



Interactive Multi-Agent Aggregation and Combination for Data Quality Assessment

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering

eingereicht von

Philipp Stöger, BSc

Matrikelnummer 11777723

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Thilo Sauter

Mitwirkung: Dipl.-Ing. Christian Stippel, BSc

Wien, 1. Februar 2026

Philipp Stöger

Thilo Sauter



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.



Interactive Multi-Agent Aggregation and Combination for Data Quality Assessment

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering

by

Philipp Stöger, BSc

Registration Number 11777723

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Thilo Sauter

Assistance: Dipl.-Ing. Christian Stippel, BSc

Vienna, February 1, 2026

Philipp Stöger

Thilo Sauter



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Philipp Stöger, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 1. Februar 2026

Philipp Stöger



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Danksagung

Zuerst möchte ich mich für die großartige Gelegenheit bedanken, im Rahmen des QuanTD-Projekts mitwirken zu dürfen. Meine Masterarbeit als Teil eines wissenschaftlichen Forschungsprojekts umzusetzen, war nicht nur eine große Motivation, sondern vor allem eine besondere Ehre. Besonders dankbar bin ich den Personen, die mir die Möglichkeit gegeben haben, diese Arbeit zu verwirklichen. Allen voran danke ich meinem Betreuer Thilo Sauter sowie dem Projektleiter Ralph Hoch, die mich in das Projekt geholt und eingebettet haben.

Mein größter Dank gilt jedoch meinem Betreuer Christian Stippel, den ich hier lieber als Freund nennen würde. Er stand mir nicht nur jederzeit bei meiner Masterarbeit zur Seite, sondern auch während meines gesamten Studiums. Durch ihn durfte ich Ralph kennenlernen, und letztlich ist er auch der Grund, warum diese Masterarbeit heute existiert. Ohne ihn wäre ich buchstäblich nicht dort, wo ich heute bin. Danke, Christian!

Außerdem möchte ich mich herzlich bei all meinen Freundinnen und Freunden bedanken, die mich in dieser Zeit nicht nur unterstützt, sondern auch tatkräftig an der User Study mitgewirkt haben. Danke für eure Geduld, Kraft und Fürsorge in dieser Zeit. Ohne euch wäre diese Arbeit ebenfalls nicht zustande gekommen. Besonderer Dank gilt dabei Natalie Puzter, die mir nicht nur den nötigen Raum gegeben, sondern mich tagtäglich dabei unterstützt hat, tiefer in die Forschung einzutauchen.

Zu guter Letzt danke ich meinen Eltern, die mir nicht nur meinen akademischen Werdegang ermöglicht, sondern mir jeden Tag Mut und Kraft zugesprochen und mir den Weg vom kleinen geliebten Dorf in die Großstadt ermöglicht haben.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Die Zuverlässigkeit datengetriebener Systeme hängt maßgeblich von ihrer „Fitness for Use“ (Gebrauchstauglichkeit) ab. Diese wird als die Fähigkeit eines Datensatzes definiert, spezifische Aufgaben oder Entscheidungskontexte zu unterstützen. In der Praxis wird diese Eigenschaft meist durch die Kombination mehrerer Kriterien der Datenqualität überprüft, statt sich auf eine einzelne Metrik zu verlassen. Aktuelle Systeme zur Datenqualitätssicherung erzwingen einen Kompromiss zwischen Ausdruckstärke und Zugänglichkeit. Während Code-basierte Frameworks detaillierte und maßgeschneiderte Prüfungen ermöglichen, setzen sie Programmierkenntnisse voraus. Visuelle Lösungen hingegen reduzieren die Bewertung auf Dashboards oder aggregierte Scores. Diese präsentieren zwar Ergebnisse, legen jedoch die zugrunde liegenden Kombinationen aus Prüfungen und Annahmen nicht offen. Zudem führt die zunehmende Integration von Large Language Models zu einem „Black-Box-Verifikationsproblem“, wodurch Nutzer:innen gezwungen sind, verborgenen, generierten Regeln blind zu vertrauen. Um diesen Herausforderungen zu begegnen, schlägt diese Arbeit ein „Glassbox“-Interaktionsparadigma vor. Sie präsentiert Design, Implementierung und Evaluation eines webbasierten Prototyps, der Datenqualität als interaktive, gerichtete, azyklische Graphen modelliert. Der Prototyp ermöglicht das interaktive Hervorheben betroffener Datenfelder mithilfe einzelner Graph-Komponenten, um die Bewertungslogik transparent zu machen. Zusätzlich integrieren wir einen KI-Assistenten, der aus natürlicher Sprache editierbare Graphen erstellt.

Die Evaluation durch eine Experten-Fokusgruppe und eine Nutzungsstudie ($N = 14$) zeigt, dass visuelle Aggregations- und Kombinationstechniken die Transparenz und Benutzbarkeit des Bewertungsprozesses verbessern. Technische Teilnehmer:innen der Studie lösten sowohl manuelle als auch KI-gestützte Aufgaben mit 100% Genauigkeit. Nicht-technische Teilnehmer erreichten 100% Genauigkeit bei der einfachen manuellen Aufgabe und 71,4% bei der komplexeren KI-gestützten Aufgabe. Teilnehmer:innen bevorzugten den KI-gestützten Prozess und bewerteten ihn mit einem „exzellenten“ System Usability Scale Ergebnis von 83,9. Trotz einzelner Ungenauigkeiten im Generierungsprozess (F_1 -Scores zwischen 0,74 und 0,83) eliminierte die KI die Hürde, komplexere Datenqualitätsabfragen ohne Vorlage zu starten. Die Ergebnisse deuten darauf hin, dass der primäre Mehrwert des KI-Assistenten in der Reduzierung des mentalen Aufwands liegt. Dadurch verlagert sich die Rolle von Fachexpert:innen von der imperativen Spezifikation hin zur deklarativen Verifikation.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

The reliability of data-driven systems depends on fitness for use, defined as a dataset’s ability to support specific downstream tasks or decision contexts. In practice, this property is checked by combining several data quality criteria rather than relying on a single metric. However, current data quality tools force a trade-off between expressiveness and accessibility. Code-based frameworks allow detailed and custom checks, but require coding expertise. Visual solutions reduce assessment to dashboards or aggregate scores, which summarise outcomes but fail to expose the underlying combinations of checks and assumptions. Furthermore, the emerging integration of Large Language Models introduces a black box verification problem, where users are forced to blindly trust opaque, generated rules.

To address these challenges, this thesis proposes a glassbox interaction paradigm. It presents the design, implementation, and evaluation of a web-based prototype that models data quality as a visually interactive Directed Acyclic Graph. The system utilises interactive row highlighting to visually link graph components to specific data entries, thereby making the assessment logic transparent. Additionally, we integrate a multi-agent AI assistant to function as a drafting engine that translates natural language into editable graph structures.

