshi, A. Textor, P. Rubel, N. Moarefvand, and A. Wagner, "SAMM Copilot: Bootstrapping semantic models with the Eclipse semantic modeling framework from domain data in JSON using large language models," in *CEUR Workshop Proceedings*, 2025. Accessed: Jan. 16, 2026. [Online]. Available: https://ceur-ws.org/Vol-4020/Paper_ID_1.pdf

[28] J. F. Sequeda, D. Allemang, and B. Jacob, "Knowledge graphs as a source of trust for LLM-powered enterprise question answering," *Journal of Web Semantics*, vol. 85, 2025. DOI: 10.1016/j.websem.2024.100858

[29] D. Allemang and J. F. Sequeda, "Increasing the accuracy of LLM question-answering systems with ontologies," *Lecture Notes in Computer Science*, vol. 15233, pp. 324–339, 2025. DOI: 10.1007/978-3-031-77847-6_18

[30] A. Gashkov, M. Eltsova, A. Perevalov, and A. Both, "Instruction-tuned language models as judges for SPARQL query correctness in knowledge graph question answering," in *CEUR Workshop Proceedings*, vol. 4020, 2025, pp. 177–194. Accessed: Jan. 16, 2026. [Online]. Available: https://ceur-ws.org/Vol-4020/Paper_ID_12.pdf

[31] D. Adam and T. Kliegr, "Traceable LLM-based validation of statements in knowledge graphs," *Information Processing and Management*, vol. 62, no. 4, 2025. DOI: 10.1016/j.ipm.2025.104128

[32] L. P. Meyer et al., "LLM-KG-Bench 3.0: A compass for semantic technology capabilities in the ocean of LLMs," *Lecture Notes in Computer Science*, vol. 15719, pp. 280–296, 2025. DOI: 10.1007/978-3-031-94578-6_16

[33] C. Kosten, F. Nooralahzadeh, and K. Stockinger, "Evaluating the effectiveness of prompt engineering for knowledge graph question answering," *Frontiers in Artificial Intelligence*, vol. 7, 2024. DOI: 10.3389/frai.2024.1454258

[34] D. M. Schmidt, R. Schubert, and P. Cimiano, "CompoST: A benchmark for analyzing the ability of LLMs to compositionally interpret questions in a QALD setting," in *The Semantic Web - ISWC 2025: 24th International Semantic Web Conference, Nara,*

Japan, November 2-6, 2025, Proceedings, Part I, Springer-Verlag, 2025, pp. 3–22. DOI: 10.1007/978-3-032-09527-5_1

[35] N. Tufek et al., "Validating semantic artifacts with large language models," *Lecture Notes in Computer Science*, vol. 15344, pp. 92–101, 2025. DOI: 10.1007/978-3-031-78952-6_9

[36] C. Kosten, P. Cudré-Mauroux, and K. Stockinger, "Spider4SPARQL: A complex benchmark for evaluating knowledge graph question answering systems," in *2023 IEEE International Conference on Big Data (BigData)*, 2023, pp. 5272–5281. DOI: 10.1109/bigdata59044.2023.10386182

[37] Z. Lin, S. Trivedi, and J. Sun, "Generating with confidence: Uncertainty quantification for black-box large language models," 2023. DOI: 10.48550/arxiv.2305.19187 arXiv: 2305.19187 [cs.CL].

[38] European Commission, "Core Public Service Vocabulary Application Profile (CPSV-AP)," Interoperable Europe Portal, 2024. Accessed: Jan. 16, 2026. [Online]. Available: https://ec.europa.eu/isa2/solutions/core-public-service-vocabulary-application-profile-cpsv-ap_en/

[39] European Commission, "Core Criterion and Core Evidence Vocabulary (CCCEV)," Interoperable Europe Portal, 2024. Accessed: Jan. 16, 2026. [Online]. Available: https://interoperable-europe.ec.europa.eu/collection/semic-support-centre/solution/core-vocabularies/core-criterion-and-core-evidence-vocabulary

[40] E. Stani, F. Barthélemy, K. Raes, M. Pittomvils, and M. A. Rodriguez, "How data vocabulary standards enhance the exchange of information exposed through APIs: The case of public service descriptions," in *Proceedings of the 13th International Conference on Theory and Practice of Electronic Governance*, 2020, pp. 807–810. DOI: 10.1145/3428502.3428626

[41] D. Brickley and R. V. Guha, "RDF Schema 1.1," W3C, W3C Recommendation, 2014. Accessed: Jan. 16, 2026. [Online]. Available: https://www.w3.org/TR/rdf-schema/

[42] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu, "Unifying large language models and knowledge graphs: A roadmap," *IEEE Transactions on Knowledge and*

Data Engineering, vol. 36, no. 7, pp. 3580–3599, 2024. DOI: 10.1109/tkde.2024.3352100

[43] E. Prud'hommeaux, G. Carothers, D. Beckett, and T. Berners-Lee, "RDF 1.1 Turtle: Terse RDF Triple Language," W3C, W3C Recommendation, 2014. Accessed: Jan. 16, 2026. [Online]. Available: https://www.w3.org/TR/turtle/

[44] H. Knublauch and D. Kontokostas, "Shapes Constraint Language (SHACL)," W3C, W3C Recommendation, 2017. Accessed: Jan. 16, 2026. [Online]. Available: https://www.w3.org/TR/shacl/

[45] E. Nuyts, M. Bonduel, and R. Verstraeten, "Comparative analysis of approaches for automated compliance checking of construction data," *Advanced Engineering Informatics*, vol. 60, p. 102 443, 2024. DOI: 10.1016/j.aei.2024.102443

