Karlsruhe Institute of Technology

Software Design and Quality

LLM-based Privacy Specification and Data Flow Diagram Derivation for Privacy Analysis

Master's Thesis of

Gabriel Gehrig

At the KIT Department of Informatics
KASTEL – Institute of Information Security and Dependability

First examiner:	Prof. Dr. Ralf Reussner
Second examiner:	Prof. Dr.-Ing. Anne Koziolek
First advisor:	M. Sc. Nicolas Boltz
Second advisor:	Dr. Ing. Tobias Hey

01. December 2025 – 01. June 2026

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Abstract

Privacy analysis of software designs requires formal artifacts derived from natural language requirements for automated detection of privacy violations. The xDECAF framework supports this by running Data Flow Analysis (DFA) on Data Flow Diagrams (DFDs) derived from functional requirements and annotated with privacy specifications derived from legal requirements. Creating these two artifact types from natural language requirements currently requires manual expert effort, which limits the practical adoption of automated privacy analysis. Existing Large Language Model (LLM)-based approaches target DFD generation in isolation and do not produce xDECAF-compatible outputs. No prior work has empirically evaluated LLM-based automation of the full DFA preprocessing chain.

Within this thesis, we designed, implemented, and evaluated two LLM-based pipelines as the first two steps toward full automation of the DFA preprocessing chain. The first derives DFDs from functional requirements using GPT-5.3-Codex. The second derives privacy label schemas and data-flow constraint rules from legal requirements using GPT-5.5. Both use chain-of-thought prompting, schema-validated outputs, and structured retry feedback to contain structural errors before downstream propagation. We evaluated both pipelines through two-part blind user studies: the first with 20 creators and 12 evaluators, the second with 10 creators and 6 evaluators. LLM-derived artifacts received directionally lower average ratings than human-created ones across all artifact types on a 1–5 scale. Comparing LLM-derived to human-created artifacts: DFD completeness and correctness averaged 2.5 vs. 4.0 and 2.9 vs. 3.6, privacy labels 2.8 vs. 3.7 and 2.7 vs. 3.5, and constraint sets 2.3 vs. 3.2 and 1.8 vs. 3.0. Issue flagging identified missing elements and hallucinations as the dominant defect types in LLM-derived DFDs, and vague labels in LLM-derived privacy label sets. One of four LLM-derived DFDs reached ratings comparable to the human group means, indicating that the pipeline can achieve near-human-quality outputs on individual runs. Four shared methodological factors prevent treating these ratings as more than directional. These are Krippendorff's α below the reliability threshold across all rating dimensions, small sample sizes, a creator-rater anchoring effect, and asymmetric rating counts. The recurrence of unreliable inter-rater agreement across both tasks suggests that consistent quality evaluation is difficult for artifact types that admit multiple valid solutions.

Zusammenfassung

Die Datenschutzanalyse von Software-Entwürfen erfordert formale Artefakte aus natürlichsprachigen Anforderungen zur automatisierten Erkennung von Datenschutzverstößen. Das xDECAF-Framework unterstützt dies durch die Datenflussanalyse (DFA) auf Datenflussdiagrammen (DFD), die aus funktionalen Anforderungen abgeleitet und mit Datenschutzspezifikationen aus rechtlichen Anforderungen annotiert werden. Die Erstellung dieser beiden Artefakttypen erfordert derzeit manuellen Expertenaufwand, was die praktische Anwendung automatisierter Datenschutzanalyse einschränkt. Bestehende Ansätze die auf großen Sprachmodellen (LLM) basieren, adressieren die DFD-Generierung isoliert und erzeugen keine xDECAF-kompatiblen Ausgaben. Keine vorherige Arbeit hat die LLM-basierte Automatisierung der vollständigen DFA-Vorverarbeitungskette empirisch evaluiert.
Im Rahmen dieser Arbeit konzipierten, implementierten und evaluierten wir zwei LLM-basierte Pipelines als die ersten zwei Schritte zur vollständigen Automatisierung der DFA-Vorverarbeitungskette. Die erste leitet DFD aus funktionalen Anforderungen mithilfe von GPT-5.3-Codex ab. Die zweite leitet Datenschutz-Label-Schemata und Datenfluss-Constraint-Regeln aus rechtlichen Anforderungen mithilfe von GPT-5.5 ab. Beide nutzen Chain-of-Thought-Prompting, schemavalidierte Ausgaben und strukturiertes Retry-Feedback, um strukturelle Fehler vor der Weitergabe einzudämmen. Wir evaluierten beide Pipelines durch zweistufige Blind-Nutzerstudien: die erste mit 20 Erstellern und 12 Bewertern, die zweite mit 10 Erstellern und 6 Bewertern. LLM-abgeleitete Artefakte erhielten über alle Artefakttypen hinweg tendenziell niedrigere Durchschnittsbewertungen als von Menschen erstellte auf einer Skala von 1 bis 5. Im Vergleich LLM vs. Mensch lagen Vollständigkeit und Korrektheit der DFDs im Durchschnitt bei 2,5 vs. 4,0 und 2,9 vs. 3,6, bei Datenschutz-Labels bei 2,8 vs. 3,7 und 2,7 vs. 3,5 sowie bei Constraint-Mengen bei 2,3 vs. 3,2 und 1,8 vs. 3,0. Die Fehlermarkierung identifizierte fehlende Elemente und Halluzinationen als dominierende Fehlertypen in LLM-abgeleiteten DFD sowie vage Labels in LLM-abgeleiteten Datenschutz-Label-Mengen. Eines von vier LLM-abgeleiteten DFDs erreichte mit den menschlichen Gruppenmittelwerten vergleichbare Bewertungen, was zeigt, dass die Pipeline in einzelnen Durchläufen nahezu menschliche Qualität erzielen kann. Vier gemeinsame methodische Faktoren verhindern, dass diese Bewertungen als mehr als richtungsweisend behandelt werden. Dies sind Krippendorffs α unterhalb der Zuverlässigkeitsschwelle über alle Bewertungsdimensionen, kleine Stichprobengrößen, ein Ersteller-Bewerter-Verankerungseffekt und asymmetrische Bewertungszahlen. Das Wiederauftreten unzuverlässiger Bewerterübereinstimmung bei beiden Aufgaben legt nahe, dass eine konsistente Qualitätsbewertung für Artefakttypen, die mehrere gültige Lösungen zulassen, schwierig zu erreichen ist.

Contents

List of Figures

List of Tables

List of Abbreviations

AI Artificial Intelligence

DFA Data Flow Analysis

DFD Data Flow Diagram

DPA Data Processing Agreement

DPbD Data Protection by Design and by Default

DPIA Data Protection Impact Assessment

DSL Domain-Specific Language

FR Functional Requirement

GDPR General Data Protection Regulation

GQM Goal Question Metric

LLM Large Language Model

MECE Mutually Exclusive and Collectively Exhaustive

ML Machine Learning

NLP Natural Language Processing

OCL Object Constraint Language

PbD Privacy by Design

PIA Privacy Impact Assessment

RAG Retrieval-Augmented Generation

RE Requirements Engineering

RQ Research Question

SDM Standard Data Protection Model

SRS Software Requirements Specification

SVO Subject-Verb-Object

UML Unified Modeling Language

1. Introduction

Software systems handle sensitive personal data across virtually every domain, from healthcare and finance to social media and critical infrastructure. As these systems grow in complexity, ensuring individual privacy has become a central concern in software engineering, as violations can cause significant harm including identity theft, financial loss, and reputational damage [17]. The European Union enacted regulations like the General Data Protection Regulation (GDPR) to safeguard personal data and privacy, but compliance remains a challenge [1, 53]. Breaux et al. [25] characterize this as a widespread, persistent, structural issue in the software industry. They identify three root causes: a cultural gap between legal and technical practice, an expertise gap, and competing priorities between legal requirements and business objectives. From a software engineering perspective, addressing such violations early in development, ideally at the design stage, avoids costly retrofitting later on [1, 97].

Data Flow Diagrams (DFDs) are diagrams that model how data moves through a software system, making them well-suited for identifying where personal data is processed and where potential privacy violations may occur [22, 92]. The Data Flow Analysis (DFA) uses these diagrams to systematically check a system design for security and privacy issues at design time [22, 92]. The xDECAF framework [22] builds on this idea by enabling automated information security violation detection, including privacy violations. xDECAF takes privacy specifications derived from applicable legal requirements as input. These specifications are expressed as annotations on the nodes and data flows of a DFD and in a machine-interpretable form. Given these inputs, xDECAF automatically checks whether a system design complies with those requirements. However, both inputs currently require substantial manual effort. Constructing a DFD from a functional system description is a labour-intensive and error-prone task requiring significant software design expertise [15]. Translating privacy requirements into machine-interpretable privacy specifications demands legal knowledge and careful interpretation of complex regulations [80]. This manual bottleneck limits the practical adoption of DFA-based privacy analysis, particularly in agile development processes where integrating time-consuming manual steps is impractical [97].

Recent advances in Large Language Models (LLMs) offer a candidate solution to this bottleneck. LLMs have demonstrated effectiveness across software engineering tasks such as code generation, documentation, and requirements analysis [14, 111], and more recently in generating design artifacts such as Unified Modeling Language (UML) diagrams [44, 105]. In the domain of privacy and Requirements Engineering (RE), LLMs have been applied to tasks such as GDPR compliance checking and deriving software-level requirements from legal texts [4]. They outperform earlier classification-based approaches by leveraging

their ability to reason across the language gap between complex legal text and technical software specifications [39]. However, this body of work operates at the level of textual requirements and compliance analysis, and does not extend to the derivation of structured, machine-interpretable privacy specifications required for DFA-based privacy analysis. While Herwanto et al. [55] have shown that LLMs can generate DFDs from user stories, their approach is limited to this input format and does not produce the privacy-annotated, xDECAF-conformant output required for automated privacy violation detection. Similarly, no prior work has automated the derivation of privacy specifications in the structured form required by DFA frameworks like xDECAF [22]. This leaves a gap: no existing work automates the derivation of the two inputs required by DFA-based analysis, namely structural DFDs and privacy specifications.

1.1. Research Objective

This thesis addresses the identified gap by designing a conceptual pipeline for DFA-based privacy analysis and implementing its first two steps. The pipeline targets the derivation from functional requirements and legal privacy requirements to the inputs required by xDECAF. Within this pipeline, this thesis realizes two of the three required automation steps: the automated derivation of structural DFDs from functional natural language descriptions (RQ1), and the automated translation of software-level privacy requirements into machine-interpretable privacy specifications (RQ2). These requirements are assumed to have been derived from legal sources such as the GDPR in a prerequisite step, with their derivation not being part of the scope of this thesis. The third step consists of automating the annotation of the derived DFDs with the derived privacy specifications to construct the final annotated DFD. This thesis identifies it as future work. Completing this step would close the path to fully automated end-to-end DFA-based privacy analysis. The following two research questions formalize the two steps addressed in this thesis:

RQ1. How does the quality of LLM-derived DFDs compare to that of human expert-created ones?

RQ2. How does the quality of LLM-derived privacy specifications compare to that of human expert-created ones?

To answer both research questions, this thesis conducted two user studies to evaluate the quality of the LLM-derived outputs.

In doing so, this thesis makes the following contributions as part of an overall conceptual pipeline for DFA-based privacy analysis, targeting the automated derivation of the two inputs required by xDECAF:

C1. An LLM-based pipeline for automatically deriving DFDs from functional requirements, realizing the first step of the pipeline and evaluated against a human expert baseline (RQ1).

C2. An LLM-based pipeline for automatically deriving privacy specifications from software-level privacy requirements, realizing the second step of the pipeline and evaluated against a human expert baseline (RQ2).

1.2. Outline

The remainder of this thesis is structured as follows. Chapter 2 introduces the necessary background on DFDs, LLMs, and the GDPR, and Chapter 3 situates the work within the broader research landscape. To provide a concrete basis for all subsequent chapters, Chapter 4 introduces the running example used throughout the thesis. Building on these foundations, Chapter 5 presents the overall conceptual design of the proposed approach and its integration into the software design process. The two research questions are then addressed in Chapters 6 and 7 respectively, each presenting the conceptual design, implementation, and evaluation for the corresponding approach. Chapter 8 discusses cross-cutting findings across both research questions, including limitations, broader implications, and directions for future work. Chapter 9 concludes the thesis.

2. Foundations

This chapter introduces the conceptual foundations underlying the design and evaluation of both Research Questions (RQs). It covers Requirements Engineering (RE) alongside privacy and data protection as the domain context (Sections 2.1 and 2.2). It then introduces annotated Data Flow Diagrams (DFDs) and constraints as the target output artifacts of RQ1 and RQ2, and their role in compliance engineering (Sections 2.3 and 2.4). Finally, it provides an overview of Large Language Models (LLMs) as the underlying automation technique (Section 2.5), and the empirical evaluation methods applied in the user studies (Section 2.6).

2.1. Requirements Engineering

In software engineering, a *requirement* captures a property or condition that a system must satisfy to address the needs and expectations of its stakeholders [59]. *Stakeholders* are individuals or organizations with a legitimate interest in or claim on a system [59]. Requirements may be expressed in various forms, such as natural language descriptions, user stories, use cases, or formal specifications [59]. *Requirements engineering* refers to the process of defining and documenting stakeholder requirements, comprising activities such as elicitation of stakeholders' needs and specification as a basis for all other system development activities [81]. Requirements can be categorised into *functional requirements*, which specify the observable behavior of a system, and *non-functional requirements*, which constrain system-wide properties, and "often apply to the system as a whole rather than individual features or services" [98]. Glinz [50] refines this further into four types. These are functional requirements, which specify a specific functionality; *performance requirements*, which pertain to operational efficiency; *specific quality requirements*, which cover quality properties such as reliability or usability; and *constraints*, which bound the solution space for design and implementation. In this view, non-functional requirements pertain to the quality of the system or constrain its design, rather than prescribing its behavior [50]. Robertson and Robertson [87] further identify *privacy requirements* as a subtype of non-functional requirements within the broader category of security requirements. They are distinguished by their legal character and cover obligations imposed by data protection regulations. In the context of this thesis, functional requirements serve as the primary input to RQ1. Privacy requirements, derived from legal sources and provided as pre-structured input, form the basis of RQ2.

2.2. Privacy and Data Protection

Privacy can be described as the right of individuals to control their personal information, to edit, manage, and delete it, and to decide for themselves when and how their personal data is shared with others [106]. *Personal data* refers to “any information relating to an identified or identifiable natural person” (Art. 4 General Data Protection Regulation (GDPR)) [40]. An *identifiable natural person* is defined as ”one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person” (Art. 4 GDPR). Data protection laws such as the GDPR regulate the processing of personal data, imposing various obligations on organizations that collect, store, and process personal data. These obligations include obtaining consent, providing transparency, and implementing appropriate security measures. They further cover ensuring data subject rights (access, rectification, and deletion), data minimization, and purpose limitation.

Privacy by Design (PbD) addresses these regulatory obligations at the software design level [27]. It advocates for embedding privacy considerations into the system design from the outset, rather than treating them as an afterthought. Its seven foundational principles mandate proactive rather than reactive privacy measures and privacy as the system default. They further require full system lifecycle protection and transparency towards users. The GDPR also emphasizes this paradigm in Art. 25 by mandating Data Protection by Design and by Default (DPbD), as a legal requirement for all organizations processing personal data. The Standard Data Protection Model (SDM) [7] developed by the German Data Protection Supervisory Authorities of the Federation and the states operationalizes these requirements in practice. It defines a structured framework of concepts, measures, and relations for modeling data protection requirements across the design, implementation, and operation of software systems. It systematically captures data protection requirements as *protection goals*, which specific measures then operationalize, and the interrelations between these concepts, such as conflicts and dependencies [7]. Furthermore, this framework enables a structured Data Protection Impact Assessment (DPIA), a systematic evaluation of risks and impacts of data processing activities on individuals’ privacy [21]. It serves to identify potential compliance issues and derive appropriate mitigation measures prior to deployment. In this thesis, the structured privacy requirements provided by the SDM serve as the evaluation input for RQ2. They provide structured legal obligations against which the LLM-derived labels and constraints are assessed.

2.3. Data Flow Analysis

DFDs are a graphical representation of the flow of data through a system from sources to sinks, through functional processes, and data stores [34]. They consist of four main components: *data flows* representing the movement of data, *processes* representing the transformation of data, *data stores* representing persistent data repositories, and *actors*

representing sources and sinks of data [34]. DeMarco's [34] notation captures these components, but lacks formal semantics. This poses challenges for automated analysis, particularly regarding exploration of multiple data flow paths, coverage of multiple confidentiality mechanisms, and support for user-defined confidentiality analyses [92].

Seifermann et al. [92] address these challenges by extending the DFD meta-model. They group actors, processes, and data stores as *nodes*, as illustrated in Figure 2.1, where actors and data stores appear as rectangles, and processes as rounded rectangles. The extended meta-model also introduces *pins* as required and provided interfaces, shown as I/O ports on node borders in Figure 2.1. Through these pins, data processing and behavioral properties of nodes and data flows can be formally described using *labels* [92]. Furthermore, both node and data flow properties can be captured as labels: *node labels* are defined on nodes themselves, while *data labels* are defined on data flows. These labels can capture various properties relevant for security and privacy analysis, such as confidentiality levels, access control policies, or data sensitivity classifications [92]. Additionally, the extended meta-model supports the propagation of labels across data flows, enabling the analysis of how data properties evolve as they move through the system. This enables the detection of potential violations of security and privacy policies [92]. Figure 2.1 illustrates this distinction in different colors, where *node labels* are defined on the nodes themselves (e.g., *Location: EU* on the User actor), while *data labels* are defined on the data flows (e.g., *Confidentiality: personal* on the data flow from the User to the buy process). *Propagated labels* are shown as dotted annotations on the data flows, representing labels that propagate from their point of assignment through subsequent data flows. For example, the *Confidentiality: personal* label propagates from the User's data flow through to the Database node. These labels support automated analysis of DFDs, such as detecting violations of access control and information flow policies [92]. This is done by comparing the labels of nodes and data flows against the defined policies and constraints.

Seifermann et al. [92] and Boltz et al. [22] introduce a Domain-Specific Language (DSL) for specifying constraints over DFDs. Each constraint can be evaluated against an annotated DFD instance. Figure 2.2 shows such a constraint, stating that data labelled as *Confidentiality: personal* must never flow to a node labelled *Location: nonEU*. Applied to Figure 2.1, this constraint is violated: the *Confidentiality: personal* label assigned to the User's data flow propagates to the Database node, which carries *Location: nonEU*, indicating that personal data flows outside the EU.

Seifermann et al. [92] and Boltz et al. [22] further introduce *labelTypes* into this extended DFD meta-model, groupings of labels that capture specific characteristics of nodes and data flows. This addition gives rise to the *labelType: labelValue* annotation pairs visible in Figure 2.1, such as *Location: EU* or *Confidentiality: personal*. These pairs enable software engineers to annotate DFD elements with privacy-relevant properties. Together, the labelTypes, labels, and constraints expressed via the DSL form a complete, machine-processable specification of a system's data flows and security policies [22, 92]. Both figures follow the notation of xDECAF, Boltz et al.'s [22] DFA for the manual creation and evaluation of annotated DFDs with constraints. xDECAF serves as the target output format for both RQ1 and RQ2.

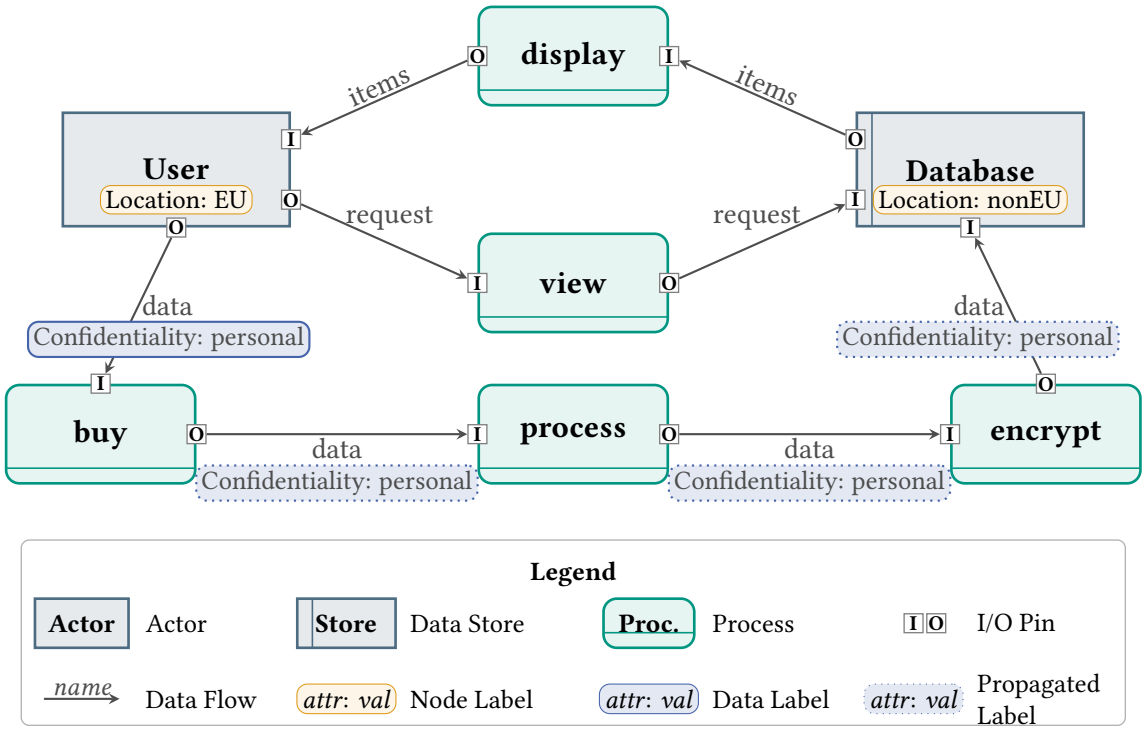


Figure 2.1.: Annotated DFD showing actors, data stores, processes, data flows, and I/O pins.

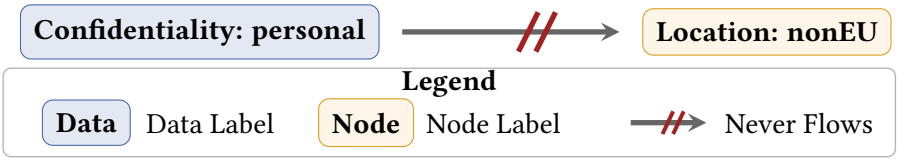


Figure 2.2.: Visual representation of a Data Flow Analysis (DFA) constraint: *Personal* data must never flow *outside of the EU*.

2.4. Compliance Analysis

Agkhibe et al. [9] identify four canonical tasks in legal and regulatory compliance engineering: *modeling, checking, analysis, and enactment. Modeling* involves sourcing and structuring requirements from legal texts. Sangaroonsilp et al. [88] illustrate this with a taxonomy of 71 privacy requirements across 7 categories. The taxonomy draws from the GDPR, ISO/IEC 29100 standards [58], and other data protection regulations. The authors apply a Goal-based Requirements Analysis method to extract these requirements from legal sources. Each requirement is formulated as an action verb–object–target result triple, for example, *OBTAIN the opt-in consent for the processing of personal data for specific purposes (R35)*. These requirements are then categorized based on similarities in their action verb, object, and target result, resulting in a structured taxonomy of privacy requirements. The final categories are *User participation*, *Notice*, *User desirability*, *Data processing*, *Breach*, *Complaint/Request* and *Security*. *Checking* involves validating formalized representations against compliance requirements. xDECAF [22] demonstrates this by performing auto-

mated violation detection on annotated DFDs. *Analysis* involves deriving insights on the compliance state. Boltz et al. [23] illustrate this by transforming annotated DFDs into a legal viewpoint, making compliance violations interpretable for both software engineers and legal experts. *Enactment* involves responding to violations to reestablish compliance, for instance by generating mitigation measures or alternative design options [23].

This thesis addresses the modeling task by automating the derivation of labels and constraints from legal requirements for DFD annotation. In doing so, it enables the subsequent checking and analysis tasks to be performed on the resulting annotated DFDs. The Sangaroonsilp taxonomy serves as the structured legal input for this derivation in RQ2.

2.5. Large Language Models

LLMs are a class of deep learning models trained on vast amounts of text data [32]. Through this training, they learn the statistical properties of human language to generate coherent and contextually relevant text. They are typically based on the Transformer architecture, which uses self-attention mechanisms to capture long-range dependencies in text [103], outperforming earlier recurrent architectures on translation benchmarks. Leveraging these capabilities, LLMs find application in various language tasks, such as text generation, summarization, translation, and question answering [32]. A key property is *few-shot prompting*: by providing a small number of examples in the input prompt, LLMs can perform a wide range of tasks without task-specific fine-tuning [26]. Effectively harnessing this property, however, requires careful design of the input prompts [111]. *Prompt engineering* refers to the systematic design and optimization of input prompts to elicit desired outputs from LLMs [104]. To support principled prompt design, Moundas et al. [73] identify five patterns, covering structured information extraction, adaptive attribute extraction, pattern matching, constraint-based output generation, and keyword-triggered extraction.

LLMs are also subject to *hallucinations*: generating plausible but factually incorrect or nonsensical responses [111]. Further limitations include sensitivity to prompt wording, factual inaccuracies, reasoning failures, and outdated knowledge. In this thesis, LLMs serve as the automation technique underlying both RQ1 and RQ2. Prompt engineering, an active area of software engineering research [12], forms the core design methodology of both pipelines.

2.6. Empirical Methods in Software Engineering

The Goal Question Metric (GQM) approach structures empirical evaluation across three levels [18]. It begins by defining goals at a conceptual level, operationalizes these as measurable questions, and identifies concrete metrics for each question. This three-level structure ensures that measurement choices remain grounded in explicit research objectives. In this thesis, the GQM approach guides the evaluation design for RQ1 and RQ2, ensuring

that each evaluation criterion traces back to an explicit research objective and that validity threats are addressed systematically.

Wohlin et al. [109] categorize the validity of empirical evaluations into four dimensions, each of which poses distinct threats that must be addressed in the study design [61].

Conclusion validity addresses whether a relationship between the evaluated artifact and the observed outcomes can be reliably established. Relevant threats include statistical power, measurement reliability, and random variation in the experimental setting. To address measurement reliability specifically, this thesis uses Krippendorff's alpha [63] to quantify inter-rater agreement where multiple evaluators assess the same artifacts. It provides a reliability coefficient applicable to ordinal, nominal, and numerical data. A value of 1 indicates perfect agreement; values above 0.8 are considered satisfactory [64]. Values between 0.67 and 0.8 permit tentative conclusions; values below 0.67 indicate unreliable agreement.

Internal validity refers to whether outcomes reflect the artifact itself rather than confounding factors such as task framing or evaluator expertise. Evaluations can employ a blind design to mitigate origin bias [95]. In such a design, evaluators do not know the origin of the artifacts they assess and thus cannot apply different standards based on how or by whom they were produced.

Construct validity examines whether the chosen metrics and evaluation methods accurately reflect the quality of LLM-derived artifacts. Quality dimensions such as completeness and correctness involve inherently subjective judgement. To standardize such assessments, five-point Likert scales [68] can be applied across these dimensions, producing ordinal data that can be analyzed statistically.

Finally, *external validity* captures the degree to which findings generalize beyond the specific study participants and artifacts. The preceding dimensions can be actively mitigated through study design. However, external validity is inherently constrained by the scope of any single evaluation, and this thesis acknowledges it as a limiting factor.

3. Related Work

This chapter situates the thesis within the broader research landscape in two key areas. The first covers the automated generation of Data Flow Diagrams (DFDs) and related artifacts (Sections 3.1, 3.2, and 3.3). The second covers the extraction and operationalization of legal compliance requirements for privacy in software systems (Sections 3.4, 3.5, 3.6, and 3.7).

3.1. Automated Data Flow Diagram Generation

With MONDRIAN, Protsko et al. [82] present one of the earliest approaches to automated DFD generation from requirements. Their motivation is the time-consuming nature of manual DFD construction and the increase of errors with growing system complexity. While the approach is constrained by its reliance on the SPSL/SPSA input format, it demonstrates that automated DFD generation from requirements has been a recognized challenge for decades. Nearly four decades later, Cheema et al. [30] investigate a semi-automated approach for DFD generation from natural language requirements. Their approach combines web scraping and Natural Language Processing (NLP) techniques to construct syntactically correct DFDs, with a focus on the extraction of DFD components. However, its reliance on pre-scraped vocabularies limits its ability to generalize to other requirements. More recently, Herwanto et al. [55] explore the potential of Large Language Models (LLMs) in automating DFD generation from user stories. Their study examines different LLMs, the effect of pre-grouping input user stories, and few-shot prompting, with outputs produced in CSV format for visualization in draw.io [37]. Three software engineering experts rate the generated DFDs on a 5-point Likert scale. The results indicate that GPT-4 outperforms other models and that completeness consistently exceeds correctness. This work demonstrates the potential of LLMs for DFD generation, while also surfacing known challenges such as hallucinations and a limited ability to capture semantic relationships within requirements. However, the approach is constrained to user stories as input and relies on active human interaction through a chat interface. Furthermore, the approach is limited to a single-prompt generation step, and its evaluation to only three raters, leaving open questions regarding the generalizability of their findings. Thus, this approach demonstrates the potential of LLMs for DFD generation, but leaves open how to integrate LLMs into a more automated pipeline supporting other types of natural language functional requirements.

3.2. LLMs for Software Engineering Artifact Generation

Transforming natural language requirements into structured software engineering artifacts has been a long-standing challenge. Earlier approaches relied on hand-crafted rules and classical NLP and Machine Learning (ML) techniques, which struggled with the ambiguity and variability of natural language [6]. With the advent of LLMs, researchers have applied them to software engineering artifact generation across tasks such as requirements modeling, diagram generation, and formal specification. These approaches show particular promise for routine artifact generation tasks, while tasks requiring domain-specific knowledge remain challenging [54]. Further challenges include hallucinations, incomplete outputs, and difficulty with complex structured requirements [54]. Belzner et al. [19] identify large context handling as an additional challenge, proposing task decomposition as a mitigation strategy alongside careful prompt engineering to introduce domain-specific knowledge. They also advocate for evaluation against ground truth through human expert assessment. Despite this broader progress on LLM-based software engineering artifact generation, DFD generation remains a comparatively underexplored area.

3.3. DFD-based privacy tools

To incorporate legal considerations into software design, Sion et al. [97] propose building a dedicated architectural viewpoint for data protection. Their meta-model captures the aspects of the General Data Protection Regulation (GDPR) [40] relevant for software design, and maps these aspects to DFD components and relationships. However, the approach is limited by its reliance on manual derivation of the data protection view for each system, which is time-consuming. Another approach by Alshareef et al. [11] focuses on the purpose limitation principle of the GDPR. Their tool integrates with draw.io [37], formerly known as diagrams.net. Through a domain-specific language, it allows specifying privacy signatures and purpose-label constraints on DFD elements, enabling checking and inference of these purpose labels. As their approach is focused on purpose limitation, it does not address other important privacy aspects such as data minimization, storage limitation, or security measures, which are also crucial for GDPR compliance. Furthermore, their approach relies on manual specification of the purpose labels, which can be time-consuming and error-prone, especially for large and complex systems.

Mollaeefar et al. [72] address this manual effort with PILLAR, a tool for LINDDUN-based privacy threat modeling using LLMs. LINDDUN [35] is a structured privacy threat modeling framework that maps specific privacy threats to DFD components and guides practitioners towards identifying privacy-enhancing solutions. PILLAR automates this process, by allowing the generation of DFDs from natural language descriptions or visual inputs, while also allowing editing and CSV export for later visualization. Furthermore, it incorporates the LINDDUN analysis process through multi-agent collaboration. Different LLM instances play distinct contributor roles, each bringing different perspectives and expertise within a virtual workspace. This automation results in an assessment of privacy threats in the

system, including their impacts, and control measures, which can be used for documentation and further mitigation efforts. Savaliya et al. [90] propose a complementary approach to address AI-specific privacy threats beyond the LINDDUN framework. They extend PILLAR's knowledge base with a model-centric privacy attack knowledge base, and apply a Retrieval-Augmented Generation (RAG) approach with LLMs for automated threat identification from DFD metadata. While PILLAR and its extension automate DFD-based privacy threat analysis, neither accounts for specific legal requirements from regulatory texts such as the GDPR. This distinction matters for ensuring formal compliance with specific GDPR provisions, as opposed to identifying threats based on general privacy principles. Furthermore, neither approach evaluates the quality of the generated DFDs, leaving open how inaccuracies in the DFD generation step may affect the downstream privacy analysis. Thus, existing approaches either rely on manual derivation of legal requirements into DFD annotations, or automate privacy threat analysis without grounding in regulatory texts and without evaluating the quality of the generated DFDs. This leaves open how to integrate legal requirements directly into the DFD generation and analysis process, and how to evaluate the quality of generated DFDs, both of which this thesis addresses.

3.4. Legal Requirements Extraction from Regulatory Texts

The extraction of legal requirements from regulatory texts has traditionally been a highly manual and time-consuming process, requiring translation of legal language into reusable technical requirements [62, 86]. A strong limitation is the effort required to extend such derivations to specific use cases or additional legal texts. To automate this process, researchers have explored NLP techniques, with advancements in overcoming ambiguity, generalization, and incompleteness challenges [57].

With LLMs showing promise in further advancing automation by increasing precision and comprehensiveness [110], researchers have explored a range of approaches. Some approaches target specific extraction tasks: automated question answering tools to extract compliance-relevant information [3], and targeted extraction of specific privacy rights such as access and portability [4]. Others pursue comprehensive representations of legal requirements, such as machine-analysable compliance criteria [2] and formal logical encodings in the form of deontic rules [56]. Abualhaija et al. [2] present a particularly broad approach, covering the full pipeline from regulatory text to structured, machine-analysable legal representations. Comparative studies confirm that LLM-based solutions significantly outperform traditional ML baselines and rule-based approaches [17]. Prompt-based approaches further outperform fine-tuned classifiers in generalizing across regulations [17, 39]. Open challenges remain around dataset quality and diversity, complexity of engineering requirements, and scalability of LLM applications [74]. Thus, these approaches collectively demonstrate the feasibility of automating legal requirement extraction from regulatory texts. However, their outputs either remain focused on specific privacy rights or target document level compliance analysis, rather than producing formal, architectural-level representations such as DFD privacy labels and constraints.

3.5. Model-based Legal Compliance Analysis

Early work on automating legal compliance analysis focused on mapping legal requirements to goal-oriented models, with the aim of alignment between legal compliance and organizational goals [47, 48, 49]. This reliance on manual mapping of legal requirements to organizational goals limits its scalability. Guerriero et al. [51] propose a framework for specifying, enforcing, and checking privacy policies in large-scale data-intensive applications. Compliance is enforced by rewriting data flows to include privacy-enforcing operators, and then verifying the resulting data flows against the specified privacy policies. While their approach integrates privacy policies directly with data flows, it operates at runtime on deployed applications and requires manual policy specification. This limits its applicability in early design stages.

Subsequent approaches have focused specifically on the GDPR and its operationalization for software design and analysis, with a focus on model-based approaches. Ahmadian et al. [5] support Privacy Impact Assessment (PIA) through model-based integration within Unified Modeling Language (UML) models. They annotate system designs with privacy-related information to calculate the impact of privacy risk and suggest mitigation measures. Torre et al. [100, 101] propose a comprehensive GDPR representation using UML, including Object Constraint Language (OCL) constraints for checking conformance of system instances against the GDPR model. The limitation of these approaches is their reliance on manual derivation of GDPR requirements into model-based representations. More recently, Lamari et al. [66] propose a more comprehensive model-driven framework including tool support for integrating GDPR compliance into software applications during early development stages. Their approach includes a UML class model of the GDPR and a Domain-Specific Language (DSL) for instantiating the metamodel in the context of a specific application. From these, the approach generates user stories and privacy enforcement database schemas to support the implementation of GDPR-compliant applications. These artifacts serve as implementation blueprints for specific applications, enabling the integration of GDPR-compliant design into software development processes. Thus, these approaches demonstrate the value of structured, model-based incorporation of legal requirements into software design. However, they rely on manual derivation of legal representations, leaving open how to automate this derivation and integrate the resulting representations into software design artifacts such as DFDs.

3.6. Automated GDPR Compliance Checking

Cejas et al. [29] propose DERECHA, a rule-based NLP system for checking the compliance of Data Processing Agreements (DPAs) with the GDPR. The system relies on a manually specified requirements schema and a set of rules for checking the presence of required elements in the DPA text. Its evaluation shows precision of 89.1% and recall of 82.4%. Similarly, Amaral et al. [13] propose Dικαιο, combining NLP and ML techniques in a multi-stage pipeline for the same DPA compliance checking task. Their evaluation shows the

ML-based approach performs on par with the rule-based DERECHA, achieving an overall precision and recall of around 83%. Shen et al. [93] further explore this problem space, utilizing the BERT text classification model [36] in a two-stage pipeline for checking GDPR compliance within DPA texts. Their approach achieves an overall accuracy close to 86% and recall exceeding 88%, though it relies on a manually annotated privacy policy dataset for training and evaluation. Cejas et al. [28] propose COMPAI, a tool for checking custom privacy policies for GDPR compliance. Users upload a privacy policy which, in combination with a questionnaire, is evaluated in a multi-stage process: the policy is transformed into a semantic representation, information types are identified, and these are checked against a conceptual model of GDPR-required information types. Okonicha et al. [75] further explore the use of different NLP techniques and optimization strategies for reducing manual effort and improving the compliance process for privacy policies. Their results show that LLM-based approaches can significantly improve the contextual understanding and classification accuracy while facing challenges such as costs and interpretability of model decisions.

Building on these LLM advancements, Das et al. [33] propose a multi-agent RAG system for compliance checking against Software Requirements Specification (SRS). Each agent is responsible for a specific aspect of the process: pre-processing via semantic chunking, compliance checking via a combination of persona prompting and chain-of-thought prompting, and compliance summary generation. They also acknowledge the role of human experts in the loop for validating outputs and providing feedback for improving the outputs. Limitations of this approach include LLM bias and limited generalizability to different software artifacts. These approaches demonstrate the feasibility of automating GDPR compliance checking for specific documents such as DPAs and privacy policies. However, they do not address the integration of legal requirements into software design artifacts such as DFDs, which is crucial for downstream automated design-time privacy analysis.