Evaluation through an expert Focus Group and a User Study ($N = 14$) demonstrates that visual aggregation and combination techniques improve the transparency and usability of the data quality assessment process. Technical participants in the User Study solved both manual and AI-assisted tasks with 100% accuracy. Non-technical participants achieved a 100% accuracy on the simple manual task and 71.4% on the more complex AI-assisted task. Overall, participants preferred the AI-assisted workflow and awarded it an “Excellent” System Usability Scale score of 83.9. Furthermore, despite some noise in the generation process (F_1 Scores ranging from 0.74 to 0.83), the AI eliminated the need to start complex data quality assessment from scratch. The findings indicate that the primary value of the AI assistant lies in reducing cognitive friction, allowing domain experts to shift their role from imperative specification to declarative verification.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Kurzfassung	ix
Abstract	xi
Contents	xiii
1 Introduction	1
1.1 Problem & Context	2
1.2 Research Questions	3
1.3 Contribution	4
1.4 Limitations & Scope	5
1.5 Outlook	5
2 Related Work	7
2.1 Landscape of Data Quality Assessment Tools	8
2.2 Metric Aggregation and Constraint-Based Approaches	9
2.3 The Role of AI and Large Language Models	9
2.4 Gap Analysis and Research Positioning	11
3 Interactive Data Quality Assessment	13
3.1 Data Quality Assessment	13
3.2 Visualisation and User Interface	18
3.3 UI/UX Design and Interaction	22
3.4 System Architecture	26
4 Multi-Agent AI Assistant	31
4.1 AI-Assisted DQ Assessment Process	31
4.2 Implementation Details	35
4.3 Agent System Design	37
5 Evaluation & Results	45
5.1 Literature Analysis	46
5.2 Focus Group	49
5.3 User Study	54
	xiii

5.4	User Study Results	57
6	Conclusion	69
6.1	Implications	70
6.2	Future Work	71
	Overview of Generative AI Tools Used	73
	List of Figures	75
	List of Tables	77
	Glossary	79
	Acronyms	81
	Bibliography	83

CHAPTER 1

Introduction

Data serves as the foundational resource for decision-making, automated processes, and Artificial Intelligence (AI) systems [CCS12, VDA12, LBH15]. However, gathering data alone does not guarantee value. The reliability of any data-driven system depends on the quality of the information it consumes [WS96]. In this context, Data Quality (DQ) is rarely a binary attribute of “good” or “bad”. Instead, established literature defines it as *Fitness for Use*, meaning that quality is determined by the specific requirements of the data consumer [PEG25].

Determining this fitness requires more than isolated technical checks. While standard profiling tools identify missing values or type mismatches, real-world quality assessments involve semantic dependencies and hierarchical logic [Sch24]. To illustrate this, consider customer contact data. A table containing customer information might pass simple validation, such as checking that the “City” field is not empty or that the “ZIP Code” contains four digits. However, assessing whether a customer is truly *contactable* requires an evaluation of multiple layers.

First, *syntactic validity* requires that the “ZIP Code” matches the country-specific Regular Expression (RegEx) format. Second, *semantic consistency* requires that “City” and “ZIP Code” correspond to one another. For example, a Graz “ZIP Code” paired with a Viennese “City” is invalid even if both fields are individually correct. Such checks are then *combined* into intermediate metrics, such as *postal-address-valid*. Finally, a top-level rule aggregates these results into a record-level decision. For instance, a customer is contactable if a correct address exists *OR* a phone number is given, which produces a boolean outcome per row (contactable / not contactable). After applying this logic to all rows, the rows are aggregated to a single DQ indicator, indicating that, for example, 40% of customers are contactable.

In this scenario, *Contactability* is not a single column in the database. Instead, it is a higher-level quality concept constructed from the aggregations and combinations of

multiple checks. Errors in low-level checks (e.g., an invalid ZIP format) propagate upward through the dependency structure and reduce the final DQ score. Understanding these dependencies (i.e., how lower-level checks contribute to higher-level indicators) helps stakeholders understand quality issues back to their sources and thereby builds trust in the underlying DQ assessment [HDBL17].

However, current DQ systems fail to represent this structure [Sch24]. Systems either generate or require code [SLS⁺18, PEG25] or flatten them into simple dashboards or visualisations [EW22] that show aggregated results. This thesis addresses this representation gap. It proposes, implements, and evaluates a system that treats DQ not as a list of checks, but as a visually explorable and editable Directed Acyclic Graph (DAG). Furthermore, it is assisted by AI to bridge the gap between technical complexity and domain knowledge.

1.1 Problem & Context

Despite the maturity of the DQ market, there is still a disconnect between the complexity of domain requirements and the capabilities of tools [WS96, KAH23]. As Chapter 2 discusses, current tooling forces a trade-off between expressiveness and accessibility.

On one side, technical frameworks allow engineers to build complex validation logic. While powerful, these solutions rely on programming expertise and are therefore excluding domain experts from the creation process [KAH23, GKHH11, EW22]. On the other side, visual tools or dashboards prioritise monitoring and reporting [RAAM25, EW22]. They present quality as a flat list of independent metrics. These flat representations hide the underlying logic and make it difficult for users to model dependencies or understand *why* a specific record fails in a larger assessment.

Large Language Models (LLMs) offer a solution to the accessibility problem by allowing domain experts to state requirements in natural language and have them translated into executable specifications (e.g., Structured Query Language (SQL)) [NYCP25, TZW⁺24]. However, this introduces a new challenge regarding verifiability and trust. If an AI system generates SQL or Python code, users must assess whether the generated logic matches their domain needs [TKLZ24]. For non-technical users, this results in a *Black Box* that they must trust blindly. For technical users, Human–AI systems are causing a shift from code implementation to validating and debugging generated code [NFLR21, FHJ⁺24].

Consequently, a gap exists for a system that combines three capabilities:

- **Hierarchical Combination & Aggregation:** The ability to compose complex metrics from simpler constraints (e.g., creating a Contactability score).
- **Verifiability by Design:** A visual representation that makes dependencies explicit and inspectable, rather than hiding them in code or opaque models.

- **AI Assistance with Verification:** A workflow where AI lowers the barrier to entry, without removing the user’s ability to validate the result.

1.2 Research Questions

To address the identified gaps, this thesis investigates the design and impact of a novel prototype that utilises a visual, graph-based approach to DQ assessment.

First, a baseline is established by analysing the shortcomings of the state-of-the-art. While functional limitations in DQ tools are well documented [EW22, PEG25], this research aims to understand the barriers that prevent non-technical stakeholders from translating their domain knowledge into executable rules. Consequently, the first research question examines the *why* behind the reliance on engineering support:

Research Question 1 *What are the functional and usability limitations of existing data quality metric tools, and in which aspects do they hinder domain experts from independently assessing data quality?*

This question is answered through a literature analysis that classifies existing solutions and identifies their functional and usability limitations.