[46] N. Ferranti, J. F. de Souza, S. Ahmetaj, and A. Polleres, "Formalizing and validating Wikidata's property constraints using SHACL and SPARQL," *Semantic Web*, vol. 15, pp. 2333–2380, 2024. DOI: 10.3233/sw-243611

[47] P. Pareti and G. Konstantinidis, "A review of SHACL: From data validation to schema reasoning for RDF graphs," in *Reasoning Web. Declarative Artificial Intelligence: 17th International Summer School 2021, Leuven, Belgium, September 8-15, 2021, Tutorial Lectures*, 2021, pp. 115–144. DOI: 10.1007/978-3-030-95481-9_6

[48] P. Hagedorn, P. Pauwels, and M. König, "Semantic rule checking of cross-domain building data in information containers for linked document delivery using the shapes constraint language," *Automation in Construction*, vol. 156, p. 105 106, 2023. DOI: 10.1016/j.autcon.2023.105106

[49] S. Ouyang, J. M. Zhang, M. Harman, and M. Wang, "An empirical study of the non-determinism of ChatGPT in code generation," *ACM Transactions on Software Engineering and Methodology*, vol. 34, pp. 1–28, 2023. DOI: 10.1145/3697010

[50] M. Schäfer, S. Nadi, A. Eghbali, and F. Tip, "An empirical evaluation of using large language models for automated unit test generation," *IEEE Transactions on Software Engineering*, vol. 50, pp. 85–105, 2023. DOI: 10.1109/tse.2023.3334955

[51] B. Donato, L. Mariani, D. Micucci, and O. Riganelli, "Studying how configurations impact code generation in LLMs: The case of ChatGPT," in *2025 IEEE/ACM 33rd International Conference on Program Comprehension (ICPC)*, 2025, pp. 442–453. DOI: 10.1109/ICPC66645.2025.00055

[52] M. Papadakis, M. Kintis, J. M. Zhang, Y. Jia, Y. L. Traon, and M. Harman, "Chapter six - mutation testing advances: An analysis and survey," *Advances in Computers*, vol. 112, pp. 275–378, 2019. DOI: 10.1016/bs.adcom.2018.03.015

[53] F. Tip, J. Bell, and M. Schäfer, "LLMorpheus: Mutation testing using large language models," *IEEE Transactions on Software Engineering*, vol. 51, pp. 1645–1665, 2024. DOI: 10.1109/tse.2025.3562025

[54] A. Gupta, M. Lu, K. Zhu, S. O'Brien, and V. Sharma, "NovelHopQA: Diagnosing multi-hop reasoning failures in long narrative contexts," 2025. DOI: 10.48550/arxiv.2506.02000 arXiv: 2506.02000 [cs.CL].

[55] Y.-C. Chen et al., "Enhancing responses from large language models with role-playing prompts: A comparative study on answering frequently asked questions about total knee arthroplasty," *BMC Medical Informatics and Decision Making*, vol. 25, no. 196, 2025. DOI: 10.1186/s12911-025-03024-5

[56] L. Wang et al., "Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models," 2023. DOI: 10.48550/arxiv.2305.04091 arXiv: 2305.04091 [cs.CL].

[57] Z. Chu, Y. Wang, L. Li, Z. Wang, Z. Qin, and K. Ren, "A causal explainable guardrails for large language models," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 1136–1150. DOI: 10.1145/3658644.3690217

[58] S. Sivarajkumar, M. Kelley, A. Samolyk-Mazzanti, S. Visweswaran, and Y. Wang, "An empirical evaluation of prompting strategies for large language models in zero-shot clinical natural language processing: Algorithm development and validation study," *JMIR Medical Informatics*, vol. 12, 2024. DOI: 10.2196/55318

[59] S. Kresevic, M. Giuffré, M. Ajčević, A. Accardo, L. Crocè, and D. Shung, "Optimization of hepatological clinical guidelines interpretation by large language models: A retrieval augmented generation-based framework," *NPJ Digital Medicine*, vol. 7, 2024. DOI: 10.1038/s41746-024-01091-y

[60] L. Liu et al., "Instruct-of-reflection: Enhancing large language models iterative reflection capabilities via dynamic-meta instruction," 2025. DOI: 10.48550/arxiv.2503.00902 arXiv: 2503.00902 [cs.CL].

[61] M. Chen et al., “Evaluating large language models trained on code,” 2021. DOI: 10.48550/arxiv.2107.03374 arXiv: 2107.03374 [cs.LG].

[62] S. Lee et al., “Reasoning abilities of large language models: In-depth analysis on the abstraction and reasoning corpus,” 2024. DOI: 10.48550/arxiv.2403.11793 arXiv: 2403.11793 [cs.CL].

[63] L.-P. Meyer, J. Frey, F. Brei, and N. Arndt, “Assessing SPARQL capabilities of large language models,” 2025. DOI: 10.48550/arXiv.2409.05925 arXiv: 2409.05925 [cs.DB].

[64] V. G. Cerf, “Large language models,” *Communications of the ACM*, vol. 66, pp. 7–7, 2023. DOI: 10.1145/3606337

[65] J. Wei et al., “Emergent abilities of large language models,” 2022. DOI: 10.48550/arxiv.2206.07682 arXiv: 2206.07682 [cs.CL].