3.7. GDPR Requirements Operationalization

Researchers have explored various approaches to operationalizing GDPR requirements within software design and development processes. These range from structured process support to formal language encodings and LLM-assisted generation of compliance-relevant artifacts. Ayala-Rivera et al. [16] propose GuideMe, a six-step systematic approach for operationalizing GDPR principles in software systems. GuideMe supports practitioners in eliciting solution requirements that link GDPR data protection obligations with the privacy controls needed to fulfil them. While it provides a structured process for operationalizing GDPR principles in software design, it relies heavily on human interaction. Nor does it help integrate legal requirements directly into software artifacts. Merigoux et al. [71] propose Catala, a domain-specific language for encoding legal texts in a way that is both human-readable and machine-executable. This allows legal and software experts to work directly from a shared representation of the legal text. Catala enables direct one-to-one encoding of legal requirements into code, but relies on manual encoding and maintenance of those requirements. Sangaroonsilp et al. [89] propose PPGen, an interactive tool for generating

GDPR-compliant privacy policies, which support software developers in writing complete privacy policies based on questionnaire inputs. Its output is limited to privacy policies, offering no connection to system design artifacts. More recently, Wijesundara et al. [107] propose GDPRShield, a tool that takes user story inputs and produces GDPR-tailored privacy descriptions using LLMs. It supports developers in identifying and resolving GDPR-relevant ambiguities in user stories and privacy descriptions. Thus, these approaches demonstrate the potential of operationalizing GDPR requirements within software development processes. However, they remain focused on textual artifacts such as privacy policies and user stories, leaving open how to reflect legal requirements directly in software design artifacts such as DFDs.

Existing research addresses LLM-based generation of software artifacts and extraction of legal requirements separately. A significant gap remains in combining these capabilities into a DFD-based privacy analysis pipeline. Specifically, it remains open how to automatically derive annotated DFDs (RQ1) and privacy constraints (RQ2) directly from software requirements and legal requirements.

4. Running Example

This chapter introduces the *Travel Planner*, a case study originally proposed by Katkalov et al. [60], which serves as the running example throughout this thesis. It provides a practical scenario to demonstrate the challenges motivating our Research Questions (RQs) and to showcase the application of our proposed methods. Section 4.1 describes the scenario and its functional requirements, and Section 4.2 outlines the applicable legal and privacy context under the General Data Protection Regulation (GDPR). Together, they give rise to the problem illustrated in Section 4.3. Section 4.4 then maps the RQs to the scenario, establishing the concrete anchor points that subsequent chapters will return to.

4.1. Scenario Description

The Travel Planner is a distributed mobile application for discovering, selecting, and booking flights, running on the user's smartphone. It connects four parties: the *Customer*, a *Travel Agency* web service, an *Airline* web service, and a *Credit Card Center* app running locally on the smartphone alongside the Travel Planner. The Functional Requirements (FRs) of the Travel Planner capture the key interactions and data flows between these parties, namely flight search criteria, booking initiation, secure payment processing, and referral commission. Table 4.1 summarizes these requirements in full. While they form the basis for deriving the system's data flows, designing for a privacy-compliant system additionally requires identifying the applicable privacy regulations.

4.2. Legal & Privacy Context

For illustrative purposes, we assume the Travel Planner operates within the European Union, making it subject to the General Data Protection Regulation (GDPR) [40]. The GDPR imposes strict requirements on the handling of personal data, including obligations regarding consent, data processing, and security. Sangaroonsilp et al. [88] translate such legal requirements into a structured taxonomy of privacy requirements for software applications. While the complete set of requirements is used when applying our methods in subsequent chapters, Table 4.2 provides an overview of a representative subset along with their application to the Travel Planner scenario. For ease of reference throughout this thesis, we label the selected requirements from Sangaroonsilp et al. [88] as **LR1**–**LR4**. With both sets of requirements established, the following section illustrates the tensions that arise when they must be reconciled in a concrete design decision.

FR-ID	Description
FR1	**Flight & Discovery** The customer sends flight search criteria (e.g., destination, dates) to the Travel Agency (TA) web service via the Travel Planner (TP) app. The Travel Agency processes the request and returns a list of suitable flight offers to the customer.
FR2	**Booking Initiation** After selecting a specific offer, the customer initiates the booking process. The Travel Planner (TP) app coordinates communication between the customer's device and the Airline (A) web service to prepare the transaction.
FR3	**Secure Payment Processing** To finalize the booking, the customer must provide payment. The Credit Card Center (CCC) app, located locally on the customer's smartphone, retrieves the stored credit card data. Upon explicit confirmation by the customer, the CCC app sends the sensitive payment details directly to the Airline for processing.
FR4	**Referral Commission** Once the Airline successfully processes the flight booking and payment, it automatically triggers a commission payment to the Travel Agency as a fee for the initial referral.

Table 4.1.: Functional requirements of the Travel Planner from Katkalov et al. [60]

LR-ID	Description
LR1	**Consent** *OBTAIN the opt-in consent for the processing of personal data for specific purposes (R35)* In the Travel Planner, explicit customer confirmation must be obtained before payment details are transmitted to the Airline **FR3**
LR2	**Data Processing: Storage** *STORE the personal data as necessary for specific purposes (R43)* The Travel Agency and Airline may store personal data on their systems related to the services provided such as booking details, but are not permitted to retain it beyond the scope of the specific purposes
LR3	**Security: Authorized Access** *ALLOW the authorised stakeholders to access personal data as instructed by a controller (R56)* Neither the Travel Agency nor the Airline may directly access the payment details stored in the Credit Card Center app **FR3**, as the Credit Card Center is a local component on the customer's device, accessible only through explicit customer action.
LR4	**Sensitive Data: Processing Justification** *PROVIDE the data subjects the sufficient explanation for the need to process sensitive personal data (R28)* The Travel Planner must provide the customer with a sufficient explanation for why their payment details are required for processing during the payment flow **FR3**, as payment information constitutes sensitive personal data requiring elevated justification.

Table 4.2.: Selected legal requirements for the Travel Planner from Sangaroonsilp et al. [88]

4.3. Illustrated Problem Statement

Consider the situation a software architect faces when designing the Travel Planner. On the one hand, **FR1**–**FR4** must be translated into a Data Flow Diagram (DFD) that correctly captures all parties, data flows, and interactions. **FR3**, for instance, describes a multi-step payment process involving a local Credit Card Center app, explicit customer confirmation, and a direct data flow to the Airline. Yet none of these are explicitly labeled as distinct entities or flows in the natural language text. Correctly identifying all entities, their roles, and the precise flows requires careful interpretation of the natural language, as it is prone to ambiguity [6]. On the other hand, the resulting design must comply with **LR1**–**LR4**. **LR1** requires that the Travel Planner obtains explicit customer consent before transmitting payment data to the Airline (**FR3**). **LR3** restricts direct access to the Credit Card Center by the Travel Agency and Airline. **LR4** demands appropriate technical protection for all personal data flows (**FR2**, **FR3**). None of these constraints are visible in the functional requirements alone. Instead, they must be derived from legal requirements and then validated against the DFD. In the Travel Planner alone, constructing the DFD requires resolving the implicit entity and flow structure across **FR1**–**FR4**. Validating the result against **LR1**–**LR4** then requires examining each requirement pair individually for conflicts. Both steps illustrate why automated support is needed, which our RQs directly address.

4.4. Mapping

Table 4.3 concretely illustrates what each RQ means in the context of the Travel Planner, though the scenario itself is illustrative rather than exhaustive. In subsequent chapters, we evaluate our methods against a different set of functional requirements and the complete privacy requirement set from Sangaroonsilp et al. [88]. The Travel Planner serves as a representative example within this broader scope.

ID	Description & Rationale
RQ1	**How does the quality of LLM-derived DFDs compare to that of human expert-created ones?** Given **FR1**-**FR4** in Table 4.1, the expected DFD involves four interacting parties: Customer, Travel Agency, Airline, and Credit Card Center. These parties exchange data flows of varying type and sensitivity such as flight search criteria (FR1,low sensitivity) and payment information (FR3, high sensitivity). This provides a concrete target against which the quality of an automatically derived DFD can be assessed.
RQ2	**How does the quality of LLM-derived privacy specifications compare to that of human expert-created ones?** The Travel Planner is subject to a range of legal requirements, such as **LR1**-**LR4** in Table 4.2, which restrict how personal data may flow and be processed within the system. To detect violations automatically, these restrictions must be represented in a machine-interpretable format that can be checked against the derived DFD. The scenario provides a mix of low-sensitivity flows (e.g., flight search criteria in FR1) and legally constrained flows (e.g., payment data crossing external service boundaries in FR3). This concretely illustrates what such a translation must capture, which flows are restricted, under what conditions and to what degree.

Table 4.3.: Mapping of RQs to the Travel Planner scenario

5. Conceptual Design

In this chapter, we present the overarching conceptual design that is shared across the Research Questions (RQs). First, we provide an overview of the overall pipeline design that shows the intended integration of RQ1 and RQ2 into an automated privacy analysis process for software designs in Section 5.1. Then, in Section 5.2, we discuss the rationale for using Large Language Models (LLMs) as the underlying automation technique for both RQs. In Section 5.3, we outline the general prompt engineering strategy applied across the different phases of the pipelines for both RQs. Section 5.4 then discusses the integration of output validation into the pipelines. Finally, we provide an overview of the shared evaluation design for both RQs in Section 5.5.

Both RQ1 and RQ2 share a common underlying challenge: extracting structured information from unstructured natural language inputs. For RQ1, this involves understanding the functional requirements and translating them into the structured format of Data Flow Diagrams (DFDs). For RQ2, it requires interpreting legal requirements and deriving machine-interpretable privacy specifications intended as input for the annotation of DFDs. This shared underlying challenge motivates treating both RQs as part of a larger conceptual design, rather than as two isolated, entirely separate concepts. Applying a consistent conceptual design across both RQs enables the same underlying automation technique and evaluation methodology to be used for both. This allows the quality of LLM-generated outputs to be assessed against human-generated baselines using the same metrics. For both RQs, the implementation focus in this thesis lies on establishing an end-to-end pipeline that produces DFDs and privacy specifications. Exhaustive optimization of individual phases is left to future work. The conceptual design outlined in this chapter thus serves as a shared foundation before the specifics of each RQ are detailed in the subsequent chapters.

5.1. Automated Pipeline Overview

Figure 5.1 illustrates the automated pipeline that forms the conceptual backbone of this thesis. It comprises three phases. This thesis addresses the first two: the generation of DFDs from functional requirements (RQ1), and the derivation of privacy specifications from legal requirements (RQ2). Annotating the generated DFDs with those specifications constitutes a future extension not covered in this thesis. Checking annotated DFDs for privacy violations has already been addressed in xDECAF [22]. This thesis therefore focuses on producing outputs that integrate into that existing approach. Both phases are automated through LLM-based processing and can be executed independently of each other, with their outputs serving as separate inputs to the annotation step.

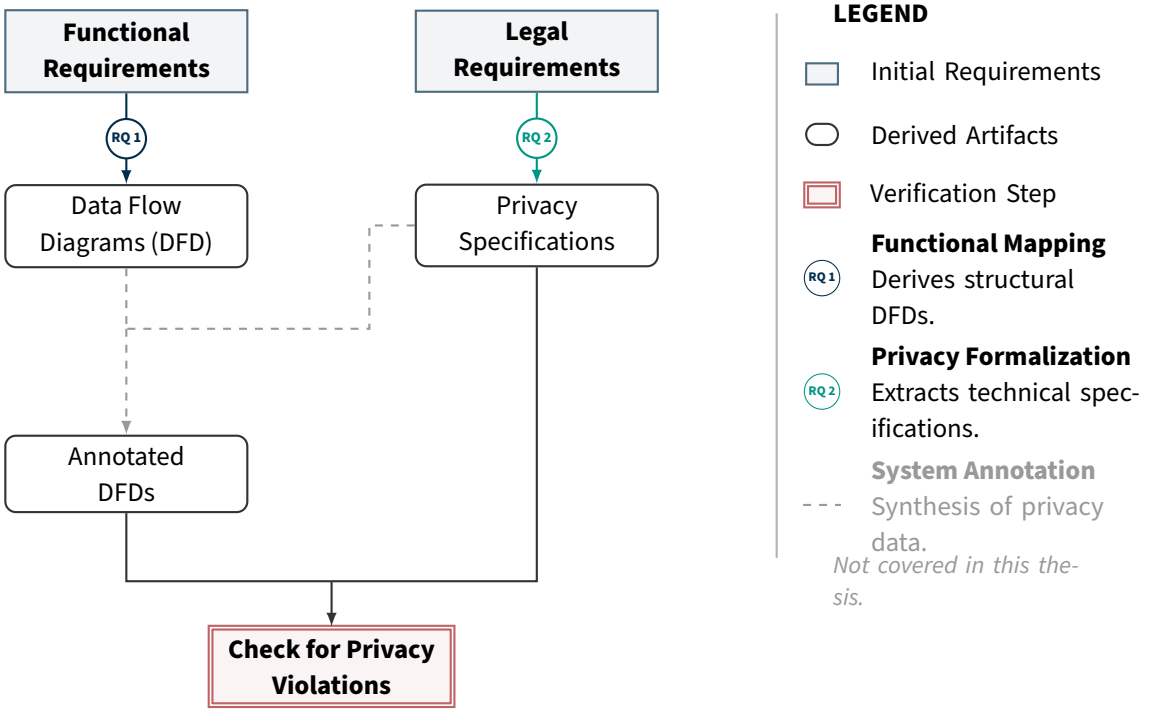


Figure 5.1.: Automated privacy analysis pipeline overview

5.2. Rationale for LLM-Based Automation

Several factors motivate the choice of LLMs as the underlying automation technique for both RQs. First, LLMs have demonstrated strong capabilities in natural language understanding and generation [26, 111], making them well-suited for tasks that require processing unstructured text inputs. Second, LLMs support few-shot prompting and in-context learning, enabling task-specific outputs without retraining or fine-tuning [26, 39]. Furthermore, LLMs have shown the ability to generate structured outputs [38, 55, 73], which is essential since both DFDs and privacy specifications require structured representations. Finally, LLMs can process large text inputs and produce outputs faster than human experts performing the same tasks manually [65]. However, LLMs also present challenges, such as hallucination and sensitivity to prompt design [39, 110], which can lead to outputs that are plausible but incorrect or inconsistent. Mitigating these risks therefore requires explicit design decisions in both the prompting strategy and the output validation, which we address in the following sections.

5.3. Prompt Engineering Strategy

Figure 5.2 illustrates the structured prompt design applied across all pipeline phases for both RQs, including example content for each component to illustrate the structure concretely. Given the sensitivity of LLMs to prompt design [39, 110], a consistent prompt structure

is essential for producing reliable outputs across the different phases. The prompt design consists of the following key components: a role or persona definition, a clear objective statement, few-shot examples with edge case guidance, input processing instructions, the dynamic input itself, and output format constraints.

The prompt opens with *role or persona prompting*, where the LLM is instructed to adopt a specific perspective or expertise relevant to the task [10, 14]. This is followed by a *clear objective statement* that specifies the intended task, which is essential for aligning the LLM's response with the desired outcome [55, 111]. The *few-shot examples* are placed before the dynamic input, as this helps the LLM apply the demonstrated patterns to the input rather than continuing to generate from the examples themselves [14]. They have been shown to be particularly effective for tasks that require specific output formats or domain knowledge, reducing the likelihood of hallucination [26, 55, 65, 110, 111]. The examples additionally include guidance on how to handle uncertain or ambiguous cases, further reducing the risk of inconsistent outputs [38]. Both aspects make few-shot examples particularly valuable for the DFD and privacy specification derivation tasks addressed in this thesis. *Input processing instructions* follow the examples, clarifying how the LLM should interpret and utilize the provided input [55, 110]. The *dynamic input* follows, representing the phase-specific artifact to be processed. Depending on the pipeline phase, this is either the initial functional or privacy requirements set, or an intermediate output produced by a prior phase. Finally, *output format constraints or strategy* either prescribe strict formatting rules for the LLM to follow, or provide guidance on which aspects of the output to prioritize. Applying them has been shown to reduce hallucination and enforce schema adherence [14, 96]. This consistent prompt structure supports reproducibility and comparability of outputs across all pipeline phases [14, 104]. Each of these components draws on established prompt engineering techniques. This thesis instantiates and combines these techniques into a domain-specific prompt template tailored to DFD and privacy specification derivation from natural language. The combination itself is not novel; its contribution lies in the domain-specific configuration and in demonstrating which components prove necessary for this class of structured artifact derivation tasks.

5.4. Output Validation

As discussed in Section 5.2, LLMs are prone to hallucination, producing outputs that are plausible but factually or structurally incorrect. Additionally, when processing inputs containing multiple distinct items, LLMs can exhibit a focus shift. This causes them to lose track of parts of the input, producing outputs that are structurally valid but incomplete in their coverage, commonly known as the "lost in the middle" phenomenon [69]. Left undetected, both types of error propagate through the pipeline and compromise the validity of downstream results. For example, when processing **FR1**–**FR4** of the Travel Planner, an LLM may produce a structurally valid DFD that captures the entities and flows from **FR1**, **FR2**, and **FR4**. The Credit Card Center and its associated data flows from **FR3** may be silently omitted. Output validation detects such omissions by verifying that every input requirement maps

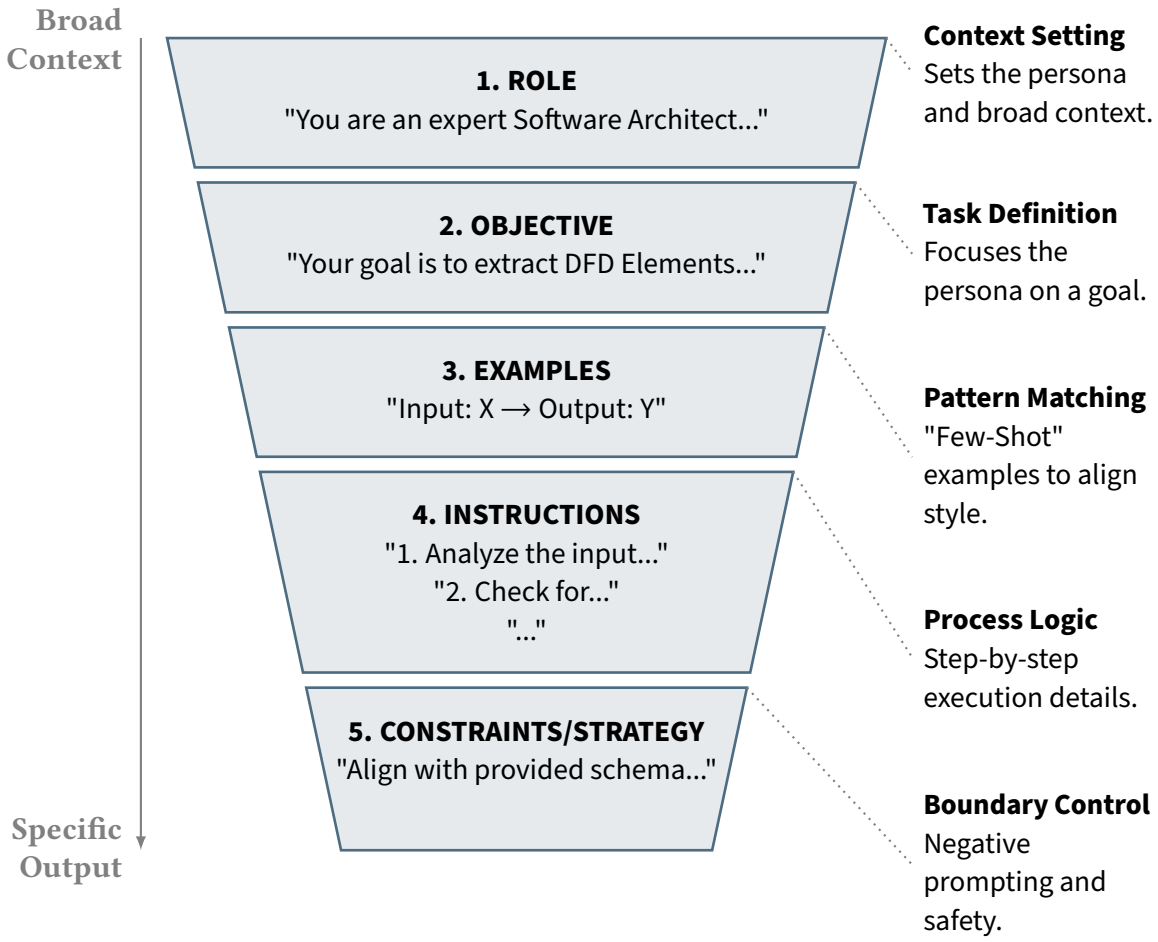


Figure 5.2.: Systematic prompt structure

to at least one output element. Grammar-constrained generation can enforce constraints expressible as formal grammars [108]. However, it cannot express dynamic constraints such as coverage completeness relative to the input, cross-field consistency, or content-level requirements. Output validation is therefore an integral part of the pipeline design. It verifies that generated outputs are structurally correct and complete with respect to the expected schema and input coverage before being passed to subsequent phases. Output validation addresses structural correctness and input coverage. Semantic correctness of the generated content is outside its scope.

5.5. Shared Evaluation Design

Figure 5.3 illustrates the shared evaluation design applied to both RQ1 and RQ2. The evaluation design combines selected elements from empirical LLM-human comparison studies described by Schneider et al. [91] and a blind peer review layer [109]. The blind

review is added to address a specific challenge: the tasks evaluated in this thesis admit multiple valid solutions, making a single-researcher comparison insufficient for an unbiased assessment. In the *preparation* phase, the researcher prepares the tasks for both the LLM-automated pipeline and human experts, ensuring both groups receive the same input data and task definitions. In the *derivation* sessions, the LLM processes the input data to generate the respective outputs, while human experts independently perform the same task to create their own outputs. In the *review* sessions, all derived artifacts are pooled together. The same participants who completed the derivation task then evaluate a randomized subset of outputs from both groups, without knowing which were generated by the pipeline and which by human experts. This blind design aims to reduce Artificial Intelligence (AI) bias in the evaluation, which has been identified as a significant concern when assessing LLM-generated outputs [65]. To allow experts to participate at their own convenience and to enable a broader reach of potential reviewers, the study is conducted as an online survey. After the review is completed, we analyze the collected ratings and feedback to assess the quality of the pipeline outputs relative to the human-generated baseline. We derive evaluation criteria using the Goal Question Metric (GQM) approach [18], establishing specific quality goals with corresponding metrics and questions to assess them. The primary quality dimensions are *correctness* and *completeness*. Correctness refers to the extent to which a derived artifact accurately reflects the intended design or requirements, while completeness refers to the extent to which it captures all relevant aspects of the intended design or requirements. This evaluation design is consistent with existing approaches for assessing LLM-generated outputs, that compare against human expert baselines [85, 112]. As the specifics of each evaluation, such as the task definition and output type, are highly dependent on the respective RQ, they are not further detailed here.

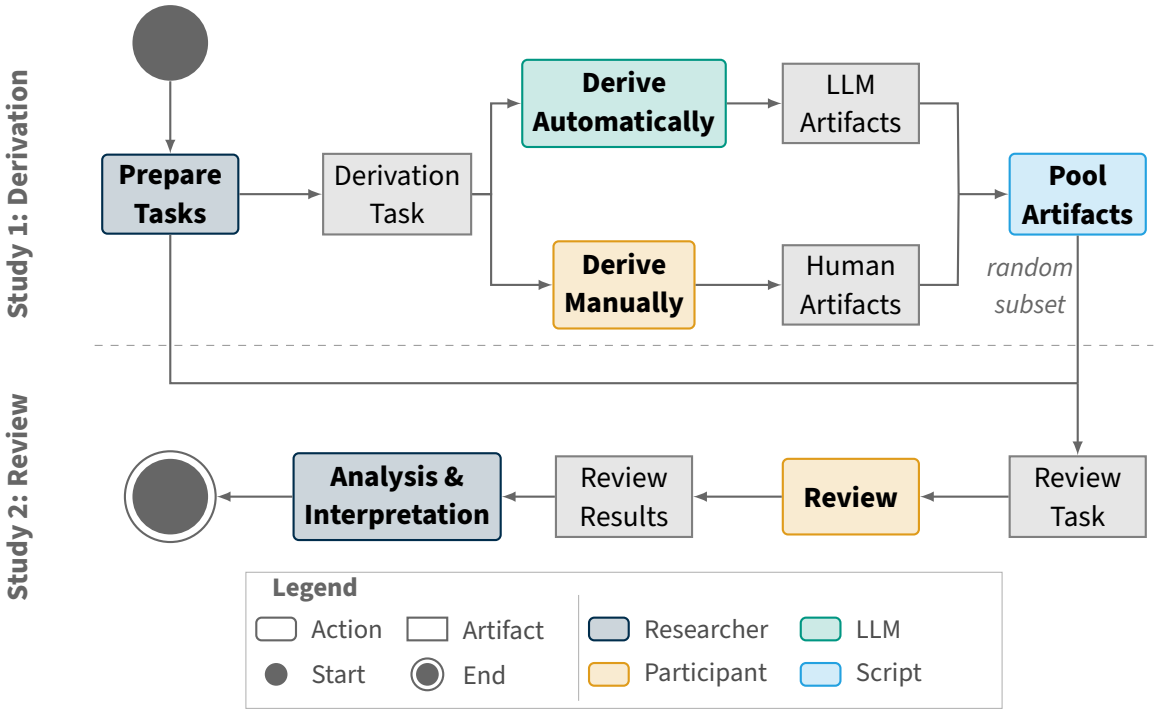


Figure 5.3.: Shared evaluation design for each RQ

6. Automated Derivation of Data Flow Diagrams

This chapter addresses RQ1, which concerns the automatic derivation of Data Flow Diagrams (DFDs) from Functional Requirements (FRs) using Large Language Models (LLMs). It covers the first phase of the automated pipeline introduced in Chapter 5 (see Figure 6.1). The contribution of this chapter is a multi-phase LLM-based pipeline that automatically derives xDECAF-compatible DFDs from natural language functional requirements. It feeds directly into the privacy analysis pipeline of Chapter 5. Throughout this chapter, the *Travel Planner* scenario introduced in Chapter 4 serves as a running example to illustrate the approach. Section 6.1 presents the conceptual design, Section 6.2 details the technical implementation, and Section 6.3 presents the evaluation and results.

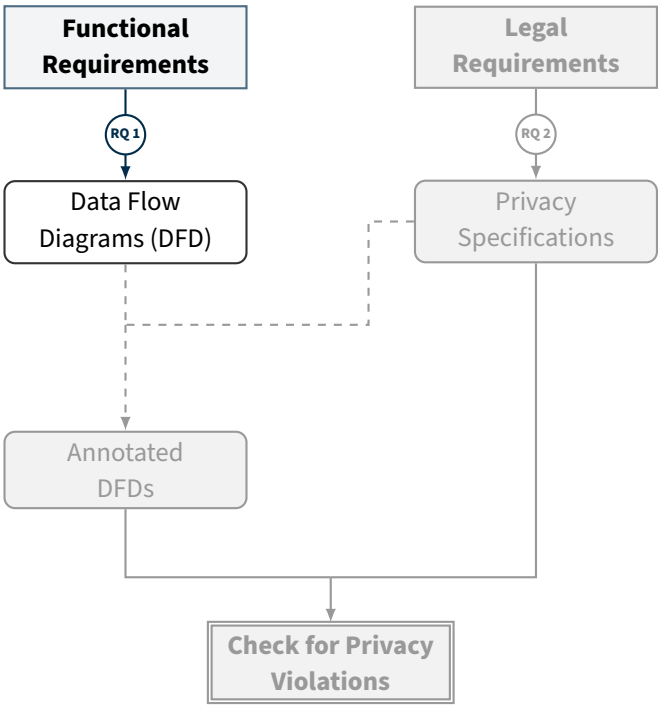


Figure 6.1.: Automated privacy analysis pipeline overview – RQ1 cutout

6.1. Concept

Deriving DFDs from FRs requires bridging a semantic gap. Functional requirements express system behavior in natural language, while DFDs encode it as a structured graph of processes, data flows, and data stores. Capturing the semantic connections between functional elements goes beyond simple pattern matching. It requires interpreting the underlying interactions and data transformations implied by the requirements [55]. To address this, we propose a multi-phase pipeline that systematically decomposes the transformation into targeted steps, as illustrated in Figure 6.2. We motivate the multi-phase design by the inherent limitations of LLMs. Processing the full derivation in a single step would exceed the effective attention span of the model [69]. It would also increase hallucination risk given the semantic complexity of the transformation [111]. By decomposing the task into focused subproblems, we ensure that each phase operates within a well-defined scope, reducing the cognitive load on the LLM per step [14, 110]. The pipeline targets Level 1 DFDs, which decompose the system into its core processes, data stores, and external entities with explicit data flows between them, as defined in Chapter 2. We select Level 1 for three reasons. First, it represents the design granularity at which privacy-relevant data flows between distinct system components become visible. At this level, flows carry labels and constraints. This makes it the natural target for Data Flow Analysis (DFA)-based compliance checking. By contrast, Level 0 (*context diagrams*) captures only system boundaries and is too coarse for privacy analysis. Level 2 and below expose implementation internals that exceed what is typically derivable from high-level functional requirements alone. Second, prior work on LLM-based DFD generation [55] operates at this level, enabling comparison with the closest related approach. Third, the xDECAF framework [22] targets Level 1 analysis. This makes Level 1 a direct requirement for integration into the target toolchain. The pipeline consists of four phases: *data pre-processing*, *element decomposition*, *wiring*, and *formatting*. We identify two additional quality assurance techniques, *reverse validation* and *determinism validation*, as potential future extensions and mark them accordingly in the figure. Across all phases, output validation as described in Chapter 5 serves as the general error handling mechanism. It ensures that outputs conform to the expected schema before they proceed to the next phase. The following subsections describe each phase in detail.

6.1.1. Data Pre-Processing

The first phase prepares the functional requirements for element extraction. Since ambiguous or conflicting requirements can propagate errors through all subsequent phases, we treat ambiguity detection as a central concern [111]. The LLM identifies and flags ambiguous statements. It then attempts to resolve them through clarification question and answer prompts, improving the clarity of the requirements before proceeding [70]. This role is similar to the *Analysis Reviewer* proposed by Zhou et al. [112]. This phase also applies iterative refinement [70, 110, 111]: the LLM rewrites the requirement text based on the clarification Q&A, producing updated requirements that replace the originals. As this thesis targets full automation, we pass the requirements to the next phase in their current state

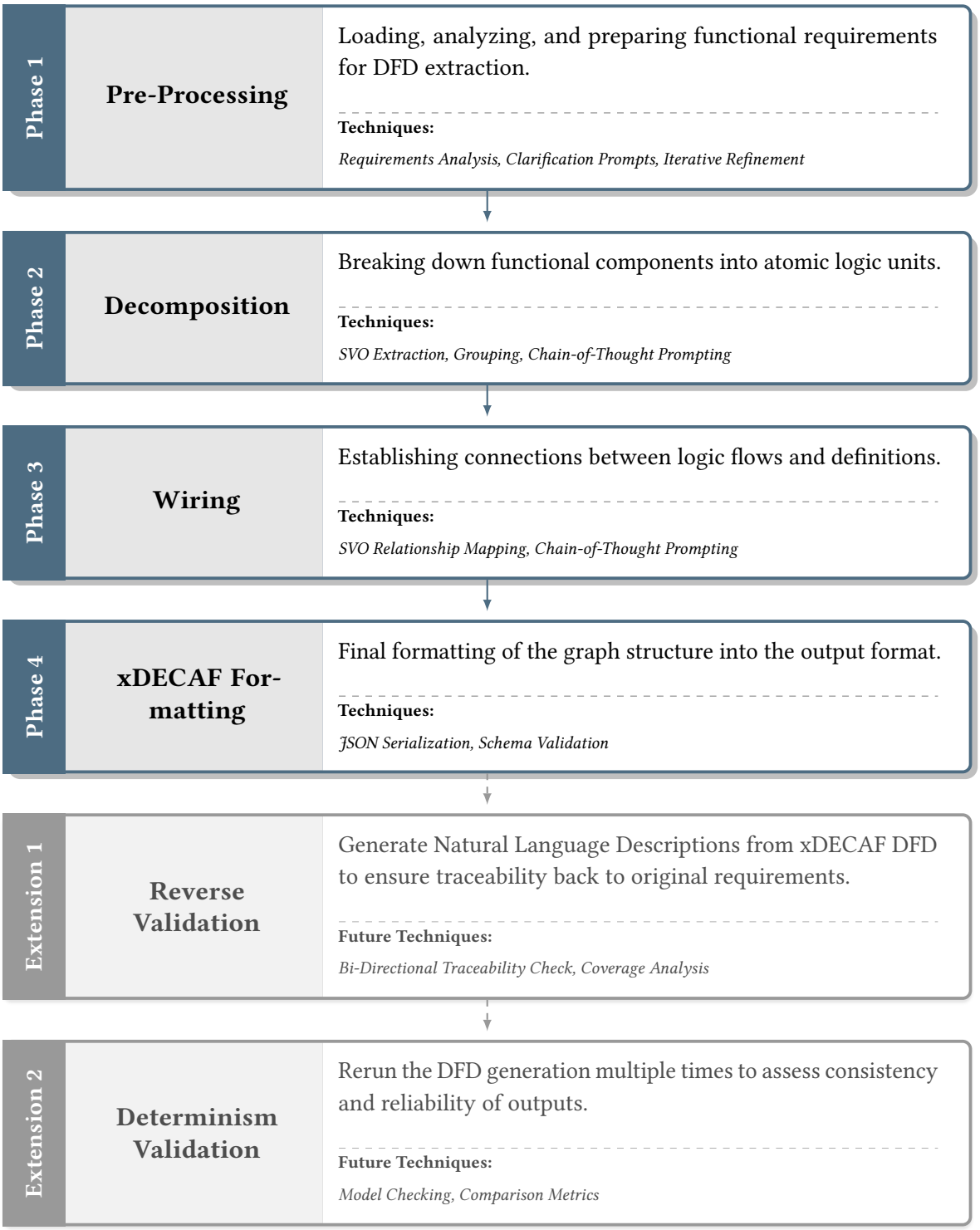


Figure 6.2.: Automated DFD derivation phase structure. Blue-bordered boxes indicate pipeline phases covered in this thesis; gray-bordered boxes mark future extensions.

when the pipeline cannot resolve the ambiguity automatically. We accept that residual ambiguity may cause missing elements or incorrect data flows in the generated DFDs.

6.1.2. Decomposition

The second phase groups the pre-processed requirements into semantic clusters, each representing a single candidate DFD element: a process, data store, or external entity. As the key adaptation to the DFD context, we map multiple related requirements to a single candidate DFD element rather than extracting elements one-to-one from individual requirements. This reflects that a DFD element often emerges from the combined semantics of several requirements. We intentionally defer data flows to the following phase, as identifying them requires a deeper understanding of the relationships between elements. We prompt the LLM to extract elements using linguistic cues, particularly Subject-Verb-Object (SVO) relationships [8, 45, 99]. Verbs indicate processes, nouns indicate data stores, and nouns with specific roles indicate external entities. Figure 6.3 illustrates the SVO-based extraction applied to **FR1** from the Travel Planner scenario. Grouping related requirements into coherent subsystems reduces duplicates and irrelevant candidates [55]. We apply Chain-of-Thought prompting to encourage the LLM to reason through inter-element relationships rather than extracting elements in isolation [14, 42, 56, 110, 111]. Output validation, as described in Chapter 5, ensures that the extracted elements conform to the expected schema before they proceed to the next phase [52]. This phase produces a clean, deduplicated set of candidate elements that forms the basis for the data flow connections established in the next phase.

6.1.3. Wiring

The previous phase identifies what the system consists of, whereas this phase determines how those elements interact. We achieve this by reasoning about the semantics of data exchange rather than the presence of individual elements. This phase constructs the wiring topology of the DFD by identifying the data flows between the semantic clusters produced in the decomposition phase. The LLM applies SVO analysis to extract data assets, the objects transferred between nodes, from the requirement text. It also identifies the relationships between nodes, determining which process produces or consumes which data from which data store or external entity. As our key adaptation to the DFD context, we determine flow direction not based on linguistic cues but by the types of the connected nodes. This follows the strict DFD topology rules to produce structurally valid DFDs within the xDECAF meta model [22]. We apply Chain-of-Thought prompting to encourage the LLM to reason through inter-node relationships systematically rather than inferring flows in isolation [14, 42, 56, 110, 111]. This phase produces a set of named data flow connections between nodes, linking the elements identified in the decomposition phase into a coherent topology ready for formatting. Figure 6.4 shows the wiring topology derived from **FR1** of the Travel Planner scenario. Data flows connect processes, data stores, and external entities, representing the interactions implied by the requirements.

FR1: Flight Search

The customer *sends* flight search criteria (e.g., destination, dates) to the Travel Agency (TA) web service via the Travel Planner (TP) app. The Travel Agency *processes* the request and *returns* a list of flight offers to the customer.

Fragment	SVO Role	DFD Element Type
Customer	Subject / Recipient	External Entity
Travel Agency	Subject / Recipient	External Entity
Travel Planner	Intermediary System	Process
flight search criteria	Object	Data Flow
flight offers	Object	Data Flow

Figure 6.3.: SVO-based element extraction applied to FR 1 of the Travel Planner scenario. Orange highlights mark external entities, blue highlights mark processes, and green highlights mark data flows.

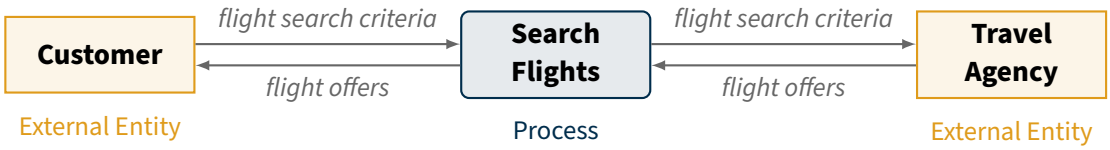


Figure 6.4.: Data flow connections derived from FR 1 of the Travel Planner scenario

6.1.4. xDECAF Formatting

The final phase takes two inputs: the semantic clusters from the decomposition phase and the data flow connections from the wiring phase. It programmatically combines them into a single xDECAF-compatible DFD representation. Unlike the preceding phases, this step does not use an LLM. Deterministic transformation rules map each element to the xDECAF schema, since non-determinism here would introduce structural errors. The result is an xDECAF-compatible DFD ready for direct integration into the privacy analysis pipeline of Chapter 5.