Then, building on the problem definition, the thesis evaluates the core contribution of the thesis: the visual combination and aggregation interaction prototype. This tests whether representing DQ logic as an interactive DAG makes the assessment process more transparent and manageable. It aims to evaluate the benefits of this approach by measuring how it affects users with different skill levels. This leads to the second research question:

Research Question 2 *To what extent do visual aggregation and combination techniques improve the efficiency and accuracy of data quality assessments for both technical and non-technical users?*

This question is addressed through a moderated Focus Group where data management experts evaluate the prototype’s interaction concepts and practical utility.

Finally, the thesis investigates the role of AI within this visual framework. Rather than treating AI as a replacement for human judgment, this research explores its potential as a collaborative assistant that helps as a *Drafting Engine* for creating the DAG. This serves to determine if integrating a multi-agent *AI Assistant* changes the user’s interaction with the tool, particularly regarding the ease of constructing complex queries, the verifiability of the output and the cognitive friction involved in the process. Therefore, the third research question addresses this incremental value:

Research Question 3 *Which further improvements does the integration of large language models offer to a data quality tool in terms of enhancing user interaction and enabling natural language query construction?*

This question is investigated through an empirical User Study that compares manual and AI-assisted workflows to measure efficiency, accuracy, and user perception.

1.3 Contribution

This thesis presents the design, implementation, and evaluation of a prototype that treats DQ logic as a visually explorable DAG. By combining visual structures with AI assistance, the work provides contributions to the domains of Human-Data Interaction (HDI), Human-Computer Interaction (HCI) and Human-AI Interaction.

First, a structured review of the state-of-the-art identifies specific limitations in existing tools, particularly regarding visual metric composition and the *Gulf of Evaluation* in AI-assisted workflows. This analysis highlights that current solutions force a trade-off between accessible monitoring and technically complex, unverifiable script-based definitions.

Second, the thesis introduces a novel interaction prototype that allows users to model Fitness for Use through nested metric aggregation. This system provides a *Glass Box* view of the underlying DQ assessment, enabling users to inspect the logic of the resulting graph. This is done through data highlighting that visually links graph nodes to affected data rows.

Third, the system integrates a multi-agent AI Assistant designed to function as a Drafting Engine within a *Human-in-the-Loop* workflow. This approach abstracts technical syntax, such as RegEx, while making sure that the resulting logic remains visible and editable within the graph. Consequently, users can leverage the benefits of using AIs in this context without losing the ability to verify and refine the underlying quality constraints.

Finally, validation through an expert Focus Group and a User Study analyses the impact of these techniques. The results indicate that visual aggregation and combination techniques improve transparency for all user types. Furthermore, the evaluation shows that AI integration primarily reduces the cognitive friction. Concretely, it helps users overcome the *Blank Canvas Problem* when starting a task, while the final assessment logic remains user-defined and verifiable.

To summarise, the core contributions of this thesis are as follows:

1. **A Critical Analysis of DQ Tools:** Identification of the “expressiveness vs. accessibility” trade-off in current tooling and requirements for verifiable Glass Box systems.

2. **A Visual Combination and Aggregation Interaction Prototype:** The design and implementation of a web-based tool that renders DQ assessments as a DAG with data highlighting.
3. **A Multi-Agent AI Assisted Workflow:** The integration of a multi-agent system that lowers the technical barrier for rule creation through natural language.
4. **Empirical Evidence on Verification:** User Study results show that AI reduces the cognitive load of task initiation and shifts the user role from specification to verification.

1.4 Limitations & Scope

The scope of this research focuses specifically on the assessment of DQ, rather than on data profiling or data cleaning. Consequently, the work treats the dataset as a read-only input to evaluate its Fitness for Use. It excludes tasks such as statistical pattern discovery or the modification of records. To investigate this assessment process, the thesis introduces a research prototype that uses hierarchical aggregation and combination of DQ metrics. This implementation supports the targeted selection of quality dimensions *Completeness*, *Correctness*, and *Validity* (compare Section 3.1.3) rather than an exhaustive library of potential checks. A central component of the scope is the visual representation through a DAG of quality checks. The graph links to the table, in which node selection evaluates the underlying quality rule and highlights invalid and valid data. Furthermore, the work explores the integration of a multi-agent AI Assistant designed to translate natural language into graphs. The evaluation of this AI component concentrates on the user interaction (specifically regarding verifiability, trust, and workflow efficiency) rather than benchmarking the code-generation accuracy of the underlying LLM. Finally, the software operates as a proof-of-concept designed to validate these interaction designs. It does not serve as a production-ready engine optimised for processing large-scale data streams.

1.5 Outlook

The remainder of this thesis is structured as follows: Chapter 2 introduces the state-of-the-art research in DQ metrics, interactive DQ assessment, and the role of AI in data management. Chapter 3 details the conceptual design and implementation of the prototype, including its interface, graph-based logic, and the underlying system architecture. Chapter 4 focuses on the integration of LLMs, describing the multi-agent AI system design used within the prototype. Chapter 5 presents the methodology and results of the Literature Analysis, Focus Group and User Study, providing answers to the research questions. Finally, Chapter 6 summarises the findings and discusses its implications for future HDI and Human–AI tools.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Related Work

DQ is a prerequisite for reliable data-driven systems [BS06, WS96]. In practice, it is measured through quality dimensions (e.g., Completeness, Validity, Correctness) and implemented via checks, constraints, and monitoring routines. Surveys of DQ tools show that systems provide support for profiling and rule execution, but expose results as individual metrics, alerts, or flat visualisations rather than as an explorable structure that makes metric dependencies and aggregation logic explicit [EW22, PEG25]. While visual approaches for creating and inspecting individual DQ metrics exist, they do not represent complex, nested compositions as an explorable object [BGK⁺18, EW22]. This becomes limiting once quality needs to be reasoned about across multiple metrics and levels of granularity, especially when stakeholders must understand *why* a score was produced and how it reflects context and intended use (Fitness for Use) [WS96, SPMM24, Sch24].

Recent work expands the capabilities of DQ systems through scalable architectures, improved interfaces, and partial automation with probabilistic inference, machine learning, and LLMs [XZX⁺24, RCIR17, XCSL25, BBB⁺24, LCQ⁺20]. At the same time, studies suggest that aggregation across checks is implemented in simple, tool-specific ways (e.g., binary pass/fail outcomes, thresholds, or summary statistics), with limited support for visually expressing and inspecting nested logic that reflects context and Fitness for Use [EW22]. As a result, users face a trade-off: either they work with expressive but technical frameworks, or they use accessible interfaces that hide logic and make verification difficult, particularly when assistance is mediated through generated code or opaque rule synthesis [XZX⁺24, GSA⁺24].