[66] Y. Fu, H.-C. Peng, L. Ou, A. Sabharwal, and T. Khot, “Specializing smaller language models towards multi-step reasoning,” 2023. DOI: 10.48550/arxiv.2301.12726 arXiv: 2301.12726 [cs.CL].

[67] H. A. Kautz, “The third AI summer: AAAI Robert S. Engelmore memorial lecture,” *AI Magazine*, vol. 43, no. 1, pp. 105–125, 2022. DOI: 10.1002/aaai.12036

[68] N. Dziri et al., “Faith and fate: Limits of transformers on compositionality,” 2023. DOI: 10.48550/arxiv.2305.18654 arXiv: 2305.18654 [cs.CL].

[69] B. Chen, Z. Zhang, N. Langrené, and S. Zhu, “Unleashing the potential of prompt engineering for large language models,” *Patterns*, vol. 6, 2023. DOI: 10.1016/j.patter.2025.101260

[70] H. Ma et al., “Fairness-guided few-shot prompting for large language models,” 2023. DOI: 10.48550/arxiv.2303.13217 arXiv: 2303.13217 [cs.CL].

[71] A. Ali, L. Wolf, and I. Titov, “Mitigating copy bias in in-context learning through neuron pruning,” 2024. DOI: 10.48550/arxiv.2410.01288 arXiv: 2410.01288 [cs.CL].

[72] Z. Fei et al., “LawBench: Benchmarking legal knowledge of large language models,” 2023. DOI: 10.48550/arxiv.2309.16289 arXiv: 2309.16289 [cs.CL].

[73] H. Naveed et al., “A comprehensive overview of large language models,” *ACM Transactions on Intelligent Systems and Technology*, vol. 16, pp. 1–72, 2023. DOI: 10.1145/3744746

[74] M. Kuziemski and G. Misuraca, “AI governance in the public sector: Three tales from the frontiers of automated decision-making in democratic settings,” *Telecommunications Policy*, vol. 44, pp. 101 976–101 976, 2020. doi: `10.1016/j.telpol.2020.101976`

[75] M. Hanisch, C. M. Goldsby, N. E. Fabian, and J. Oehmichen, “Digital governance: A conceptual framework and research agenda,” *Journal of Business Research*, vol. 162, p. 113 777, 2023. doi: `10.1016/j.jbusres.2023.113777`

A Code for Pipeline Core

This appendix lists the full python code of the *pipeline core*, the main executable part of the pipeline. It is intended to give a high-level overview only, and for that reason it does not include any of the functions that make up the custom utility libraries, which are used throughout the code. For the entirety of the source code, refer to the dissertation's GitHub repository [10], under the `src` directory.