6.1.5. Potential Extensions

We identify two additional quality assurance techniques as potential future extensions to this pipeline. Both could reduce missing-element and structural errors in the generated DFDs. *Reverse Validation* would establish traceability between the generated DFD and the original functional requirements. It would verify that each DFD reflects all requirements, and that each DFD element traces back to at least one requirement. This forms an additional correctness criterion beyond schema validation, confirming that the generated DFD is semantically complete with respect to the input. Another potential extension is *Determinism Validation*, which would address a fundamental property of LLMs: identical inputs can produce varying outputs across runs. Such variability can cause structurally inconsistent DFDs for the same requirements. By re-running the pipeline on the same requirements and comparing the resulting DFDs, we could flag runs where node sets or flow topologies differ as indicators of unreliable pipeline behaviour.

6.2. Implementation

This section describes how we realize the pipeline introduced in Section 6.1. The focus lies on establishing a functional end-to-end pipeline rather than exhaustively optimizing individual phases, as noted in Chapter 5. For each phase, we cover the data used, the prompt design, and the key implementation choices. The functional requirements of the Travel Planner scenario from Chapter 4 serve as concrete examples throughout.

6.2.1. Implementation Setup

The implementation builds on four key choices: the underlying model, the programming language and API, the data formats for inter-phase communication, and the tooling for output validation and evaluation. A comparative evaluation of LLMs for software engineering tasks [94] informed our model selection. We use GPT-5.3 [76]. This choice reflects the availability of more capable models at the time of implementation. For the implementation language, we use Python 3 [84], which offers extensive libraries for natural language processing, data manipulation, and integration with LLMs. We also use the OpenAI Async Chat Completions API [78] to facilitate efficient and scalable interactions with the LLMs. The pipeline follows a modular design with separate modules per phase. This enables independent refinement, debugging, and reuse of components across phases. For inter-phase data exchange, we use YAML [20] for prompt inputs and JSON [24] for structured outputs. A comparison of YAML, JSON, CSV, and unstructured formats for LLM inputs found YAML to achieve the best input processing performance [38]. Its higher readability and lower syntactic overhead reduce token usage and processing costs. The same comparison identified JSON as producing the highest quality structured outputs [38]. Additionally, xDECAF [22] requires JSON as the input format for DFDs and privacy specifications. For output validation, we use Pydantic [83], a widely used data validation library for Python.

We chose it for its expressive schema definitions and detailed error messages, which can be fed back to the LLM as structured correction feedback. For the online evaluation, we use SoSci Survey [67]. It supports the blind peer review design outlined in Section 5.5. Its highly customizable question types and PHP-based logic enable the required randomization and branching.

Within the implementation, we apply a consistent execution model across all LLM processing phases. Each phase consists of one or more processing *steps*. Each step follows the structured flow illustrated in Figure 6.5. Depending on the requirements of the phase, steps can be executed *sequentially*, *in parallel*, or in a *looped* manner. Sequential execution is the default. We apply parallel execution when a phase can be decomposed into independent sub-tasks that each require focused LLM processing and can be handled in separate runs. We use looped execution when the input cannot be decomposed into independent sub-tasks but requires iterative processing to cover all relevant aspects in successive passes. Each step begins by loading the associated *prompt template* and *dynamic input data*, which are combined to *build* the filled prompt. The pipeline sends this prompt to the LLM through the OpenAI Async API and receives an *output*. On receiving the output, the pipeline then *validates* it against the Pydantic model associated with the prompt, which defines the expected structure and constraints. If the output is *valid*, the pipeline stores it and passes it to the next step or phase. If the output is *invalid*, the pipeline raises an *error* and triggers a retry. Pydantic provides detailed error messages as feedback to the LLM to refine the output in the next attempt. The default retry limit of three balances robustness against processing cost. We adapt this limit based on the requirements of a given phase. This mechanism reduces structurally invalid outputs by catching schema violations before they propagate to subsequent phases. We expect this to also reduce hallucination risk, as targeted correction feedback guides the LLM toward structurally valid outputs. This execution model, developed as part of this thesis, enables modular, phase-based pipeline construction.

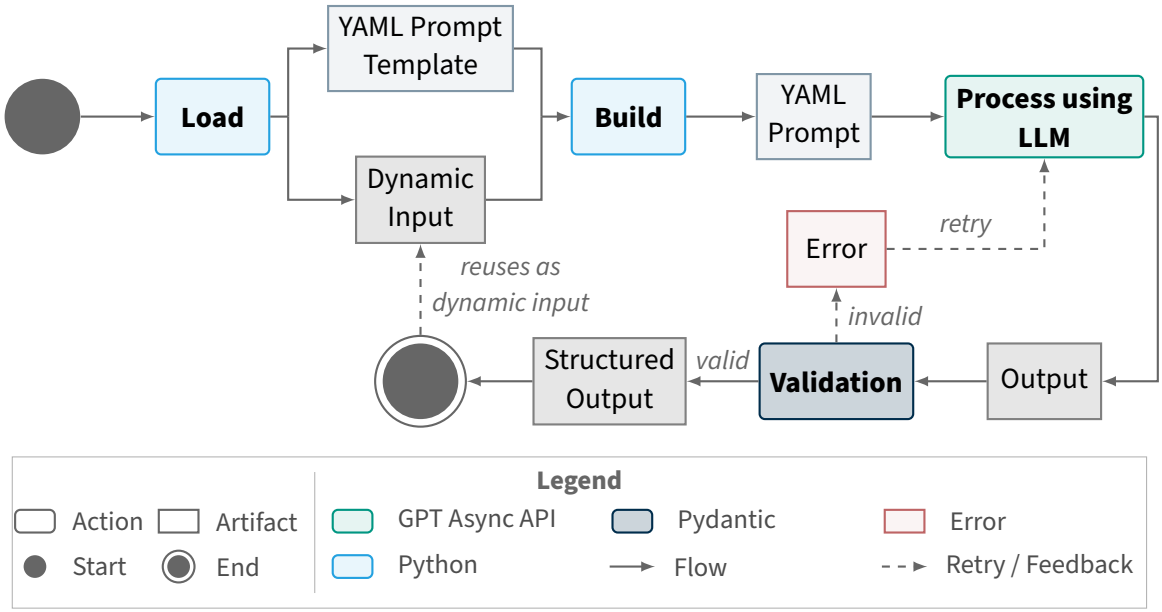


Figure 6.5.: Single step execution model

6.2.2. Dataset Selection

To build and refine the pipeline and its prompt templates, we identify a dataset of functional requirements suited to DFD derivation. We prioritize system-specific descriptions over general requirements datasets, since these describe complete systems with consistent actor and component naming. Consistent naming is a prerequisite for meaningful DFD extraction. Availability and prior use in related work on automated diagram generation serve as further selection criteria. The *User Stories Dataset* [31] is the only dataset satisfying all three criteria. It comprises 22 requirement sets with over 50 requirements each from real software projects. Related work on LLM-based DFD generation has already applied it [55]. We use it as the primary dataset for all pipeline development and prompt refinement. The Travel Planner scenario, by contrast, serves solely as a running example for illustration within this thesis. Before applying the pipeline phases, we load the requirements into a list of structured requirement objects, which serves as the initial input for the pre-processing phase.

6.2.3. Data Pre-Processing

The first phase prepares the functional requirements for the subsequent extraction and wiring phases. Its emphasis lies on identifying and resolving ambiguities in the requirements text, as described in Section 6.1.1. It receives the raw functional requirements from the selected dataset as input. The phase produces a refined requirement set with reduced ambiguity for the subsequent phases. We implement it as a loop of *ambiguity detection*, *clarification questioning*, *clarification answering*, and *requirements updating*. This loop iteratively resolves ambiguities in the requirements. The refined output then advances to the next phase.

Prompt Design Each step in this phase uses an individual prompt template. Figure 6.6 shows an excerpt for the *ambiguity detection* step. We frame the role as a *hostile* Senior Requirements Analyst, an adversarial persona. The hypothesis is that this framing counteracts the LLM's tendency to make benign assumptions about vague requirements and increases its sensitivity to potential ambiguities. Empirical comparison with a neutral expert persona falls outside the scope of this work. Three few-shot examples cover both ambiguous and unambiguous requirements. They teach the LLM what to flag and what to pass through without raising false positives. A structured chain-of-thought sequence then guides the LLM through a systematic per-requirement analysis of four defined ambiguity types: lexical, referential, scope, and process ambiguity. This reduces the risk of inconsistent coverage from ad-hoc reasoning [14, 110]. To handle large requirement sets, we design the prompt to operate on batches rather than the full set at once. It provides the full document as context while limiting the analysis scope to a manageable subset. This mitigates coverage gaps from long-context attention degradation [69]. The LLM assigns each requirement one of three readiness levels based on the severity of detected issues: ready to proceed, requiring minor clarification, or requiring major rework. This classification directly gates whether the requirement proceeds to the clarification loop. Finally, a reflection step asks the LLM to

justify its classifications and flag any borderline cases. This serves as an additional check against inconsistent decisions.

Output Validation The pipeline validates the output of the *ambiguity detection* step against a Pydantic schema that enforces both structural and logical correctness. Structurally, enum constraints ensure that each detected issue carries a valid severity level and ambiguity type, and that the overall readiness classification is one of the three defined values. Beyond structural checks, a cross-field validator enforces logical consistency. If the LLM identifies no issues, the validator requires the readiness classification to be set to ready. This catches outputs that are structurally valid but internally contradictory. A model-level validator additionally normalises field-name hallucinations. Rather than failing outright, it remaps common aliases to their expected counterparts, such as *overview* to *summary*. This approach is consistent with the retry mechanism described in Section 5.4. We encode the loop termination condition as a final validator. The phase only advances once all requirements in the batch are classified as ready. If the pipeline reaches the maximum number of iterations, the phase proceeds with the best-effort output. For the *clarification answering* step, the schema additionally enforces content-level constraints on the answer field. It requires an Subject-Verb-Object (SVO)-ready sentence and explicitly forbids vague responses such as *The System* or *It depends*.

Running Example We apply the data pre-processing phase to the Travel Planner scenario from Chapter 4. It takes **FR1** to **FR4** as text input and produces the output shown in Table 6.1 after two iterations of the ambiguity detection and clarification loop. The phase resolves ambiguities by inferring specific details about each involved component. These include the exact fields in the flight offer list returned by the Travel Agency, and the data points validated during booking preparation. They also cover the conditions under which the Credit Card Center retrieves stored card tokens versus encrypted card numbers. Additionally, they specify the data included in the commission payable record created by the Airline web service. Table 6.1 shows the refined requirements produced by this phase for the Travel Planner example. However, the added detail reflects the LLM inferring plausible information rather than input from actual stakeholders or domain experts. This reflects an inherent trade-off of the fully automated approach. In the absence of real stakeholder input, the LLM resolves ambiguity by inferring plausible but unverified details, which may not reflect the actual system.

6.2.4. Decomposition

This phase groups the refined requirements from the pre-processing phase into candidate DFD elements using semantic clustering, as described in Section 6.1.2. We implement it as a single semantic clustering step. The LLM groups related requirements into functional aggregates using SVO-based linguistic cues. This maps multiple requirements to a single candidate element rather than extracting elements one-to-one from individual requirements. We explicitly defer data flows to the Wiring phase. This step focuses solely on node

```
ROLE:
  # Hostile Senior Requirements Analyst persona that treats every vague term as a
      CRITICAL
  # blocker -- enforces strict ambiguity detection without making assumptions.

OBJECTIVE:
  # Analyze each requirement in the batch for ambiguities that prevent clear SVO
      parsing
  # and DFD element classification; classify each issue by type and severity.

EXAMPLES:
  # Three few-shot JSON examples: one ambiguous requirement with vague adjectives and
  # undefined scope, one well-formed requirement ready for extraction, and one
  # passive-voice requirement with unclear actors and storage details.

INSTRUCTIONS:
  # Seven-step pipeline applied to each requirement in the batch:
  #   STEP_0 -- Ingest full requirements document for context; isolate the target
      requirement.
  #   STEP_1 -- Review four ambiguity classes (LEXICAL, REFERENTIAL, SCOPE, PROCESS)
  #             with definitions, examples, and detection strategies; review severity
       guidelines.
  #   STEP_2 -- Analyze text sentence by sentence; create an issue object per
      ambiguity
  #             with description, type, and severity.
  #   STEP_3 -- Classify overall DFD readiness as READY, NEEDS_MINIMAL_CLARIFICATIONS,
  #             or NEEDS_MAJOR_REWORK based on detected issues.
  #   STEP_5 -- Self-reflect on completeness and severity justifications; write
      summary.
  #   STEP_6 -- Construct final JSON output object; enforce strict schema and
      unchanged req_id/text.
  #   STEP_7 -- Repeat for all requirements in the batch.
  # FINAL_STEP -- Aggregate all outputs into a single JSON array; enforce schema
      compliance.

INPUT_DATA:
  # Two input slots: requirements_set (full document for context only) and
  # requirements_set_batch (the specific subset to process in this run).

STRATEGY:
  # Three directives: systematic step-by-step analysis for full ambiguity coverage,
  # contextual understanding via the full requirements set, and justified reasoning
  # captured in the summary field of each output object.
```

Figure 6.6.: Ambiguity detection prompt template (excerpt)

FR	Refined Requirement Text
FR1	The Travel Agency web service *validates the* flight search criteria from the Travel Planner app, *queries Airline inventory and fare web services and the Travel Agency cached flight-offer repository, includes only flight offers with matching destination, matching departure and return or segment travel dates, sufficient available seats in the requested fare class, and bookable fare status, excludes flight offers with missing destination, missing travel-date, unavailable-seat, or non-bookable-fare data, formats the flight offer list with offer identifier, airline or carrier, flight number or segment identifiers, origin, destination, departure and arrival date-times, itinerary details, fare class, total price, currency, available seat indicator, bookable fare status, and offer expiration timestamp ordered by total price ascending and departure time ascending,* and returns the *formatted* flight offer list to the Travel Planner app.
FR2	The Travel Planner app *validates the selected offer identifier, itinerary details, fare class, price, travel dates, passenger reference or passenger details, and booking-session context against the active flight-offer list and Booking Session Store, packages the validated fields into an Airline web service booking-preparation request using a passenger reference for an existing passenger profile or passenger details for a new or edited passenger,* forwards the request to the Airline web service, routes Airline responses to the customer's device, *maintains the booking session identifier, selected offer identifier, itinerary details, fare class, price, travel dates, passenger data, Airline request correlation identifier, and latest Airline response status in an application-controlled in-memory Booking Session Store on the customer's device, and clears the Booking Session Store after booking completion, cancellation, timeout, or app session termination.*
FR3	The Credit Card Center app *accesses the OS-protected secure keychain or secure-enclave-backed Local Secure Card Vault on the customer's smartphone* after the Customer confirms payment through *biometric or device passcode* verification, *generates a signed device authorization token, retrieves and sends a stored card token when a usable token exists and the Airline web service payment-processing endpoint accepts it or retrieves and sends an encrypted primary account number when no usable token exists or the endpoint requires encrypted card-number credentials, and* sends the *payment amount or booking reference, cardholder name, expiration date, billing reference, and signed device authorization token* to the Airline web service *payment-processing endpoint.*
FR4	The Airline web service *Booking-and-Payment Completion Process evaluates confirmed reservation or ticketing status, approved or captured payment status, and a valid Travel Agency referral identifier, creates a commission payable record containing commission payable identifier, booking identifier, ticket or reservation identifier, Travel Agency identifier, Travel Agency referral identifier, payment transaction identifier, commission amount, currency, payable status, creation timestamp, and settlement instruction status in the Airline commission payables ledger until settlement and financial reconciliation are complete, and* sends a commission payment *instruction to the Airline payment service when it supports outbound commission disbursement for the Travel Agency or to the banking interface when bank-transfer settlement is required or the Airline payment service does not support the settlement route.*

Green italic text marks details inferred by the LLM beyond the original functional requirements.

Table 6.1.: Refined requirements after data pre-processing phase

identification. The phase produces a deduplicated set of candidate DFD elements with their assigned types, which serves as the structural input for the subsequent Wiring phase.

Prompt Design Figure 6.7 shows an excerpt of the prompt template for the semantic clustering step. We frame the role as a *Domain-Driven Design (DDD) Architect and Systems Analyst*. This grounds the LLM's reasoning in architectural and domain knowledge rather than surface-level textual patterns. It supports the identification of meaningful functional groupings. One detailed example demonstrates the many-to-few mapping of multiple related requirements onto two candidate DFD elements. This establishes the expected grouping behaviour before the LLM processes the actual input. A structured chain-of-thought sequence then guides the LLM through a systematic analysis using SVO-based linguistic cues: subjects identify external entities, verbs identify processes, and nouns identify data stores. An explicit atomicity validation step follows. It requires the LLM to verify that each identified group represents a single, cohesive DFD element before finalising the output. Finally, the strategy section enforces a *verb-noun* naming convention for all process elements. It also instructs the LLM to group by functional context rather than by superficial keyword co-occurrence. This reduces the risk of fragmented or redundant candidates.

Output Validation Once the LLM generates the output, the pipeline validates it against a Pydantic schema that enforces structural, cross-field, and completeness constraints. Structurally, each cluster must be typed as one of the three DFD element types: process, data store, or external entity. The schema also allows the LLM to flag clusters as non-functional requirements. The pipeline then excludes these rather than mapping them to DFD elements. This acts as a safety valve for requirements that escaped the data pre-processing phase. A cross-field validator additionally checks that each cluster's identifier reflects its element type, since misclassifications would otherwise propagate silently to subsequent phases. Each cluster also requires the LLM to provide a chain-of-thought justification for the grouping decision. This enforces deliberate reasoning and guards against superficial or unjustified clusters. For completeness, a validator ensures that every input requirement is either assigned to a cluster or explicitly marked as unassigned, since silent omissions would otherwise go undetected. Finally, analogous to the ambiguity detection loop in the previous phase, each cluster carries an atomicity flag that serves as the loop termination gate. The phase retries until all clusters are atomic, or proceeds with the best-effort output if the pipeline reaches the iteration limit. We consider a cluster atomic if it represents a single, cohesive DFD element that cannot be further decomposed without losing its functional integrity.

Running Example Table 6.2 shows the output of the semantic clustering phase applied to the refined requirements from Table 6.1. The LLM identifies four candidate processes, four candidate data stores, and five candidate external entities. Each cluster references the specific requirements that contributed to its identification. It names each process cluster using a verb-noun pair that reflects the core action of the grouped requirements and marks all data stores and external entities as atomic. When the pipeline reaches the iteration

```
ROLE:
  # DDD Architect and Systems Analyst persona -- performs Functional Decomposition by
  # organizing requirements into cohesive DFD nodes (Processes, Data Stores, External
       Entities).

OBJECTIVE:
  # Three identification tasks: (1) External Entities outside the system boundary,
  # (2) Internal Data Stores for persistence, (3) Internal Processes as cohesive
  # functional units -- with a CRITICAL rule to name groups by action, not system
      boundary.

EXAMPLES:
  # One detailed few-shot example with nine requirements clustered into two PROCESS
      groups
  # (Process Payments, Manage User Access), two DATA_STORE groups (User Database,
  # Inventory Database), and one EXTERNAL_ENTITY group (External User), each with
  # reasoning traces, contribution notes, and atomicity flags.

INSTRUCTIONS:
  # Five-step reasoning sequence applied per group:
  #   STEP_0 -- Parse each requirement into SVO triplets for traceability.
  #   STEP_1 -- Identify persistence nouns (store, persist, cache, log) and map to
      DATA_STORE groups.
  #   STEP_2 -- Identify action verbs (process, validate, manage) and form PROCESS
      groups
  #             by shared verbs and related data lifecycle nouns.
  #   STEP_3 -- Identify external actors (user, third-party) and map to
      EXTERNAL_ENTITY groups.
  #   STEP_4 -- Validate atomicity per group; justify decision in reasoning_trace.
  #   STEP_5 -- Compile final output per the cluster schema; list unassigned
      requirements separately.

INPUT_DATA:
  # Single input slot: requirements_set -- cleaned requirements from Phase 1.

STRATEGY:
  # Six directives: prioritize functional cohesion over keyword matching; use domain
      knowledge;
  # maintain traceability; ignore "The System" as subject -- use the action instead;
  # group by functional context; name every PROCESS group with a Verb-Noun pair.
```

Figure 6.7.: Semantic clustering prompt template (excerpt)

limit, however, the LLM flags all four process clusters as non-atomic. The phase therefore proceeds with the best-effort output rather than fully validated atomic clusters.

ID	Name	FR(s)	Atomic
Processes			
P-001	Search Flight Offers	**FR1**	✗
P-002	Prepare Booking Request	**FR2**	✗
P-003	Authorize Card Payment	**FR3**	✗
P-004	Settle Commission Payable	**FR4**	✗
Data Stores			
DS-001	Cached Flight-Offer Repository	**FR1**	✓
DS-002	Booking Session Store	**FR2**	✓
DS-003	Local Secure Card Vault	**FR3**	✓
DS-004	Commission Payables Ledger	**FR4**	✓
External Entities			
E-001	Travel Planner App	**FR1**, **FR2**	✓
E-002	Airline Web Service	**FR1**, **FR2**, **FR3**	✓
E-003	Customer	**FR2**, **FR3**	✓
E-004	Airline Payment Service	**FR4**	✓
E-005	Banking Interface	**FR4**	✓

✗ non-atomic at max iterations; ✓ atomic

Table 6.2.: Candidate DFD elements identified by the semantic clustering phase

6.2.5. Wiring

This phase takes the candidate DFD elements produced by the decomposition phase as input and identifies the data flows between them. It wires these nodes into a connected DFD topology. We implement it as a single data flow extraction step using a single-pass, global analysis approach. Rather than processing pairs of nodes in isolation, the LLM considers the entire set of candidate elements and the requirements embedded within each cluster. From this global view, it determines flow direction and data content. The phase produces a set of named data flow edges. Each edge specifies a source node, a target node, and the data object in transit. We pass these edges to the subsequent phase for final assembly into the DFD representation.

Prompt Design Figure 6.8 shows an excerpt of the prompt template for the wiring step. We frame the role as a *Senior Security Architect and DFD Specialist* familiar with the xDECAF specification. This grounds the LLM's reasoning in both security-oriented data flow analysis

and the structural requirements of the xDECAF target format. The prompt includes two detailed few-shot examples that demonstrate the expected output format and the reasoning process for determining flow direction based on node types and requirement context. These establish a clear precedent for the LLM before it processes the actual input. A structured chain-of-thought sequence guides the LLM through a systematic analysis. It first extracts SVO triplets from each cluster's linked requirements to identify data assets. It then determines flow direction based on node roles and the topology rules embedded in the prompt. Finally, it constructs data flow objects with the required fields. An explicit self-validation step then requires the LLM to check each constructed flow against the topology rules before finalising the output. This serves as a final guard against structurally invalid flows. Constraints additionally enforce strict schema adherence and require all identified flows to be included in the output.

Output Validation Once the LLM generates the output, the pipeline validates it against a Pydantic schema that enforces format constraints on each field. Specifically, flow identifiers, data asset names, and node references must each conform to defined patterns that reflect their expected structure and type. Topology rules such as referential integrity, the Interface Law, and the Passive Store Law require cross-object validation that exceeds Pydantic's field-level constraints. We therefore rely on the prompt's self-validation step to enforce them.

Running Example Table 6.3 shows the output of the wiring phase applied to the candidate DFD elements from Table 6.2. The LLM identifies 19 data flows connecting the various processes, data stores, and external entities. It names each flow after the specific data asset being transferred. For example, the pipeline identifies `DF-001` from `E-001` (Travel Planner App) to `P-001` (Search Flight Offers). This flow captures the submission of flight search criteria described in **FR1**. `DF-004` from `P-001` (Search Flight Offers) to `DS-001` (Cached Flight-Offer Repository) captures the querying of cached offer candidates relevant in **FR2**. The pipeline then passes this output to the xDECAF Formatting phase for final assembly. That phase combines the nodes and flows into a complete DFD representation of the system.

6.2.6. xDECAF Formatting

This phase takes the candidate DFD elements from the decomposition phase and the data flows from the wiring phase as input, and produces a complete DFD representation in the xDECAF format. Because this is a deterministic transformation into a fixed target schema, we implement it as a scripted mapping rather than an LLM-driven step. The phase produces a single file containing the complete DFD representation of the system in the JSON format required by xDECAF.

```
ROLE:
  # Senior Security Architect and DFD Specialist (xDECAF-aware) -- constructs the
  # Wiring Diagram (topology) connecting pre-defined Functional Nodes via Data Flows.

OBJECTIVE:
  # Analyze requirements text to define all Data Flows between the given nodes
  # and specify the Data Asset carried by each flow.

EXAMPLES:
  # Two few-shot examples: (1) a PROCESS-to-DATA_STORE flow for validated transaction
  # data between Process Payments and Transaction Cache; (2) bidirectional
  # EXTERNAL_ENTITY-to-PROCESS flows (User Commands and Command Results) between
  # External User and User Command Processing.

INSTRUCTIONS:
  # Four topology rules enforced throughout:
  #   1. Conservation of Flow -- every PROCESS must have at least one incoming and one
       outgoing flow
  #      (no Black Holes or Miracles).
  #   2. Interface Law -- EXTERNAL_ENTITY nodes must communicate through a PROCESS.
  #   3. Passive Store Law -- DATA_STORE nodes cannot connect to other DATA_STORE
      nodes.
  #   4. Referential Integrity -- source/target IDs must exactly match Phase 2
      group_ids.
  #
  # Four-step pipeline:
  #   STEP_1 -- Extract SVO triplets from grouped requirements to identify Data Asset
      names.
  #   STEP_2 -- Determine flow direction per node roles (E, P, DS) and topology rules.
  #   STEP_3 -- Construct a Data Flow Object (flow_id, name, source_node_id,
  #             target_node_id, description) per identified asset and direction.
  #   STEP_4 -- Validate all flow objects against topology rules; adjust or flag
      violations.

INPUT_DATA:
  # Single input slot: semantic_clustered_requirements -- Functional Nodes from Phase
      2.
  # Only these nodes may be used; creating new nodes or undescribed flows is
      forbidden.

CONSTRAINTS:
  # Four output constraints: strict Pydantic schema compliance, no thinking text in
      output,
  # all identified flows must be included, topology rules must be strictly followed.
```

Figure 6.8.: Wiring prompt template (excerpt)

ID	Name	Source	Target
DF-001	Flight Search Criteria	E-001	P-001
DF-002	Airline Inventory Fare Query	P-001	E-002
DF-003	Airline Inventory Fare Data	E-002	P-001
DF-004	Cached Flight Offer Query	P-001	DS-001
DF-005	Cached Flight Offers	DS-001	P-001
DF-006	Formatted Flight Offer List	P-001	E-001
DF-007	Selected Offer Booking Details	E-001	P-002
DF-008	Booking Session Context	DS-002	P-002
DF-009	Booking Preparation Request	P-002	E-002
DF-010	Airline Booking Response	E-002	P-002
DF-011	Customer Airline Response	P-002	E-003
DF-012	Booking Session State	P-002	DS-002
DF-013	Payment Confirmation	E-003	P-003
DF-014	Stored Card Credentials	DS-003	P-003
DF-015	Payment Authorization Data	P-003	E-002
DF-016	Booking Payment Referral Status	E-002	P-004
DF-017	Commission Payable Record	P-004	DS-004
DF-018	Commission Payment Instruction	P-004	E-004
DF-019	Bank Transfer Commission Instruction	P-004	E-005

Node IDs refer to clusters in Table 6.2. Blue = process, gray = data store, orange = external entity.

Table 6.3.: Data flows identified by the wiring phase

Running Example The pipeline assembles the 13 nodes from Table 6.2 and the 19 data flows from Table 6.3 into an xDECAF-compatible DFD representation. The resulting file is ready for direct integration into the xDECAF toolchain for privacy analysis. Figure 6.9 shows the generated DFD as rendered by the xDECAF tool. The diagram includes all nodes and flows assembled in the previous phases.

Together, the four phases build a complete end-to-end pipeline that transforms raw functional requirements from natural language into an xDECAF-compatible DFD. This DFD serves as the artefact evaluated in the following section. The data companion set [46] contains all prompt templates, validation schemas, and the prototype implementation code for the pipeline.

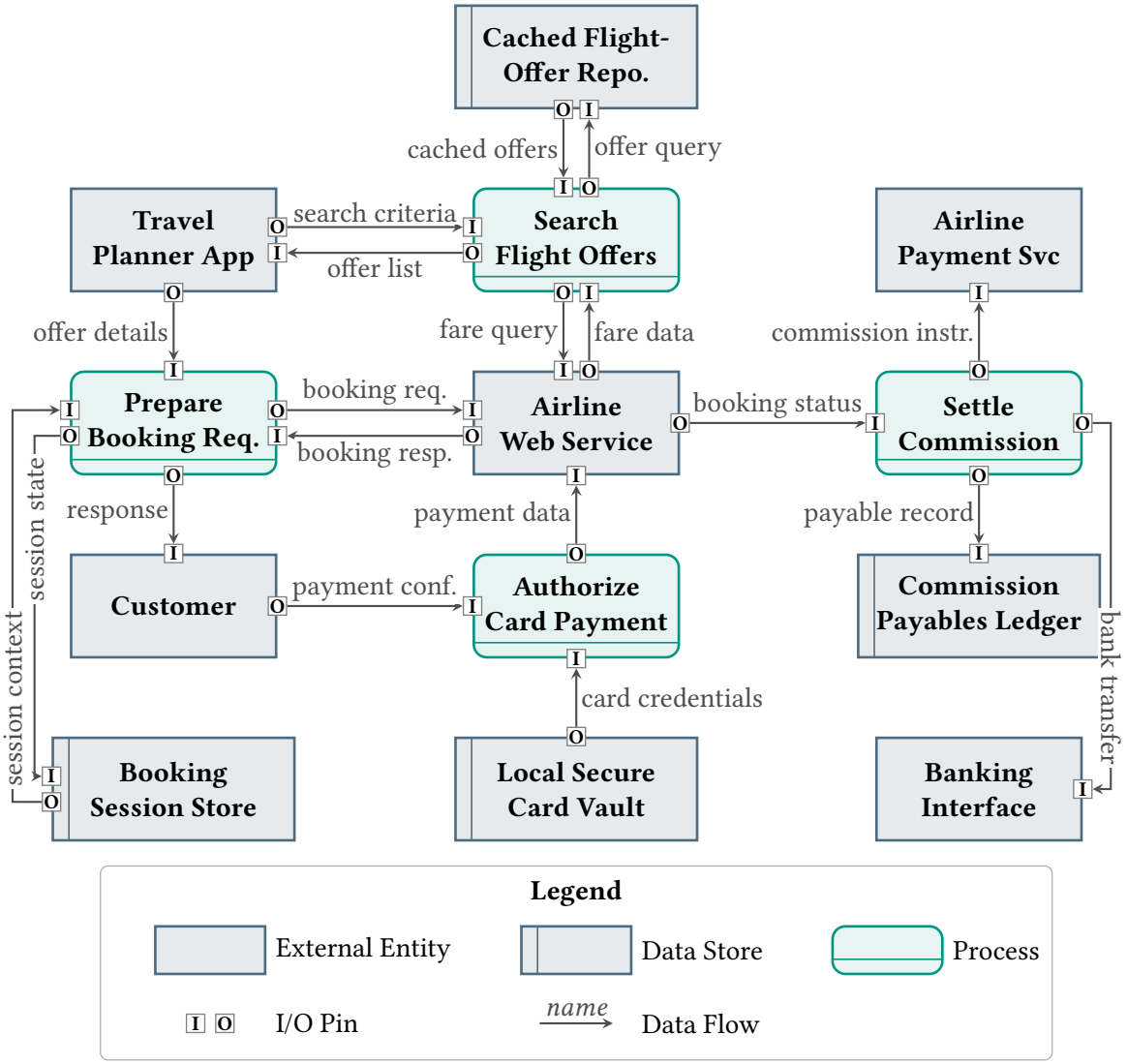


Figure 6.9.: Resulting DFD visualization in xDECAF for the Travel Planner scenario

6.3. Evaluation

Following the evaluation design outlined in Chapter 5, this section details how that design applies to RQ1. It instantiates the Goal Question Metric (GQM) plan [18] with goals, questions, and metrics for DFD quality assessment. It also describes the two-part user study through which we collect these metrics. We then present the results, discuss their implications, and close with a threats-to-validity assessment.

6.3.1. Goals, Questions and Metrics

Goal 1 defines the evaluation goal as assessing the quality of LLM-derived DFDs relative to those created by human experts, as posed by RQ1. We operationalize quality along two primary dimensions, *correctness* and *completeness*. This follows established evaluation criteria for LLM-generated artefacts [54, 55, 65]. Since DFDs admit multiple valid representations of the same requirements, we cannot assess either dimension against an absolute standard. Evaluation therefore depends on a human-baseline comparison. Expert ratings provide the basis for quality judgements. **Question 1.1** and **Question 1.2** address correctness and completeness as the primary quality dimensions. **Question 1.3** provides a complementary perspective by examining what types of issues occur in each group. This captures qualitative differences that quantitative ratings alone cannot distinguish. Finally, **Question 1.4** addresses rating reliability. Reliable ratings are essential for interpreting the validity of the expert ratings on which the comparison depends.

Metric 1.1.1 and **Metric 1.2.1** capture correctness and completeness using expert Likert ratings [68] applied to both LLM-derived and expert-created DFDs. This enables a direct group comparison based on the same criteria. **Metric 1.3.1** measures the prevalence of each issue type (missing element, hallucination, syntactic error, and logic error) by recording which evaluators flagged its presence per DFD. It then compares these proportions between LLM-derived and expert-created DFDs. **Metric 1.4.1** quantifies rating reliability for the Likert dimensions using Krippendorff's alpha (interval scale) [63]. We choose this metric for its ability to handle ordinal and interval data, unequal numbers of raters per item, and missing observations. **Metric 1.4.2** extends this to issue-category flagging by reporting raw percent agreement alongside Krippendorff's alpha (nominal scale). Both are reported because Krippendorff's alpha for binary outcomes is sensitive to category prevalence [41]. The raw agreement rate makes the full picture of rater agreement visible regardless of the prevalence distribution. Figure 6.10 provides a visual overview of the instantiated GQM plan.

6.3.2. Evaluation Design

As outlined in Chapter 5 and illustrated in Figure 5.3, the evaluation design for RQ1 relies on an online survey that consists of two parts. For both parts, we recruited participants from our academic and professional networks. We targeted individuals with prior experience

Goal 1: Evaluating the quality of LLM-derived DFDs compared to expert-created DFDs.

Question 1.1: *(Correctness)* How does the correctness of LLM-derived DFDs compare to that of expert-created DFDs with respect to the functional requirements?

Metric 1.1.1: Expert Likert rating (1-5) for semantic correctness of each DFD (LLM-derived and expert-created) in relation to the provided functional requirements, enabling group comparison.

Question 1.2: *(Completeness)* How does the completeness of LLM-derived DFDs compare to that of expert-created DFDs with respect to the functional requirements?

Metric 1.2.1: Expert Likert rating (1-5) for completeness of each DFD (LLM-derived and expert-created) in relation to the provided functional requirements, enabling group comparison.

Question 1.3: *(Detected Issues)* Which types of quality issues are identified in LLM-derived DFDs compared to expert-created DFDs, and how prevalent is each issue type across evaluators?

Metric 1.3.1: Proportion of evaluators flagging each issue type (missing element, hallucination, syntactic error, logic error) for LLM-derived vs. expert-created DFDs.

Question 1.4: *(Rating Reliability)* To what extent do experts agree in their assessment of DFD quality?

Metric 1.4.1: Krippendorff's alpha (interval scale) among experts on correctness and completeness ratings.

Metric 1.4.2: Raw percent agreement and Krippendorff's alpha (nominal scale) for inter-rater reliability of issue-category flagging per DFD.

Figure 6.10.: GQM plan for the evaluation of RQ1

in data flow analysis, software design, or threat modelling. In *Part 1*, a group of experts independently create DFDs based on the same functional requirements. Their submissions form the human baseline for comparison. In parallel, the LLM generates its own DFDs from the same requirements using the pipeline described in Section 6.2. We set the number of LLM-derived DFDs to four to keep the evaluation workload in *Part 2* feasible. This trades a larger sample for a manageable per-participant review time. Before creating their DFDs, participants generate a participant ID for subsequent use and complete a self-assessment covering their experience with DFDs and threat modelling, as well as their professional role. We collect the self-assessment before the task to avoid retrospective bias [109]. This ensures that self-reported skill levels reflect prior experience rather than impressions formed during the task. Next, participants receive a running example concretely illustrating the requirements and expected DFD elements. This ensures a shared understanding of the task before they begin. Figure 6.11 shows this example from the xDECAF documentation[1]. After the running example, participants receive the task instructions including the functional requirements for a hypothetical online bookstore, as shown in Figure 6.12. To minimize

[1] `https://dataflowanalysis.org/examples/models/dfd/cma.html`

tool-specific bias, experts are free to use any DFD tool of their choice, provided they adhere to the supplied guidelines. After participants upload their DFD, *Part 1* closes with a brief feedback survey on perceived task difficulty.

Running Example

Use Case Description: Conference Management App (CMA)

The Conference Management App (CMA) is a secure platform used by academic communities to manage the submission and evaluation of research papers. The system must process three types of users: Authors, Reviewers, and the Conference Chair.

Core Functional Requirements:

1. Profile & Consent Management: When any user accesses the CMA, the system retrieves their specific permissions and role data from the user store.
   Users can submit privacy updates to change their consent settings, which the system then records back into the consent registry.
2. Search & Retrieval: Any user can provide a search query (e.g. title or keyword). The system queries the manuscript repository to find matching entries and returns paper details to the user for reading.
3. Paper Submission: An author submits a manuscript package containing the document, conference id, and co-author names. The system initializes this as a new entry in the manuscript repository with an unpublished status.
4. Reviewer Assignment & Conflict Check: The conference chair requests a conflict analysis for a specific paper. The system reads the manuscript repository and compares it against a list of conflict metadata (e.g., past co-authorships). It then generates a candidate reviewer list for the chair. The chair then sends a reviewer assignment to the system, which is stored in the review database.
5. Peer Review: An assigned reviewer evaluates the paper and submits a review report (score and comments). The system records this report in the review database.
6. Final Decision: The conference chair retrieves the consolidated review reports for a paper and issues a final decision (accept or decline). The system updates the paper's status in the manuscript repository and notifies the author of the outcome.