A recurring theme across tool surveys is that DQ systems optimise for execution and monitoring, but provide less support for communicating how results are produced in a way that users can inspect, question, and adapt to domain intent [EW22, PEG25]. This challenge becomes more noticeable once systems move beyond single checks toward composed assessments, where context and stakeholder priorities determine which violations

matter and how they should be aggregated [WS96, SPMM24]. This chapter positions the thesis within this landscape.

The research focuses on two gaps that appear across prior work:

1. the lack of *hierarchical* metric composition that remains readable for humans, and
2. the lack of *verifiable* interaction paradigms once AI assistance is introduced.

The proposed prototype targets the space in between. It supports nested metric aggregation and combination, while keeping the process visible and inspectable through a visual Glass Box representation that can be validated by both experts and non-experts.

2.1 Landscape of Data Quality Assessment Tools

DQ toolings range from lightweight open-source libraries to large enterprise platforms. Across this spectrum, surveys report limitations that show up regardless of product maturity. Tools target technical users, interfaces emphasise monitoring over explanation, and systems provide only limited support for expressing and inspecting complex logic [EW22, PEG25].

2.1.1 Target Audience and Accessibility

A substantial share of DQ tools is designed for technical stakeholders such as data engineers and data scientists, while domain experts are treated as consumers of results rather than authors of quality logic. For instance, open-source solutions assume programming or configuration expertise to define checks, integrate them into pipelines, and interpret the resulting outputs [EW22, PEG25]. Similarly, systems that allow automated data repairs can be effective at finding tuples, but the reasoning behind the outcome is difficult to inspect for non-technical users [RCIR17]. This creates a practical barrier. If users cannot easily reconstruct *why* a rule fired or *how* a result was produced, it becomes harder to verify, adapt, and trust automated suggestions, particularly in AI-assisted settings [XZX⁺24, GSA⁺24].

2.1.2 Functionality and Visualisation

Existing tools focus on profiling and validation in operational workflows: computing descriptive statistics, applying predefined constraints, and reporting violations or trends over time [EW22, PEG25]. These capabilities are valuable for monitoring and governance, but they treat checks as a flat set of independent assertions. As a consequence, they offer limited support for expressing nested combinations of metrics that capture semantic dependencies, even though such a hierarchical composition is central to Fitness for Use assessments [Sch24, PEG25].

In terms of interaction, DQ information is presented through high-level visualisation that summarises violations or trends. Such visualisations work well for continuous monitoring, but they make the logical structure of an overall score explicit. Non-technical users cannot see how individual rules contribute to an aggregated or combined result [PEG25, Sch24, EW22]. Even when visual tools support creating and inspecting individual metrics, they do not usually expose complex compositions as an explorable object that lets users trace and validate how higher-level quality judgments are formed [BGK⁺18]. As a result, users need to fall back to manual querying or technical inspection to understand why a dataset received a certain assessment and which specific records result in that outcome.

2.2 Metric Aggregation and Constraint-Based Approaches

A central challenge in DQ assessment is moving from isolated checks toward a cohesive notion of Fitness for Use [WS96, SPMM24]. In practice, quality is a collection of independent assertions: constraints are executed per column or per row, and results are reported as pass/fail outcomes, violation counts, or summary statistics [EW22, PEG25]. This execution model is effective for enforcing local rules, but it provides limited support for representing higher-level quality concepts that depend on multiple checks and their semantic relationships. Schrott [Sch24] directly addresses this limitation by treating metric definition and aggregation as a compositional problem. Instead of keeping constraints as a flat checklist, his work formulates DQ validation hierarchically and shows how complex requirements can be expressed by combining constraints into nested structures. This thesis builds on that foundation. In particular, it builds on the idea that composite metrics can capture domain semantics more naturally than independent rule sets, and that aggregation should reflect stakeholder intent [Sch24, WS96].

At the same time, existing implementations of such expressive constraint compositions still remain engineering-oriented. Schrott’s CobADQ [Sch24] is provided as a Python framework and Command-Line Interface (CLI), which offers computational flexibility but assumes that users can author and inspect the underlying definitions programmatically. In contrast, the broader tool ecosystem also shows that widely adopted frameworks for quality verification and testing (e.g., pipeline-integrated constraint and test suites) are commonly operated through code or configuration, reinforcing the separation between expressive logic and accessible interaction [SLS⁺18, PEG25].

To the best of our knowledge, there is still a lack of systems that combine deep, hierarchical aggregation logic with a Graphical User Interfaces (GUIs) that allows users to inspect dependencies, trace how outcomes are formed, and validate assessments without requiring direct code-level reviewing.

2.3 The Role of AI and Large Language Models

AI has been part of DQ research for years, for example, through probabilistic repair and learning-based approaches that help detect and fix inconsistencies in tabular

data [RCIR17]. More recently, LLMs have been explored as an interface layer for DQ tasks, with the goal of translating informal domain requirements into executable checks and rules [XCSL25, Sch25, AIMS25]. This line of work emphasises accessibility: non-expert users can express intent in natural language and receive structured checks or recommendations.

2.3.1 Enterprise AI Integration

Enterprise platforms start to integrate AI functionality and agent-like features into data management and DQ products. Informatica positions its CLAIRE AI engine (CLAIRE Agents, CLAIRE GPT and CLAIRE Copilot¹) as components that support discovery, pipeline tasks, or the identification of DQ issues [Inf25]. Similarly, Ataccama ONE² provides guided rule creation workflows and with Ataccama ONE AI³ explicitly allows generating rules with AI. While these integrations can be powerful in practice, they are embedded in ecosystems, and their AI-supported logic is not transparent in a way that enables lightweight verification.

2.3.2 Generative AI and No-Code Solutions

Beyond enterprise suites, LLM-based tooling support “no-code” workflows where users upload a dataset and specify checks in natural language. This improves accessibility, but it also makes verification a central issue. Systems execute user intent by generating code or query fragments that remain hard to review for users without programming expertise. Empirical studies in Human–AI analysis show that users adopt workarounds to verify AI-assisted analyses, and that verification can become difficult when the underlying logic is produced by an AI rather than authored directly by the user [GSA⁺24]. Systems such as WaitGPT explicitly target this gap by making generated data analysis code and intermediate steps visible and steerable, highlighting that transparency mechanisms are necessary once LLMs become part of analytical workflows [XZX⁺24]. However, WaitGPT still requires an abstract understanding of how code-based data analysis works.