Listing A.1: Pipeline Core Python source code

```
"""Core pipeline module for the neuro-symbolic dissertation,
   orchestrating LLM-based SHACL generation and validation."""

# Standard library imports
import json
import time

# Third-party imports
from pydantic import BaseModel
from rdflib import Graph, Namespace
from typing import List

# Local imports
from src.graph_utils import get_semantic_hash,
   resolve_node_path, validate_shacl_syntax, visualize_graph
from src.llm_utils import call_gemini, call_gemini_json,
   call_gemini_pdf, initialize_gemini_client, reflect,
   with_retries
from src.parsing_utils import read_txt

# Constants for Turtle graph prefixes
PREFIXES = """@prefix ex: <http://example.org/> .
@prefix cccev: <http://data.europa.eu/m8g/> .
@prefix cpsv: <http://purl.org/vocab/cpsv#> .
```

```
@prefix dct: <http://purl.org/dc/terms/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

"""

# Standard headers for SHACL shapes
STANDARD_HEADERS = """
@prefix : <http://example.org/schema#> .
@prefix ex: <http://example.org/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
"""

def run_main_pipeline(ctx, artifact_dir: str, progress_bar,
   DOCUMENT_NAME: str, PROMPT_VERSION: str, GEMINI_MODEL: str,
    current_run_id: int):
    """Runs the main neuro-symbolic pipeline for SHACL
       generation and validation.

    Orchestrates the end-to-end process: document
       summarization, information model creation,
    graph generation, SHACL synthesis, validation,
       visualization.

    Args:
        ctx: Context dictionary to store results and metadata.
        artifact_dir: Directory to save generated artifacts.
        progress_bar: Progress bar object for updates.
        DOCUMENT_NAME: Name of the document being processed.
        PROMPT_VERSION: Version of prompts to use.
        GEMINI_MODEL: Gemini model name for LLM calls.
        current_run_id: ID of the current experiment run.

    Returns:
        Updated context dictionary with results and execution
            time.
    """
    # Initialize Gemini client
    initialize_gemini_client(model_name=GEMINI_MODEL)

    execution_start_time = time.time()

    ### 1.1 Document -> Preconditions Summary
    progress_bar.set_description(f"Run {current_run_id}:
        Generating Preconditions Summary...")

    file_path = f"Precondition documents/{DOCUMENT_NAME}.pdf"
    prompt = read_txt(f'Prompts/{PROMPT_VERSION}/summarization
        .txt')
    preconditions_summary = with_retries(call_gemini_pdf,
        prompt, file_path)
    # Optional: reflexion
    if PROMPT_VERSION == 'Reflexion':
        preconditions_summary = reflect([prompt],
            preconditions_summary)

    # Save artifact
    with open(f"{artifact_dir}/{DOCUMENT_NAME} preconditions
        summary.txt", "w") as f:
        f.write(preconditions_summary)

    ### 1.2. Preconditions Summary + Citizen Schema (TTL) ->
        Information Model (JSON)
    progress_bar.set_description(f"Run {current_run_id}:
        Generating Information Model...")

    citizen_schema = read_txt(f"Citizens/{DOCUMENT_NAME}
        schema.ttl")

    class Paths(BaseModel):
        path: List[str]
        datatype: str

    class InformationConcept(BaseModel):
        name: str
        related_paths: List[Paths]  # Links the concept to citizen data available

    class Constraint(BaseModel):
        name: str
        desc: str
        constrains: List[InformationConcept]

    schema = list[Constraint]

    # Formulate prompt content and call Gemini
    prompt = read_txt(f'Prompts/{PROMPT_VERSION}/preconditions_to_JSON.txt')
    content = [prompt, preconditions_summary, citizen_schema]
    info_model_str = with_retries(call_gemini_json, content, schema)
    # Optional: reflexion
    if PROMPT_VERSION == 'Reflexion':
        info_model_str = reflect(content, info_model_str, schema)

    # Save artifact
    with open(f"{artifact_dir}/{DOCUMENT_NAME} information model.json", "w") as f:
        f.write(info_model_str)

    ### 1.3 Information Model (JSON) -> Public Service Graph (TTL)
    # Parse JSON string
    info_model = json.loads(info_model_str)
    service_name = DOCUMENT_NAME

    triples = [PREFIXES]
    triples.append(f"ex:{service_name} a cpsv:PublicService .\
       n\n")

    # Convert constraints + concepts into triples
    for constraint in info_model:
        constraint_name = constraint["name"]
        constraint_desc = constraint["desc"].replace('"', '\\"
           ')

        # Public service -> holdsRequirement -> constraint
        triples.append(f"ex:{service_name} cpsv:
           holdsRequirement ex:{constraint_name} .\n")

        # Constraint node
        triples.append(f'ex:{constraint_name} a cccev:
           Constraint ; dct:description "{constraint_desc}" .\
           n')

        # InformationConcept nodes
        for concept in constraint.get("constrains", []):
            concept_name = concept["name"]

            # Link constraint to concept
            triples.append(f"ex:{constraint_name} cccev:
               constrains ex:{concept_name} .\n")

            # Declare information concept
            triples.append(f'ex:{concept_name} a cccev:
               InformationConcept .\n')

        triples.append("\n")  # Spacing for readability

    service_graph_ttl = "".join(triples)

    # Log semantic hash
    ctx["Service Graph Hash"] = get_semantic_hash(
        service_graph_ttl)

    # Save artifact
    with open(f"{artifact_dir}/{DOCUMENT_NAME} service graph.
        ttl", "w") as f:
        f.write(service_graph_ttl)

    ### 1.4. Graph Visualization / Inspection
    visualize_graph(f"{artifact_dir}/{DOCUMENT_NAME} service
        graph.ttl")  # Also saves its own HTML artifact

    ### 2.1. Information Model (JSON) -> SHACL-spec (JSON)
    shacl_spec_json = []

    for constraint in info_model:
        # Rename for clarity downstream
        shape_name = constraint["name"].replace("_condition",
            "_shape")
        desc = constraint["desc"]

        concepts = []
        # Iterate concepts (e.g., family_income,
            residency_city)
        for concept in constraint.get("constrains", []):
            related_paths = []

            paths_source = concept.get("related_paths", [])

            for rp in paths_source:
                # Capture the path and datatype (URI vs
                    Literal)
                related_paths.append({
                    "path": rp["path"],
                    "datatype": rp["datatype"]
                })

            concepts.append({
                "name": concept["name"],
                "related_paths": related_paths
            })

        shacl_spec_json.append({
            "shape_name": shape_name,
            "desc": desc,
            "concepts": concepts
        })

    # Save artifact
    with open(f"{artifact_dir}/{DOCUMENT_NAME} shacl-spec.json", "w") as f:
        json.dump(shacl_spec_json, f, indent=2)

    ### 2.2. SHACL-spec (JSON) + Citizen Schema (TTL) -> SHACL Shapes (TTL)
    progress_bar.set_description(f"Run {current_run_id}: Generating SHACL Shapes...")

    # Convert JSON to string for prompt
    shacl_spec_str = json.dumps(shacl_spec_json)
    prompt = read_txt(f'Prompts/{PROMPT_VERSION}/shacl_spec_to_shacl_ttl.txt')
    content = [prompt, shacl_spec_str, citizen_schema]

    shacl_shapes = with_retries(call_gemini, content)
    # Optional: reflexion
    if PROMPT_VERSION == 'Reflexion':
        shacl_shapes = reflect(content, shacl_shapes)

    # Cleanup Gemini markdown formatting
    shacl_shapes = shacl_shapes.strip("‘").replace("turtle", "
").replace("ttl", "").strip()

    # Strip existing headers to avoid duplicates/conflicts
    lines = shacl_shapes.split('\n')
    body_lines = [line for line in lines if not line.strip().lower().startswith('@prefix')]
    clean_body = '\n'.join(body_lines)
    # Prepend correct headers
    shacl_shapes = STANDARD_HEADERS + "\n" + clean_body

    # Log validation results
    ctx["SHACL Graph Hash"] = get_semantic_hash(shacl_shapes)
    is_valid, error_stage, error_message = validate_shacl_syntax(shacl_shapes)
    ctx["SHACL Valid Syntax"] = is_valid
    ctx["SHACL Error Type"] = error_stage
    ctx["SHACL Error Message"] = error_message

    # Save artifact
    with open(f"{artifact_dir}/{DOCUMENT_NAME} shacl shapes.ttl", "w") as f:
        f.write(shacl_shapes)

    ### 3.1 Public Service Graph (TTL) + Citizen Graph (TTL) + Information Model (JSON) -> Citizen-Service Graph (TTL)
    EX = Namespace("http://example.org/")
    SC = Namespace("http://example.org/schema#")

    # Load service and citizen TTLs and info model
    citizen_ttl = f"Citizens/{DOCUMENT_NAME} eligible.ttl"

    # Realize them into graphs
    unified_graph = Graph()
    unified_graph.parse(data=service_graph_ttl, format="turtle")
    citizen_graph = Graph()
    citizen_graph.parse(citizen_ttl, format="turtle")

    # Merge citizen triples into main graph
    for triple in citizen_graph:
        unified_graph.add(triple)

    # Automatically determine the root citizen node
    root_candidates = list(citizen_graph.subjects(predicate=
       None, object=SC.Applicant))
    citizen_root = root_candidates[0]

    # Add mapsTo edges
    for constraint in info_model:
        for concept in constraint["constrains"]:
            concept_uri = EX[concept["name"]]

            for path_obj in concept["related_paths"]:
                path_list = path_obj["path"]
                dtype = path_obj["datatype"]

                # Pass the datatype to the resolver
                subject_nodes = resolve_node_path(
                   citizen_graph, citizen_root, path_list,
                   dtype)

                for subj in subject_nodes:
                    # Connect the Information Concept to the
                       Data Node
                    unified_graph.add((concept_uri, EX.mapsTo,
                        subj))

    # Serialize unified graph into TTL and save to file
    unified_graph.serialize(f"{artifact_dir}/{DOCUMENT_NAME}
       citizen-service graph.ttl", format="turtle")

    ### 3.2 Visualize the unified graph
    visualize_graph(f"{artifact_dir}/{DOCUMENT_NAME} citizen-
       service graph.ttl")

    # This marks the end of the main pipeline.
    ctx["Execution Time"] = round(time.time() -
       execution_start_time)

    return ctx
```