Calibration: Exemplary extracted DFD for CMA Use Case

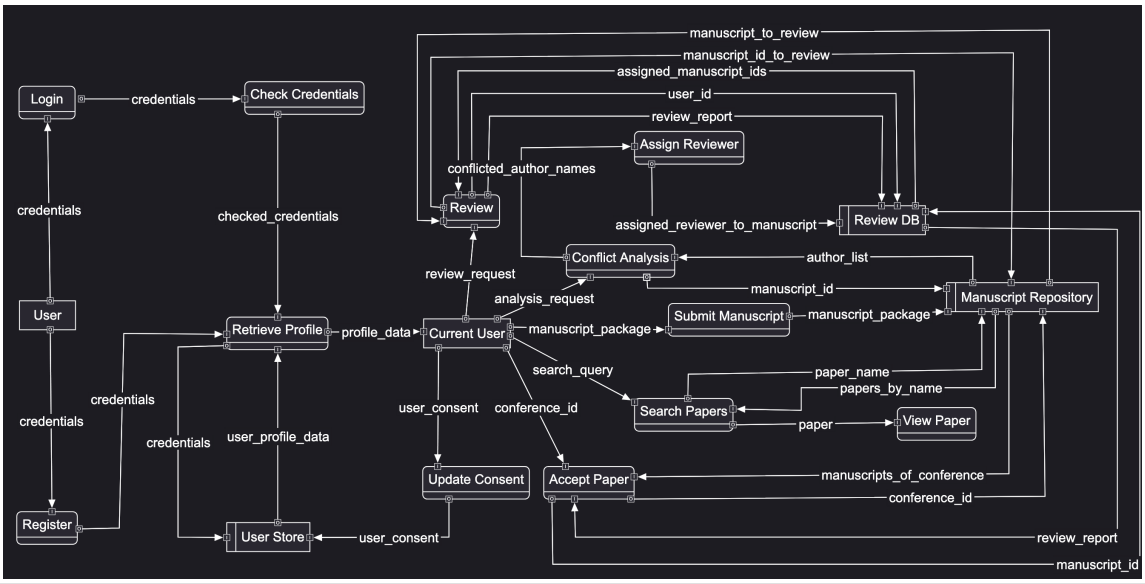


Figure 6.11.: Running example provided to participants in *Part 1* and *Part 2* of user study 1

In *Part 2*, the same participants evaluate the quality of the collected DFDs in a *blind study* setup: they know they will see a mix of LLM-derived and expert-created DFDs, but do not know which is which. We require participants to re-enter their participant ID before the evaluation begins. This ensures that no one evaluates their own DFD and eliminates a direct source of evaluation bias. After participants review the running example from *Part 1*, each rates four DFDs in a random order: two LLM-derived and two expert-created. The SoSci Survey platform determined this assignment at session start by randomly drawing from the DFD sets with the fewest ratings received so far. This ensured even coverage across

The "Chapters" Online Bookstore

The "Chapters" Online Bookstore is an automated platform designed to manage the purchasing process for customers and the fulfillment process for warehouse staff. The system serves as a central hub connecting online shoppers with inventory management and external payment services.

Core Functional Requirements:

1. Search & Browse: The customer sends search criteria (e.g., title or author) to the system. The system queries the inventory database to find matching entries and returns book details (price, availability) to the customer.
2. Order Processing & Payment: When a customer decides to buy, they submit order & payment details to the system. The system must first send the payment data to the external payment service. The payment service processes the transaction and returns a payment status (confirmation or rejection).
3. Finalize Transaction: If the payment is confirmed, the system performs two actions: it stores the record in the order database as a transaction entry and sends an order receipt back to the customer.
4. Inventory Stock Update: Once an order is successfully stored, the system automatically sends an inventory update to the inventory database to deduct the quantity of books sold from the current stock levels.
5. Order Fulfillment: The warehouse manager requests pending shipping orders from the system. The system retrieves these from the order database. The manager then uses this information to pack and ship the physical items to the customer.

Derive a complete Data Flow Diagram including all processes, external entities, data stores and main flows of the system.

Figure 6.12.: Task description for participants in *Part 1* of user study 1

all DFDs regardless of final participant numbers, since pure randomisation only balances reliably at large counts. We also randomized the presentation order within the assigned set of four per participant to distribute order effects across DFDs. Participants assess each DFD on correctness and completeness using the metrics defined in section 6.3.1. The survey displays the DFD being evaluated alongside the original functional requirements. This ensures participants can compare against the specification at all times. Figure 6.13 shows the rating interface presented to participants in *Part 2*. We chose four DFDs to balance coverage against cognitive load [109]. Presenting too many risks fatigue-driven rating drift [61], while too few limits the statistical power of the comparison [109]. We finalize each rating upon submission, preventing participants from revising earlier assessments in response to later DFDs. This avoids anchoring effects [102]. *Part 2* closes with a feedback question on the aspects participants focused on during evaluation. The data companion set [46] contains all study materials, including task instructions, rating scales, and consent forms.

Task: DFD Quality Assessment

Below, you are presented with a Data Flow Diagram (DFD) and the Use Case Description from which it was derived.

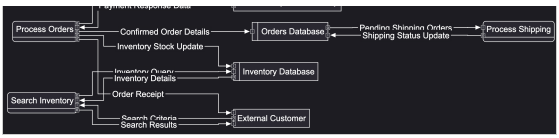
Instructions:

1. **Consult the Ground Truth:** Use the provided text as the absolute reference. Base your ratings **STRICTLY** on this description, not on your personal knowledge of bookstores.
2. **Identify Errors:** Check for Omissions (information in the text missing from the DFD) and Hallucinations (information in the DFD not found in the text).
3. **Standardized Rating:** Answer the questions below to evaluate the diagram's completeness, correctness, and overall quality.

Note: Each diagram has been produced by a different source (Human or AI). Sources remain anonymous to ensure an objective evaluation.

Data Flow Diagram

(Click image to zoom)



Use Case Description

The "Chapters" Online Bookstore

The "Chapters" Online Bookstore is an automated platform and the fulfillment process for warehouse staff. The system serves as a central hub connecting online shoppers with inventory management and external payment services.

Core Functional Requirements:

[… functional requirements omitted, see Figure 6.12]

10. Rate the DFD based on the following dimensions:

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Completeness: The diagram includes all relevant elements that I perceive as important.	○	○	○	○	○
Correctness: The elements included in the diagram accurately reflect the requirements.	○	○	○	○	○
Confidence: I am confident in my ability to accurately judge this specific diagram.	○	○	○	○	○

11. What Issues did you identify?

Please flag any identified issues (select all that apply)

- ☐ Missing Element: Something in the text is missing in the diagram.
  - Please specify...
- ☐ Hallucination: Something is in the diagram that was NOT in the text.
  - Please specify...
- ☐ Syntactic Error: (e.g., a data flow directly between two data stores).
  - Please specify...
- ☐ Logic Error: (e.g., data flowing the wrong direction).
  - Please specify...
- ☐ Vague Labeling: Process or flow names are unclear.
  - Please specify...
- ☐ Other: (Please specify in the text box below).

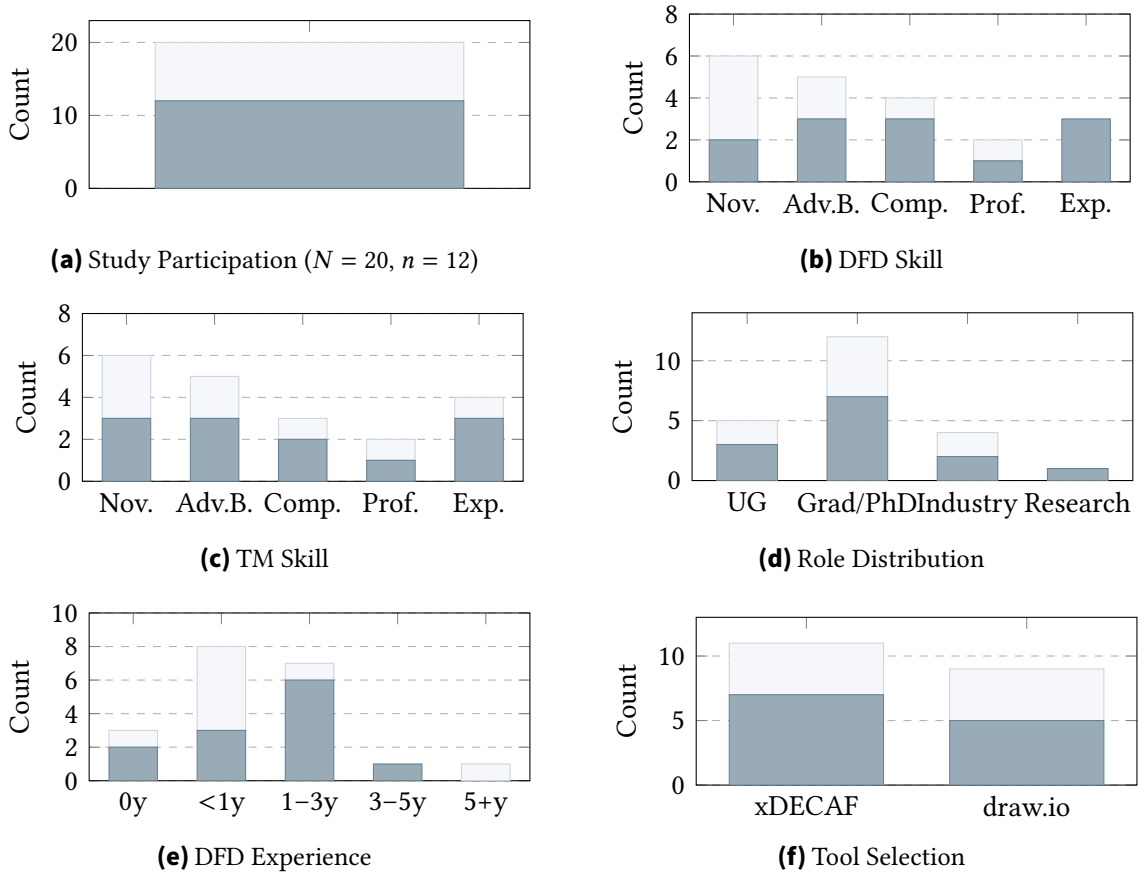
12. Any additional thoughts/insights you have?

Notes

Figure 6.13.: *Part 2* evaluation interface presented to participants in user study 1

6.3.3. Results

Having described the study design, we now present the data collected from both parts. 20 participants completed *Part 1* and submitted a valid DFD. Of these, 12 also completed *Part 2*, evaluating a total of 19 DFDs (15 human-created, 4 LLM-derived). The pool of evaluated DFDs is smaller than the number of *Part 1* participants because we included only DFDs submitted before *Part 2* opened for evaluation. This ensures all evaluators drew from the same fixed pool of artifacts. *Part 1* recruitment remained open throughout *Part 2* to maximise participant numbers. Figure 6.14 summarises the demographics of all 20 *Part 1* and 12 *Part 2* participants. We provide the full per-participant breakdown in Table A.1.



(a) Study Participation ($N = 20$, $n = 12$)

(b) DFD Skill

(c) TM Skill

(d) Role Distribution

(e) DFD Experience

(f) Tool Selection

Figure 6.14.: Participant demographics for *User Study 1* ($N = 20$ *Part 1*, $n = 12$ also completed *Part 2*). DFD = Data Flow Diagram; TM = Threat Modelling. Role counts exceed N as participants could select multiple roles.

As shown in Figure 6.14, self-assessed DFD skill spans the full five-point scale across all 20 *Part 1* participants. Novice is the most frequent level ($n = 6$). Of the 12 who also completed *Part 2*, Novices were fewer ($n = 2$). This suggests that participants with higher self-assessed DFD skill were more likely to complete both parts of the study. Graduate/PhD students

($n = 12$) dominate the role distribution across all *Part 1* participants. The remaining roles are Undergraduate students ($n = 5$), Industry Practitioners ($n = 4$), and Academic Researchers ($n = 1$). Prior DFD experience across all *Part 1* participants is concentrated at the lower end. The largest groups report no experience ($n = 3$) or less than one year ($n = 8$). Among the *Part 2* completers the most common band was one to three years ($n = 6$), consistent with the skill trend above. Tool use was split between xDECAF [22] ($n = 11$) and draw.io [37] ($n = 9$). No participant used PlantUML or another tool.

Table 6.4 summarises the self-assessment results collected after participants created their DFDs from the provided functional requirements. Participants rated the clarity of the use-case description highly ($Avg = 4.35$, $SD = 0.49$). They rated the mental demand of the task as moderately low ($Avg = 2.45$, $SD = 0.94$). Participants rated notation ($Avg = 3.65$, $SD = 0.81$), coverage ($Avg = 3.65$, $SD = 0.88$), and detail ($Avg = 3.50$, $SD = 0.83$) all moderately. This indicates a mixed level of confidence among participants in their created DFDs. Additionally, participants could provide open-text feedback on challenges they encountered. The responses are available in the appendix (Table A.2). Responses were generally brief. The most common themes were uncertainty about the level of detail to include in the DFDs and challenges related to the clarity of boundaries and interactions in the use-case description.

Item	Statement	Avg (SD)
Use-case clarity	The use-case description provided clear information about how data moves through the system.	4.35 (0.49)
Mental demand	It was mentally demanding to transform the text into a diagram structure.	2.45 (0.94)
Notation confidence	I am confident that I followed the correct DFD notation rules (e.g. using proper symbols).	3.65 (0.81)
Coverage confidence	I am confident that my diagram captures all the requirements mentioned in the text.	3.65 (0.88)
Detail confidence	I am confident that the requirements I modelled are represented in full detail.	3.50 (0.83)

Likert scale 1 (Strongly Disagree) – 5 (Strongly Agree). $N = 20$. SD = standard deviation.

Table 6.4.: *Part 1* creator self-assessment ratings after DFD creation

With the human baseline established in *Part 1*, *Part 2* yields the blind quality ratings on which the core comparison rests. Regarding **Metric 1.1.1** and **Metric 1.2.1**: human-created DFDs received higher average ratings on both dimensions. Completeness averaged $Avg = 3.96$ ($SD = 1.27$) for human-created DFDs versus $Avg = 2.46$ ($SD = 1.32$) for LLM-derived ones. Correctness averaged $Avg = 3.62$ ($SD = 1.21$) versus $Avg = 2.92$ ($SD = 1.18$). Table 6.5 summarises these results. Each participant rated two human-created and two LLM-derived DFDs, but the pool contained more human-created DFDs overall. Therefore, LLM-derived DFDs received more ratings on average ($Avg = 6.00$, $SD = 0.00$) than human-created DFDs ($Avg = 1.60$, $SD = 0.51$). This asymmetry means the two group averages carry different levels of precision: LLM-derived average are based on six ratings per DFD while human-

created averages rest on at most two ratings per DFD. The human group average is therefore considerably less stable, and readers should treat observed group differences as indicative rather than precise estimates of a quality gap.

Type	DFDs	Total Ratings	Ratings/DFD Avg (SD)	Completeness Avg (SD)	Correctness Avg (SD)	Confidence Avg (SD)
Human	15	24	1.60 (0.51)	3.96 (1.27)	3.62 (1.21)	4.46 (0.72)
LLM	4	24	6.00 (0.00)	2.46 (1.32)	2.92 (1.18)	4.42 (0.83)
Overall	19	48	2.53 (1.90)	3.21 (1.49)	3.27 (1.23)	4.44 (0.77)

Likert scale 1–5; *Avg* = average, *SD* = standard deviation.

Table 6.5.: Aggregated completeness, correctness, and confidence ratings by DFD origin.

At the individual DFD level, Figure 6.15 compares all 4 LLM-derived DFDs across the three rating dimensions. Completeness ranges from a low of $Avg = 1.83$ ($SD = 0.98$) for LLM_1 to a high of $Avg = 3.17$ ($SD = 1.47$) for LLM_4. Correctness similarly ranges from a low of $Avg = 2.33$ ($SD = 1.21$) for LLM_1 to a high of $Avg = 3.67$ ($SD = 0.82$) for LLM_4. In contrast, confidence ratings are consistently high across all LLM-derived DFDs. They range narrowly from $Avg = 4.00$ ($SD = 0.89$) for LLM_4 to $Avg = 4.67$ ($SD = 0.82$) for LLM_2. This suggests evaluators felt certain in their assessments regardless of the DFD's quality level.

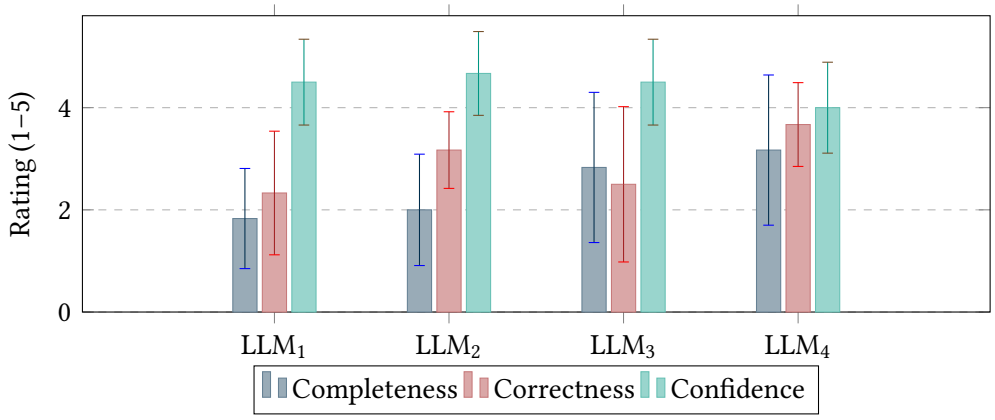


Figure 6.15.: Per-DFD ratings for the 4 LLM-derived DFDs (Likert scale 1–5; error bars show SD).

Human-created DFDs similarly showed variability across individuals, as expected given the mixed-expertise participant pool. We provide the full per-DFD breakdown for both groups in Table A.3.

Regarding **Metric 1.3.1**: evaluators flagged specific issues in each DFD and could optionally leave free-text comments. Table 6.6 lists the frequency of each issue type per diagram across all DFDs. LLM-derived DFDs had a higher average number of issues per evaluator ($Avg = 2.08$, $SD = 0.75$) compared to human-created DFDs ($Avg = 1.37$, $SD = 1.11$). The dominant issue types in LLM-derived DFDs were missing elements ($Avg = 0.88$ per evaluator) and hallucinations ($Avg = 0.50$ per evaluator). Human-created DFDs showed

lower absolute rates across all reported categories ($Avg = 0.27$ and $Avg = 0.20$ respectively). Syntactic errors were rare in both groups (LLM: $Avg = 0.12$, human: $Avg = 0.07$ per evaluator).

DFD	T	N	Miss.	Hall.	Syn.	Logic	Vague	Other	Tot.	/Eval
H_{126225}	H	1	0	0	0	0	1	0	1	1.00
H_{131244}	H	2	0	1	0	0	0	2	3	1.50
H_{216739}	H	1	1	0	0	0	1	0	2	2.00
H_{343962}	H	2	0	0	0	1	2	0	3	1.50
H_{356787}	H	2	1	0	1	0	0	0	2	1.00
H_{388389}	H	1	0	0	0	1	0	0	1	1.00
H_{671858}	H	2	1	0	0	1	1	1	4	2.00
H_{688508}	H	2	1	0	0	0	2	1	4	2.00
H_{719176}	H	2	0	1	0	1	0	1	3	1.50
H_{731262}	H	1	0	0	0	0	0	0	0	0.00
H_{770487}	H	2	1	0	0	0	0	2	3	1.50
H_{781453}	H	2	0	0	0	0	0	0	0	0.00
H_{835392}	H	1	0	1	0	0	0	0	1	1.00
H_{872246}	H	1	0	0	0	0	0	0	0	0.00
H_{876646}	H	2	2	2	1	1	2	1	9	4.50
LLM_1	L	6	6	4	1	2	2	1	16	2.67
LLM_2	L	6	6	2	1	1	2	1	13	2.17
LLM_3	L	6	4	5	1	1	2	2	15	2.50
LLM_4	L	6	5	1	0	0	0	0	6	1.00

T: H = Human, L = LLM. Miss. = Missing Element, Hall. = Hallucination, Syn. = Syntactic Error, Logic = Logic Error, Vague = Vague Labelling, O = Other. Tot. = sum of flagged issue types; /Eval = average per evaluator.

Table 6.6.: Per-DFD evaluator counts by issue type. Each cell reports how many evaluators flagged that issue type as present (binary flag per evaluator); N = number of evaluators per DFD.

Table 6.7 reveals further qualitative differences in the evaluator comments for each LLM-derived DFD. LLM_1 received the most flagged issues: all 6 evaluators identified missing elements and 4 identified hallucinations. One evaluator characterised it as having "half of the diagram missing". LLM_3 received a more mixed assessment: one evaluator noted that the DFD "reflects most parts of the diagram" and that the missing elements were not critical. LLM_4 had fewer issues than the others. One evaluator raised only a single marginal concern, prefacing it with "One could argue...". This suggests evaluators found no clear deficiencies in LLM_4 beyond that single marginal concern. One evaluator identified LLM_2 as "definitely AI". Its output apparently had structural or stylistic characteristics distinctive enough to infer its origin from the DFD content alone. We provide the corresponding counts by issue type and evaluator comments for the human-created DFDs in Table A.4 as additional reference.

Addressing **Metric 1.4.1**, we compute Krippendorff's alpha on the LLM-derived DFDs only, as human-created DFDs received only two ratings each, which is below the minimum for a

DFD	M	H	S	L	V	O	T	Evaluator Comments
LLM_1	6	4	1	2	2	1	16	*"It seems half of the diagram is missing; the person modelling this either hasnt had much experience or didn't understand the requirements of a Level 1 diagram correctly. There are modelling inconsistencies and a different naming syntax differentiation nodes and edges would have been nice."*
LLM_2	6	2	1	1	2	1	13	*"definitely AI / non-human ;D"*
LLM_3	4	5	1	1	2	2	15	*"How could this happen?"* *"I think this DFD cleanly reflects most parts of the diagram. It could maybe be more specific (e.g. divide search criteria in title and author flows) but overall I think it reflects the requirements perfectly"*
LLM_4	5	1	0	0	0	0	6	*"One could argue that "Record Order Transaction" and "Update Inventory Stock" can be summarized as one process, especially since the only purpose of "Update Inventory Stock" is to forwards data from "Record Order Transaction". There was no requirement to separate these processes."*

M = Missing, H = Hallucination, S = Syntactic Error, L = Logic Error, V = Vague, O = Other, T = Total.

Table 6.7.: Issue types and evaluator comments for each LLM-derived DFD. Each cell shows the number of evaluators who identified that issue type (binary flag per evaluator).

reliable alpha estimate. We apply an interval-scale metric function, which is appropriate for Likert-scale data [43]. Table 6.8 reports the results. Completeness ratings yielded $\alpha = 0.065$ (*unreliable*), while correctness ratings yielded $\alpha = 0.097$ (*unreliable*). Both values fall well below the threshold for reliable inter-rater agreement ($\alpha \geq 0.80$) and below the tentative threshold ($\alpha \geq 0.667$) as per Krippendorff [63]. This indicates disagreement in the absolute ratings assigned to the same DFD by different evaluators.

Dimension	Krippendorff's α	Interpretation	Items (N≥2)
Completeness	0.065	unreliable	4
Correctness	0.097	unreliable	4

Table 6.8.: Inter-rater reliability (Krippendorff's α, interval scale) for completeness and correctness ratings on LLM-derived DFDs ($N = 12$ raters, 4 DFDs; human-created DFDs excluded as each received only 2 raters). Thresholds: $\alpha \geq 0.80$ reliable, ≥ 0.667 tentative [64].

For **Metric 1.4.2**, Table 6.9 reports both raw pairwise percent agreement and Krippendorff's alpha (nominal scale) per issue category. The two measures are reported jointly because high category prevalence suppresses Krippendorff's alpha even when empirical agreement is substantial (the prevalence paradox) [41]. When most raters flag a category on most DFDs,

the chance-agreement baseline rises alongside the observed agreement. This leaves almost no room for alpha to register agreement above chance. With only four LLM-derived DFDs and near-universal flagging for categories such as missing elements, alpha is not informative as a standalone measure. The raw agreement rate provides the more interpretable figure in this context.

Issue Category	Agree.%	Krippendorff's α	Items (N≥2)
Missing Element	78.3%	0.051	4
Hallucination	56.7%	0.169	4
Syntactic Error	75.0%	-0.095	4
Logic Error	70.0%	-0.035	4
Vague Labeling	60.0%	-0.022	4
Other	70.0%	-0.035	4

Table 6.9.: Raw pairwise percent agreement and Krippendorff's α (nominal scale) per issue category for LLM-derived DFDs ($N = 12$ raters, 4 DFDs). Agreement is at category level only; raters flagging the same category may have identified different specific instances. Alpha thresholds: reliable ≥ 0.80, tentative ≥ 0.667, unreliable < 0.667. [64]

At the close of *Part 2*, participants self-reported on study experience and execution quality via three items, summarised in Table 6.10. On enjoyment, 9 of 12 participants responded positively (3 "yes", 6 "rather yes"), while 3 responded "rather not". Regarding task completion, 11 participants confirmed completing all tasks as instructed, while one reported partial completion with uncertainty. Results on attention span show that 7 participants completed the questionnaire without distraction, 4 reported being distracted once, and 1 reported being distracted several times.

Finally, Table 6.11 breaks down the ratings for human-created DFDs by creator background: DFD skill, Threat Modelling skill, and DFD experience. Completeness and correctness averages range from approximately 2.3 to 4.7 across all creator subgroups, with no clear trend by skill or experience level. Expert-level creators' DFDs scored the lowest on completeness in both the DFD Skill (2.33) and TM Skill (2.43) groupings. Confidence shows no clear monotonic trend: DFD Skill Expert creators' DFDs scored the highest (5.00) while those by Competent creators scored the lowest (3.60).

The data companion set [46] contains the raw study data and the measured metrics.

Item	Response	N
SC03	Yes	3
	Rather yes	6
	Rather not	3
SC04	Completed all tasks as instructed	11
	Partially / with uncertainty	1
SC05	No distractions	7
	Distracted once	4
	Distracted several times	1

SC03: *Did you like participating in this study?* **SC04**: *Have you carried out all the tasks as described?* **SC05**: *Could you complete the questionnaire without being distracted?* $N = 12$ Part-2 participants.

Table 6.10.: *Part 2* participant self-report on enjoyment (SC03), task completion (SC04), and attention (SC05).

Grouping	Level	N_c	N_{rat}	Completeness	Correctness	Confidence
DFD Skill	Novice	6	10	4.70	4.20	4.80
	Adv. Beginner	3	4	4.50	4.25	4.50
	Competent	3	5	3.20	2.60	3.60
	Proficient	1	2	3.50	3.50	4.00
	Expert	2	3	2.33	2.67	5.00
TM Skill	Novice	5	10	4.60	4.00	4.80
	Adv. Beginner	4	5	4.60	4.20	3.80
	Competent	2	2	4.50	3.50	4.50
	Expert	4	7	2.43	2.71	4.43
DFD Experience	0 years	3	6	4.67	3.83	4.83
	<1 year	6	9	4.67	4.33	4.33
	1-3 years	5	7	2.57	2.57	4.43
	5+ years	1	2	3.50	3.50	4.00

N_c = creator DFDs in subgroup; N_{rat} = ratings received. Ratings on human-created DFDs only; subgroup sizes are very small.

Table 6.11.: Average completeness, correctness, and confidence ratings received by human-created DFDs, grouped by creator DFD skill, TM skill, and DFD experience. Results are exploratory only.

6.3.4. Discussion

This section interprets the results in relation to participant composition, the quality gap between LLM-derived and human-created DFDs, inter-rater reliability, and the validity boundaries of the observed findings. The participant pool spans the full self-assessed skill spectrum for both DFD creation and threat modelling. Most participants reported either no prior DFD experience ($n = 3$) or less than one year ($n = 8$). This composition reflects the realistic range of practitioners who create and evaluate DFDs in practice: a pool restricted to experts would set an unrealistically high human baseline, while a mixed pool yields a comparison closer to real-world conditions. The academic dominance reflects the recruitment setting and limits generalisability to industrial contexts, though the inclusion of 4 Industry Practitioners partially offsets this limitation. The experience subgroup analysis (Table 6.11) reveals a counterintuitive pattern: DFDs by higher-skill creators scored lower on completeness and correctness than those by lower-skill creators. However, with subgroup sizes as small as $N_c = 1$, we cannot reliably interpret this pattern: it likely reflects sampling noise rather than a true skill effect. We therefore treat the pooled ratings as a pragmatic baseline while acknowledging that the skill composition of creators introduces variation that the small subgroup sizes prevent us from quantifying. All participants used either xDECAF or draw.io. xDECAF enforces a formal DFD grammar, while draw.io imposes no structural constraints. This may introduce variation in the structural quality of human-created DFDs independent of participant skill. Restricting all participants to the xDECAF tool would have ensured structural consistency but may have introduced tool-learning overhead and reduced participation. Allowing tool choice therefore represents a trade-off between instrument consistency and study accessibility.

Beyond tool choice, the *Part 1* self-assessment supports the validity of the study setting. The high use-case clarity rating indicates participants understood the functional requirements as intended. The moderately low mental demand score suggests the task was cognitively accessible across skill levels without being trivial. Moderate confidence ratings for notation, coverage, and detail are consistent with the mixed-expertise pool and leave meaningful room for quality variation across human-created DFDs. The per-DFD rating spread observed in *Part 2* confirms this variation. The *Part 2* enjoyment responses reinforce this: 9 of 12 evaluators responded positively (Table 6.10). Therefore, the evaluation workload was manageable, with 11 of 12 participants confirming they completed all tasks as instructed.

Regarding **Question 1.1** and **Question 1.2**: human-created DFDs received higher average ratings than LLM-derived ones on both dimensions, though both gaps carry higher uncertainty for the human group due to the asymmetric rater counts. An additional factor limiting causal interpretation is that all *Part 2* evaluators had previously derived their own DFDs from the same requirements in *Part 1*. Their ratings may therefore reflect alignment with their own prior solution rather than objective quality. If human-created DFDs cluster closer to typical solution patterns than LLM outputs do, this anchor effect could inflate the observed quality gap independently of actual pipeline limitations. The average correctness gap is approximately 0.7 Likert points while the completeness gap is approximately 1.5 points, indicating that coverage of requirements is the primary shortfall. Within the LLM-

derived group, quality varies across both dimensions. LLM_4 approaches human correctness ($Avg = 3.67$) and completeness ($Avg = 3.17$) levels. LLM_1, by contrast, falls well short ($Avg = 2.33$ and $Avg = 1.83$ respectively). This within-group spread is consistent with the non-deterministic nature of LLM outputs. The same pipeline configuration produced all four DFDs from the same requirements. Therefore, the variability reflects output variance across runs rather than any structural difference in the inputs.

Regarding **Question 1.3**: missing elements were the dominant issue in LLM-derived DFDs, flagged by between 4 and 6 evaluators per DFD. Hallucinations were the second most common type. Syntactic errors were rare across both groups. Human-created DFDs show a qualitatively different profile. Vague labelling is the dominant concern rather than missing elements or hallucinations. Human and LLM errors therefore concentrate in distinct areas. LLM_4 again stands out: evaluators identified on average only 1 issue type, compared to 2.17–2.67 for the other three. This reinforces the within-group variability observed in the Likert ratings.

Regarding **Question 1.4**: the low inter-rater reliability for completeness ($\alpha = 0.065$) and correctness ($\alpha = 0.097$) reflects evaluator disagreement about absolute quality levels rather than a methodological error. This is consistent with the acknowledged subjectivity of DFD evaluation: multiple valid representations can exist for the same requirements, and there is no single ground truth against which raters can anchor their judgements. We therefore urge caution when interpreting absolute ratings: with $\alpha = 0.065$ for completeness and $\alpha = 0.097$ for correctness, the group averages carry wide uncertainty. Directional comparisons remain meaningful provided the disagreement is not systematically biased by rater background, which the subgroup analysis provides exploratory support for. We did not formally test whether this directional ordering is consistent across individual raters, which remains an additional source of uncertainty. However, this does not address the creator-rater confound. All *Part 2* evaluators previously created their own DFD in *Part 1* and may therefore rate artefacts that match their own solution more favourably than those that diverge from it. Future studies would need a larger sample of DFD outputs and more raters per item to reduce this uncertainty.

Beyond being a power limitation, the near-zero inter-rater reliability raises a construct validity question independent of sample size. We remain uncertain whether any rating instrument can reliably operationalize correctness and completeness for DFD artifacts that admit multiple valid solutions. Here, evaluators are individually confident yet collectively disagree. This suggests the constructs themselves may be evaluator-relative rather than artifact-intrinsic. Future work should explore whether reference solutions, explicit scoring rubrics, or functional evaluation criteria can yield more reliable quality measures for this artifact type. A concrete candidate is a functional criterion: whether the DFD enables correct violation detection when fed into xDECAF.

By contrast, issue-category flagging, yields more interpretable agreement. It is a binary judgement of whether a given issue type is present or absent per DFD. Raw pairwise agreement of 56.7%–78.3% indicates moderate to substantial empirical agreement. The corresponding nominal-scale alpha values are near zero or negative, but this reflects the prevalence paradox (see Section 6.3.3) rather than true disagreement. The raw rates therefore

indicate that evaluators showed greater consensus in identifying issue types than the Likert ratings imply.

Regarding the overall **Goal 1**: LLM-derived DFDs fall below human-created ones on both completeness and correctness. The completeness gap is more pronounced, though the human group averages carry higher uncertainty due to fewer ratings per DFD. The dominant issue types reveal where the quality gap concentrates. Missing elements and hallucinations dominate LLM-derived DFDs, while vague labelling dominates human-created ones. The LLM thus captures structural patterns well enough to keep syntactic errors rare ($Avg = 0.12$ per evaluator) but struggles with coverage and factual grounding. The near-zero inter-rater reliability ($\alpha = 0.065$ for completeness, $\alpha = 0.097$ for correctness) also limits the precision of both group averages. This reflects the inherent subjectivity of DFD evaluation. Since DFDs admit multiple valid representations, raters have no single ground truth to anchor their judgements. Despite this uncertainty, completeness ratings, correctness ratings, and issue type presence all indicate the same relative ordering: LLM_4 performs best and LLM_1 worst. Output non-determinism likely explains the within-group spread: the same pipeline configuration produced all four DFDs from the same requirements. The pipeline aimed to mitigate missing elements and hallucinations. Remaining issues may therefore partly stem from the deliberate trade-off introduced by the requirements clarification step in the data pre-processing phase. It allows the LLM to fill gaps in the specification at the cost of an increased hallucination risk. Similarly, the atomicity gate in the semantic clustering phase flagged all four process clusters as non-atomic at maximum iteration (Section 6.2.4). The gate thus cannot yet distinguish decomposable clusters from intentionally aggregated Level-1 processes. Additionally, the topology rules in the wiring phase remain prompt-enforced rather than schema-validated, so violations propagate silently to the final DFD. Together, these limitations explain why LLM-derived DFDs are not yet a reliable substitute for human-created ones. Targeted refinements to the atomicity gate and topology validation are direct levers for closing the remaining quality gap. LLM_4, however, already shows this is achievable: it received quality ratings comparable to those of human-created DFDs on both dimensions ($Avg = 3.17$ for completeness, $Avg = 3.67$ for correctness).

6.3.5. Threats to Validity

Where the preceding discussion interprets the results, this subsection addresses methodological threats. We structure it using the four validity categories of Wohlin et al. [109]: conclusion, internal, construct, and external validity.

Conclusion Validity The most significant threat to conclusion validity is the small number of LLM-derived DFDs ($N = 4$). This precludes reliable inferential comparison between groups and limits claims about the magnitude of the quality gap to descriptive observations. Within-group variability across the four LLM-derived DFDs further suggests that a larger sample would not necessarily yield more stable quality estimates. The spread likely reflects output variability rather than sampling noise. Given the near-zero inter-rater reliability and small sample, no formal power analysis can determine whether the observed quality

gap reflects a true population effect. With 12 evaluators and high within-rater variance, the study could only have detected very large effect sizes. The observed completeness gap of approximately 1.5 points may sit near this threshold, while the correctness gap of approximately 0.7 points likely falls below it.

Rater count asymmetry compounds this: each LLM-derived DFD received six ratings while human-created DFDs received at most two. This precision asymmetry may overstate the apparent clarity of the quality gap. Low inter-rater reliability for the Likert dimensions ($\alpha = 0.065$ for completeness, $\alpha = 0.097$ for correctness) further limits the precision of the group averages. High evaluator disagreement inflates the variance around each point estimate.

Implementation reliability is a further concern: the LLM pipeline is non-deterministic. The spread across LLM_1–LLM_4 already demonstrates this: the same pipeline configuration produced DFDs with substantially different quality profiles on the same requirements. These four outputs therefore represent one specific realisation of the pipeline rather than a stable characterisation of its behaviour. The observed quality gap may partly reflect which particular outputs the pipeline generated.

Beyond pipeline variability, online, uncontrolled survey conditions expose ratings to variation in participant attention across sessions. Self-report data (Table 6.10) confirms this threat materialised: 5 of 12 evaluators reported at least one distraction, and one participant reported partial task completion with uncertainty. Both likely introduced noise into a subset of ratings.

Internal Validity The blind study design mitigates direct evaluation bias: raters evaluated DFDs without knowing their origin. Randomising the presentation order additionally counteracts anchoring effects and attention drift across the session. However, one evaluator identified LLM_2 as "definitely AI" based on its content. Some LLM-derived DFDs may therefore carry structural or stylistic signals that allow raters to infer their origin. If raters detect such signals, prior expectations about LLM quality rather than the DFD content alone may skew their ratings.

A selection bias arises from the volunteer recruitment: self-selected participants from academic and professional networks tend to be more motivated and domain-engaged than the general population of DFD practitioners. The dropout pattern from *Part 1* to *Part 2* further compounds this: as shown in Figure 6.14, *Part 2* completers skewed toward more experienced participants. *Part 2* completers skewed toward experience: novices accounted for only $n = 2$, while the one-to-three-year band accounted for $n = 6$.

Sequential evaluation within each *Part 2* session introduces a maturation concern: rating quality may degrade due to fatigue or improve due to learning across the session. Randomisation of presentation order distributes this effect across DFDs rather than eliminating it.

Finally, we have no control over whether participants communicated about the study before or during participation. Shared evaluation criteria or primed quality concerns from such communication could reduce the independence of ratings.

Construct Validity Because DFDs admit multiple valid representations of the same requirements, neither completeness nor correctness admits an objective reference: no ground-truth DFD exists to compare against. Expert Likert ratings are therefore not a simplification of an objective measure but the only viable option available. They capture perceived quality relative to each rater's mental model of what a correct and complete DFD should look like. Relatedly, we did not accompany either construct with a formal scoring rubric. Raters therefore operationalised completeness and correctness independently. This lack of a shared pre-operational explication directly contributes to the low inter-rater reliability and weakens the claim that ratings capture a single well-defined quality dimension. Issue detection is similarly subjective. Two raters flagging the same category may have identified different specific instances, and we did not formally standardise the threshold for what constitutes a reportable issue. Using both Likert ratings and issue profiles as independent measures partially mitigates mono-method bias. LLM-derived DFDs rank consistently across both instruments, providing cross-measure validation of the observed quality differences.