Furthermore, open-source tools have begun adopting this paradigm. OpenRefine, a widely used interactive data wrangling environment [Ham13], supports AI⁴ through an LLM extension⁵ that transform natural language into data cleaning, semantic enrichment, anomaly detection and content generation workflows. In the DQ domain, SodaGPT positions itself as “generative AI for data quality” and allows users to describe checks in natural language that are then translated into executable validations [Mas23]. These systems represent an important step toward accessibility, but they also intensify the Gulf of Evaluation: if the tool generates complex logic in the background, users still need ways

¹<https://www.informatica.com/platform/claire-ai.html>

²<https://www.ataccama.com/platform>

³<https://www.ataccama.com/ai>

⁴<https://openrefine.org/blog/2025/02/21/AIExtension>

⁵<https://github.com/sunilnatraj/llm-extension>

to verify logic without becoming dependent on using CLIs, or reading code [GSA⁺24, XZX⁺24].

2.4 Gap Analysis and Research Positioning

Despite the discussed advancements, an important gap remains in the design space of DQ systems. Across both open-source and enterprise ecosystems, tools tend to fall into one of three recurring patterns.

First, expressive approaches for defining and aggregating complex quality logic are engineering-centric. They require programming expertise to model constraints and compose metrics, as illustrated by Schrott’s constraint-based aggregation framework [Sch24]. Second, newer AI-supported workflows improve accessibility but reduce transparency. If checks or transformations are generated through LLMs as code, SQL, or hidden logic, non-experts face difficulty verifying whether the generated solution matches domain intent [GSA⁺24, XZX⁺24]. Third, user interfaces remain structurally flat: they present metrics, alerts, and summary indicators, but rarely visualise how metrics depend on each other or how multiple checks combine into a final assessment [EW22, PEG25].

This thesis aims to bridge these gaps by introducing a prototype that supports nested metric aggregation while remaining accessible through a transparent visual interface. Rather than competing with “all-in-one” commercial suites, this work focuses on a specific workflow: defining, combining, and verifying DQ constraints in a way that makes dependencies explicit and outcomes traceable.

2.4.1 A Visual and Hybrid Paradigm

To the best of our knowledge, prior work has not systematically compared fully manual and AI-assisted workflows specifically for hierarchical DQ assessment in a way that keeps the aggregation structure inspectable by users. This work differentiates itself through three design strategies aimed at increasing interpretability and supporting Verifiability by Design.

First, DQ logic is represented as a DAG, making dependencies and aggregation steps readable rather than hiding them behind code or simple visual indicators.

Second, the system provides interactive feedback on the impact of metrics. Users can select nodes in the graph and immediately see which records are affected through visual highlighting in the data table. This enables rapid validation of how rules and compositions behave on data.

Third, the approach adopts a hybrid human-in-the-loop workflow where the AI acts as a Drafting Engine. The LLM proposes metric configurations as understandable natural language statements that users can inspect and adjust, while the system keeps the underlying aggregation structure visible. By avoiding workflows that require non-

experts to audit generated code, the design keeps the user in control of the final quality logic [GSA⁺24, XZX⁺24].

2.4.2 Bridging the Expertise Gap in Data Quality Assessment

A central objective of this research is to broaden access to complex DQ tasks. This thesis investigates whether metric aggregation can be performed effectively by both experts and non-experts, and whether AI assistance changes performance, usability and perceived workload across these groups. To answer this, our study design compares two user groups as they complete representative tasks in a fully manual mode and in a hybrid, AI-assisted mode.

In addition, the work is positioned against the state-of-the-art by incorporating qualitative expert feedback through a Focus Group. Together with the quantitative results from the User Study, this allows for an evaluation of whether the prototype supports the interpretability of quality logic. Specifically, the assessment focuses not only on whether the tool detects issues, but also on whether it enables users to understand, justify, and refine complex hierarchical assessments.

We argue that before data can be monitored or cleaned at scale, it must first be understood. The contribution of this thesis is therefore a bridge between raw data and complex quality assessment through a verifiable, visual, and AI-supported Glass Box system.

Interactive Data Quality Assessment

This chapter details the design and implementation of the built prototype. It outlines the functional capabilities of the tool, defining the hierarchical structure of DQ tasks and the specific mechanisms used to assess DQ tasks. Furthermore, it describes the GUI paradigms designed to visualise these dependencies, followed by a technical overview of the system architecture.

3.1 Data Quality Assessment

The core concept of the prototype is a framework that allows users to construct data quality rules through a hierarchical approach. Rather than using isolated checks, the system models DQ as a DAG, where individual tasks are represented as connected nodes to form a comprehensive assessment logic.

Throughout this chapter, we use an address-quality scenario as a running example. A Comma-Separated Values (CSV) file contains address columns such as “Street”, “ZIP”, and “City”. The assessment accepts addresses from either Vienna or Graz and treats an address as valid only if (1) the ZIP and city checks match the same location and (2) the street entry follows a required format.

3.1.1 File Management and Task Initialisation

The workflow begins with the upload of tabular data. The system supports the management of CSV files, allowing users to upload datasets for analysis or remove obsolete files. Once a dataset is selected, a Detail View serves as the workspace for defining the quality assessment task.

In the running example, the workflow starts by uploading the addresses CSV. Figure 3.1 shows the *Dataset Inspection Table* after the upload, which exposes the respective columns and serves as an initial overview before any node is created.

Index	LastName	Street	City	ZIP
1	Stekl	Rotenturmstr. 45	Wien	1010
2	Puszter	Piaristeng. 20	Graz	8010
3	Pichler	Mautner-Markhof-Gasse 15	Graz	8010
4	Winkelmüller	Favoritenstr	Wien	1100
5	Moser	Währinger Str. 76	Wien	4180
6	Karisch	Vienna 57	Win	1210
7	Quistorp	Leopoldauerstr 6	Graz	1210

Page 1 of 12

« < > »

Figure 3.1: Dataset Inspection Table: Default view (no node selected).

3.1.2 Data Quality Tasks

In this workspace, quality tasks are represented as nodes within a DAG. The structure is hierarchical: leaf nodes represent atomic checks on data values, while parent nodes combine or aggregate these results to produce higher-level insights. Edges within the graph represent the flow of logic, pointing from parent nodes to their children, thereby establishing dependencies. The workspace allows for multiple independent trees to coexist, and the final root(s) of the tree represent the overall result(s) of the quality assessment.

The system distinguishes between three primary node types: *Metric Nodes*, *Combination Nodes*, and *Aggregation Nodes*. Based on the granularity taxonomy derived from Schrott, each type operates at a specific level of granularity, ranging from individual values to entire rows/columns, and finally to the dataset level [Sch24].

Figure 3.2 instantiates this hierarchy for the running example. Leaf nodes check “ZIP”, “City”, and “Street” columns. Two city-specific Combination Nodes (*Valid Viennese Address* and *Valid Graz Address*) enforce that “ZIP” and “City” values both pass for the same location. An *Austrian City* node accepts either city predicate, while *Valid Address* additionally requires a valid “Street” entry. The Aggregation Node compresses the final vector into a single dataset-level score.