B Serialized Citizen-Service Graph

The `ttl` code below is one citizen-service graph created as an artifact of the pipeline, that corresponds to the simple Parental Leave document use case. It is visualized with *PyVis* and depicted in Figure 3.4.

Listing B.1: Full citizen-service graph Turtle for the Parental Leave document

```
@prefix cccev: <http://data.europa.eu/m8g/> .
@prefix cpsv: <http://purl.org/vocab/cpsv#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix ex: <http://example.org/> .
@prefix ns1: <http://example.org/schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ex:Dimitris a ns1:Applicant,
        ns1:Person ;
    ns1:birthDate "1990-05-15"^^xsd:date ;
    ns1:hasChild ex:Nikolas ;
    ns1:hasEmployment ex:DimitrisJob .

ex:parental_leave a cpsv:PublicService ;
    cpsv:holdsRequirement ex:child_age_condition,
        ex:employment_duration_condition,
        ex:employment_status_and_sector_condition .

ex:child_age_condition a cccev:Constraint ;
    cccev:constrains ex:child_birth_date ;
    dct:description "The beneficiary's child must be under the
        age of eight (8)." .

ex:child_birth_date a cccev:InformationConcept ;
    ex:mapsTo ex:Nikolas .

```

```
ex:employment_duration_condition a cccev:Constraint ;
    cccev:constrains ex:employment_start_date ;
    dct:description "The beneficiary must have completed at
       least one (1) year of continuous employment (or
       successive fixed-term contracts) with the same employer
       ." .

ex:employment_sector a cccev:InformationConcept ;
    ex:mapsTo ex:DimitrisJob .

ex:employment_start_date a cccev:InformationConcept ;
    ex:mapsTo ex:DimitrisJob .

ex:employment_status_and_sector_condition a cccev:Constraint ;
    cccev:constrains ex:employment_sector,
        ex:is_currently_employed ;
    dct:description "The beneficiary must be employed under a
       dependent employment relationship or a salaried mandate
        in the private or public sector." .

ex:is_currently_employed a cccev:InformationConcept ;
    ex:mapsTo ex:DimitrisJob .

ex:Nikolas a ns1:Person ;
    ns1:birthDate "2022-03-10"^^xsd:date .

ex:DimitrisJob a ns1:EmploymentRecord ;
    ns1:employmentSector "Private" ;
    ns1:employmentStartDate "2023-01-01"^^xsd:date ;
    ns1:isActive true .
```

C Citizen Schemas & Golden Citizens

This appendix provides the full `ttl` files that describe the LLM-generated citizen schemas used as context for the pipeline, as detailed in Section 3.2.2, along with their respective hand-crafted "Golden Citizen" instance graphs described in Section 3.3.1. As per the description of the Experimental Design on Section 3.3, there is one pair of files per document use case.