Quality ratings measure the final DFD as a whole and cannot reveal which pipeline phase introduced a given defect, leaving the phase-level dimension of quality unobserved. Isolating individual phase contributions would require controlled ablation studies rather than a quality rating survey.

Rater behaviour introduces two further threats. The interaction of testing and treatment creates a creator-rater confound: participants created DFDs in *Part 1* before evaluating them in *Part 2*. This may have sensitised them to quality criteria and led to more critical ratings than those of a naive evaluator who had not engaged in the same task. In particular, evaluators may rate artifacts that align with their own derivation choices more favourably than those that take a different approach, independently of objective quality. Finally, knowing that the evaluation contained a mix of human and LLM-derived DFDs, some participants may have attempted to identify the origin of each DFD. Those who succeeded may have adjusted their ratings based on prior expectations about LLM quality rather than the content alone. The "definitely AI" comment for LLM_2 confirmed this hypothesis-guessing effect at least once.

External Validity Participant demographics limit generalisability to industrial practice. Most participants came from an academic background: 12 of 20 were Graduate or PhD students. This may not reflect the demographics of practitioners who create and evaluate DFDs in industry.

Beyond participant composition, the study covers a single hypothetical use case: an online bookstore. It evaluates one LLM configuration, GPT-5.3-Codex [76], with one prompting setup. Results may therefore not generalise to other contexts, use cases, or LLM configurations. Observed quality differences are also specific to this requirements context. Different specifications, particularly more complex or ambiguous ones, may produce a different quality profile. Expanding to multiple use cases or model configurations without a proportional increase in participants would dilute the already-small per-DFD rating counts further. Each additional DFD per evaluator conflicts with the cognitive load constraint that shaped the

study design. Improving generalisability therefore requires a larger participant pool rather than simply adding more inputs or configurations.

Study conditions also diverge from industrial practice in several ways. Practitioners evaluate DFDs within a broader threat modelling workflow and hold prior domain knowledge of the system being modelled. Study evaluators, by contrast, encountered the requirements for the first time at evaluation time and assessed DFDs in a blind format using Likert scales. Neither condition reflects how practitioners judge DFD quality in the field.

Finally, we evaluate the pipeline on one model configuration at a specific point in time, and LLM capabilities evolve rapidly. Quality levels observed with one model version may not hold for later versions or current alternatives. The absolute results are therefore time-sensitive in a way that purely human studies are not. Readers should not assume these findings hold for requirements with higher ambiguity, team-scale deployment settings, or LLM backends other than GPT-5.3-Codex [76]. As a proprietary frontier model, GPT-5.3-Codex may not remain accessible to independent researchers seeking to replicate these results.

Downstream validation is also absent: we have not tested whether LLM-derived DFDs, when fed into xDECAF, produce meaningful compliance analyses or propagate errors that compromise violation detection.

7. Automated Derivation of Privacy Specifications

This chapter addresses RQ2, which concerns the automatic translation of privacy-relevant legal requirements into machine-interpretable privacy specifications (see Figure 7.1). We contribute a multi-phase Large Language Model (LLM)-based pipeline that automatically derives these specifications from natural language legal requirements. The resulting structured annotations serve as input to the privacy analysis pipeline that Chapter 5 introduces [22]. As in Chapter 6, the *Travel Planner* scenario from Chapter 4 grounds the approach throughout. It supplies privacy-relevant legal requirements as input, illustrating how the pipeline derives the corresponding specifications. The chapter proceeds as follows: Section 7.1 develops the conceptual design, Section 7.2 covers the implementation, and Section 7.3 presents the evaluation and results.

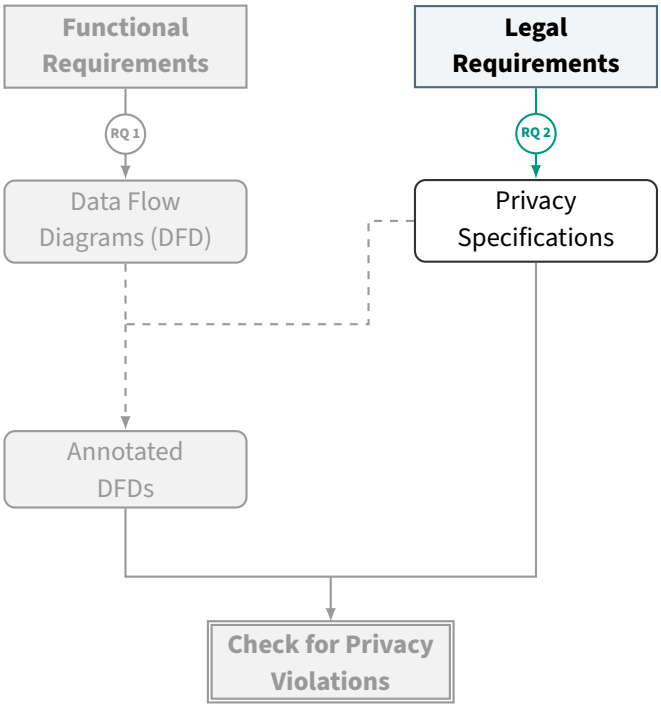


Figure 7.1.: Automated privacy analysis pipeline overview – RQ2 cutout

7.1. Concept

The xDECAF framework for Data Flow Diagram (DFD)-based privacy analysis [22] requires two distinct types of machine-interpretable annotations to enable automated privacy compliance checking. The first type, *labels*, attaches privacy-relevant metadata to data flows and nodes in the DFD. The second type, *constraints*, specifies the logical conditions that a flow must satisfy to comply with applicable legal requirements. These two types differ not only in form but in the nature of what they represent. Labels capture *what* privacy-relevant facts characterise a given DFD element, while constraints capture *how* those facts relate as verifiable conditions. This distinction motivates treating them as separate derivation targets. Although the input already derives from privacy law as software-level requirements, natural language alone is not sufficient for automated analysis. Thus, the pipeline requires a further transformation into structured, machine-processable form before it can apply them for DFD annotation or compliance checking. The conceptual pipeline developed in this section addresses this through four phases, as Figure 7.2 illustrates. The distinct nature of labels and constraints, combined with the limitations of current LLM capabilities, motivates this multi-phase structure. Deriving both labels and constraints in a single step would exceed the effective attention range of the model [69] and increase hallucination risk [111]. Instead, separating the derivation into focused phases reduces task complexity per phase and allows the output of each phase to serve as structured context for the next [14, 110]. Specifically, *Requirements Filtering* selects the privacy-relevant requirements from the input set. Following this, *Label Derivation* extracts privacy-relevant metadata from the filtered requirements and structures it into labels. Then, *Schema Aggregation* clusters and refines the derived labels into a global privacy schema. Finally, *Constraint Derivation* derives logical constraints from the finalized label set for use in privacy violation detection within the xDECAF framework. Across all phases, the pipeline applies the prompt engineering techniques and output validation mechanisms described in Chapter 5. Each phase depends on the output of the previous one, forming a sequential derivation chain from raw requirements to a complete privacy specification.

7.1.1. Requirements Filtering

Not all software-level privacy requirements are equally relevant for DFD annotation and analysis. Requirements addressing personal data handling, consent, or data retention are directly useful for deriving labels and constraints. Purely functional or performance-related requirements are not: they describe the connections and data flows that the DFD already captures, not the privacy-relevant properties of the data. Additionally, some requirements may pertain to privacy at runtime or organizational processes, rather than the static data flows of design-time DFD analysis. Passing the full unfiltered set to subsequent phases introduces noise that distorts label derivation and reduces the precision of the derived specifications [33, 111]. As the adaptation specific to the DFD context, this phase classifies each requirement not only for privacy relevance but also for annotability at design time. The resulting filtered subset serves as the sole input to the subsequent phase. The classification

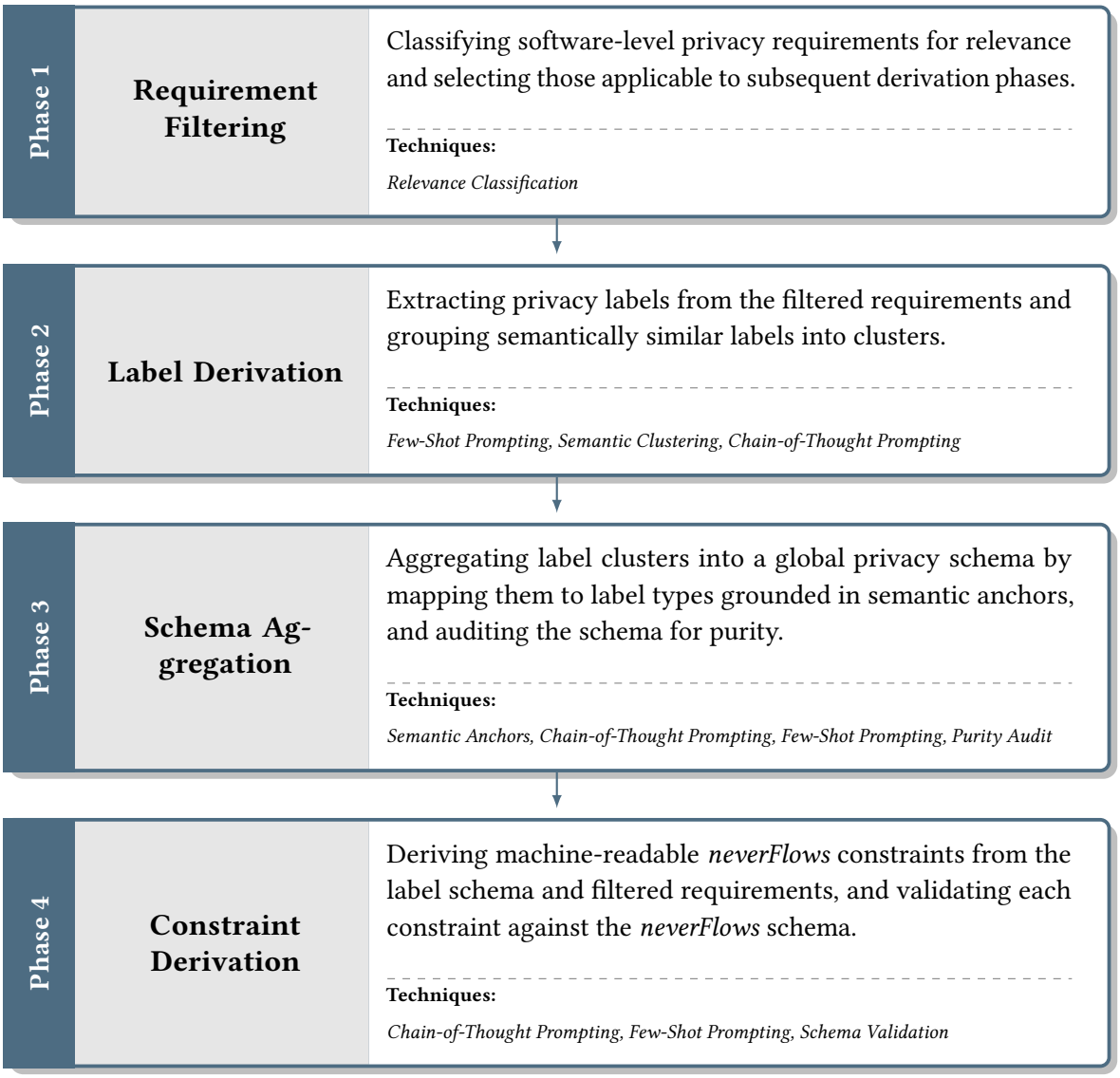


Figure 7.2.: Conceptual pipeline structure for privacy specification derivation

approach is binary per requirement, keeping the output unambiguous for downstream processing. The literature reports that LLMs produce inconsistent outputs when interpreting graded rating scales [91], and a binary relevance decision avoids this ambiguity.

7.1.2. Label Derivation

Building on the filtered requirements, this phase extracts privacy-relevant metadata and structures it into labels for attachment to DFD elements. Each label represents a specific privacy-relevant fact about a data flow or node, such as the type of personal data involved, the legal basis for processing, or the applicable retention period. Chain-of-thought prompting structures the extraction into explicit reasoning steps, verifying each label before including it in the output [14, 56]. Additionally, the pipeline provides few-shot examples to constrain

the LLM's output to the required label format: a privacy-relevant fact specifying the label type, a concrete value, and the target DFD element [26, 39]. To deduplicate the label set, the pipeline applies embedding-based semantic clustering to group synonymous labels. For example, **LR1** and **LR2** both constrain data handling to specific purposes and may independently produce multiple purpose-limitation labels. The clustering step then aims to merge such synonymous labels into a single canonical label. This phase produces a set of clustered labels that capture the relevant privacy metadata for the DFD elements. The *Schema Aggregation* phase then refines this set into a global privacy schema.

7.1.3. Schema Aggregation

While the previous phase produces a broad set of clustered labels, this phase aggregates them into a reusable global privacy label schema organised around *label types*. Label types are canonical privacy categories such as data type, legal basis, or retention period, each grouping a set of concrete label values as required by the xDECAF framework [22]. Organising the schema by label types ensures that every DFD node and flow draws its annotation from the same finite set of types. It also enables the *Constraint Derivation* phase to derive constraints from a structured label space rather than an unstructured flat list, which the literature warns against [96]. To guide this aggregation toward established privacy concepts, the pipeline provides a set of semantic anchors grounded in LINDDUN [35] and the GDPR [40]. Each anchor names a canonical privacy dimension, such as *data sensitivity* or *consent status*. The pipeline directs the LLM to use that name when a cluster captures the same privacy concern the anchor was designed to represent. The anchor set is non-exhaustive, covering privacy dimensions from those two frameworks, and serves solely as naming guidance for the LLM. When a cluster does not match any anchor, the pipeline instructs the LLM to propose a new label type and justify it. This keeps the schema open to domain-specific privacy dimensions. Figure 7.3 illustrates label types and their values for LR1 of the Travel Planner running example. It shows how a single requirement motivates both a data label and a node label grounded in the semantic anchors. Chain-of-thought prompting guides the LLM through the mapping of clusters to label types by structuring the reasoning into explicit steps, verifying each assignment before including it in the label schema [14, 110]. The pipeline provides few-shot examples to illustrate what constitutes a valid label assignment and a well-formed label type [26, 39]. This phase further refines the label set by verifying that each label type is orthogonal to the others and that each label value belongs to exactly one type [96]. The resulting global privacy schema provides an orthogonal, non-overlapping label set for DFD annotation and serves as the sole input to the following phase.

7.1.4. Constraint Derivation

The pipeline derives machine-readable *neverFlows* constraints from the label schema that the previous phase produces, for use in xDECAF's automated privacy violation detection [22]. Each constraint specifies a forbidden data flow state. Figure 7.4 illustrates an example in the context of the Travel Planner running example from Chapter 4. The pipeline takes the filtered

LR1: Consent

OBTAIN the opt-in consent for the processing of personal data for specific purposes.

Fragment	Label Type	Derived Label
personal data	`DataSensitivity` *sensitivity tier of the personal data; absent for non-personal flows*	**DataSensitivity: regular** *regular personal data (GDPR Art.4(1))*
opt-in consent	`ConsentStatus` *whether valid consent is active for the data flow*	**ConsentStatus: granted** *active consent from the data subject exists*

Figure 7.3.: Label types and their values illustrated on LR1 of the Travel Planner scenario. Highlighted fragments indicate the requirement text that motivates each label type. Blue badges are data labels, annotating data flows.

requirements from *Requirements Filtering* as a second input alongside the label schema. This grounds each constraint in the privacy dimension of its source requirement. The pipeline processes each requirement independently to ensure each constraint reflects only its source requirement. Chain-of-thought prompting guides the LLM through the constraint derivation by structuring the reasoning into explicit steps. Few-shot examples illustrate what constitutes a valid *neverFlows* constraint [26, 39]. The examples cover a diverse range of constraint patterns to enable the LLM to generalise across unseen requirements.

The *neverFlows* formalism captures only data-flow prohibitions, limiting the types of privacy obligations the pipeline can express. Many General Data Protection Regulation (GDPR) obligations are affirmative, requiring, for example, that a system collect consent before processing personal data or present a privacy notice at collection time. The pipeline addresses this by guiding the LLM to formulate label values in their positive state, such as *ConsentStatus: granted* instead of *ConsentStatus: not granted*. The absence of that positive value on data flowing to the affected component then constitutes the *neverFlows* violation condition.

A schema validation step then checks each constraint against the *neverFlows* schema before it enters the constraint set [14, 110]. Together, the constraint set and the label schema form the privacy specification for xDECAF integration.

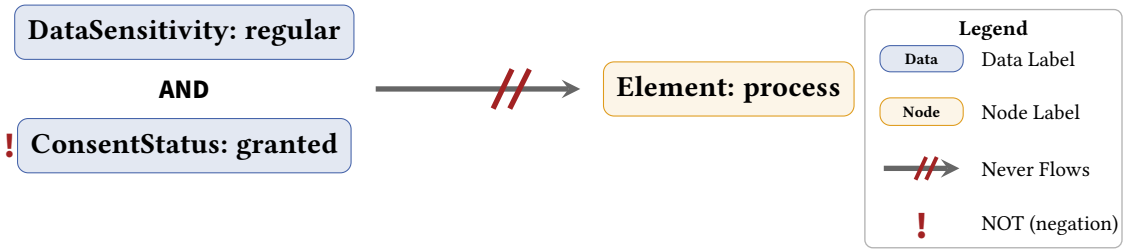


Figure 7.4.: A compound *neverFlows* constraint derived from LR1: data labelled *DataSensitivity: regular* and *NOT ConsentStatus: granted* must never flow to an *Element: process*.

7.2. Implementation

Translating the conceptual pipeline from Section 7.1 into a working system requires a suitable input dataset, prompts designed to guide the LLM through each derivation step, and output validation at each phase boundary. The following subsections address each phase in turn. They begin with dataset selection and then cover prompt design and execution details for each of the four phases. Throughout, the *Travel Planner* scenario from Chapter 4 grounds the implementation in a concrete example. The implementation builds on the Implementation Setup from Section 6.2.1. It uses GPT-5.5 [77] as the underlying model, which was not yet available during RG1.

7.2.1. Dataset Selection

The pipeline requires a dataset of software-level privacy requirements already derived from legal sources, covering a broad range of privacy categories applicable to DFD annotation contexts. Table 7.1 surveys the available options against these criteria. Among the surveyed options, the Sangaroonsilp et al. [88] dataset meets all of these criteria. It provides 71 requirements derived from GDPR and other privacy laws, spanning seven privacy categories, and targets software system contexts. Partially usable datasets are either limited in scope (e.g., restricted to specific GDPR rights) or lack well-defined requirement boundaries. The pipeline therefore excludes the remaining datasets, which address DPA compliance or tool outputs rather than software-level requirements.

7.2.2. Requirement Filtering

The first phase classifies each input requirement on two dimensions: whether it is privacy-relevant in a software system context, and whether an architect can express it as a persistent annotation on a DFD element. Only requirements satisfying both conditions proceed to the *Label Derivation* phase. At this stage, the pipeline excludes privacy-relevant but non-DFD-annotatable requirements (e.g., organisational obligations). Classification runs as a parallel batch process, processing the full requirement set in batches of five requirements.

Dataset	Size	Characteristics	Source	Usable?	Reason
71 Privacy Requirements	71 Req.	7 groups of privacy req.	[88]	✓	Software Req. derived from GDPR and other privacy laws, grouped by privacy category
Case Study on GDPR	108 Req.	LLM Annotated	[4]	(✓)	GDPR-derived req., limited to Rights of Access and Right to Portability, limited applicability to DFD context
GDPR to Software Req. (Kneuper)	≈35 Req.	Software Req.	[62]	(✓)	Manually extracted software req. from GDPR, in 11 subgroups
GDPR to Software Req. (Ringmann)	≈30 Req.	Software Req.	[86]	(✓)	Manually derived GDPR software req., in 9 groups, but not clearly defined
DPA Completeness	169 DPAs	DPA	[17]	✗	DPAs, not legal sources for software req.
Annotated DPAs	180	ML-based Annotations	[13]	✗	DPA compliance verification, not software req.
DPA Compliance Req.	45	DPA	[29]	✗	Req. for DPA documents, not software systems

Table 7.1.: Overview of available datasets for privacy requirements extraction. ✓ = directly usable, (✓) = partially usable, ✗ = not usable.

A batch of five keeps each prompt short enough to avoid output truncation caused by token limits.

Prompt Design The prompt, as Figure 7.5 illustrates, uses a *Privacy Analyst and Data Protection Expert* role to ground the LLM's reasoning in domain expertise. This role directs the model to apply privacy principles and software-system context to each classification decision. Classification proceeds in two explicit sequential steps. Step 1 assesses privacy relevance. Step 2 tests DFD annotability using a design-time criterion: whether an architect can determine the annotation value from system design alone, without runtime measurement or external verification. Multiple few-shot examples cover both annotatable and non-annotatable cases. These examples demonstrate the expected output format and

the reasoning behind each classification decision. To avoid losing privacy coverage through over-filtering, the prompt instructs the LLM to prefer `TRUE` for DFD annotability when in doubt. Downstream phases can further filter requirements, but cannot recover those dropped at this stage. Finally, the prompt provides the full requirement set as context alongside each batch, enabling the LLM to interpret individual requirements in relation to the whole set rather than in isolation.

Output Validation A Pydantic schema validates the output by requiring both boolean flags and mandatory reasoning strings for each classification. This ensures the LLM cannot produce bare classification decisions. Where a requirement appears in multiple processing iterations with conflicting results, the pipeline retains the more inclusive result (`TRUE` for both dimensions) and concatenates the reasoning strings. This follows the recall-over-precision strategy from the prompt design. After merging, a final programmatic filter removes all requirements where either flag remains `FALSE`. The remaining subset proceeds to the *Label Derivation* phase.

Running Example Table 7.2 shows the classification results for the four Travel Planner legal requirements alongside one rejected requirement from the full dataset. All four pass both dimensions and proceed to the *Label Derivation* phase. For **LR4**, the *PROVIDE* verb could suggest a runtime notification obligation, but the classifier identifies it as DFD-annotatable. The classifier reasons that data sensitivity is a design-time property observable from the data model without runtime measurement. In contrast, the classifier rejects REQ-013 despite its privacy relevance, with the LLM reasoning: *"Maintaining a record is an organizational documentation/governance obligation. The record itself is not a property annotatable on a DFD element by inspecting architecture; it requires maintaining an external inventory or compliance artifact."*

7.2.3. Label Derivation

The second phase takes the filtered requirements and produces a set of candidate privacy label clusters through four sequential steps. First, label derivation runs in parallel batches of 20, producing a flat set of candidate labels across the full requirement set. A batch of 20 spans multiple privacy categories, keeping each prompt within a length that preserves consistent label granularity. Second, a clustering step organizes these labels into semantic groups in parallel batches of 20, grouping similar labels from different requirements together. Third, each cluster receives the original label details from the derivation step, preserving the source requirement context. Finally, a Jaccard-based deduplication step merges duplicate clusters arising from independent parallel batches, using a similarity threshold of 0.5. A threshold of 0.8 proved too strict during development, failing to merge clusters with broad semantic overlap across varying levels of specificity. At 0.5, clusters sharing a common privacy dimension merge even when their label names differ in specificity. Deduplicated label clusters then proceed to the *Schema Aggregation* phase. Together, these four steps produce the first draft of the label schema. The *Schema Aggregation* phase refines and finalizes it.

```
ROLE:
  # Lead Privacy Analyst and Data Protection Expert persona -- grounds LLM reasoning
      in domain expertise for privacy-relevance and DFD-annotability classification.

OBJECTIVE:
  # For each requirement in the batch, decide: (1) is_privacy_relevant -- whether it
      concerns personal data protection in a software system context; (2)
      is_dfd_annotatable -- whether it can be expressed as a persistent annotation on
      a DFD element (process, data store, data flow, or external entity); (3) provide
       reasoning for both decisions.

EXAMPLES:
  # Fourteen few-shot examples spanning annotatable cases (encryption at rest/in
      transit, data subject rights, cross-border transfer notification, consent
      gating, per-element access authorisation) and non-annotatable cases (DPO
      contact details, internal record-keeping, controller identity, generic access-
      control policy, runtime consent tracking); each example provides
      reasoning_privacy and reasoning_dataflow, demonstrating the design-time vs
      runtime distinction.

INSTRUCTIONS:
  # Two-step pipeline per requirement in requirements_set_batch:
  #   STEP_1 -- Assess privacy relevance: determine whether the requirement concerns
      personal data, data protection, confidentiality, privacy rights, or obligations
       tied to personal data handling; data subject rights are always privacy-
      relevant regardless of framing.
  #   STEP_2 -- Assess DFD annotability using the design-time criterion: can an
      architect determine the annotation value for a specific DFD element from system
       design alone, without runtime measurement or external verification? TRUE if
      the requirement constrains data flows, storage, processing conditions,
      component capabilities, or transparency obligations that vary across elements;
      FALSE for purely organisational obligations (filing reports, contact details,
      training).

INPUT_DATA:
  # Two slots: requirements_set (full legal document for context only) and
      requirements_set_batch (the specific subset to classify in this run).

STRATEGY:
  # Three directives: systematic step-by-step analysis; use the full requirements set
       for contextual interpretation; prefer TRUE for is_dfd_annotatable when in
      doubt -- downstream phases can filter further but cannot recover dropped
      requirements.
```

Figure 7.5.: Filter requirements prompt (excerpt)

ID	Requirement	Privacy Relevant	DFD-Annotatable	Result
LR1	OBTAIN the opt-in consent for the processing of personal data for specific purposes	✓	✓	passed
LR2	STORE the personal data as necessary for specific purposes	✓	✓	passed
LR3	ALLOW the authorised stakeholders to access personal data as instructed by a controller	✓	✓	passed
LR4	PROVIDE the data subjects the sufficient explanation for the need to process sensitive personal data	✓	✓	passed
REQ-013	MAINTAIN a record of personal data processing activities	✓	✗	rejected

Table 7.2.: Requirement filtering results for the Travel Planner legal requirements and one rejected requirement from the full dataset. ✓ = true, ✗ = false.

Prompt Design The derivation prompt uses a *Privacy Analyst and DFD Annotation Architect* role, combining domain expertise in privacy with practical knowledge of how to represent privacy dimensions as DFD annotations. Figure 7.6 shows an excerpt. A Chain-of-Thought approach structures the reasoning into four explicit steps. First, the LLM identifies the privacy dimension each requirement addresses and groups requirements by dimension, classifying each label as a node or data label. Second, it checks whether each candidate label name matches an established privacy dimension by comparing the architectural question it answers. Third, it consolidates the within-batch label set before producing the final output. The prompt includes multiple few-shot examples to address specific derivation challenges. These cover grouping related requirements under one label and splitting requirements that co-occur in legal texts but address different architectural questions. Additional examples illustrate how to avoid micro-labels that model a single requirement state rather than a reusable dimension. Dedicated examples also cover the node-versus-data label distinction, the no-negation-pair rule, and misclassification cases such as data sensitivity. One design consideration concerns the source of few-shot examples: if they drew from the same dataset as the pipeline input requirements, the LLM might reproduce known label patterns from its training data or in-context examples rather than deriving them independently. To mitigate this, the few-shot examples use synthetic privacy requirements constructed independently of the Sangaroonsilp taxonomy. This ensures the LLM cannot directly map input requirements to in-context label names. Additionally, because the dataset is part of the LLM's pre-training data, we do not use it as an evaluation benchmark. Any match between derived labels and the data set could reflect memorization rather than genuine derivation from the requirements. To prevent the LLM from combining multiple privacy dimensions into a single label, the

prompt prohibits “and”, “&”, and “/” in any label name. During development, label names containing these characters consistently indicated that two distinct dimensions had been merged rather than separated. As in the *Requirement Filtering* phase, the prompt provides the full requirement set as context alongside each batch, but instructs the LLM to derive labels only from the current batch.

The clustering prompt follows the same structural principles but shifts the reasoning objective from derivation to grouping. It uses an *Ontology Engineer and Privacy Architect* role, focusing on grouping the derived labels by their shared architectural question. Its Chain-of-Thought steps follow the same read-group-assign-finalise structure. A Mutually Exclusive and Collectively Exhaustive (MECE) constraint requires that exactly one cluster covers each label and that every input label appears in some cluster. As in the derivation prompt, multiple few-shot examples illustrate correct merges and necessary splits. These also cover anti-patterns such as catch-all cluster names, micro-clusters, and composite names containing “and”.

Output Validation A Pydantic schema validates the derivation step output. It enforces all required fields for each label: a name, a definition, and a list of values (each with a code and description). Each label must also carry the applicable DFD element types, the source requirement IDs, and a reasoning string. A coverage validator ensures every requirement ID in the batch appears in at least one label’s source requirements, rejecting outputs that silently drop requirements. A micro-label guard rejects single-value labels whose sole label values is a trivial paraphrase of the label name. Such labels model one specific requirement state rather than a reusable privacy dimension. Any validation failure invokes the pipeline’s retry mechanism (Section 6.2.1), passing the error details as feedback to the LLM for correction. The pipeline merges parallel batch outputs by label name, combining value lists and source requirement IDs for same-name labels. Prior to the *Schema Aggregation* phase, the deduplication step ensures the phase receives a duplicate-free cluster set.

Within the clustering step, a MECE validator ensures the LLM neither hallucinates nor drops labels. Every assigned label must appear in the derivation output, and every derivation output label must appear in some cluster. The pipeline defers semantic correctness validation of the clusters to the *Schema Aggregation* phase.

Running Example Table 7.3 shows the derived labels and assigned clusters for **LR1**–**LR4**. In this example, each requirement maps to a distinct label, and the clustering step assigns each label to its own cluster, as all four address different architectural dimensions. It names the cluster for **LR3** Access_Authorization rather than *Access Control*. Cluster names derive from the architectural question the label answers, independently of the label name itself. **LR4** is the only requirement that produces a data label. Its applicable DFD element types are *data flow* and *data store*. **LR1**–**LR3** produce node labels. For **LR1** and **LR2**, the pipeline also includes *data flow* in the DFD elements to support downstream constraint generation. The derived values are unrefined candidates that proceed to the *Schema Aggregation* phase for consolidation.

```
ROLE:
  # Privacy Analyst and DFD Annotation Architect -- derives reusable privacy labels
      from requirements, one label per architectural privacy dimension.

OBJECTIVE:
  # (1) Group same-dimension requirements into ONE label with multiple values.
  # (2) Keep labels abstract enough to apply consistently across systems.
  # (3) Classify as node_label (processes, data stores, external entities) or
      data_label (data flows and elements in motion).
  # (4) Trace every label back to its source requirements.

EXAMPLES:
  # Eleven examples: Ex.1-2 grouping and splitting by dimension; Ex.3-5 structure
      rules (no micro-labels, no negation pairs, include data_flow when a flow
      carries a violation); Ex.6-7 dimension boundaries ("and" in a name signals a
      merged label); Ex.8-11 four misclassification patterns (Automated Decision
      Oversight, Data Sensitivity, Data Protection vs Access Control, Data Subject
      Rights vs Purpose Limitation).

INSTRUCTIONS:
  # Four-step pipeline per batch:
  #   STEP_1  -- identify each requirement's privacy dimension.
  #   STEP_2  -- group same-dimension requirements into one candidate label; stop if
      name contains "and", "&", or "/" -- split; node_label for component properties,
      data_label for data; add data_flow when a flow can carry a violation; 2-5
      affirmative values only.
  #   STEP_2b -- canonical dimension check: rename to the canonical name if the
      architectural question matches; justify any novel label name.
  #   STEP_3  -- merge near-synonyms; list all related_reqs per label.
  #   STEP_4  -- output node_labels and data_labels; every requirement must appear in
      at least one label's related_reqs.

INPUT_DATA:
  # Two slots: relevance_classification (full filtered requirements list, context
      only) and relevance_classification_batch (subset to derive labels from in this
      run).

CONSTRAINTS:
  # SINGLE_DIMENSION_RULE: one atomic dimension per label -- never use "and", "&", or
      "/" in a label name.
  # CONSISTENT_NAMING: use the same label name for the same dimension across batches.
```

Figure 7.6.: Derive privacy labels prompt (excerpt)

LR	Derived Label	Type	DFD Elements	Cluster
LR1	Consent Mechanism	node	process, external_entity, data_flow	CONSENT_MECHANISM
LR2	Purpose Limitation	node	process, data_flow	PURPOSE_LIMITATION
LR3	Access Control	node	process, data_store, external_entity	ACCESS_AUTHORIZATION
LR4	Data Sensitivity	data	data_flow, data_store, external_entity	DATA_SENSITIVITY

Table 7.3.: Phase 2 output for the Travel Planner legal requirements: derived labels (2a) and their assigned clusters (2b). Type indicates node label (annotates DFD elements) or data label (annotates data flows and stores).

7.2.4. Schema Aggregation

The previous phase produces labels at the requirement-batch level. These labels may be redundant across batches, span multiple dimensions within a single cluster, or contain requirements that slipped through Phase 1 filtering. Aggregating these directly into a global schema in a single step would require the LLM to handle four tasks simultaneously. It would need to resolve dimension overlaps, reject non-annotatable clusters, derive architectural questions, and populate label values. Thus, this task would exceed reliable LLM attention and increase hallucination risk [69, 111]. This phase therefore decomposes the aggregation into focused sequential steps.

First, the *cluster dimension mapping* step refines cluster definitions by ensuring all associated requirements answer the same architectural question. It splits clusters whose requirements span multiple dimensions and rejects clusters whose requirements are not DFD-annotatable at design time. Then, the *aggregate labels* step consolidates the clusters into label types, using the semantic anchors introduced in Section 7.1.3. Next, the pipeline attaches label values from Phase 2 as historical candidates and subjects the label types to a cross-label purity audit. This step follows an actor-critic structure. A dedicated critic prompt reviews all label types simultaneously for dimension purity violations, incorrect DFD element assignments, and runtime states presented as design-time properties. The *label refinement* step then takes this critique as input and re-derives the final values for each label type from its source requirements.

Prompt Design The *cluster dimension mapping* prompt evaluates each Phase 2 cluster on two criteria: whether all its requirements address the same dimension, and whether any are DFD-annotatable at design time. The step accepts clusters where all requirements address the same annotatable dimension. It splits clusters whose requirements span multiple dimensions. The step rejects clusters with no annotatable requirements. This classification directly drives the programmatic *BuildDimensionGroups* step. It groups the accepted and

split clusters by canonical dimension, producing one dimension group per distinct privacy dimension.

The *aggregate labels* prompt maps the resulting dimension groups to label types, deriving an architectural question for each dimension. Its design follows directly from the semantic anchors in Section 7.1.3. That section already establishes all necessary design decisions.

The *purity audit* prompt applies an actor-critic pattern. Rather than auditing each label type in isolation, it reviews the full set of label types simultaneously. This is a deliberate design choice, as the full schema must be present to detect dimension purity violations. A violation occurs when a value answers the wrong architectural question, or when another label type already owns that concept. The prompt instructs the LLM to act as a critic only, not as a schema designer. It must identify each problem and flag it with the specific label type that owns the concept. Its output is not a revised schema but a structured set of per-label critique notes that feed directly into the refinement step.

The *label refinement* prompt, shown in Figure 7.7, operates on one label type at a time and introduces the design decisions most specific to this phase. It receives the label type skeleton, the source requirements, the Phase 2 historical values, and any purity critique from the audit step. The central design decision restricts label values to design-time-observable properties. Every value must describe a persistent property an architect can observe from system design alone, explicitly excluding runtime states, log-readable properties, and documentation inventories. A cross-label boundary check (Step 2b) prevents values from duplicating concepts that other label types own, complementing the purity critique. The refinement step treats purity critique notes from the audit as correction instructions, not suggestions, ensuring it addresses every violation the audit flags.

Output Validation The earlier steps of this phase use coverage validators to prevent requirement loss. For *cluster dimension mapping*, an output validator checks that every input requirement ID appears in exactly one tag. For *aggregate labels*, a validator enforces that every dimension group becomes a label type and that the pipeline either rescues or explicitly rejects every candidate-reject group with a reason. A validator on the *purity audit* output checks that every critique entry names an actual label type from the input. This prevents the LLM from hallucinating label names that would make critique notes unactionable.

The *label refinement* step carries validators that enforce the design-time constraint on label values. An inapplicable-label check rejects label values that indicate the label does not apply to any DFD element type (process, data store, data flow, or external entity). If such a value appears in the output, the pipeline omits the label rather than including it as a placeholder. An affirmative-state check rejects label values that name what the system must do rather than the compliant state it is in (e.g., *consent required* rather than *consent obtained*). This prevents label values unsuited for downstream constraint generation, as described in Section 7.1.4. Two structural validators preserve schema consistency across refinement iterations. The first rejects any output that renames a label type provided as input, ensuring each label type retains its identity across refinement iterations. The second rejects label values that prefix the label type name. This prevents the LLM from generating

```
ROLE:
  # Expert Privacy Architect and Data Flow Modeler -- receives a LabelType skeleton
      with source requirements and historical data; derives a clean set of
      label_values and confirms correct dfd_elements.

OBJECTIVE:
  # Populate the LabelType skeleton by synthesizing label_values from source
      requirements, applying all value quality rules, and confirming DFD element
      coverage; every value must be a persistent design-time property grounded in the
       requirements.

EXAMPLES:
  # Eight examples covering value quality failures and cross-label contamination:
      runtime states reformulated as design-time capabilities (Ex.1, 5, 7); cross-
      label duplicates dropped to the owning LabelType (Ex.2, 3); negation pairs, non
      -exclusive disclosure topics, and near-synonyms collapsed to canonical values (
      Ex.4, 6, 8).

INSTRUCTIONS:
  # Four-step pipeline per LabelType:
  #   STEP_1  -- confirm dfd_elements; always include data_flow when a requirement
      prohibits data from reaching an inappropriate destination.
  #   STEP_2  -- synthesize label_values: STATE labels require mutually exclusive
      values, CAPABILITY labels allow co-existing values; no unknown/negation/
      terminal-action states; collapse synonyms; values must be design-time
      observable -- reformulate runtime states as persistent capabilities; codes name
      a possessed state, not an obligation; target 2-4 values; snake_case, no
      LabelType-name prefix.
  #   STEP_2b -- cross-labeltype boundary check: each LabelType owns its dimension
      exclusively; do not duplicate values owned by another LabelType; apply
      value_exclusion_notes if provided.
  #   STEP_2c -- purity critique: VALUE PURITY ISSUE drops the flagged value; DFD
      ASSIGNMENT ISSUE removes the flagged element.
  #   STEP_3  -- define a precise trigger_condition per value.

INPUT_DATA:
  # hydrated_label_types_batch: LabelType skeleton with source requirements,
      historical values, and optional purity_critique and value_exclusion_notes.

CONSTRAINTS:
  # SINGLE_DIMENSION_FOCUS: one atomic privacy dimension per LabelType.
  # CONSTRAINT_GENERATION: include data_flow when a violated label could propagate
      via a flow to a wrong destination.
  # NO_HALLUCINATION: no invented values; ARCHITECTURE_FIRST: values observable from
      system design only.
```

Figure 7.7.: Refine privacy labels prompt (excerpt)

redundant label values that repeat the label type name rather than describing a specific property state.