To instantiate any node in the system, regardless of its type, the user provides the following attributes to ensure the graph remains human-readable and self-documenting:

- **Name:** A unique identifier used for visualisation in the graph (e.g., “Check Email Format”).

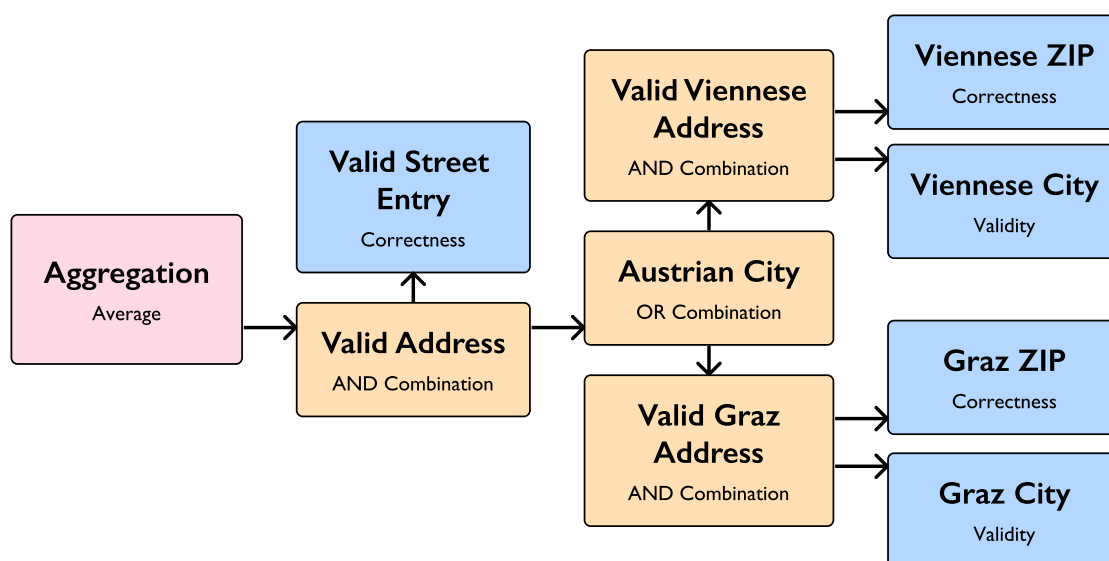


Figure 3.2: Conceptual DAG for the address-quality running example, showing Metric (blue), Combination (orange), and Aggregation (pink) Nodes.

- **Description (Optional):** A human-understandable explanation of the node’s purpose, helping in interpreting the assessment logic.

3.1.3 Metric Nodes (Value-Level)

Metric Nodes operate at the value-granularity, i.e., the finest level of granularity. A Metric Node is applied to a specific column within the dataset and evaluates every entry against a defined condition. Unlike Combination Nodes or Aggregation Nodes, which operate on the results of other nodes, Metric Nodes operate directly on the raw dataset. Therefore, they require an additional parameter:

- **Column:** A specific column of the dataset (e.g., “ZIP”) to be evaluated.

While the underlying architecture is designed to be extensible, allowing for the future addition of complex or domain-specific checks, the current prototype focuses on three foundational metric types that cover basic DQ requirements. The output of this system’s Metric Nodes are binary vectors of length n (where n is the number of rows in the dataset), where 1 indicates a pass and 0 indicates a failure.

Completeness The Completeness metric evaluates the presence of data. It addresses the fundamental quality dimension of whether the required information is missing [WS96]. Conceptually, it marks any missing or null values as errors. Formally, let $v_{i,c}$ be the value at row i in column c . The Completeness function $M_{completeness}$ is defined as:

$$M_{completeness}(v_{i,c}) = \begin{cases} 1 & \text{if } v_{i,c} \neq \emptyset \\ 0 & \text{if } v_{i,c} = \emptyset \end{cases} \quad (3.1)$$

This metric requires no additional user parameters beyond selecting the target column.

Validity The Validity metric verifies domain conformance, i.e., whether a data value conforms to a defined domain of allowed entries. This is particularly useful for categorical data (e.g., “Country” or “Status”) where only a specific set of values is permissible. The user provides a set of valid strings, S , as an input parameter. Formally, for a given set of allowed values $S = \{s_1, s_2, \dots, s_k\}$, the Validity function $M_{validity}$ is defined as:

$$M_{validity}(v_{i,c}, S) = \begin{cases} 1 & \text{if } v_{i,c} \in S \\ 0 & \text{if } v_{i,c} \notin S \end{cases} \quad (3.2)$$

In the running example, Validity nodes evaluate the column “City” through Validity to restrict values to allowed strings for Vienna or Graz.

Correctness The Correctness metric assesses format conformance, i.e., whether a value adheres to a specific structural format, such as an email address, phone number, or date pattern. It leverages RegEx to provide a flexible mechanism for pattern matching. Formally, given a RegEx pattern P , the Correctness function $M_{correctness}$ is defined as:

$$M_{correctness}(v_{i,c}, P) = \begin{cases} 1 & \text{if } match(v_{i,c}, P) \text{ is true} \\ 0 & \text{otherwise} \end{cases} \quad (3.3)$$

This node type requires the user to provide the RegEx pattern P as a configuration parameter.

In the running example, Metric Nodes evaluate the columns “ZIP” and “Street”. The ZIP checks use Correctness to validate a location-specific ZIP pattern. Furthermore, the street check uses Correctness to validate the structural street format. Naively, it checks if strings end with at least one digit (house number).

3.1.4 Combination Nodes (Row-Level / Column-Level)

Combination Nodes transform the assessment granularity from the value level to the row/column level. Unlike Metric Nodes, which use raw data columns, combination nodes accept a set of other nodes as input. Depending on the chosen operator, they require either a single child node (unary operation) or multiple child nodes (n-ary operation). The output of a combination node remains a vector of length n (where n is the number of rows), which effectively creates a new synthetic feature column that can be further combined or aggregated.

Logical Operators The logical operators AND, OR, and NOT perform standard Boolean algebra row-by-row. Formally, let $C = \{c_1, c_2, \dots, c_k\}$ be the set of input vectors from k child nodes, where $c_{j,i}$ represents the value of the j -th child at row i . For the AND operator, the result vector R at row i is defined as $R_i = \prod_{j=1}^k c_{j,i}$ (assuming binary inputs), returning 1 only if all children are 1. The OR operator is defined as $R_i = \max(c_{1,i}, \dots, c_{k,i})$, returning 1 if at least one child is 1. The NOT operator is unary (taking only one child c_1) and flips the bit value: $R_i = 1 - c_{1,i}$.