Student Housing Use Case

Listing C.1: The Citizen Schema RDFS for the Student Housing document

```
@prefix : <http://example.org/schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

# ==========================================
# 1. CORE ENTITY
# ==========================================

:Person a rdfs:Class ;
    rdfs:comment "A human individual." .

:Applicant a rdfs:Class ;
    rdfs:subClassOf :Person ;
    rdfs:comment "A person actively applying for a public
       service. Used as the Focus Node for SHACL validation."
       .

# --- Basic Attributes ---
```

```
:citizenshipCountry a rdf:Property ;
    rdfs:domain :Person ;
    rdfs:range xsd:string ;
    rdfs:comment "ISO code (e.g., 'GR', 'FR')." .

:birthDate a rdf:Property ;
    rdfs:domain :Person ;
    rdfs:range xsd:date .

:isDependent a rdf:Property ;
    rdfs:domain :Person ;
    rdfs:range xsd:boolean ;
    rdfs:comment "Flag to explicitly mark a family member as a
        dependent." .

# --- Relationships ---
:hasParent a rdf:Property ;
    rdfs:domain :Person ;
    rdfs:range :Person .

:hasChild a rdf:Property ;
    rdfs:domain :Person ;
    rdfs:range :Person .

# ==========================================
# 2. RESIDENCE & HOUSING
# ==========================================

:Residence a rdfs:Class ;
    rdfs:comment "A place where a person lives, either rented
       or owned." .

:Municipality a rdfs:Class ;
    rdfs:comment "The city or administrative area of a
       residence." .

```

```
# --- Residence Properties ---
:hasCurrentResidence a rdf:Property ;
    rdfs:domain :Person ;
    rdfs:range :Residence ;
    rdfs:comment "Where the student is currently living." .

:hasFamilyResidence a rdf:Property ;
    rdfs:domain :Person ;
    rdfs:range :Residence ;
    rdfs:comment "The permanent home of the parents/family." .

:locatedIn a rdf:Property ;
    rdfs:domain :Residence ;
    rdfs:range :Municipality .

:residenceType a rdf:Property ;
    rdfs:domain :Residence ;
    rdfs:range xsd:string ;
    rdfs:comment "Values: 'Rented', 'Owned', 'Guest'." .

:leaseStartDate a rdf:Property ;
    rdfs:domain :Residence ;
    rdfs:range xsd:date ;
    rdfs:comment "If rented, when the lease started." .

# ==========================================
# 3. INCOME
# ==========================================

:IncomeRecord a rdfs:Class ;
    rdfs:comment "Financial data for a specific year." .

:hasIncome a rdf:Property ;
    rdfs:domain :Person ;
    rdfs:range :IncomeRecord .

```

```
:incomeYear a rdf:Property ;
    rdfs:domain :IncomeRecord ;
    rdfs:range xsd:gYear .

:amount a rdf:Property ;
    rdfs:domain :IncomeRecord ;
    rdfs:range xsd:decimal .

# ==========================================
# 4. REAL ESTATE ASSETS
# ==========================================

:RealEstateProperty a rdfs:Class ;
    rdfs:comment "A distinct physical property owned by
       someone." .

:ownsRealEstate a rdf:Property ;
    rdfs:domain :Person ;
    rdfs:range :RealEstateProperty .

:propertyAreaM2 a rdf:Property ;
    rdfs:domain :RealEstateProperty ;
    rdfs:range xsd:decimal .

:propertyLocation a rdf:Property ;
    rdfs:domain :RealEstateProperty ;
    rdfs:range :Municipality .

:municipalityPopulation a rdf:Property ;
    rdfs:domain :Municipality ;
    rdfs:range xsd:integer .

# ==========================================
# 5. EDUCATION
# ==========================================

:EducationRecord a rdfs:Class ;
    rdfs:comment "Academic status for a specific period." .

:hasEducation a rdf:Property ;
    rdfs:domain :Person ;
    rdfs:range :EducationRecord .

:cityOfStudy a rdf:Property ;
    rdfs:domain :EducationRecord ;
    rdfs:range :Municipality ;
    rdfs:comment "Links the education record to the City/
        Municipality of the University." .

:hasValidAcademicID a rdf:Property ;
    rdfs:domain :EducationRecord ;
    rdfs:range xsd:boolean .

:passedCoursesCount a rdf:Property ;
    rdfs:domain :EducationRecord ;
    rdfs:range xsd:integer .

:totalCoursesCount a rdf:Property ;
    rdfs:domain :EducationRecord ;
    rdfs:range xsd:integer .
```