Running Example Figure 7.8 shows the finalized label types for the *Travel Planner* requirements after Schema Aggregation. Schema Aggregation consolidates *Purpose Limitation* from four Phase 2 label values to a single *purpose bound* value, and *Data Sensitivity* from two Phase 2 label values to *sensitive personal data*. Of these four label types, *Data Sensitivity* is the only data label, and the remaining three are node labels. The finalized schema proceeds to the *Constraint Derivation* phase.

LR	Label Type	Values (refined)	DFD Elements
LR1	**consent_mechanism**	`opt_in_gated`, `withdrawal_supported`, `consent_preferences_enforced`, *+ 3 further values*	process, data_flow
LR2	**purpose_limitation**	`purpose_bound` *(consolidated from 4 raw values)*	process, data_store, data_flow
LR3	**access_control**	`controller_instructed_access`	process, data_store, data_flow
LR4	**data_sensitivity**	`sensitive_personal_data` *(consolidated from 2 raw values)*	data_flow, data_store

Figure 7.8.: Finalized label types after Schema Aggregation for the Travel Planner legal requirements. Orange badges indicate node labels; blue badges indicate data labels.

7.2.5. Constraint Derivation

The final phase takes the finalized label schema from Phase 3 and the filtered requirements from Phase 1. It derives *neverFlows* constraints for automated privacy violation detection within the xDECAF framework [22], as described in Section 7.1.4. A pre-sorting step routes each requirement into one of two paths based on how many label types it intersects. Requirements touching a single dimension follow a dedicated path with a more focused and cost-efficient prompt. All others follow a unified path with access to the full label schema. The pipeline then processes each requirement individually in parallel, with the full label schema as context so the LLM can combine any label types when formulating the constraint. We merge the resulting constraints and compile the final set into the xDECAF constraint Domain-Specific Language (DSL) syntax.

Prompt Design The unified constraint prompt uses a *Privacy Logic Engineer* role, shifting the focus from extraction and analysis to formal logical translation. The prompt processes each requirement in isolation with the full label schema as context. A five-step Chain-of-Thought structures the derivation, as Figure 7.9 shows. Three design decisions merit explicit

justification. A *neverFlows* constraint must always check for the non-compliant state, never the compliant one. This design choice follows from how xDECAF analysis works: it detects the presence of a forbidden state, not the absence of a good one. Some requirements have no invertible data-flow prohibition, such as pure positive rights, notification duties, and system design obligations. The prompt explicitly instructs the LLM to return an empty constraint list for these. This prevents generating logically unsound constraints. Finally, any compound subject combining multiple labels must be grounded in the specific requirement text rather than inferred from general privacy reasoning. This ensures that AND/OR combinations reflect actual legal intent. Four few-shot examples illustrate the main constraint patterns. They cover single label to DFD element type, cross-label targeting (subject flows to a non-compliant state of another label), NOT wrapper (absence of a required safeguard), and AND compound subject.

The single-dimension path uses a simplified variant of the same prompt. We make the primary label type explicit as the focal subject, reducing the schema navigation burden for requirements that cleanly map to one dimension.

Output Validation The primary validation rejects two types of structurally invalid constraints. Self-contradictory ones arise when the left and right side of the *neverFlows* constraint share the same label and value. Same-dimension ones arise when a single-condition left and right side reference the same privacy dimension. Both indicate a misunderstanding of the requirement text. Additionally, two topology validators prevent unsupported constraint shapes. The first rejects vertex-to-data constraints, since a system component cannot logically flow into a data payload. The second rejects vertex-to-vertex constraints, which the xDECAF DSL does not support. A hallucination check validates that every label name and value in the output exists in the input schema, preventing the LLM from referencing labels or values absent from the derived schema.

Furthermore, an action paradox check guards against a specific semantic inversion: a requirement mandating an action should not produce a constraint that blocks the corresponding compliant state. It inspects whether the source requirement contains verbs such as *erase*, *delete*, *archive*, *notify*, *inform*, *document*, or *record*, and whether any non-negated value in the derived constraint contains the same verb. Negated values are excluded from this check, since blocking the absence of a compliant state is the intended pattern.

Running Example Figure 7.10 shows the four *neverFlows* constraints derived from the Travel Planner requirements. **LR1**–**LR3** each produce a NOT-wrapper constraint targeting a generic DFD element type. **LR1** constrains data lacking an opt-in consent gate from reaching a process. **LR2** constrains data without purpose-bound storage from reaching a data store. **LR3** constrains data without authorised access control from reaching an external entity. **LR4** produces a cross-label constraint: data classified as sensitive personal data must never flow to a component annotated with partial rather than full transparency disclosure. It combines the data label from **LR4** with the *transparency obligation* label derived from the broader requirement set.

```
ROLE:
  # Privacy Logic Engineer specializing in GDPR compliance -- translates legal
      requirements into JSON neverFlows constraints for DFD static analysis tooling.

OBJECTIVE:
  # Given ONE requirement and the full LabelType set, derive JSON neverFlows
      constraints; each specifies a data state that must never reach a DFD element
      type or the non-compliant state of another label.

EXAMPLES:
  # Six examples covering the full constraint vocabulary: A -- single label to element
       type (corrupted -> PROCESS); B -- cross-label to non-compliant state (always
      BAD value as object, not compliant); C -- NOT wrapper for absent safeguard (!
      encrypted -> DATA_STORE); D -- AND compound on subject (sensitive AND unverified
      ); E -- AND compound on object (non-EU AND !enforced storage); F -- SKIP: system
      obligations return [] -- not expressible as a data-flow prohibition.

INSTRUCTIONS:
  # (1) Skip unless a data state (Q1) and a forbidden destination (Q2) can both be
      named from the label set; also skip when the relevant label describes a system
      capability rather than a property of the data.
  # (2) Most specific label value naming the problem; {NOT: code} for absent
      safeguards; subject_type always "data".
  # (3) "Element" for any-class DFD target; cross-label for conditional target;
      always the NON-COMPLIANT object value.
  # (4) Names/codes must match input exactly; snake_case name with blocking verb;
      description quotes the REQ-ID.
  # (5) One constraint per state x target; max 2 per requirement.

TARGET_DSL_FORMAT:
  # data <label.value> neverFlows vertex <type | label.value>; NOT: !label.value; AND
      : space-separated; OR: comma-separated.

INPUT_DATA:
  # unified_constraints_input_batch: one requirement with REQ-ID and the full
      all_label_types list with value codes.

CONSTRAINTS:
  # subject_type always "data" -- vertex-to-vertex not supported; labels/values must
      match input exactly -- never copy from examples.
  # Object for Element: process/data_store/external_entity only; for cross-label: NON
      -COMPLIANT/bad state only.
  # Field "related_req_ids" exactly; relation "neverFlows"; return [] for system
      obligations and non-invertible rights.
```

Figure 7.9.: Constraint derivation prompt (excerpt)

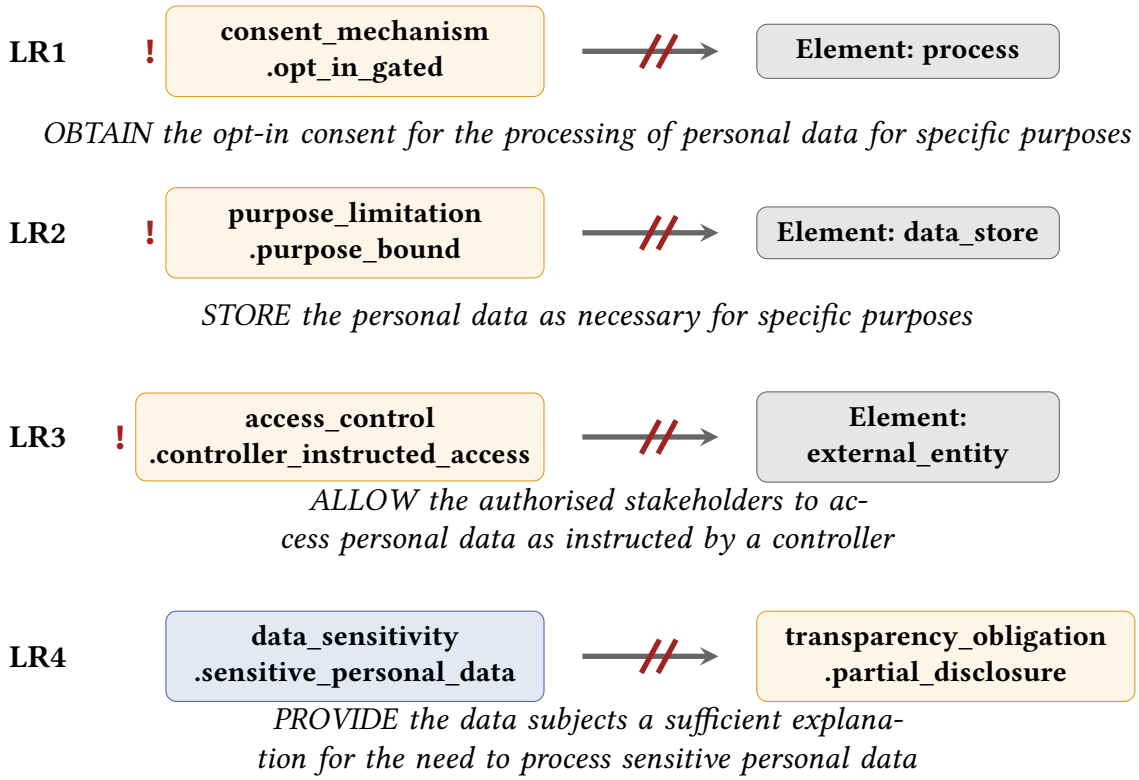


Figure 7.10.: *neverFlows* constraints derived from the Travel Planner legal requirements. Blue badges are data labels; orange badges are node labels; grey badges are generic DFD element types. The ! prefix denotes negation.

The data companion set [46] contains all prompt templates, validation schemas, and the prototype implementation code for the pipeline.

7.3. Evaluation

Following the shared evaluation design outlined in Chapter 5, this section details the design's application to RQ2. It instantiates the Goal Question Metric (GQM) plan [18] with specific goals, questions, and metrics for privacy specification quality assessment, and describes the two-step user study we conducted to collect those metrics. It further presents the results, discusses them, and addresses threats to validity.

7.3.1. Goals, Questions, and Metrics

Goal 2 defines the evaluation goal as assessing the quality of LLM-derived privacy specifications, as posed by RQ2. The approach detailed in Section 7.1 derives both privacy labels and logical constraints. Since no objective quality standard exists for either artifact type in isolation, we operationalise quality through comparison with expert-created specifications. Expert ratings thus provide the basis for quality judgements. As the two artifact types differ in their nature, we define all metrics separately for both *privacy labels* and *logical constraints*. This enables a fine-grained analysis of where the automated approach succeeds or falls short. **Question 2.1** and **Question 2.2** address *correctness* and *completeness* as the primary quality dimensions. **Question 2.3** provides a complementary perspective by examining what types of quality issues occur in each artifact type and at what frequency. This captures qualitative differences that quantitative ratings alone cannot distinguish. Finally, **Question 2.4** addresses rating reliability, which is essential for interpreting the validity of the expert ratings on which the comparison depends. Figure 7.11 provides a visual overview of the instantiated GQM plan.

Metric 2.1.1 and **Metric 2.2.1** capture the correctness and completeness of privacy labels using expert Likert ratings [68] applied to both LLM-derived and expert-created label sets. This enables a direct group comparison on the same criteria. **Metric 2.1.2** and **Metric 2.2.2** apply the same rating approach to logical constraints. They separately assess whether the derived *neverFlows* rules are logically sound and cover the relevant privacy violations implied by the software-level requirements. **Metric 2.3.1** and **Metric 2.3.2** measure the prevalence of each issue type (missing, incorrect, hallucinated, and vague), separately for privacy labels and logical constraints. For each type, prevalence is the proportion of evaluators who flagged its presence. The resulting proportions compare the two groups. **Metric 2.4.1** and **Metric 2.4.2** quantify rating reliability for the Likert dimensions using Krippendorff's alpha (interval scale) [63]. This measure handles ordinal data, unequal numbers of raters per item, and missing observations. **Metric 2.4.3** and **Metric 2.4.4** extend this to issue-category flagging, separately for privacy labels and logical constraints. They report raw percent agreement alongside Krippendorff's alpha (nominal scale). Krippendorff's alpha for binary outcomes is sensitive to category prevalence (the relative frequency of each category across all rated items) [41]. When one category dominates, alpha can be suppressed even if raw agreement is high. Reporting both measures therefore ensures the full picture of rater agreement is visible regardless of the prevalence distribution.

Goal 2: Evaluating the quality of LLM-derived privacy specifications compared to expert-derived ones.

Question 2.1: *(Correctness)* To what extent do LLM-derived privacy specifications correctly reflect the given software-level privacy requirements compared to expert-derived ones?

Metric 2.1.1: Expert Likert rating (1–5) for semantic correctness of LLM-derived privacy labels compared to expert-derived ones.

Metric 2.1.2: Expert Likert rating (1–5) for logical correctness of LLM-derived constraints compared to expert-derived ones.

Question 2.2: *(Completeness)* To what extent do LLM-derived privacy specifications cover all relevant aspects of the given software-level privacy requirements?

Metric 2.2.1: Expert Likert rating (1–5) for completeness of LLM-derived privacy labels compared to expert-derived ones.

Metric 2.2.2: Expert Likert rating (1–5) for completeness of LLM-derived constraints compared to expert-derived ones.

Question 2.3: *(Detected Issues)* Which types of quality issues are identified in LLM-derived privacy specifications compared to expert-derived ones, and how prevalent is each issue type across evaluators?

Metric 2.3.1: Proportion of evaluators flagging each issue type (missing, incorrect, hallucinated, vague) for privacy labels in LLM-derived vs. expert-derived specifications.

Metric 2.3.2: Proportion of evaluators flagging each issue type (missing, incorrect, hallucinated, vague) for logical constraints in LLM-derived vs. expert-derived specifications.

Question 2.4: *(Rating Reliability)* To what extent do experts agree in their assessment of privacy specification quality?

Metric 2.4.1: Krippendorff's alpha (interval scale) among experts on correctness and completeness ratings for privacy labels.

Metric 2.4.2: Krippendorff's alpha (interval scale) among experts on correctness and completeness ratings for logical constraints.

Metric 2.4.3: Raw percent agreement and Krippendorff's alpha (nominal scale) for inter-rater reliability of issue-category flagging for privacy labels.

Metric 2.4.4: Raw percent agreement and Krippendorff's alpha (nominal scale) for inter-rater reliability of issue-category flagging for logical constraints.

Figure 7.11.: GQM plan for the evaluation of RQ2

7.3.2. Evaluation Design

The evaluation design for RQ2 relies on a two-part online survey, as outlined in Chapter 5 and Figure 5.3. We recruited participants from a pool of academic and industry experts in software engineering and the legal domain. In *Part 1*, participants completed a demographic survey. The survey preceded the task so that task experience would not influence participants' self-reported background [109]. They then worked through a running example to familiarise themselves with the derivation task. Figure 7.12 and Figure 7.13 show the requirements and task instructions used in the actual derivation task. We derived these requirements from the Standard Data Protection Model (SDM) [7]. We selected and adapted those accessible to participants with varying levels of expertise and feasible within the study time constraints. The selected requirements cover a limited cross-section of privacy dimensions rather than the full 71-requirement dataset we used to develop the pipeline. Participants then independently derived a set of privacy labels and logical constraints from these requirements. This derivation provides the human baseline for comparison. In parallel, the LLM generated four independent sets from the same requirements using the pipeline described in Section 7.2. This number balances coverage of output variability against participant cognitive load [61, 109]. The part concluded with a self-assessment questionnaire on task difficulty and confidence. Each participant also generated an anonymous ID to link their *Part 1* submission to their *Part 2* ratings. To retain the ID across sessions, participants could either download it directly or request that the platform send it to their email address. The platform did not retain the email-to-ID mapping, preserving participant anonymity.

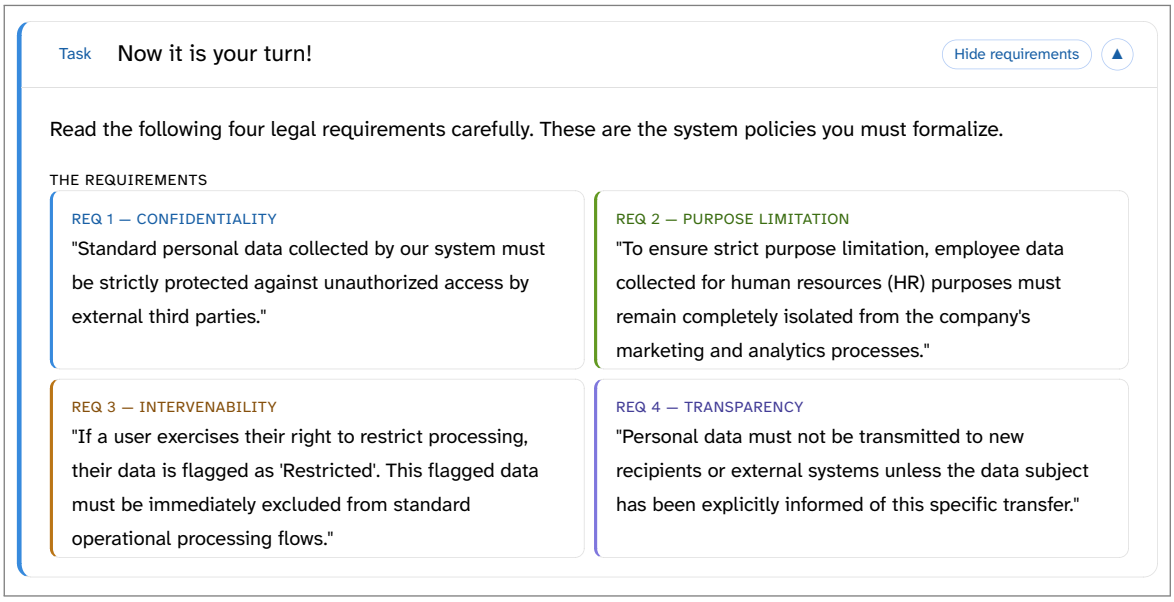


Figure 7.12.: Presented requirements in *Part 1* of user study 2

In *Part 2*, participants first reviewed the same running example as in *Part 1*, alongside explicit rating criteria and issue type definitions. We included this step to calibrate participants' rating scales. They then evaluated four sets of privacy labels and logical constraints randomly sampled from the pool we collected in *Part 1*. At session start, the SoSci Survey online

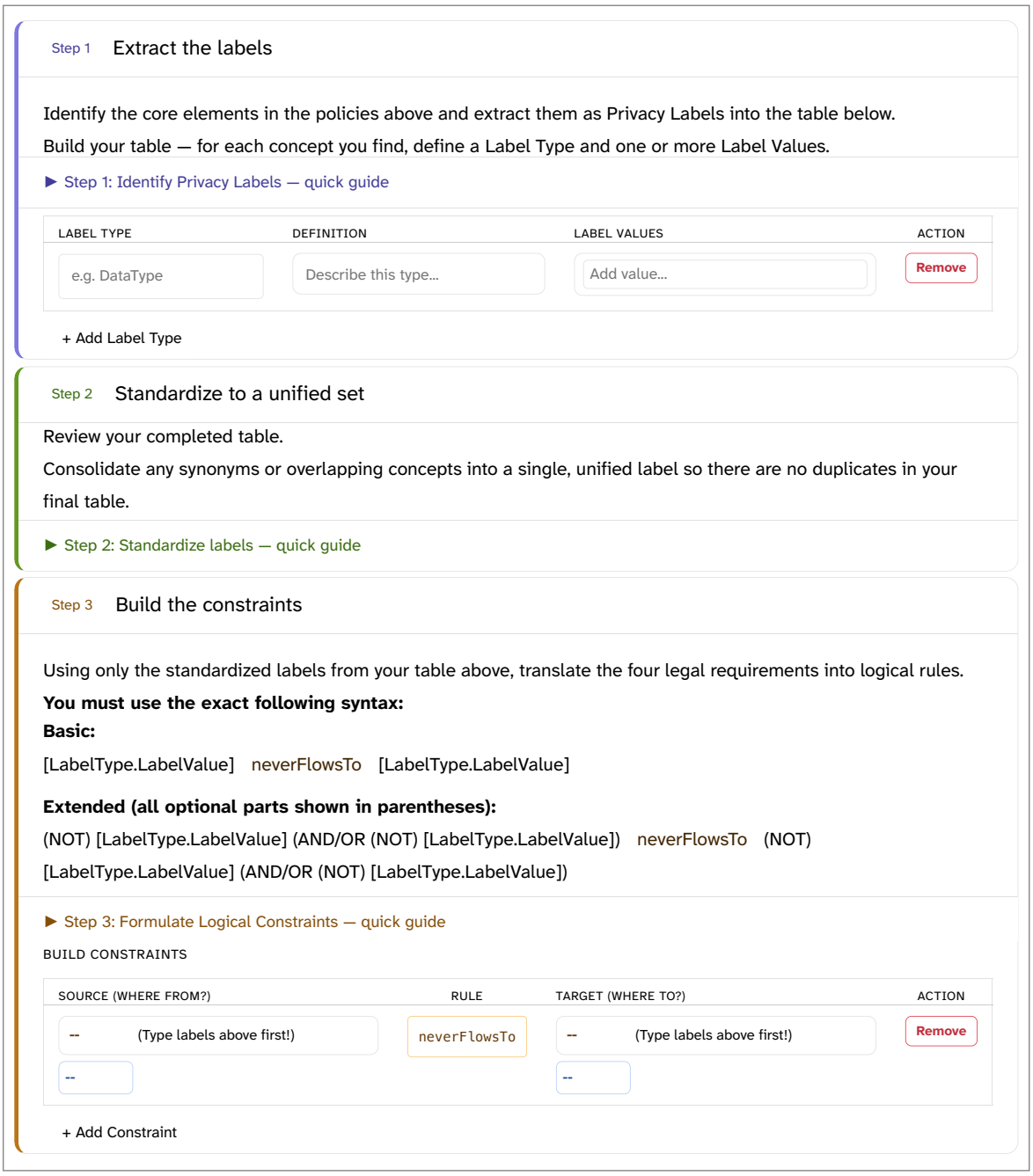


Figure 7.13.: *Part 1* task pane of user study 2 (2)

platform randomly drew two sets from each group for each participant to evaluate, ensuring no participant rated their own submission. Within each group, it prioritised artifacts with the fewest reviews to distribute evaluations evenly. It also randomised the presentation order of the four sets. This randomisation mitigates order effects and ensures a balanced distribution of evaluations across artifacts within each group [109]. The survey design prevented participants from revising earlier ratings after proceeding, avoiding anchoring effects [102]. The running example remained accessible throughout the session to support

consistent interpretation of the task. Figure 7.14 and Figure 7.15 show the evaluation task instructions and rating panes presented to participants in *Part 2*. For each set, participants provided Likert ratings for correctness and completeness. They also flagged any issues separately for labels and constraints. The part concluded with a brief self-assessment on attention and motivation. The data companion set [46] contains all study materials for both parts.

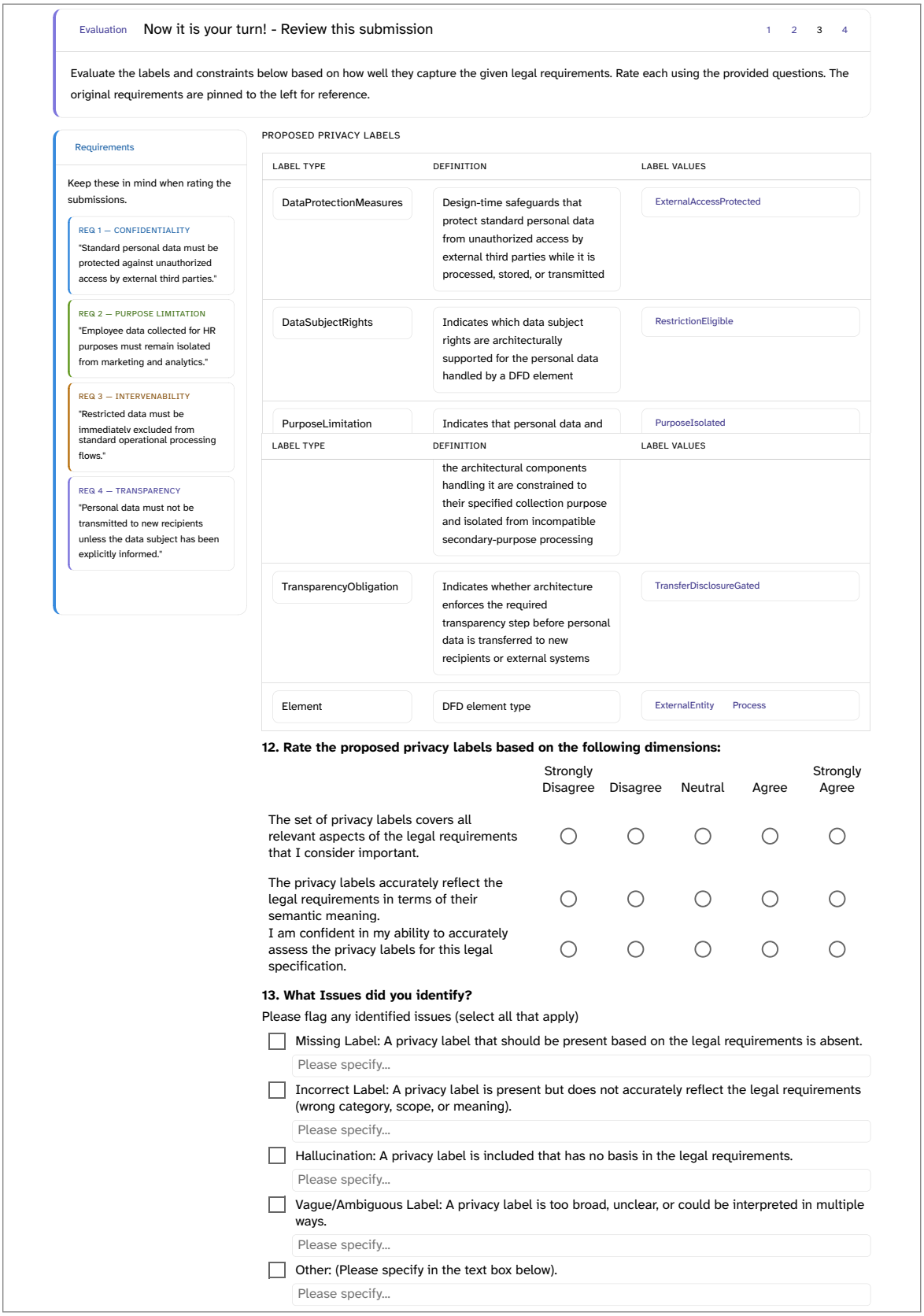


Figure 7.14.: *Part 2* task pane of user study 2 (1)

PROPOSED CONSTRAINTS

SOURCE	RULE	TARGET
NOT DataProtectionMeasures.ExternalAccessProtected	neverFlowsTo	Element.ExternalEntity
DataSubjectRights.RestrictionEligible	neverFlowsTo	Element.Process
NOT PurposeLimitation.PurposeIsolated	neverFlowsTo	Element.Process
NOT TransparencyObligation.TransferDisclosureGated	neverFlowsTo	Element.ExternalEntity

14. Rate the proposed constraints based on the following dimensions:

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
The set of logical constraints covers all relevant relationships and rules that I consider necessary to capture the legal requirements.	○	○	○	○	○
The logical constraints accurately reflect the legal requirements in terms of their semantic meaning and logical structure.	○	○	○	○	○
I am confident in my ability to accurately assess the logical constraints for this legal specification.	○	○	○	○	○

15. What Issues did you identify?

Please flag any identified issues (select all that apply)

- ☐ Missing Constraint: A logical rule or relationship that should be present based on the legal requirements is absent.
  Please specify...
- ☐ Incorrect Constraint: A constraint is present but does not correctly capture the legal rule (e.g., wrong condition, wrong scope).
  Please specify...
- ☐ Hallucination: A constraint is included that has no basis in the legal requirements.
  Please specify...
- ☐ Contradictory Constraint: Two or more constraints are logically inconsistent with each other.
  Please specify...
- ☐ Overly Restrictive / Too Permissive: A constraint captures the right idea but is calibrated incorrectly (too strict or too lenient).
  Please specify...
- ☐ Other: (Please specify in the text box below).
  Please specify...

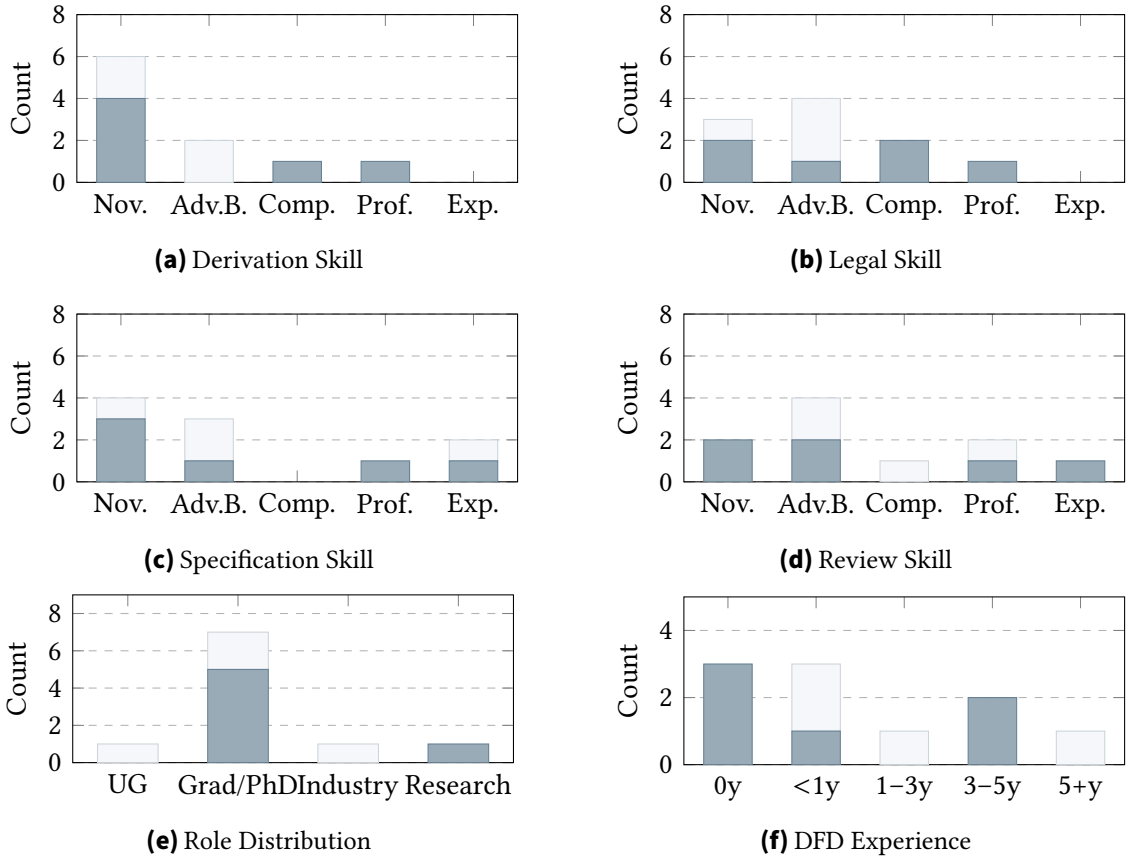
16. Any additional thoughts/insights you have?

Notes

Figure 7.15.: *Part 2* task pane of user study 2 (2)

7.3.3. Results

10 participants completed *Part 1* of the study, creating sets of privacy labels and logical constraints from the provided software-level requirements. Of these, 6 returned to assess the collected sets in *Part 2*. 9 of the 10 created sets entered the evaluation pool. We excluded those submitted after *Part 2* data collection started, so that evaluators had access to the full pool at the time of rating. These evaluators provided 24 ratings for privacy labels and logical constraints, along with issue flags for each evaluated set. In addition to the 9 human-created sets, participants evaluated 4 LLM-derived sets, resulting in a total of 13 evaluated artifact sets. Figure 7.16 summarises the demographic background of participants. Table A.5 in the appendix lists the full per-participant breakdown.



(a) Derivation Skill **(b)** Legal Skill **(c)** Specification Skill **(d)** Review Skill **(e)** Role Distribution **(f)** DFD Experience

Figure 7.16.: Participant demographics for *User Study 2* ($N = 10$ *Part 1*, $n = 6$ also completed *Part 2*). Skill columns show self-assessed level on a Novice–Expert scale: Deriv. = experience deriving technical specifications from legal texts; Legal = familiarity with privacy law and regulations; Spec. = familiarity with formal specification languages or constraint modelling; Review = experience reviewing technical artefacts. Role counts exceed N as participants could select multiple roles.

After completing the derivation task, participants rated their own experience in *Part 1*, as Table 7.4 presents. Participants reported clarity of the requirements (*Avg*=3.90). They

also found that the *LabelType: LabelValue* format allowed them to express requirements in a structured way (*Avg*=4.00). They reported moderate confidence in the correctness of their *Part 1* submissions (*Avg*=3.70), in the translation into the notation (*Avg*=3.50), and in completeness (*Avg*=3.40).

Item	Statement	Avg (SD)
Legal clarity	Legal requirements were clear and easy to understand.	3.90 (0.99)
Label system nuance	The LabelType/LabelValue system allowed capturing nuances.	4.00 (0.94)
Constraint confidence	I am confident that constraints accurately describe the restrictions.	3.70 (0.82)
Translation ease	Translating legal concepts into the notation was straightforward.	3.50 (1.08)
Coverage confidence	I am confident that my constraints cover all relevant aspects.	3.40 (1.26)

Likert scale 1 (Strongly Disagree) – 5 (Strongly Agree). N = 10. SD = standard deviation.

Table 7.4.: *Part 1* creator self-assessment ratings after privacy specification creation

Table 7.5 summarises the free-text creator notes on task difficulty. Participants most frequently reported difficulty in interpreting the requirements and deciding on the appropriate level of detail for labels and constraints. They also reported difficulty translating the requirements into the specified formats.

ID	Note
P01	Lack of experience
P02	mental translation into the "never flows to"-system; to stay in this "rigid" system
P03	I wasn't quite sure with which level of granularity I was supposed to define labels
P04	Unsureness about what is data and what node label
P05	More information about the general system workflow would have been beneficial. Only the logical attributes make it hard to derive correct label types.
P06	Constraints window only allowed one logical operation (or/and) while some constraints could be expressed with more
P08	A lot of reasoning felt short and maybe hard to exract from the final formal constraints; capturing this may help in later (manual/automated) traceability
P09	Some cases/requirements appeared to be very similar with subtle differences

Free-text responses to: *"Was there anything you found particularly difficult about the task? (optional)"*

Table 7.5.: *Part 1* creator notes on task difficulty (optional, free-text)

We present expert ratings on the privacy labels collected in *Part 2* in Table 7.6. Evaluators rated LLM-derived sets 3.00 times on average, while human-created sets received 1.33 ratings on average. The lower rating count per human set results in a less reliable baseline for that group. For both completeness and correctness, human-created sets received higher average

ratings (3.67 and 3.50 respectively) compared to LLM-derived sets (2.83 and 2.67). Human-created sets thus scored above the scale midpoint of 3 on average, while LLM-derived sets scored below it on average. Confidence levels were above the midpoint of the scale for both groups, with 3.92 for human-created sets and 3.75 for LLM-derived sets.

Type	Set	Ratings	Ratings/Set	Labels		
			Avg (SD)	Compl. Avg (SD)	Corr. Avg (SD)	Conf. Avg (SD)
Human	9	12	1.33 (0.50)	3.67 (1.16)	3.50 (1.09)	3.92 (0.67)
LLM	4	12	3.00 (0.00)	2.83 (1.19)	2.67 (1.30)	3.75 (0.87)
Overall	13	24	1.85 (0.90)	3.25 (1.23)	3.08 (1.25)	3.83 (0.76)

Compl. = Completeness; Corr. = Correctness; Conf. = Confidence. Likert scale 1–5; Avg = average, SD = standard deviation.

Table 7.6.: *Part 2* evaluator Likert ratings for privacy labels.

Expert ratings on logical constraints, summarised in Table 7.7, follow a similar pattern: human-created sets received higher average ratings for both completeness (3.17) and correctness (3.00) compared to LLM-derived sets (2.25) and (1.83). Confidence levels were above the midpoint of the scale for both groups, with an average of 3.92 for human-created sets and 3.83 for LLM-derived sets.

Type	Set	Ratings	Ratings/Set	Constraints		
			Avg (SD)	Compl. Avg (SD)	Corr. Avg (SD)	Conf. Avg (SD)
Human	9	12	1.33 (0.50)	3.17 (1.11)	3.00 (1.13)	3.92 (0.67)
LLM	4	12	3.00 (0.00)	2.25 (1.22)	1.83 (1.19)	3.83 (0.72)
Overall	13	24	1.85 (0.90)	2.71 (1.23)	2.42 (1.28)	3.88 (0.68)

Compl. = Completeness; Corr. = Correctness; Conf. = Confidence. Likert scale 1–5; Avg = average, SD = standard deviation.

Table 7.7.: *Part 2* evaluator Likert ratings for logical constraints.

Figure 7.17 shows that all LLM-derived sets received lower average ratings for completeness and correctness compared to human-created sets. We also find that no LLM-derived set consistently outperforms the others across all dimensions. Table A.6 in the appendix lists the complete ratings by artifact.