The running example uses AND to define city-specific predicates: *Valid Viennese Address* combines *Viennese ZIP* and *Viennese City* nodes, and *Valid Graz Address* combines *Graz ZIP* and *Graz City* nodes. The example then uses OR to form a new *Austrian City* node, which returns 1 if either city predicate is satisfied. Finally, *Valid Address* node uses AND to require both *Austrian City* and *Valid Street Entry* (compare Figure 3.8).

Statistical Operators The statistical operators MAX and MIN evaluate the extremes among the child nodes for each specific row. While logically similar to OR and AND for binary data, these operators preserve magnitude when applied to non-binary inputs (e.g., if potentially custom metric nodes return probability scores). The MAX operator outputs the highest value found among the children for that row: $R_i = \max_{j=1}^k (c_{j,i})$. Conversely, the MIN operator outputs the lowest value: $R_i = \min_{j=1}^k (c_{j,i})$.

Arithmetic Operators Arithmetic operators allow for quantitative assessments of a row's quality. The SUM operator calculates the total score of a row by adding the values of all child nodes: $R_i = \sum_{j=1}^k c_{j,i}$. The AVG operator computes the mean score, representing the percentage of passed checks for that specific row: $R_i = \frac{1}{k} \sum_{j=1}^k c_{j,i}$.

3.1.5 Aggregation Nodes (Table-Level)

Aggregation Node transform the granularity to the table level. Unlike Combination Nodes, which output a vector, an aggregation node accepts a single child node (either a Metric Node or Combination Node) and compresses its entire output vector into a single scalar value. This scalar typically represents the final answer to the underlying DQ question.

Arithmetic Aggregation SUM and AVG provide the total count or the ratio of valid data points across the entire dataset. Formally, let v be the input vector of size n from the child node. The SUM operator calculates the total accumulated value: $S = \sum_{i=1}^n v_i$. The AVG operator provides the final quality score (e.g., the percentage of valid rows in the file): $S = \frac{1}{n} \sum_{i=1}^n v_i$. The LEN operator returns the total number of rows in the dataset, $S = n$.

The running example aggregates the output of the *Valid Address* node using AVG. This aggregation returns the share of valid addresses across the dataset and displays the result directly on the aggregation node inside the DAG.

Statistical Aggregation The statistical operators MAX and MIN are used to identify the best or worst performing row in the entire dataset. The MAX operator returns the highest single value present in the input vector: $S = \max(v)$. The MIN operator returns the lowest single value: $S = \min(v)$. In a binary context (0/1), MIN effectively acts as a global “All Valid” check (returning 0 if even a single row fails), while MAX acts as an “Any Valid” check.

3.1.6 Advanced Logic with Lark Grammar

To support complex quality assessment scenarios that exceed the capabilities of standard logical or arithmetic operators, the tool allows for context-free languages parsing using Lark grammar¹ through Schrott’s CobADQ library [Sch24]. This feature provides a programmable layer on top of the visual node structure, allowing users to define specific constraints for Combination Nodes and Aggregation Nodes. The grammar enables the encoding of Effectively Propositional Logic formulas. Structurally, the language consists of pairs of constraints and aggregation functions. The constraints are composed of predicates connected by standard logical operators (such as AND, OR, IMPLIES) and may include quantifiers like FORALL or EXISTS. These predicates allow for the comparison of values using standard operators (e.g., $<$, $>$, $=$, \neq). Furthermore, the grammar supports arithmetic expressions for aggregation. These expressions can combine numeric literals, variables, and a set of special functions to calculate results based on the input data. The supported functions mirror the standard operators ($\text{avg}()$, $\text{len}()$, $\text{min}()$, $\text{max}()$, and $\text{sum}()$), but can be combined into complex formulas (e.g., weighting specific inputs differently before averaging).

Integration and Limitations In the prototype, this functionality is accessible via a CUSTOM operator type. To utilise this feature, a user must first connect the formula-dependent child nodes to the Combination Nodes and Aggregation Nodes. Subsequently, the user can manually input the grammar directly into a dedicated “Constraints” input field, referencing the connected children as variables within the formula. However, it is important to note that this text-based approach is a deviation from the low-code, visual paradigm of the prototype. Consequently, this advanced feature is not integrated into the AI Assistant extension (see Chapter 4), nor is it part of the user evaluation described in Chapter 5. While the backend architecture supports these custom parsers, the frontend integration remains a proof-of-concept. Therefore, detailed user support for writing complex Lark grammar is considered out of scope for this prototype and remains a subject for future work.

3.2 Visualisation and User Interface

To make the abstract logic of the DAG accessible, the GUI focuses on two primary visualisation strategies: the interactive graph canvas and the synchronised data tables.

¹<https://github.com/lark-parser/lark>

3.2.1 Interactive DAG Visualization

The DQ assessment task is visualised as a DAG rendered on an interactive canvas that supports panning and zooming. The graph is automatically laid out from left to right, visually guiding the user from general aggregation tasks down to specific metric checks. A “Recentre” control adjusts the viewport to ensure that the entire graph is visible with minimal padding.

Figure 3.3 shows the GUI of the final DAG in respect to the running example. The pink Aggregation Node on the left reveals the final result of the address-quality assessment task, showing that more than 42% of the data are faulty, i.e., do not hold correct addresses for Vienna or Graz regarding the defined DQ rules.

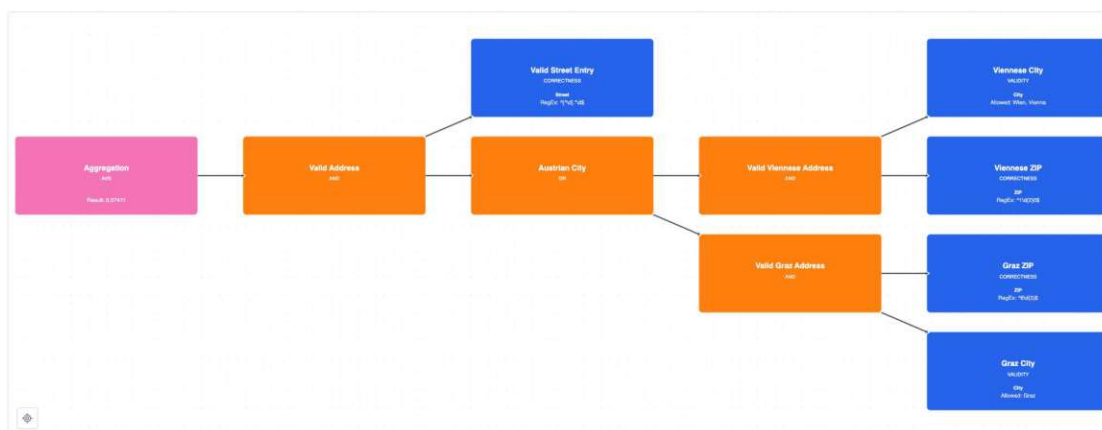


Figure 3.3: GUI representation of the address-quality assessment DAG.