Listing C.2: The Golden Citizen for the Student Housing document

```
@prefix : <http://example.org/schema#> .
@prefix ex: <http://example.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

# ==========================================
# 1. GEOGRAPHY
# ==========================================

ex:Athens a :Municipality ;
```

```
    :municipalityPopulation 664046 .

ex:Thessaloniki a :Municipality ;
    :municipalityPopulation 315196 .

ex:TinyVillage a :Municipality ;
    :municipalityPopulation 500 .

# ==========================================
# 2. THE APPLICANT
# ==========================================

ex:Katerina a :Applicant, :Person ;
    :citizenshipCountry "GR" ;
    :birthDate "2003-05-20"^^xsd:date ; # 22 Years old
    :hasParent ex:Dad, ex:Mom ;
    :isDependent true ;
    # Housing & Education
    :hasCurrentResidence ex:StudentApartment ;
    :hasFamilyResidence ex:FamilyHome ;
    :hasEducation ex:KaterinaEduRecord ;
    :hasIncome ex:KaterinaIncome .

# ==========================================
# 3. THE FAMILY (Parents + Sibling)
# ==========================================

ex:Dad a :Person ;
    :birthDate "1970-01-01"^^xsd:date ;
    :hasChild ex:Katerina, ex:Fonis ;
    :hasIncome ex:DadIncome ;
    :ownsRealEstate ex:DadMainProperty .

ex:Mom a :Person ;
    :birthDate "1975-03-15"^^xsd:date ;
    :hasChild ex:Katerina, ex:Fonis ;
    :hasIncome ex:MomIncome ;
    :ownsRealEstate ex:MomVillageHouse .

ex:Fonis a :Person ;
    :birthDate "2015-06-01"^^xsd:date ; # 10 Years old (Minor)
    :isDependent true ;
    :hasParent ex:Dad, ex:Mom .

# ==========================================
# 4. RESIDENCES & LEASES
# ==========================================

ex:StudentApartment a :Residence ;
    :locatedIn ex:Athens ;
    :residenceType "Rented" ;
    :leaseStartDate "2024-09-01"^^xsd:date .

ex:FamilyHome a :Residence ;
    :locatedIn ex:Thessaloniki ;
    :residenceType "Owned" .

# ==========================================
# 5. INCOME RECORDS
# ==========================================

ex:KaterinaIncome a :IncomeRecord ;
    :incomeYear "2024"^^xsd:gYear ;
    :amount 0.0 .

ex:DadIncome a :IncomeRecord ;
    :incomeYear "2024"^^xsd:gYear ;
    :amount 25000.0 .

ex:MomIncome a :IncomeRecord ;
    :incomeYear "2024"^^xsd:gYear ;
    :amount 7000.0 .
```

```

# ==========================================
# 6. EDUCATION RECORD
# ==========================================

ex:KaterinaEduRecord a :EducationRecord ;
    :cityOfStudy ex:Athens ;
    :hasValidAcademicID true ;
    :passedCoursesCount 6 ;
    :totalCoursesCount 10 .

# ==========================================
# 7. REAL ESTATE ASSETS
# ==========================================

ex:DadMainProperty a :RealEstateProperty ;
    :propertyLocation ex:Thessaloniki ;
    :propertyAreaM2 150.0 .

ex:MomVillageHouse a :RealEstateProperty ;
    :propertyLocation ex:TinyVillage ;
    :propertyAreaM2 80.0 .
```

Parental Leave Use Case

Listing C.3: The Citizen Schema RDFS for the Parental Leave document

```
@prefix : <http://example.org/schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

# ==========================================
# 1. CORE ENTITY
# ==========================================

:Person a rdfs:Class ;
    rdfs:comment "A human individual." .

:Applicant a rdfs:Class ;
    rdfs:subClassOf :Person ;
    rdfs:comment "A person actively applying for the parental
       leave allowance. Used as the Focus Node for SHACL
       validation." .

# --- Basic Attributes ---
:birthDate a rdf:Property ;
    rdfs:domain :Person ;
    rdfs:range xsd:date ;
    rdfs:comment "Used to calculate age (e.g., for the child).
       " .

# --- Relationships ---
:hasChild a rdf:Property ;
    rdfs:domain :Person ;
    rdfs:range :Person ;
    rdfs:comment "Links a parent to their child." .

# ==========================================
# 2. EMPLOYMENT STATUS (Crucial for Eligibility)
# ==========================================

:EmploymentRecord a rdfs:Class ;
    rdfs:comment "A record of an active or past employment
       contract." .

:hasEmployment a rdf:Property ;
    rdfs:domain :Person ;
    rdfs:range :EmploymentRecord ;
    rdfs:comment "Links the applicant to their job details." .

# --- Employment Properties ---
```

```
:employmentSector a rdf:Property ;
    rdfs:domain :EmploymentRecord ;
    rdfs:range xsd:string ;
    rdfs:comment "The sector of employment. Expected values: '
       Private', 'Public'." .

:employmentStartDate a rdf:Property ;
    rdfs:domain :EmploymentRecord ;
    rdfs:range xsd:date ;
    rdfs:comment "The date the current employment or
       continuous contract series began." .

:isActive a rdf:Property ;
    rdfs:domain :EmploymentRecord ;
    rdfs:range xsd:boolean ;
    rdfs:comment "True if the employment is currently active."
        .
```

Listing C.4: The Golden Citizen for the Parental Leave document

```
@prefix : <http://example.org/schema#> .
@prefix ex: <http://example.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

# ==========================================
# 1. THE APPLICANT (Dimitris)
# ==========================================

ex:Dimitris a :Applicant, :Person ;
    # Personal Info
    :birthDate "1990-05-15"^^xsd:date ; # 35 Years old
    :hasChild ex:Nikolas ;

    # Employment Status
    :hasEmployment ex:DimitrisJob .
# ==========================================
# 2. THE CHILD (Nikolas)
# ==========================================

ex:Nikolas a :Person ;
    :birthDate "2022-03-10"^^xsd:date .

# ==========================================
# 3. EMPLOYMENT RECORD
# ==========================================

ex:DimitrisJob a :EmploymentRecord ;
    :employmentSector "Private" ;
    :isActive true ;
    :employmentStartDate "2023-01-01"^^xsd:date .
```

D Scenarios

This appendix provides the full YAML files that describe the scenarios applied for mutation testing. As per the description of the Experimental Design on Section 3.3, there is one file per document use case.