Table 7.8 summarizes the issue flagging results for privacy labels. Issue flags per evaluator ranged from 0.5 to 2.0 for human-created sets and from 0.67 to 2.0 for LLM-derived sets. For human-created sets, Missing Label accounted for 37.5% of the 16 issue type flags detected in this group, followed by Vague/Ambiguous at 25%. For LLM-derived sets, Vague/Ambiguous accounted for 50% of the 18 issue type flags detected in this group, followed by Missing Labels at 22.2%. Hallucinated Label accounted for the smallest share for LLM-derived sets

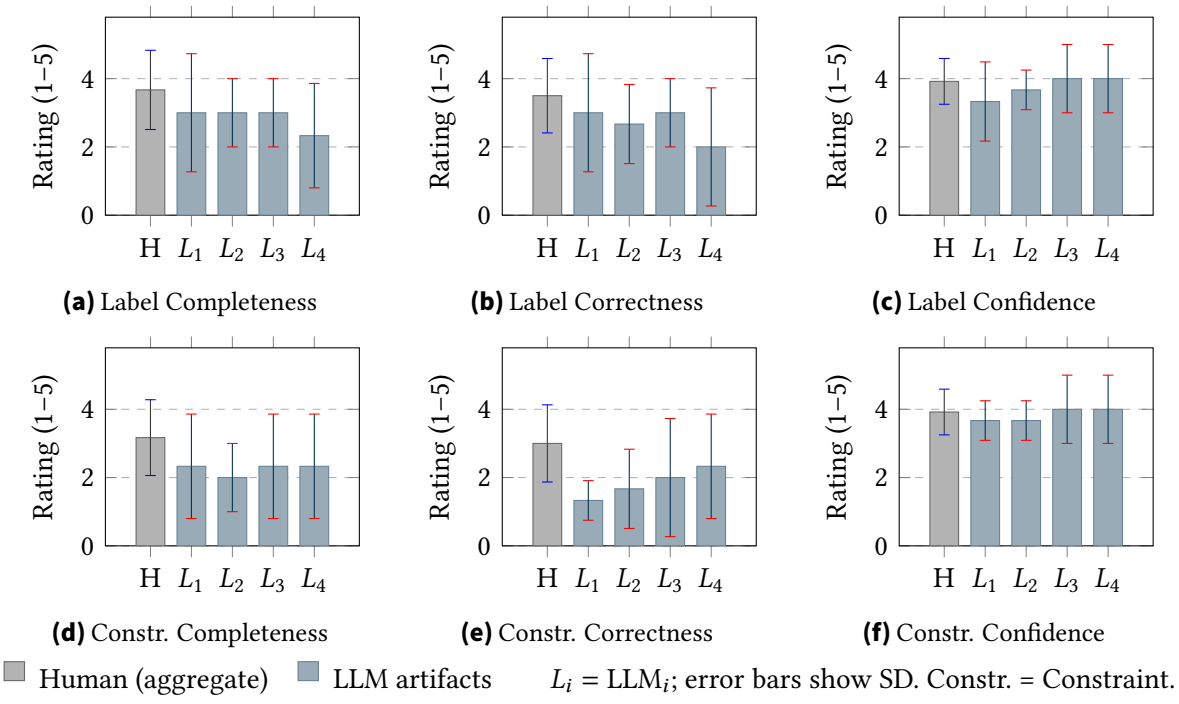


Figure 7.17.: Per-dimension ratings for the 4 LLM-derived artifacts compared to the human aggregate ($n_H = 9$). Likert scale 1–5; error bars show SD.

at 5.6% of the 18 issue type flags, compared to 18.75% of the 16 for human-created sets. For human-created sets, Incorrect Label accounted for the smallest share at 6.25% of the 16 issue type flags, compared to 16.7% of the 18 for LLM-derived sets.

Label Set	T	N	Miss.	Incorr.	Hall.	Vague	Other	Tot.	/Eval
$H_{1450856}$	H	2	1	0	0	1	0	2	1.00
$H_{2229857}$	H	1	1	0	0	0	1	2	2.00
$H_{4458842}$	H	1	1	0	1	0	0	2	2.00
$H_{4473222}$	H	2	0	0	0	0	1	1	0.50
$H_{6418222}$	H	2	1	1	1	1	0	4	2.00
$H_{7649803}$	H	1	0	0	1	0	0	1	1.00
$H_{8178528}$	H	1	0	0	0	1	0	1	1.00
$H_{9318487}$	H	1	1	0	0	1	0	2	2.00
$H_{9648566}$	H	1	1	0	0	0	0	1	1.00
LLM_{01}	L	3	0	0	0	1	1	2	0.67
LLM_{02}	L	3	1	0	0	3	0	4	1.33
LLM_{03}	L	3	2	1	0	3	0	6	2.00
LLM_{04}	L	3	1	2	1	2	0	6	2.00

T: H = Human, L = LLM; N = number of evaluators. Miss. = Missing Label, Incorr. = Incorrect Label, Hall. = Hallucination, Vague = Vague/Ambiguous Label, Tot. = total issues flagged, /Eval = issues per evaluator.

Table 7.8.: Issue flag counts by type for each evaluated privacy label set

For logical constraints, summarised in Table 7.9, human-created sets varied more widely in flags per evaluator (0.0–2.0) than LLM-derived sets (1.0–1.67). Incorrect Constraint accounted for the largest share in both groups, at 33.3% of the 15 issue type flags for human-created sets and 44.4% of the 18 for LLM-derived sets. Contradictory Constraint accounted the least share in both groups, with no flags in human-created sets and 5.6% of the 18 issue type flags for LLM-derived sets. For LLM-derived sets, Hallucinated Constraints share the least share at 5.6% of the 18 issue type flags, compared to 13.3% of the 15 for human-created sets.

Constr. Set	T	N	Miss.	Incorr.	Hall.	Contr.	OverP.	Other	Tot.	/Eval
$H_{1450856}$	H	2	0	1	0	0	2	0	3	1.50
$H_{2229857}$	H	1	0	1	0	0	0	0	1	1.00
$H_{4458842}$	H	1	1	1	0	0	0	0	2	2.00
$H_{4473222}$	H	2	0	0	0	0	1	0	1	0.50
$H_{6418222}$	H	2	1	0	2	0	1	0	4	2.00
$H_{7649803}$	H	1	0	1	0	0	0	0	1	1.00
$H_{8178528}$	H	1	0	0	0	0	0	0	0	0.00
$H_{9318487}$	H	1	1	1	0	0	0	0	2	2.00
$H_{9648566}$	H	1	1	0	0	0	0	0	1	1.00
LLM_{01}	L	3	1	2	0	1	0	1	5	1.67
LLM_{02}	L	3	2	2	0	0	0	1	5	1.67
LLM_{03}	L	3	0	2	0	0	1	0	3	1.00
LLM_{04}	L	3	1	2	1	0	1	0	5	1.67

T: H = Human, L = LLM; N = number of evaluators. Miss. = Missing Constraint, Incorr. = Incorrect Constraint, Hall. = Hallucination, Contr. = Contradictory, OverP. = Over/Too Permissive, Tot. = total issues flagged, /Eval = issues per evaluator.

Table 7.9.: Issue flag counts by type for each evaluated logical constraint set

Table 7.10 summarizes the inter-rater reliability for completeness and correctness ratings using Krippendorff's alpha [63] for interval data. Across all dimensions, reliability is low, with alpha values between -0.11 and 0.09, indicating no rater agreement beyond chance.

Dimension	Krippendorff's α	Interpretation	Items (N≥2)
Label Completeness	-0.111	unreliable	7
Label Correctness	0.002	unreliable	7
Constraint Completeness	-0.097	unreliable	7
Constraint Correctness	0.088	unreliable	7

Table 7.10.: Inter-rater reliability for Likert ratings

Similarly, Table 7.11 shows inter-rater reliability for issue flagging, computed using both Krippendorff's alpha for nominal data and raw percent agreement. We observe low inter-rater reliability for issue flagging as well, with alpha values between -0.15 and 0.23 for labels.

For constraints, Hallucination yielded the highest alpha at 0.51, still below the reliability threshold of 0.667, while the remaining categories spanned between -0.39 and 0.04.

Issue Category	Agree.%	Krippendorff's α	Interpretation	Items (N≥2)
Labels				
Missing Label	42.9%	-0.095	unreliable	7
Incorrect Label	66.7%	-0.150	unreliable	7
Hallucination	76.2%	0.233	unreliable	7
Vague/Ambiguous Label	52.4%	0.142	unreliable	7
Other	76.2%	0.026	unreliable	7
Constraints				
Missing Constraint	57.1%	0.042	unreliable	7
Incorrect Constraint	47.6%	-0.072	unreliable	7
Hallucination	90.5%	0.513	unreliable	7
Contradictory Constraint	90.5%	-0.333	unreliable	7
Over/Too Permissive	52.4%	-0.136	unreliable	7
Other	81.0%	-0.394	unreliable	7

Table 7.11.: Inter-rater reliability for issue type flagging

We summarize the self-assessment ratings for attention and motivation in Table 7.12. Evaluators reported limited motivation, with 2 ratings each for "Rather yes", "Rather no", and "No". Only one participant reported a distraction, with the rest reporting none.

Item	Response	N
SC03	Rather yes	2
	Rather not	2
	No	2
SC04	Completed all tasks as instructed	4
	Partially / with uncertainty	1
SC05	No distractions	5
	Distracted once	1
	Distracted several times	0

SC03: *Did you like participating in this study?* **SC04**: *Have you carried out all the tasks as described?* **SC05**: *Could you complete the questionnaire without being distracted?* $N = 6$ Part-2 evaluators.

Table 7.12.: Attitude and engagement of participants

Comparing rated sets by creator background, we find in Table 7.13 that all subgroups have at least one creator within each background dimension. We observe no pattern of higher ratings linked to creator background. Ratings do not differ by experience in deriving specifications from legal texts, familiarity with privacy law, familiarity with formal specification languages, or experience reviewing technical artifacts.

Grouping	Level	N_c	N_{rat}	Labels		Constraints	
				Compl.	Corr.	Compl.	Corr.
Deriv. Skill	Novice	4	6	4.00	3.67	3.83	3.67
	Adv. Beginner	2	3	3.33	3.67	2.33	2.67
	Competent	1	1	4.00	2.00	4.00	2.00
	Proficient	1	1	2.00	3.00	2.00	2.00
Legal Skill	Novice	2	4	3.50	3.50	2.75	2.75
	Adv. Beginner	3	4	4.00	3.75	3.75	3.75
	Competent	2	2	3.00	2.50	3.00	2.00
	Proficient	1	1	4.00	4.00	4.00	4.00
Spec. Skill	Novice	2	3	3.33	3.00	3.33	3.00
	Adv. Beginner	3	5	4.40	4.40	3.60	3.80
	Proficient	1	1	2.00	3.00	2.00	2.00
	Expert	2	2	3.00	2.00	3.00	2.00
Review Skill	Novice	2	3	3.33	3.00	3.33	3.00
	Adv. Beginner	2	3	4.00	4.33	3.00	3.33
	Competent	1	2	5.00	4.50	4.50	4.50
	Proficient	2	2	2.00	2.50	2.00	2.00
	Expert	1	1	4.00	2.00	4.00	2.00
DFD Exp.	0 years	3	4	3.50	3.25	3.50	3.25
	<1 year	1	2	4.00	4.50	2.50	3.00
	1-3 years	1	2	5.00	4.50	4.50	4.50
	3-5 years	2	2	3.00	2.50	3.00	2.00
	5+ years	1	1	2.00	2.00	2.00	2.00

N_c = creator artifacts in subgroup; N_{rat} = ratings received. Compl. = Completeness, Corr. = Correctness. Deriv. = experience deriving specifications from legal texts; Legal = familiarity with privacy law; Spec. = familiarity with formal specification languages; Review = experience reviewing technical artifacts. Ratings on human artifacts only; subgroup sizes are very small.

Table 7.13.: Average completeness and correctness ratings received by human-created label and constraint sets, grouped by creator background. Results are exploratory only.

The data companion set [46] contains the raw study data and the measured metrics.

7.3.4. Discussion

Human-created sets received higher average ratings than LLM-derived sets across all rating dimensions. Human-created sets averaged 1.33 and LLM-derived sets 3.00 across labels and constraints combined. Every individual LLM-derived set received lower average ratings than the human aggregate, as shown in Figure 7.17. The near-zero inter-rater reliability across all Likert dimensions (α between -0.111 and 0.088, Table 7.10) prevents treating the magnitude of this gap as a precise measurement. Thus, the comparison is directional only. With only 6 evaluators, formal rank-order consistency analysis is not feasible.

Beyond being a methodological limitation, the consistently unreliable Likert ratings are a substantive finding in themselves. Different evaluators applied different standards when judging the correctness and completeness of the same artifact. These dimensions therefore resist operationalisation as shared criteria for privacy specifications. The issue flagging results reinforce this: evaluators frequently flagged different issue types for the same artifact, consistent with the unreliable alpha values reported in Table 7.11. A confidence-agreement paradox further sharpens the picture: evaluators were individually certain of their judgements while collectively failing to agree on them. Evaluators were indeed individually moderately confident: LLM-derived constraint sets received 3.83 and human-created sets 3.92. Taken together, these results show that evaluators do not share a common standard of correctness or completeness for privacy specifications. This may reflect genuine heterogeneity in expert understanding rather than a gap in evaluation criteria alone.

An additional confound is that all evaluators completed the derivation task in *Part 1* before rating others' work in *Part 2*. Their own solution therefore served as an implicit reference point. This anchor effect is not directionally neutral. If human-created artifacts resemble typical solution patterns more closely than LLM-derived ones, evaluators may rate them more favourably. This would reflect similarity to their own work rather than actual quality. Such a bias could systematically inflate the observed human-LLM quality gap. At the same time, prior task exposure gave evaluators familiarity with the requirements and notation before reviewing others' work.

Among issue types, evaluators flagged incorrect constraints most frequently across both human-created and LLM-derived sets, and every constraint set received at least four issue type flags per evaluator on average. The issue flagging results carry more evidential weight than the Likert ratings for one specific category: constraint hallucination yields the highest inter-rater reliability among all issue categories ($\alpha = 0.51$, Table 7.11), though it remains unreliable. All other issue categories show near-zero or negative agreement. We observe incorrect constraints in both groups, indicating that the derivation task itself is error-prone independent of creator background. This is consistent with the creator self-assessments in Table 7.4: creators rated translating requirements into the notation at 3.50 on a five-point scale. Table 7.5 corroborates this, with creators noting specific challenges such as selecting the appropriate level of granularity and staying within the notation's logical constraints. Label quality may also propagate into constraint quality: one evaluator noted *"Some of the mistakes in Proposed Constraints follow from mistakes in Proposed Privacy Labels"*. If this

cascade holds, errors from the label derivation phase compound in the constraint derivation phase, a dynamic that applies to both human and LLM-generated artifacts.

We find in Figure 7.17 that no single LLM-derived set consistently outperformed the others across all rating dimensions. This indicates consistent output quality across pipeline runs rather than high run-to-run variance. At the same time, we observe vague or ambiguous labels and incorrect constraints across all four LLM-derived sets. The pipeline's validation and consistency checks therefore do not fully eliminate these failure modes. Their co-occurrence across the same sets is consistent with the cascade described above: a vague label may underspecify the intended privacy constraint, propagating into constraint incorrectness.

The subgroup analysis, presented in Table 7.13, shows no detectable pattern between creator background and the ratings their artifacts received. Evaluators did not consistently rate artifacts from more experienced creators higher than artifacts from less experienced ones. The creator notes in Table 7.5 offer a possible explanation: creators cite task-specific difficulties such as granularity decisions, the node-versus-data label distinction, and capturing constraint reasoning. These are not difficulties that domain experience would resolve. With the available subgroup sizes, the observed pattern is not statistically conclusive. A study with larger subgroups is needed to determine whether creator experience predicts artifact ratings.

Despite the quality gap, LLM-derived sets averaged 2.83, 2.67 on a five-point scale for completeness and correctness on labels. For constraints, LLM-derived sets averaged 2.25, 1.83 on the same scale. Given the near-zero inter-rater reliability, however, these averages should not be treated as precise quality estimates. Rater disagreement inflates variance around each average, and the directional comparison is a noisy estimate rather than a precise measurement of the quality gap.

One important qualification is the evaluation scope: the study used only four requirements from a single software context. The pipeline's derivation logic was built using a 71-requirement dataset, covering a broader range of requirement categories than the four used in this evaluation. Thus, the evaluation exercises only a small slice of that development context. With only four inputs, the aggregation logic that groups and refines labels across diverse requirement categories has limited material to work with. At the scale for which it was developed, the aggregation logic operates over a more diverse set of requirement categories. We therefore interpret the observed quality levels as an evaluation under constrained conditions, not as a characterisation of performance on larger requirement sets.

Despite these quality limitations, the LLM explanations included in the implementation expose the reasoning behind each derived label and constraint. This allows targeted review and correction rather than full re-derivation. Finally, the limited evaluator motivation reported in Table 7.12 is itself informative: most evaluators reported limited or no motivation to complete the rating task. The task is compliance-relevant yet perceived as burdensome even by domain experts, reinforcing the case for automated derivation support.

Returning to the evaluation **Goal 2**, these findings provide a directional answer to RQ2. Human-created sets received higher correctness and completeness ratings than LLM-derived sets (**Question 2.1**, **Question 2.2**), with the gap consistent across all four LLM-derived

sets. The issue type analysis (**Question 2.3**) shows that incorrect constraints are the most frequently flagged issue type for both groups. This indicates that the derivation task itself is error-prone regardless of creator background. The near-zero inter-rater reliability (**Question 2.4**) limits this comparison to directional conclusions and prevents precise quantification of the quality difference. These consistent below-human ratings establish a quality baseline and identify concrete improvement targets. We cannot, however, claim practical utility from these results. No effort comparison was conducted, and whether the exposed reasoning traces enable correction at lower cost than full re-derivation remains an open question.

7.3.5. Threats to Validity

We discuss threats to validity along the four dimensions described by Wohlin et al. [109] for empirical software engineering research.

Conclusion Validity The primary threat is low statistical power. With 10 creators, 6 evaluators, and only 4 LLM-derived sets, the study cannot support formal significance testing, and subgroup comparisons are exploratory only. Given the near-zero IRR and the small number of evaluators, the study could only have detected very large effect sizes. The reliability of the rating instrument poses a second, independent threat. Krippendorff's α falls below the reliability threshold across all rating dimensions, meaning individual ratings carry substantial uncertainty and only directional patterns admit meaningful interpretation. We aligned raters through written criteria rather than a dedicated calibration session, which may have left raters with differing understandings of correctness and completeness. The heterogeneity of the participant pool across all four skill dimensions compounds this. Individual differences between evaluators may dominate over the human-versus-LLM treatment effect, making observed rating gaps difficult to attribute to artifact origin alone. A third threat concerns the balance of the comparison itself. LLM-derived sets received on average 3.00 ratings each, compared to 1.33 for human-created sets, meaning the human baseline is estimated from fewer observations and is therefore less stable. Low evaluator motivation, as reported in Table 7.12, adds further noise to individual rating judgements on both sides of the comparison.

Internal Validity Several design choices protect internal validity. The blind design ensured evaluators did not know whether a set was human- or LLM-derived, reducing AI bias and confirmation bias. Random sampling of which sets each evaluator rated reduced order effects, and no participant rated their own submission. Despite these precautions, two confounds threaten causal interpretation. First, all evaluators completed the derivation task in *Part 1* on the same requirements before rating others' work in *Part 2*. This prior experience may have shaped evaluators' implicit quality standards. It is therefore difficult to determine whether rating differences reflect artifact quality or evaluator expectation. In particular, evaluators may rate artifacts that align with their own derivation approach more favourably than those that diverge from it, independently of actual quality. Second, we cannot rule out a learning effect within *Part 2*. As evaluators assessed four sets sequentially, their familiarity

with the rating criteria and the *neverFlows* notation improved, and later-evaluated sets may have received more calibrated ratings than earlier ones. Two selection effects further limit the interpretability of the human baseline. We recruited participants from our academic and professional networks, over-representing individuals with prior familiarity with privacy specification and potentially missing a broader practitioner population. Additionally, one creator's artifact was excluded because it was submitted after *Part 2* data collection started. If early submission correlates with motivation or task familiarity, this exclusion criterion may introduce a minor selection bias. A separate mortality concern arises from the 10 Part 1 creators: 4 did not proceed to *Part 2*. The remaining 6 evaluators cover a diverse range of backgrounds, partially mitigating this concern. Finally, task instrumentation introduced noise throughout. Evaluators reported difficulty with the node-versus-data label distinction and the *neverFlows* notation (Table 7.5). Evaluating four full sets in a single session also risks fatigue in later rating slots, despite the random assignment of sets.

Construct Validity Construct validity concerns arise from two sources: the measurement instruments and the representation of the constructs under study. Regarding measurement, inter-rater reliability below the reliability threshold across all rating dimensions shows that evaluators did not operationalise correctness and completeness consistently. The confidence dimension compounds this: high confidence ratings alongside near-zero agreement indicate that the measure captures individual decisiveness rather than shared certainty about artifact quality. The *neverFlows* formalism introduces a further instrument-specific bias. The formalism may influence which issue types evaluators can identify and how easily they can judge constraint quality, independently of constraint quality itself. Regarding construct representation, two opposing effects threaten the validity of the human baseline as a proxy for expert quality. Human creators worked in a survey setting without access to full system documentation or legal resources that practitioners ordinarily consult when deriving privacy specifications. This makes the human baseline a potentially conservative estimate of true expert quality. Conversely, creators knew their work would be peer-rated and likely applied more care than in a naturalistic setting. This may inflate the baseline relative to typical practitioner output. The net effect on the human baseline is uncertain. We therefore interpret the quality gap with both directions in mind. Finally, using a single LLM and a single requirement set means a narrow sample characterises each construct (*LLM-derived privacy specification quality* and *human expert quality*). The study therefore characterises each construct under limited sampling conditions.

External Validity External validity is constrained along four dimensions. First, evaluation used a single four-requirement set, which may not represent the diversity of privacy contexts and regulatory frameworks encountered in practice. The evaluation scope differs substantially from the development context: the pipeline was developed using a full 71 privacy requirements dataset, while the evaluation used only four requirements from a single context. The evaluation therefore exercises only a small slice of the input diversity for which the pipeline was developed, potentially putting the LLM at a disadvantage that would not persist at broader deployment. In particular, a four-requirement input does not fully exercise the aggregation logic that groups and refines labels across a diverse set of requirement categories, as that logic has limited material to operate on. Second, the

evaluation used a single LLM and a fixed pipeline configuration, resulting in findings that may not generalise to other models or prompt designs. The evaluation is also tied to the *neverFlows* formalism specific to the xDECAF framework. Findings about issue prevalence and output quality may therefore not transfer to other constraint languages or privacy analysis frameworks. Third, we recruited the participant pool from our academic and professional networks, over-representing individuals with prior familiarity with privacy specification tasks. This limits the generalisability of findings about human baseline quality and evaluation behaviour to the broader practitioner population. The intended users of such a pipeline, including software architects, legal engineers, and data protection officers, are not well-represented by an academic sample. Finally, the study ran at a fixed point in LLM development. Given the rapid pace of model capability improvements, findings about pipeline output quality are time-specific and may not reflect the performance of more recent or future LLM releases. As a proprietary frontier model, GPT-5.5 may not remain accessible to independent researchers seeking to replicate these results. A further external validity threat is the absence of downstream validation. We have not tested whether LLM-derived privacy specifications, when used as inputs to xDECAF, produce meaningful compliance analyses or propagate errors that compromise violation detection.

8. Cross-Study Discussion

This chapter synthesizes the findings from Chapters 6 and 7 in relation to the conceptual design outlined in Chapter 5. While Sections 6.3 and 7.3 evaluated each research question in isolation, this chapter examines what the combined findings imply for the conceptual design as a whole. In Section 8.1, we first synthesize findings across RQ1 and RQ2 to identify cross-cutting patterns. We then reflect on the limitations of the research in Section 8.2. This complements the study-level threats to validity discussed in Sections 6.3.5 and 7.3.5. Finally, we discuss research and practice implications and the future work directions they motivate in Section 8.3.

8.1. Cross-Study Synthesis

Both research questions investigated Large Language Model (LLM)-based derivation at different stages of Data Flow Analysis (DFA) preprocessing: RQ1 on Data Flow Diagram (DFD) generation from functional requirements, and RQ2 on privacy specification derivation from legal requirements. We conducted the studies independently, and their sample sizes preclude significance testing. Despite the small sample sizes, both yield a consistent directional finding: human-created artifacts received higher quality ratings than LLM-derived ones across all evaluated dimensions. This agreement across two structurally different tasks provides directional evidence within the methodological constraints shared by both studies. It does not establish generalizability beyond these settings: both studies recruited from the same personal and professional network, used the same measurement instrument, and each targeted a single use case. The structural differences between the two tasks strengthen the cross-study consistency as evidence. However, the shared recruitment network and measurement instrument mean the two studies cannot be treated as fully independent replications.

The quality gap profiles differ between the two studies. For RQ1, the completeness gap is more pronounced than the correctness gap, indicating the LLM captures DFD structure reasonably well but falls short on coverage of requirements. The gap in RQ2 is more uniform across completeness and correctness for both labels and constraints, with no single dimension standing out as the primary contributor.

The two studies also reveal contrasting patterns in LLM output consistency. For RQ1, quality varied widely across the four LLM-derived DFDs: LLM_4 approached human quality levels on both dimensions, while LLM_1 fell short. The four LLM-derived sets in RQ2, by contrast, clustered consistently below the human aggregate, with no single set outperforming the

others. Across RQ2, the pipeline therefore produced lower run-to-run variance than RQ1, but without the higher-quality outlier outputs that RQ1 showed. One cross-study confound warrants explicit acknowledgement: the two pipelines used different model versions (GPT-5.3-Codex [76] for RQ1 and GPT-5.5 [77] for RQ2). Differences in consistency patterns between the two studies may therefore partly reflect model version differences rather than pipeline design differences alone. Cross-study comparisons of output variance should be interpreted with this model version difference in mind.

The issue type distributions reveal a shared but asymmetric pattern. For RQ1, syntactic errors were rare across both groups, indicating the LLM correctly applies DFD grammar. The dominant issues for LLM-derived DFDs were missing elements and hallucinations, while evaluators primarily flagged human-created DFDs for vague labelling. For RQ2, hallucinations were less prevalent than in RQ1: for labels, vague or ambiguous labels were the dominant issue for LLM-derived sets, while missing labels dominated for human-created sets. For constraints, evaluators flagged incorrect constraints as the most frequent issue type for both groups. This indicates that the *neverFlow*s formalism introduces difficulty independent of artifact origin. The LLM's dominant issue type therefore varies by task. Missing elements and hallucinations dominate in RQ1, while vague labels characterize LLM-derived sets in RQ2.

Both studies yielded unreliable Krippendorff's alpha values across Likert dimensions, limiting comparisons to directional conclusions. We attribute this to two shared causes. First, neither task has a single correct solution. Both tasks admit multiple valid outputs. Within RQ1, different DFD representations can satisfy the same requirements. Within RQ2, different privacy label and constraint sets can satisfy the same legal requirements. Raters therefore lacked a shared ground truth. Second, in both studies evaluators were individually confident in their ratings yet collectively disagreed. This may indicate that correctness and completeness criteria for these artifact types are genuinely ambiguous, not merely difficult to apply. Alternatively, this may reflect heterogeneous rater understandings of what quality means for these artifact types: a question of differing quality conceptions rather than shared criteria applied inconsistently. That both causes recur across two structurally different tasks is itself a methodological finding. Operationalizing shared quality measures for artifact types that admit multiple valid solutions is not specific to this thesis. It constitutes an open challenge for the broader field of LLM-generated artifact evaluation, and one that larger samples alone cannot resolve. More data would produce more stable Likert estimates but would not address the underlying ambiguity.

Subgroup analyses for both RQ1 and RQ2 show no systematic trend in artifact quality across skill levels or domain experience. Open-text notes from participants in both studies point to task-specific difficulties that domain experience did not resolve. For RQ1, these included coverage and labelling precision. For RQ2, these covered granularity decisions and notation constraints. Given the small sample sizes, these findings are exploratory.

8.2. Limitations

The cross-study patterns identified in Section 8.1 hold within the following methodological boundaries. First, the sample sizes for both studies are small, which precludes statistical significance testing. All results are therefore directional and exploratory. Second, each study used a single use case. The findings may therefore not generalize to the full range of requirement and privacy specification variability encountered in practice. Third, both pipelines target specific formalisms defined within xDECAF [22]: RQ1 targets its DFD notation, and RQ2 targets its DFD annotation and *neverFlows* constraint notation. We did not evaluate transfer to other DFD dialects or privacy constraint representations. Furthermore, we did not examine whether the *neverFlows* formalism is expressively adequate for the full range of General Data Protection Regulation (GDPR) obligations. If the formalism cannot capture certain obligations, the pipeline's practical utility is bounded independently of output quality.

Fourth, the evaluation relied on subjective quality ratings from human evaluators, since both tasks admit multiple valid outputs without a shared ground truth. This introduced rater variability and unreliable inter-rater agreement, as Sections 6.3.5 and 7.3.5 discuss. A related structural limitation is the creator-rater confound: all *Part 2* evaluators in both studies had previously completed the derivation task on the same requirements in *Part 1*. Their own prior solutions may therefore have anchored their ratings in *Part 2*. If human-created artifacts cluster closer to the solutions participants produced themselves than LLM outputs do, this anchor effect systematically inflates the observed gap regardless of artifact quality. A further structural issue is asymmetric rating counts: LLM-derived artifacts received more ratings per item. The LLM group average is therefore more stable, while the human group average is more susceptible to fluctuation. The qualitative evaluator comments provide partial counter-evidence against a purely methodological explanation of the quality gap: for RQ1, evaluators described specific structural and coverage defects in LLM-derived DFDs (e.g., "half of the diagram missing") rather than style mismatches. This suggests at least some of the quality gap reflects genuine content-level differences. Unreliable inter-rater agreement, the creator-rater confound, and asymmetric rating counts each independently shift the group comparison in the observed direction. Together, they preclude any determination of whether the directional finding reflects a true quality difference, a style-alignment bias, or a combination of both. The true magnitude of any underlying quality difference therefore remains undetermined.

Fifth, the RQ2 evaluation used only four requirements from a legal privacy context. We developed the pipeline against a dataset of 71 legal privacy requirements [88] and designed it to handle large requirement sets. The evaluation exercises only a small slice of that development context. The RQ2 pipeline derives a reusable label schema by aggregating requirements. Four inputs provide insufficient material for this aggregation logic to operate as intended. The evaluation therefore captures a restricted and potentially unrepresentative operating point, and may place the LLM at a disadvantage that would diminish with larger requirement sets. Sixth, LLMs continue to evolve rapidly, so the findings may not generalize to future model generations. Also, we did not assess or control for input requirement

quality. The findings are therefore specific to the quality of the requirements used in each study. Higher-quality inputs may yield higher LLM output quality. Seventh, we evaluated a single pipeline implementation per research question, so the findings may not generalize to alternative pipeline designs or configurations. Finally, both pipelines are functional prototypes with known quality gaps that were not addressed in this study. The findings do not represent the ceiling of LLM-based derivation.

8.3. Implications and Future Work

Building on the synthesis outlined in Section 8.1 and limitations outlined in Section 8.2, we discuss research directions and practice implications.

Research Implications Closing the quality gap requires both pipeline improvements and methodological advances. For RQ1, non-deterministic LLM behavior that prompt design alone cannot stabilize motivates investigation of iterative refinement approaches. Two structural gaps in the pipeline also limit current findings. The atomicity gate cannot yet distinguish genuinely decomposable process clusters from intentionally aggregated Level-1 processes. Topology rules enforced through prompting rather than schema validation allow violations to propagate silently to the final artifact. Because incorrect constraints were the dominant issue type for both LLM-derived and human-created sets, the *neverFlows* formalism introduces difficulty independent of artifact origin. We should not attribute constraint issues solely to LLM limitations. We have not yet quantified whether label errors cascade into constraint errors through the derivation pipeline.

These findings open the following research directions. For RQ1, iterative prompting with structured feedback loops could reduce run-to-run variance by allowing the LLM to refine outputs across successive passes [110]. Addressing the atomicity gate and topology validation gaps would enable structured feedback loops to function as effective quality gates rather than best-effort filters. For RQ2, an explicit quality gate between the label and constraint derivation phases would allow practitioners to detect label errors before they propagate into constraint errors downstream. Meta-models for privacy labels could constrain the LLM to derive labels within predefined boundaries, thereby addressing the label consistency issues observed in this study [96]. Retrieval-augmented generation represents a further direction for RQ2: grounding label derivation in retrieved examples from an existing legal requirements corpus could reduce vague and incorrect labels without requiring model fine-tuning. Future work should also investigate whether label errors cascade into constraint errors and, if so, how much label quality independently contributes to constraint quality. Closing the quality gap also requires a shift in evaluation paradigm. As Section 8.1 shows, the unreliable inter-rater agreement may partly reflect heterogeneous quality conceptions rather than evaluators applying shared criteria inconsistently. Larger samples alone would produce more stable Likert estimates without resolving that underlying ambiguity. Future studies should therefore assess functional correctness rather than perceived artifact quality on abstract scales. One concrete criterion is whether executing LLM-derived DFDs and privacy

specifications in xDECAF correctly detects known privacy violations. Such a criterion provides a shared, objective reference point that subjective Likert ratings cannot supply. As Section 8.2 outlines, both studies used small participant samples. Once future work establishes shared quality criteria, replication with larger samples would enable significance testing and allow us to quantify the quality gap rather than only characterize its direction. Future studies should additionally recruit evaluators who did not complete the derivation task, providing a comparison group unaffected by the creator-rater confound. Comparing ratings from creator-evaluators and non-creator evaluators within the same study directly bounds the confound's contribution to the observed quality gap. Such a comparison would also establish whether the directional finding persists when the study design eliminates the anchor effect. A controlled effort comparison between LLM-assisted and fully manual derivation would determine whether the pipelines reduce practitioner effort, a question the current studies did not address. Comparing time-to-acceptable-artifact and residual error rates across conditions would establish whether the pipeline's drafting value outweighs its correction overhead.

Practice Implications Beyond these research directions, the findings carry concrete implications for practitioners using or extending the pipelines. Both implementations expose derivation reasoning that supports targeted review: practitioners can focus correction effort on specific elements rather than re-deriving artifacts from scratch. For RQ1, reviewers should prioritize completeness over correctness. The completeness gap appears more pronounced than the correctness gap, and missing elements are the dominant LLM issue type. Hallucinations ranked second. The requirements clarification step partly causes them, as the pipeline fills ambiguous specification gaps with LLM-generated assumptions. We therefore recommend resolving input ambiguities before running the pipeline to reduce hallucination frequency. Given the substantial run-to-run variance, we recommend generating multiple pipeline outputs. We recommend selecting the highest-quality candidate for expert review. This reduces the risk of starting from a poor draft without additional prompt engineering. Expert correction remains necessary regardless. Practitioners extending the RQ1 pipeline should address the atomicity gate and topology validation gaps before deploying outputs in production. For RQ2, practitioners should review labels before constraints rather than treating them as independent outputs, to detect whether label errors have cascaded into constraint errors. Across both pipelines, the output validation and retry mechanism detects structural errors before they propagate downstream. Practitioners extending either pipeline should preserve this mechanism. Removing it allows invalid outputs to reach subsequent phases undetected.

A human-in-the-loop approach targeting the label review step in RQ2 would allow practitioners to detect and correct label errors before constraint derivation runs. Whether this reduces the overall correction burden relative to fully automated derivation remains an empirical question for future effort comparison studies. Prompt optimization, including automated prompt generation and tuning, would lower per-run prompt engineering effort by treating the prompt as an optimizable parameter rather than a manually maintained artifact.

Cross-Cutting Implications A natural next extension of this work is the third step of the conceptual pipeline from Chapter 5: annotating DFDs with the derived privacy specifications. Until this step is implemented and evaluated, we cannot validate the end-to-end feasibility of LLM-based DFA automation. We cannot yet determine whether partially correct DFDs and privacy specifications produce analyzable compliance results when combined. They may instead propagate errors that invalidate the downstream analysis. This step would produce the first fully analyzed design artifact within this pipeline from natural language requirements. It constitutes the key feasibility test for the integrated pipeline. Two cross-cutting techniques could also improve both pipelines. Multi-agent debate frameworks let multiple LLMs evaluate and critique each other's outputs [79]. We could apply this pattern to both pipelines to target the ambiguity-driven errors observed in the requirements and derivation steps. We could fine-tune LLMs on domain-specific corpora of requirements and design artifacts to address the domain-specific error patterns and run-to-run variance observed across both studies. This approach requires labelled training data.

9. Conclusion

The xDECAF framework [22] supports privacy compliance analysis of software designs through Data Flow Analysis (DFA). Its input is a privacy-annotated Data Flow Diagram (DFD). Creating it requires a DFD derived from functional requirements and privacy specifications derived from legal requirements. Both steps currently require substantial manual effort. We investigated whether Large Language Model (LLM)-based pipelines can automate these two derivation steps.

We designed a three-step conceptual pipeline for LLM-based derivation of privacy-annotated DFDs from natural language requirements and implemented the first two steps. Both implementations used task-specific prompts, output schema validation, and retry mechanisms to catch structural errors before downstream propagation. We evaluated both pipelines through online user studies with human experts, each following a two-round blind study design and an individual Goal Question Metric (GQM)-based evaluation plan.

For RQ1, evaluators rated LLM-derived DFDs lower than manually created ones on both completeness and correctness, with the completeness gap more pronounced. Missing elements and hallucinations were the dominant issue types for LLM-derived outputs, while human-created DFDs drew flags primarily for vague labelling. One run received ratings closer to human quality levels than the other two, indicating non-trivial run-to-run variance. Inter-rater agreement was unreliable across all dimensions, with evaluators individually confident yet collectively disagreeing about artifact quality. For RQ2, evaluators rated LLM-derived labels and constraints lower than manually created ones across all four evaluated dimensions, with no single LLM-derived set approaching human quality levels. Incorrect constraints were the dominant issue type for both groups, indicating that the *neverFlows* formalism used by xDECAF introduced difficulty independent of artifact origin. Run-to-run variance across the LLM-derived sets was lower than in RQ1. Inter-rater agreement was similarly unreliable across all dimensions.

Four factors limit confidence in the findings. First, both studies used limited participant samples: we recruited 20 participants for RQ1 in Part 1 and 12 evaluators in Part 2. For RQ2, we recruited 10 participants in Part 1 and 6 evaluators in Part 2. Each study also evaluated a single use case and a single LLM configuration. Second, the creator-rater confound: evaluators who rated artifacts in Part 2 had previously derived their own solutions for the same inputs in Part 1. Their prior solutions may therefore have anchored their ratings toward their own work. As LLM-derived artifacts are likely more structurally dissimilar to participants' own solutions than human-created ones, this anchoring may systematically inflate the observed quality gap. Third, inter-rater agreement was unreliable across all Likert dimensions, meaning no quality dimension produced a reliable group consensus.