Listing D.1: YAML description of scenarios for the Student Housing document

```
- id: SCN_00
  description: "Conforms to all preconditions."
  expected_violation_count: 0
  actions:
    - type: no_action

- id: SCN_01
  description: "Not an EU citizen."
  expected_violation_count: 1
  actions:
    - type: patch_node
      turtle: ex:Katerina :citizenshipCountry "CA" .

- id: SCN_02
  description: "No academic ID."
  expected_violation_count: 1
  actions:
    - type: patch_node
      turtle: ex:KaterinaEduRecord :hasValidAcademicID false .

- id: SCN_03
  description: "Base income is too high."
  expected_violation_count: 1
  actions:
    - type: patch_node
      turtle: ex:DadIncome :amount 26001.0 .
```

```

- id: SCN_04
  description: "Income threshold decreased due to no dependent
      children."
  expected_violation_count: 1
  actions:
    - type: patch_node
      turtle: ex:Fonis :isDependent false .

- id: SCN_05
  description: "Study city is the same as family residence
     city."
  expected_violation_count: 1
  actions:
    - type: patch_node
      turtle: |
        ex:Trikala a :Municipality ; :municipalityPopulation
           50000 .
        ex:StudentApartment :locatedIn ex:Trikala .
        ex:KaterinaEduRecord :cityOfStudy ex:Trikala .
        ex:FamilyHome :locatedIn ex:Trikala ; :residenceType "
           Rented" .

- id: SCN_06
  description: "Parent owns property in study city."
  expected_violation_count: 1
  actions:
    - type: patch_node
      turtle: ex:DadMainProperty :propertyLocation ex:Athens .

- id: SCN_07
  description: "Lease too short."
  expected_violation_count: 1
  actions:
    - type: patch_node
      turtle: ex:StudentApartment :leaseStartDate "2025-11-30"
```

```
        ^^xsd:date .

- id: SCN_08
  description: "Family property too big."
  expected_violation_count: 1
  actions:
    - type: patch_node
      turtle: ex:DadMainProperty :propertyAreaM2 201.0 .

- id: SCN_09
  description: "Property threshold decreased due to large
     municipality."
  expected_violation_count: 1
  actions:
    - type: patch_node
      turtle: ex:TinyVillage :municipalityPopulation 3001 .

- id: SCN_10
  description: "Bad academic performance."
  expected_violation_count: 1
  actions:
    - type: patch_node
      turtle: ex:KaterinaEduRecord :passedCoursesCount 4 .
```

Listing D.2: YAML description of scenarios for the Parental Leave document

```
- id: SCN_00
  description: "Conforms to all preconditions."
  expected_violation_count: 0
  actions: []

- id: SCN_01
  description: "Tenure too short."
  expected_violation_count: 1
  actions:
    - type: patch_node
      turtle: ex:DimitrisJob :employmentStartDate "2025-08-01"
```

```
          ^^xsd:date .

- id: SCN_02
  description: "Child is older than 8 years."
  expected_violation_count: 1
  actions:
    - type: patch_node
      turtle: ex:Nikolas :birthDate "2015-05-20"^^xsd:date .

- id: SCN_03
  description: "Unemployed."
  expected_violation_count: 1
  actions:
    - type: patch_node
      turtle: ex:DimitrisJob :isActive false .

- id: SCN_04
  description: "Wrong employment sector."
  expected_violation_count: 1
  actions:
    - type: patch_node
      turtle: ex:DimitrisJob :employmentSector "Freelance" .
```

E Glossary of Terms

This glossary provides definitions for the terms utilized throughout this dissertation. The terms are categorized by their domain of origin.

Artificial Intelligence & Large Language Models

Hallucination

A phenomenon in Large Language Models where the system generates text that is syntactically correct but factually incorrect or logically inconsistent with the input context.

In-Context Learning (ICL)

The ability of an LLM to perform a task after being shown a few examples within the prompt, without any permanent updates to its underlying neural weights.

Chain-of-Thought (CoT) Prompting

A prompting technique that encourages an LLM to generate intermediate reasoning steps before arriving at a final answer, often used to improve performance on complex logical or mathematical tasks.

Temperature

A hyperparameter in LLM generation that controls the randomness of the output. A temperature of 0 aims for the most probable/deterministic response, while higher values increase creativity and variability.

Model-as-a-Service (MaaS)

A cloud-computing model where pre-trained Large Language Models are hosted by a provider (e.g., Google, OpenAI) and accessed by developers via an API, removing the need for local hardware at the cost of dependency on the provider's infrastructure.

Semantic Web & Knowledge Engineering

Knowledge Graph (KG)

A structured representation of information using a graph-based data model, where nodes represent entities and edges represent the relationships between them, grounded in a formal ontology.

RDFS (Resource Description Framework Schema)

A formal way of representing properties, and the relationships between those properties, within a specific domain of interest. This work utilizes RDFS to define the "Citizen" and "Public Service" schemas.

Turtle (Terse RDF Triple Language)

A syntax and file format for the Resource Description Framework (RDF) that is designed to be both machine-readable and easily edited by humans.

SHACL (Shapes Constraint Language)

A World Wide Web Consortium (W3C) standard for validating RDF graphs against a set of conditions. In this work, it serves as the primary symbolic engine for eligibility verification.

SPARQL (SPARQL Protocol And RDF Query Language)

An RDF query language and data access protocol used to retrieve and manipulate data stored in Resource Description Framework (RDF) format.

Graph Edit Distance (GED)

A metric for measuring the similarity between two graphs by calculating the minimum number of operations (node/edge insertions, deletions, or substitutions) required to transform one graph into the other.

Abstract Syntax Tree (AST)

A tree representation of the abstract syntactic structure of source code (like SPARQL). Comparing ASTs allows for measuring logical similarity rather than just textual overlap.

Administrative Law & Public Governance

Once-Only Principle (OOP)

A European Union e-government strategy aimed at ensuring that citizens and businesses only have to provide standard information to administrations once, with authorities then sharing this data internally.

Digital Sovereignty

The capacity of a state to exert authority over its digital infrastructure, including the "code" of its laws, without being dependent on proprietary, foreign-controlled ecosystems.