Fourth, rating counts were asymmetric: LLM-derived artifacts received more ratings per item than human-created ones, making the human group average less stable. The unreliable inter-rater agreement points to a broader methodological problem. Quality constructs such as correctness and completeness are not reliably operationalizable for artifact types that admit multiple valid solutions.

Contribution C1 is the LLM-based DFD derivation pipeline and contribution C2 is the privacy specification derivation pipeline. Both expose the derivation reasoning behind each generated element. This allows practitioners to target corrections at specific elements rather than re-deriving the artifact from scratch. The empirical evaluations establish a quality baseline and improvement targets for both tasks.

Despite their current quality limitations, both contributions have relevance beyond the xDECAF ecosystem. For practitioners and researchers within xDECAF, the evaluations establish a quality baseline for LLM-based automation of the two manual DFA preprocessing steps and identify the dominant failure modes that future work should address. For researchers building LLM-based pipelines for structured artifact derivation from natural language, three design elements transfer to comparable tasks: domain-specific prompt engineering with chain-of-thought decomposition and schema-constrained output validation, a structured retry mechanism for error containment, and a two-round blind evaluation design. These elements address challenges common to structured artifact derivation and are not tied to the neverFlows formalism or the privacy domain.

The quality gaps identified above point to a direct next step: implementing the third pipeline step, which annotates DFDs with derived privacy specifications. This step is the central feasibility test for the integrated pipeline, because it is the first point at which the pipeline combines all three outputs. Evaluating it requires a shift from subjective Likert ratings to functional evaluation criteria: testing whether LLM-derived artifacts, when executed together in xDECAF, correctly detect known privacy violations. Identifying which pipeline step causes failures additionally requires shared quality criteria for intermediate artifacts that admit multiple valid solutions. Establishing such criteria requires specifying which structural properties a correct artifact must satisfy—a definitional problem that larger evaluation samples cannot resolve.

Data Availability The data companion set [46] contains all prompts, validation schemas, prototype implementation code, user study materials, raw data, and measured metrics produced in this thesis.

Bibliography

[1] Abdel-Jaouad Aberkane. "Automated GDPR-Compliance in Requirements Engineering". In: International Conference on Advanced Information Systems Engineering (CAiSE). CEUR, 2021, pp. 21–29.

[2] Sallam Abualhaija, Marcello Ceci, and Lionel Briand. "Legal Requirements Analysis: A Regulatory Compliance Perspective". In: *Handbook on Natural Language Processing for Requirements Engineering*. Ed. by Alessio Ferrari and Gouri Ginde. Springer Nature Switzerland, 2025, pp. 209–242. DOI: 10.1007/978-3-031-73143-3_8.

[3] Sallam Abualhaija et al. "Automated Question Answering for Improved Understanding of Compliance Requirements: A Multi-Document Study". In: International Requirements Engineering Conference (RE). IEEE, 2022, pp. 39–50. DOI: 10.1109/RE54965.2022.00011.

[4] Sallam Abualhaija et al. "LLM-assisted Extraction of Regulatory Requirements: A Case Study on the GDPR". In: International Requirements Engineering Conference (RE). IEEE, 2025, pp. 142–154. DOI: 10.1109/RE63999.2025.00023.

[5] Amir Shayan Ahmadian et al. "Supporting privacy impact assessment by model-based privacy analysis". In: Symposium on Applied Computing (SAC). Pau France: ACM, 2018, pp. 1467–1474. DOI: 10.1145/3167132.3167288.

[6] Sharif Ahmed, Arif Ahmed, and Nasir U. Eisty. "Automatic Transformation of Natural to Unified Modeling Language: A Systematic Review". In: International Conference on Software Engineering Research, Management and Applications (SERA). IEEE, 2022, pp. 112–119. DOI: 10.1109/SERA54885.2022.9806783.

[7] AK Technik. "The Standard Data Protection Model". In: Independent Data Protection Supervisory Authorities of the Federation and the Länder. Vol. 3.0a. 2022.

[8] Haluk Akay and Sang-Gook Kim. "Extracting functional requirements from design documentation using machine learning". In: *Procedia CIRP* 100 (2021), pp. 31–36. DOI: 10.1016/j.procir.2021.05.005.

[9] Okhaide Akhigbe, Daniel Amyot, and Gregory Richards. "Information Technology Artifacts in the Regulatory Compliance of Business Processes: A Meta-Analysis". In: *E-Technologies*. Ed. by Morad Benyoucef, Michael Weiss, and Hafedh Mili. Vol. 209. Cham: Springer International Publishing, 2015, pp. 89–104. DOI: 10.1007/978-3-319-17957-5_6.

[10] Osamah AlDhafer, Irfan Ahmad, and Sajjad Mahmood. "An end-to-end deep learning system for requirements classification using recurrent neural networks". In: *Information and Software Technology (IST)* 147 (2022). DOI: 10.1016/j.infsof.2022.106877.

[11] Hanaa Alshareef et al. “Precise Analysis of Purpose Limitation in Data Flow Diagrams”. In: International Conference on Availability, Reliability and Security (ARES). Vienna Austria: ACM, 2022, pp. 1–11. DOI: 10.1145/3538969.3539010.

[12] Domenico Amalfitano et al. “A Research Roadmap for Augmenting Software Engineering Processes and Software Products with Generative AI”. In: *Transactions on Software Engineering and Methodology* (Jan. 30, 2026). DOI: 10.1145/3788879.

[13] Orlando Amaral, Sallam Abualhaija, and Lionel Briand. “ML-Based Compliance Verification of Data Processing Agreements against GDPR”. In: International Requirements Engineering Conference (RE). IEEE, 2023, pp. 53–64. DOI: 10.1109/RE57278.2023.00015.

[14] Chetan Arora, John Grundy, and Mohamed Abdelrazek. “Advancing Requirements Engineering through Generative AI: Assessing the Role of LLMs”. In: *Generative {AI} for Effective Software Development*. Springer, 2024. DOI: 10.1007/978-3-031-55642-5_6.

[15] Caroline M Ashworth. “Structured systems analysis and design method (SSADM)”. In: *Information and Software Technology (IST)* 30.3 (1988), pp. 153–163. DOI: 10.1016/0950-5849(88)90062-6.

[16] Vanessa Ayala-Rivera and Liliana Pasquale. “The Grace Period Has Ended: An Approach to Operationalize GDPR Requirements”. In: International Requirements Engineering Conference (RE). 2018, pp. 136–146. DOI: 10.1109/RE.2018.00023.

[17] Muhammad Ilyas Azeem and Sallam Abualhaija. “A Multi-solution Study on GDPR AI-enabled Completeness Checking of DPAs”. In: *Empirical Software Engineering* 29.4 (2024), p. 96. DOI: 10.1007/s10664-024-10491-3.

[18] Victor R Basili, Gianluigi Caldiera, and H Dieter Rombach. “The Goal Question Metric Approach”. In: *Encyclopedia of software engineering* (1994).

[19] Lenz Belzner, Thomas Gabor, and Martin Wirsing. “Large Language Model Assisted Software Engineering: Prospects, Challenges, and a Case Study”. In: International conference on bridging the gap between AI and reality. Ed. by Bernhard Steffen. Vol. 14380. Springer, 2024, pp. 355–374. DOI: 10.1007/978-3-031-46002-9_23.

[20] Oren Ben-Kiki, Clark Evans, and Ingy döt Net. *YAML Ain’t Markup Language*. URL: https://yaml.org/ (visited on 05/31/2026).

[21] Felix Bieker et al. “A Process for Data Protection Impact Assessment Under the European General Data Protection Regulation”. In: Annual Privacy Forum. Ed. by Stefan Schiffner et al. Springer, 2016, pp. 21–37. DOI: 10.1007/978-3-319-44760-5_2.

[22] Nicolas Boltz et al. “An Extensible Framework for Architecture-Based Data Flow Analysis for Information Security”. In: European Conference on Software Architecture. Springer, 2023. DOI: 10.48550/arXiv.2403.09402.

[23] Nicolas Boltz et al. *Bridging Legal and Technical Diciplines: A Model-Based Framework for Continuous Legal Assessments*. 2025.

[24] Tim Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. Request for Comments RFC 8259. Internet Engineering Task Force, Dec. 2017. DOI: 10.17487/RFC8259.

[25] Travis D. Breaux and Thomas Norton. "Legal Accountability as Software Quality: A U.S. Data Processing Perspective". In: International Requirements Engineering Conference (RE). IEEE, 2022, pp. 101–113. DOI: 10.1109/RE54965.2022.00016.

[26] Tom B. Brown et al. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2020, pp. 1877–1901. DOI: 10.48550/arXiv.2005.14165.

[27] Ann Cavoukian. "Privacy by Design: The 7 Foundational Principles". In: *Information and privacy comissioner of Ontario, Canada* 5 (2009), p. 12.

[28] Orlando Amaral Cejas, Sallam Abualhaija, and Lionel C. Briand. "CompAi: A Tool for GDPR Completeness Checking of Privacy Policies using Artificial Intelligence". In: International Conference on Automated Software Engineering. Sacramento CA USA: ACM, 2024, pp. 2366–2369. DOI: 10.1145/3691620.3695353.

[29] Orlando Amaral Cejas et al. "NLP-Based Automated Compliance Checking of Data Processing Agreements Against GDPR". In: *Transactions on Software Engineering* 49.9 (2023), pp. 4282–4303. DOI: 10.1109/TSE.2023.3288901.

[30] Sehrish Munawar Cheema, Saman Tariq, and Ivan Miguel Pires. "A natural language interface for automatic generation of data flow diagram using web extraction techniques". In: *Journal of King Saud University - Computer and Information Sciences* 35.2 (2023), pp. 626–640. DOI: 10.1016/j.jksuci.2023.01.006.

[31] Fabiano Dalpiaz. *Requirements data sets (user stories)*. Version 2. 2024. DOI: 10.17632/7zbk8zsd8y.2.

[32] Daniel Jurafsky and James H. Marin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd. 2025.

[33] Souvick Das et al. "A Multi-Agent RAG Framework for Regulatory Compliance Checking of Software Requirements". In: *Transactions on Software Engineering and Methodology* (2025). DOI: 10.1145/3785472.

[34] Tom DeMarco. "Structure Analysis and System Specification". In: *Software pioneers: contributions to software engineering* (2011), pp. 529–560. DOI: 10.1007/978-3-642-59412-0_33.

[35] Mina Deng et al. "A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements". In: *Requirements Engineering* 16.1 (Mar. 2011), pp. 3–32. DOI: 10.1007/s00766-010-0115-7.

[36] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: North American chapter of the association for computational linguistics: human language technologies. Vol. 1. 2019, pp. 4171–4186.

[37] draw.io Ltd. *draw.io*. URL: https://www.drawio.com/ (visited on 05/31/2026).

[38] Ashraf Elnashar, Jules White, and Douglas C. Schmidt. "Enhancing structured data generation with GPT-4o evaluating prompt efficiency across prompt styles". In: *Frontiers in Artificial Intelligence* 8 (2025), p. 1558938. DOI: 10.3389/frai.2025.1558938.

[39] Romina Etezadi et al. "Classifier or Prompt: A Case Study on Legal Requirements Traceability". In: *Empirical Software Engineering* 31.4 (2026), p. 85. DOI: 10.1007/s10664-026-10827-1.

[40] European Union. "General Data Protection Regulation". In: *Official Journal of the European Union* (2016).

[41] Alvan R. Feinstein and Domenic V. Cicchetti. "High agreement but low Kappa: I. the problems of two paradoxes". In: *Journal of Clinical Epidemiology* 43.6 (Jan. 1990), pp. 543–549. DOI: 10.1016/0895-4356(90)90158-L.

[42] Nick Feng et al. "Normative Requirements Operationalization with Large Language Models". In: International Requirements Engineering Conference (RE). IEEE, 2024, pp. 129–141. DOI: 10.1109/RE59067.2024.00022.

[43] Pere J. Ferrando et al. "Likert Scales: A Practical Guide to Design, Construction and Use". In: *Psicothema* 37.4 (2025), pp. 1–15. DOI: 10.70478/psicothema.2025.37.24.

[44] Alessio Ferrari, Sallam Abualhaija, and Chetan Arora. "Model Generation with LLMs: From Requirements to UML Sequence Diagrams". In: International Requirements Engineering Conference Workshops (REW). IEEE, 2024, pp. 291–300. DOI: 10.1109/REW61692.2024.00044.

[45] Megh Gala. *Unified Modeling Language (UML) generation from user requirements in natural language*. 2023.

[46] Gabriel Gehrig. *Data Companion Set - Master's Thesis - Gabriel Gehrig*. 2026. DOI: 10.5281/zenodo.20037873.

[47] Sepideh Ghanavati, Daniel Amyot, and Liam Peyton. "Compliance Analysis Based on a Goal-oriented Requirement Language Evaluation Methodology". In: International Requirements Engineering Conference (RE). IEEE, 2009, pp. 133–142. DOI: 10.1109/RE.2009.42.

[48] Sepideh Ghanavati, Daniel Amyot, and André Rifaut. "Legal goal-oriented requirement language (legal GRL) for modeling regulations". In: International Workshop on Modeling in Software Engineering. MiSE 2014. ACM, 2014, pp. 1–6. DOI: 10.1145/2593770.2593780.

[49] Sepideh Ghanavati et al. "Goal-oriented compliance with multiple regulations". In: International Requirements Engineering Conference (RE). IEEE, 2014, pp. 73–82. DOI: 10.1109/RE.2014.6912249.

[50] M. Glinz. "On Non-Functional Requirements". In: International Conference on Requirements Engineering. Delhi: IEEE, Oct. 2007, pp. 21–26. DOI: 10.1109/RE.2007.45.

[51] Michele Guerriero, Damian Andrew Tamburri, and Elisabetta Di Nitto. "Defining, enforcing and checking privacy policies in data-intensive applications". In: International Conference on Software Engineering for Adaptive and Self-Managing Systems. ACM, 2018, pp. 172–182. DOI: 10.1145/3194133.3194140.

[52] Jiacheng Guo et al. "Temporal Consistency for LLM Reasoning Process Error Identification". In: Findings of the Association for Computational Linguistics: EMNLP 2025. Association for Computational Linguistics, 2025. DOI: 10.18653/v1/2025.findings-emnlp.1205.

[53] Majid Hatamian et al. "A privacy and security analysis of early-deployed COVID-19 contact tracing Android apps". In: *Empirical Software Engineering* 26.3 (2021), p. 36. DOI: 10.1007/s10664-020-09934-4.

[54] Arshia Hemmat et al. "Research directions for using LLM in software requirement engineering: a systematic review". In: *Frontiers in Computer Science* 7 (2025), p. 1519437. DOI: 10.3389/fcomp.2025.1519437.

[55] Guntur Budi Herwanto. "Automating Data Flow Diagram Generation from User Stories Using Large Language Models". In: *Workshop on Natural Language Processing for Requirements Engineering* (2024).

[56] Elias Horner et al. "From Legal Texts to Defeasible Deontic Logic via LLMs: A Study in Automated Semantic Analysis". In: *e-prints* (2025). DOI: 10.48550/ARXIV.2506.08899.

[57] Mitra Bokaei Hosseini and Travis Breaux. "Privacy Requirements Acquisition and Analysis". In: *Handbook on Natural Language Processing for Requirements Engineering*. Ed. by Alessio Ferrari and Gouri Ginde. Springer Nature Switzerland, 2025, pp. 243–278. DOI: 10.1007/978-3-031-73143-3_9.

[58] ISO. "ISO/IEC 29100:2024 Information technology — Security techniques — Privacy framework". In: *International Organization for Standardization* 2 (2024). URL: https://www.iso.org/standard/85938.html.

[59] ISO, IEC, and IEEE. "ISO/IEC/IEEE 29148:2018 Systems and software engineering — Life cycle processes — Requirements engineering". In: *International Organization for Standardization* (2018), pp. 1–104. DOI: 10.1109/IEEESTD.2018.8559686.

[60] Kuzman Katkalov et al. "Model-Driven Development of Information Flow-Secure Systems with IFlow". In: International Conference on Social Computing. IEEE, 2013, pp. 51–56. DOI: 10.1109/SocialCom.2013.14.

[61] B.A. Kitchenham et al. "Preliminary guidelines for empirical research in software engineering". In: *Transactions on Software Engineering (TSE)* 28.8 (2002), pp. 721–734. DOI: 10.1109/TSE.2002.1027796.

[62] Ralf Kneuper. "Translating Data Protection into Software Requirements". In: International Conference on Information Systems Security and Privacy. SCITEPRESS, 2020, pp. 257–264. DOI: 10.5220/0008873902570264.

[63] Klaus Krippendorff. *Computing Krippendorff's Alpha-Reliability*. 2011.

[64] Klaus Krippendorff. *Content Analysis: An Introduction to Its Methodology*. SAGE Publications, Inc., 2019. DOI: 10.4135/9781071878781.

[65] Madhava Krishna et al. "Using LLMs in Software Requirements Specifications: An Empirical Evaluation". In: International Requirements Engineering Conference (RE). IEEE, 2024, pp. 475–483. DOI: 10.1109/RE59067.2024.00056.

[66] Selena Lamari et al. "Assisting the early development stages of privacy-aware software: The PRIAM tooled metamodel for GDPR". In: *Information and Software Technology* (2026). DOI: 10.1016/j.infsof.2026.108065.

[67] D.J. Leiner. *SoSci Survey*. Version 3.8.03. 2024. URL: https://www.soscisurvey.de/ (visited on 05/31/2026).

[68] R. Likert. "A technique for the measurement of attitudes". In: *Archives of Psychology* 22 140 (1932), pp. 55–55.

[69] Nelson F Liu et al. "Lost in the Middle: How Language Models Use Long Contexts". In: *Transactions of the association for computational linguistics* (2024).

[70] Fabrizio Marozzo. "Iterative Resolution of Prompt Ambiguities Using a Progressive Cutting-Search Approach". In: International Conference on Artificial Intelligence Applications and Innovations. Springer, 2025, pp. 210–224. DOI: 10.1007/978-3-031-96228-8_16.

[71] Denis Merigoux, Nicolas Chataing, and Jonathan Protzenko. "Catala: a programming language for the law". In: *ACM on Programming Languages* 5 (ICFP 2021), pp. 1–29. DOI: 10.1145/3473582.

[72] Majid Mollaeefar et al. "PILLAR: LINDDUN Privacy Threat Modeling Using LLMs". In: European Symposium on Security and Privacy Workshops (EuroS&PW). IEEE, 2025, pp. 278–286. DOI: 10.1109/EuroSPW67616.2025.00038.

[73] Max Moundas, Jules White, and Douglas C Schmidt. "Prompt Patterns for Structured Data Extraction from Unstructured Text". In: Conference on Pattern Languages of Programs, People, and Practices (PLoP). ACM, 2024, pp. 1–15. DOI: 10.64346/PLoP2024p24.

[74] Johannes J. Norheim et al. "Challenges in applying large language models to requirements engineering tasks". In: *Design Science* 10 (2024), e16. DOI: 10.1017/dsj.2024.8.

[75] Ozioma Okonicha and Andrey Sadovykh. "Enhancing GDPR Compliance Through NLP: Automated Policy Evaluation and Legal Document Analysis". In: *System Design in Software Engineering*. Ed. by Radek Silhavy and Petr Silhavy. Springer Nature Switzerland, 2026, pp. 408–423. DOI: 10.1007/978-3-031-94770-4_34.

[76] OpenAI. *Introducing GPT-5.3-Codex*. Feb. 5, 2026. URL: https://openai.com/index/introducing-gpt-5-3-codex/ (visited on 05/31/2026).

[77] OpenAI. *Introducing GPT-5.5*. Apr. 23, 2026. URL: https://openai.com/index/introducing-gpt-5-5/ (visited on 05/31/2026).

[78] OpenAI. *OpenAI Python API library*. URL: https://developers.openai.com/api/reference/python/ (visited on 05/31/2026).

[79] Marc Oriol et al. “Multi-Agent Debate Strategies to Enhance Requirements Engineering with Large Language Models”. In: International Requirements Engineering Conference (RE). IEEE, 2025, pp. 527–534. DOI: 10.1109/RE63999.2025.00063.

[80] Paul N. Otto and Annie I. Anton. “Addressing Legal Requirements in Requirements Engineering”. In: International Requirements Engineering Conference. IEEE, 2007, pp. 5–14. DOI: 10.1109/RE.2007.65.

[81] Pohl. *Requirements Engineering: Fundamentals, Principles, and Techniques*. 2nd ed. 2025. ISBN: 978-3-662-69204-2.

[82] L. Protsko, P. Sorenson, and J. Demblay. “Automatic Generation of Data Flow Diagrams From A Requirements Specification Language”. In: *ICIS 1984 Proceedings* (1984). URL: https://aisel.aisnet.org/icis1984/19.

[83] Pydantic. *Pydantic Validation*. URL: https://pydantic.dev/docs/validation/2.12/get-started/ (visited on 05/31/2026).

[84] Python. *Python 3.12 documentation*. 2023. URL: https://docs.python.org/3.12/ (visited on 05/31/2026).

[85] Gokul Rejithkumar and Preethu Rose Anish. “NICE: Non-Functional Requirements Identification, Classification, and Explanation Using Small Language Models”. In: International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). IEEE, 2025, pp. 284–295. DOI: 10.1109/ICSE-SEIP66354.2025.00031.

[86] Sandra Domenique Ringmann, Hanno Langweg, and Marcel Waldvogel. “Requirements for Legally Compliant Software Based on the GDPR”. In: *On the Move to Meaningful Internet Systems. OTM 2018 Conferences*. Ed. by Hervé Panetto et al. Vol. 11230. Springer International Publishing, 2018, pp. 258–276. DOI: 10.1007/978-3-030-02671-4_15.

[87] Suzanne Robertson and James Robertson. *Mastering the Requirements Process: Getting Requirements Right*. 4th ed. Addison-Wesley, 2024. 656 pp. ISBN: 978-0-13-796950-0.

[88] Pattaraporn Sangaroonsilp et al. “A taxonomy for mining and classifying privacy requirements in issue reports”. In: *Information and Software Technology* 157 (2023), p. 107162. DOI: 10.1016/j.infsof.2023.107162.

[89] Pattaraporn Sangaroonsilp et al. *Interactive GDPR-Compliant Privacy Policy Generation for Software Applications*. 2024. DOI: 10.48550/arXiv.2410.03069.

[90] Gautam Savaliya et al. “PriMod4AI: Lifecycle-Aware Privacy Threat Modeling for AI Systems using LLM”. In: Workshop on LLM Assisted Security and Trust Exploration (LAST-X). 2026. DOI: 10.14722/last-x.2026.23080.

[91] Kurt Schneider, Farnaz Fotrousi, and Rebekka Wohlrab. “A Reference Model for Empirically Comparing LLMs with Humans”. In: International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS). IEEE, 2025, pp. 130–134. DOI: 10.1109/ICSE-SEIS66351.2025.00018.

[92] Stephan Seifermann et al. “Detecting violations of access control and information flow policies in data flow diagrams”. In: *Journal of Systems and Software* 184 (2022). DOI: 10.1016/j.jss.2021.111138.

[93] Wuqiang Shen et al. “An NLP-Based Method for Assessment of GDPR Compliance in Data Privacy Agreements”. In: International Conference on Artificial Intelligence, Automation and High Performance Computing. ACM, 2024, pp. 466–471. DOI: 10.1145/3690931.3691009.

[94] Mohammed Latif Siddiq et al. *Assessing the Software Security Comprehension of Large Language Models*. 2025. DOI: 10.48550/arXiv.2512.21238.

[95] Boyce Sigweni and Martin Shepperd. “Using blind analysis for software engineering experiments”. In: International Conference on Evaluation and Assessment in Software Engineering. ACM, 2015, pp. 1–6. DOI: 10.1145/2745802.2745832.

[96] Anmol Singhal and Travis Breaux. “Legal Requirements Translation from Law”. In: International Requirements Engineering Conference (RE). IEEE, 2025, pp. 205–217. DOI: 10.1109/RE63999.2025.00028.

[97] Laurens Sion et al. “An Architectural View for Data Protection by Design”. In: International Conference on Software Architecture (ICSA). IEEE, 2019, pp. 11–20. DOI: 10.1109/ICSA.2019.00010.

[98] Ian Sommerville. *Software engineering*. Tenth edition. Always learning. Pearson, 2016. ISBN: 978-1-292-09613-1 978-1-292-09614-8.

[99] Sonal Sonawane and Shubha Puthran. “Extracting use case elements from requirement documents: a natural language processing approach”. In: *Requirements Engineering* 30.3 (2025), pp. 283–310. DOI: 10.1007/s00766-025-00445-6.

[100] Damiano Torre et al. “Modeling data protection and privacy: application and experience with GDPR”. In: *Software and Systems Modeling* 20.6 (2021), pp. 2071–2087. DOI: 10.1007/s10270-021-00935-5.

[101] Damiano Torre et al. “Using Models to Enable Compliance Checking Against the GDPR: An Experience Report”. In: International Conference on Model Driven Engineering Languages and Systems (MODELS). ACM, 2019, pp. 1–11. DOI: 10.1109/MODELS.2019.00-20.

[102] Amos Tversky and Daniel Kahneman. “Judgment under Uncertainty: Heuristics and Biases”. In: *science* 185 (1974), pp. 1124–1131.

[103] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in neural information processing systems* 30 (2017).

[104] Hugo Villamizar et al. “Prompts as Software Engineering Artifacts: A Research Agenda and Preliminary Findings”. In: International Conference on Product-Focused Software Process Improvement. Springer, 2026, pp. 470–478. DOI: 10.1007/978-3-032-12089-2_32.

[105] Beian Wang et al. “How LLMs Aid in UML Modeling: An Exploratory Study with Novice Analysts”. In: International Conference on Software Services Engineering (SSE). IEEE, 2024, pp. 249–257. DOI: 10.1109/SSE62657.2024.00046.

[106] Alan F Westin. “Privacy And Freedom”. In: *Washington and Lee Law Review* 25 (1968).

[107] Tharaka Wijesundara, Mathew Warren, and Nalin Arachchilage. *GDPRShield: AI-Powered GDPR Support for Software Developers in Small and Medium-Sized Enterprises*. 2025. DOI: 10.48550/arXiv.2505.12640.

[108] Brandon T. Willard and Rémi Louf. *Efficient Guided Generation for Large Language Models*. Aug. 19, 2023. DOI: 10.48550/arXiv.2307.09702.

[109] Claes Wohlin et al. *Experimentation in Software Engineering*. Vol. 236. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. DOI: 10.1007/978-3-662-69306-3.

[110] Mohammad Amin Zadenoori et al. *Large Language Models (LLMs) for Requirements Engineering (RE): A Systematic Literature Review*. 2025. DOI: 10.48550/arXiv.2509.11446.

[111] Wayne Xin Zhao et al. *A Survey of Large Language Models*. 2023. DOI: 10.48550/arXiv.2303.18223.

[112] Xiyu Zhou et al. "Using LLMs in Generating Design Rationale for Software Architecture Decisions". In: *Transactions on Software Engineering and Methodology* (2025). DOI: 10.1145/3785010.

A. Appendix

A.1. Additional Results of User Study for RQ1

ID	DFD Skill	TM Skill	Role(s)	DFD Exp.	Tool	Eval?
P01	Adv. Beginner	Novice	Student (Grad/PhD)	<1 year	draw.io	✓
P02	Competent	Adv. Beginner	Student (UG)	<1 year	xDECAF	✓
P03	Proficient	Expert	Industry Practitioner	5+ years	xDECAF	✗
P04	Expert	Expert	Student (UG), Academic Researcher	1-3 years	xDECAF	✓
P05	Novice	Novice	Student (Grad/PhD)	0 years	draw.io	✓
P06	Expert	Expert	Student (Grad/PhD)	1-3 years	xDECAF	✓
P07	Adv. Beginner	Adv. Beginner	Student (Grad/PhD)	1-3 years	xDECAF	✓
P08	Competent	Expert	Student (Grad/PhD)	1-3 years	xDECAF	✓
P09	Adv. Beginner	Adv. Beginner	Student (UG)	<1 year	xDECAF	✗
P10	Competent	Competent	Student (UG)	1-3 years	xDECAF	✓
P11	Novice	Adv. Beginner	Student (UG)	<1 year	xDECAF	✗
P12	Novice	Competent	Student (Grad/PhD)	<1 year	draw.io	✗
P13	Novice	Novice	Student (Grad/PhD)	0 years	xDECAF	✓
P14	Novice	Novice	Student (Grad/PhD)	0 years	draw.io	✗
P15	Novice	Novice	Student (Grad/PhD)	<1 year	xDECAF	✗
P16	Adv. Beginner	Adv. Beginner	Student (Grad/PhD)	<1 year	draw.io	✓
P17	Expert	Proficient	Industry Practitioner	1-3 years	draw.io	✓
P18	Proficient	Competent	Industry Practitioner	3-5 years	draw.io	✓
P19	Competent	Proficient	Student (Grad/PhD), Industry Practitioner	1-3 years	draw.io	✗
P20	Adv. Beginner	Novice	Student (Grad/PhD)	<1 year	draw.io	✗

Table A.1.: Demographic overview of study participants, user study 1 ($N = 20$). *DFD Skill* describes the participant's self-assessed experience with DFDs, and *TM Skill* reports their self-assessed experience with threat modelling. *Eval?* indicates whether the participant also completed Part 2.

ID	Challenges / Feedback
P01	choosing the details degree
P02	One challenge I encountered was to think about the level of detail in which I have to dive into, which was not that clear to me.
P04	The responsibilities (e.g. what a process or external entity, like requesting pending orders, belongs to) are not very clear.
P05	what to explicitly represent as an entity, how to represent automatic processes and conditional processes
P07	I was not sure in how much detail I needed Function Nodes. For example the entire buy block is one function node, but I could have easily split it. I was not sure if system needed to be its own I/O node or if that was sufficiantly encapsulated by the function nodes. I was not sure whether the sending stuff to customer part needed to be modelled as it happens outside of the system itsself
P09	I wasn't always sure what to call the flows/processes (e.g. Inventory Update, deduct_quantity). Also I wasn't sure whether to model the physical items or if this model was supposed to only show the digital system.
P10	The level of detail with which the system is to be modeled is not described as clearly as I would have wished / Some parts of the system are left a little bit vague. I feel confident in my guess but in practice, models may differ in detail.
P11	In what depth the specific functions of the system need to be modeled - e.g. the intro example modeled a full login and registration, despite not being in described in the instructions in that detail
P14	- How to correctly name the data flowing between processes - Is sending a stateless request also a dataflow? - Unclear if physical processes are part of modelling
P15	Unsure, if "the system" is a single service or different ones
P17	AFAIK, there are no "official" symbols for DFDs, as this has not been standardized yet. There are multiple authors writing on DFD methodology, and they all might be use their own symbols and notation rules. In addition, there are further evolutions of DFDs such as threat models, which even extend notation rules and symbols by trust zones / groups / zones. Apart from that, the introducing example of this study was unnecessary complex, especially as its only purpose was to explain how DFDs work.
P18	Stencils that have been required from the description are not available easy to use in draw.io
P19	Unclear how / if actors should be modelled. Unclear how internal systems should be modelled, i.e. as a separate entity or implicitly within a process.

Blank and 'none' responses omitted. $N_{\text{responses}} = 13$ of 20 participants.

Table A.2.: Part 1 creator open-text responses to "What challenges did you encounter?"

DFD	Type	N	Completeness Avg (SD)	Correctness Avg (SD)	Confidence Avg (SD)
H_{126225}	H	1	5.00	5.00	5.00
H_{131244}	H	2	4.50 (0.71)	2.50 (0.71)	4.50 (0.71)
H_{216739}	H	1	1.00	4.00	5.00
H_{343962}	H	2	4.50 (0.71)	4.50 (0.71)	5.00 (0.00)
H_{356787}	H	2	4.50 (0.71)	3.50 (2.12)	3.00 (0.00)
H_{388389}	H	1	4.00	4.00	4.00
H_{671858}	H	2	3.50 (0.71)	3.50 (0.71)	4.00 (1.41)
H_{688508}	H	2	1.50 (0.71)	2.00 (0.00)	4.00 (0.00)
H_{719176}	H	2	4.50 (0.71)	4.00 (0.00)	4.50 (0.71)
H_{731262}	H	1	5.00	5.00	4.00
H_{770487}	H	2	4.50 (0.71)	4.50 (0.71)	5.00 (0.00)
H_{781453}	H	2	5.00 (0.00)	4.50 (0.71)	5.00 (0.00)
H_{835392}	H	1	4.00	2.00	4.00
H_{872246}	H	1	5.00	5.00	5.00
H_{876646}	H	2	3.00 (1.41)	2.00 (0.00)	5.00 (0.00)
LLM_1	L	6	1.83 (0.98)	2.33 (1.21)	4.50 (0.84)
LLM_2	L	6	2.00 (1.09)	3.17 (0.75)	4.67 (0.82)
LLM_3	L	6	2.83 (1.47)	2.50 (1.52)	4.50 (0.84)
LLM_4	L	6	3.17 (1.47)	3.67 (0.82)	4.00 (0.89)

Table A.3.: Per-DFD completeness, correctness, and confidence ratings (**H** = human-created, **L** = LLM-derived; N = number of evaluators; Likert scale 1–5).

DFD	M	H	S	L	V	O	T	Evaluator Comments
H_{126225}	0	0	0	0	1	0	1	*"Very specific and complete DFD."*
H_{131244}	0	1	0	0	0	2	3	*"No"*
H_{216739}	1	0	0	0	1	0	2	—
H_{343962}	0	0	0	1	2	0	3	—
H_{356787}	1	0	1	0	0	0	2	*"flow labels are specific (good), but it includes some overly granular processes (e.g. receive payment status). also the overall process "chapters system" kind of mixes level 0 with level 1 dfd"*
H_{388389}	0	0	0	1	0	0	1	—
H_{671858}	1	0	0	1	1	1	4	—
H_{688508}	1	0	0	0	2	1	4	*"Looks very bad"*
H_{719176}	0	1	0	1	0	1	3	*"Very nice format, very readable. Different naming conventions for nodes/edges might be useful."*
H_{731262}	0	0	0	0	0	0	0	—
H_{770487}	1	0	0	0	0	2	3	—
H_{781453}	0	0	0	0	0	0	0	—
H_{835392}	0	1	0	0	0	0	1	—
H_{872246}	0	0	0	0	0	0	0	—
H_{876646}	2	2	1	1	2	1	9	—

M = Missing Element, H = Hallucination, S = Syntactic Error, L = Logic Error, V = Vague Labeling, O = Other, T = Total.

Table A.4.: Issue type counts and evaluator comments for human-created Data Flow Diagrams (DFDs) (**H** = human; *N* = evaluators; */Eval* = issues per evaluator).

A.2. Additional Results of User Study for RQ2

ID	Deriv.	Legal	Spec.	Review	Role(s)	DFD Exp.
P01	Novice	Novice	Novice	Novice	Student (Grad/PhD)	0 years
P02	Novice	Proficient	Novice	Novice	—	0 years
P03	Novice	Adv. Beginner	Adv. Beginner	Competent	Student (UG)	1-3 years
P04	Proficient	Competent	Proficient	Proficient	Student (Grad/PhD)	3-5 years
P05	Adv. Beginner	Novice	Adv. Beginner	Adv. Beginner	Student (Grad/PhD)	<1 year
P06	Competent	Competent	Expert	Expert	Student (Grad/PhD), Academic Researcher	3-5 years
P07	Novice	Adv. Beginner	Adv. Beginner	Adv. Beginner	Student (Grad/PhD)	0 years
P08	Adv. Beginner	Adv. Beginner	Expert	Proficient	Industry Practitioner	5+ years
P09	Novice	Adv. Beginner	Novice	Adv. Beginner	Student (Grad/PhD)	<1 year
P10	Novice	Novice	Novice	Adv. Beginner	Student (Grad/PhD)	<1 year

Skill columns show self-assessed level on Novice–Expert scale. Deriv. = Experience deriving technical specifications from legal texts; Legal = Familiarity with privacy law and regulations; Spec. = Familiarity with formal specification languages or constraint modelling; Review = Experience reviewing technical artefacts.

Table A.5.: Demographic overview of study participants, user study 2 ($N = 10$). Including their self assessed skills, job roles and experience with DFDs

Artifact	T	N	**Labels** Avg (SD)			**Constraints** Avg (SD)		
			Compl.	**Corr.**	**Conf.**	**Compl.**	**Corr.**	**Conf.**
$H_{1450856}$	H	2	4.00 (1.41)	4.50 (0.71)	4.50 (0.71)	2.50 (0.71)	3.00 (1.41)	4.50 (0.71)
$H_{2229857}$	H	1	4.00	4.00	4.00	4.00	4.00	4.00
$H_{4458842}$	H	1	2.00	3.00	3.00	2.00	2.00	3.00
$H_{4473222}$	H	2	5.00 (0.00)	4.50 (0.71)	4.00 (0.00)	4.50 (0.71)	4.50 (0.71)	4.00 (0.00)
$H_{6418222}$	H	2	3.00 (1.41)	2.50 (0.71)	4.00 (0.00)	3.00 (1.41)	2.50 (0.71)	4.00 (0.00)
$H_{7649803}$	H	1	4.00	2.00	3.00	4.00	2.00	3.00
$H_{8178528}$	H	1	4.00	4.00	3.00	4.00	4.00	3.00
$H_{9318487}$	H	1	2.00	2.00	4.00	2.00	2.00	4.00
$H_{9648566}$	H	1	4.00	4.00	5.00	2.00	2.00	5.00
LLM_{01}	L	3	3.00 (1.73)	3.00 (1.73)	3.33 (1.16)	2.33 (1.53)	1.33 (0.58)	3.67 (0.58)
LLM_{02}	L	3	3.00 (1.00)	2.67 (1.16)	3.67 (0.58)	2.00 (1.00)	1.67 (1.16)	3.67 (0.58)
LLM_{03}	L	3	3.00 (1.00)	3.00 (1.00)	4.00 (1.00)	2.33 (1.53)	2.00 (1.73)	4.00 (1.00)
LLM_{04}	L	3	2.33 (1.53)	2.00 (1.73)	4.00 (1.00)	2.33 (1.53)	2.33 (1.53)	4.00 (1.00)

T: H = Human, L = LLM. Compl. = Completeness, Corr. = Correctness, Conf. = Confidence. Scale 1–5.

Table A.6.: All evaluator Likert ratings by artifact, user study 2