Visual Encoding and Colour Strategy To enhance scannability and reduce cognitive load, nodes are categorised using a semantic colour coding scheme designed to establish a visual hierarchy based on colour temperature and contrast. Metric Nodes are rendered in blue, Combination Nodes are displayed in orange, and Aggregation Nodes are coloured pink to highlight the final result and visually mark the root of the tree. This encoding uses principles in visualisation colour design: cool hues (e.g., blue) tend to be perceived as more distant and less salient than warm hues, and are therefore well-suited for less prominent elements, whereas warmer hues are more effective for attention-demanding foreground elements [WGM⁺08].

Node Anatomy and Content Geometrically, each node is rendered as a rounded rectangle designed to serve as a summary card for its details. The layout follows a simple hierarchy where the user-defined name of the node is displayed at the top of the header. Beneath the name, the node body lists the identifying operator (for Combination Nodes and Aggregation Nodes) or the metric type (for Metric Nodes). To minimise repeated navigation into a node, the most important configuration details are shown directly on the node. Metric Nodes display the target column, while Correctness and Validity nodes

include their configuration, such as the specific RegEx pattern or value list. Unique to Aggregation Nodes is the final calculated score displayed directly within the node body, allowing for an immediate assessment of the overall quality. To preserve layout integrity, all strings exceeding the available width are truncated with an ellipsis (“...”).

The interface uses visual highlighting to communicate interaction states. A currently selected node is rendered with a purple border and a drop-shadow effect. Furthermore, the system provides feedback during asynchronous operations. Nodes currently being processed are pulsed in grey. Nodes that encounter calculation errors are highlighted in red.

DAG Layout The spatial arrangement of the nodes is determined by the *Sugiyama method* for layered graph drawing [STT07]. It organises vertices into layers (or ranks) such that edges predominantly point in a single direction. The benefit of this layout strategy is to minimise edge crossings and bends, to reduce visual clutter. Figure 3.4 visualises the incremental construction of the graph of the running example. The figure displays the sequence of layout as nodes are added, showing how the algorithm integrates new elements while maintaining structure.

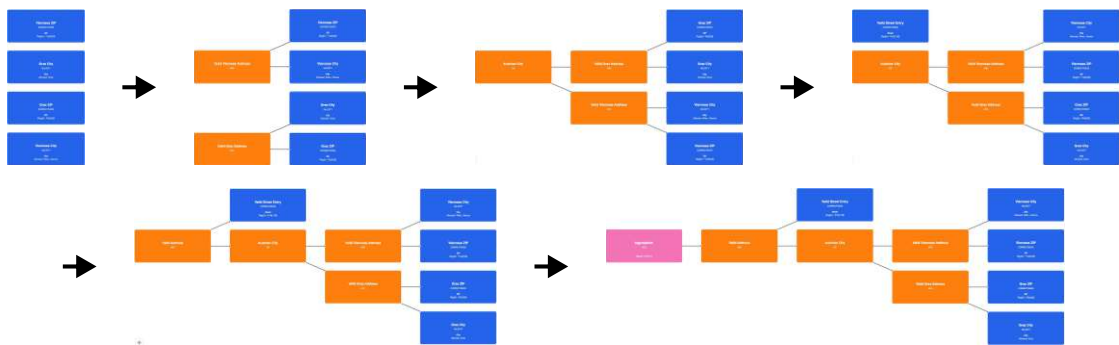


Figure 3.4: Visualisation of incremental graph construction and layout adaptation.

The standard Sugiyama method implementation aligns layers vertically from top to bottom. Our application rotates the graph orientation by 90 degrees, resulting in a left-to-right layout, where the final root nodes (typically Aggregation Nodes) appear on the left and the flow of logic progresses towards the leaf nodes (Metric Nodes) on the right. We choose this orientation to align with the left-to-right reading direction of Western scripts to reflect how processing moves from a final quality score to finer-grained values.

3.2.2 Tables

Supporting the abstract logic of the DAG, the GUI provides two tabular views: the *Dataset Inspection Table* and the *Node Result Table*. These tables are used to validate the computed results by showing the underlying raw data and guide for defining metric nodes (e.g., identifying column values).

Both tables are designed for handling large datasets. They feature fixed headers and indices to maintain context during vertical scrolling and support horizontal scrolling for high-dimensional datasets. To ensure performance and usability, the data is paginated. Crucially, the pagination logic of the *Node Result Table* is synchronised with the *Dataset Inspection Table*. Navigating to a specific page in one view updates the other to ensure the user always compares corresponding rows across both views.

Dataset Inspection Table

The *Dataset Inspection Table* provides a full, read-only view of the uploaded tabular data. While it displays the raw dataset by default, it acts as an interactive visualisation layer when coupled with the DAG. Clicking on a node within the graph triggers distinct visual highlighting patterns in the table, effectively projecting the node’s logic onto the raw data. The highlighting behaviour adapts based on the granularity of the selected node:

Value-Level (Metric Nodes) When a Metric Node is selected, the specific column it evaluates is highlighted with a bold font weight. Values that fail the metric check (i.e., return 0) are rendered with a red background, allowing users to spot data errors. Figure 3.5 showcases this behaviour of the running example by highlighting malformed “ZIP” entries for Vienna after selecting the corresponding Metric Node.



Index	LastName	Street	City	ZIP
1	Steki	Rotenturmstr. 45	Wien	1010
2	Puzster	Piaristeng. 20	Graz	8010
3	Pichler	Mautner-Markhof-Gasse 15	Graz	8010
4	Winkelmüller	Favoritenstr.	Wien	1100
5	Moser	Währinger Str. 76	Wien	4180
6	Karisch	Vienna 57	Win	1210
7	Quistorp	Leopoldauerstr 6	Graz	1210

Figure 3.5: Dataset Inspection Table: Viennese ZIP Metric Node anomalies.

Row-/Column-Level (Combination Nodes) Selection of a Combination Node highlights all child columns (the columns used by the Metric Nodes feeding into this combination). Additionally, the table dynamically appends a temporary fixed column at the far right, labelled with the node’s operator name. This column displays the calculated result for that row. If the combined result for a row is 0 (failure), the entire row is highlighted in red to signify a record-level quality issue. Continuing the running example, Figure 3.6 showcases this behaviour for an AND combination of the *Viennese ZIP* and *Viennese City* Metric Nodes for Vienna.

Table-Level (Aggregation Nodes) For Aggregation Nodes, the table highlights all columns involved in the calculation tree to indicate the scope of the aggregation. However, since the result is a scalar value (valid for the entire dataset), no specific row or cell

3. INTERACTIVE DATA QUALITY ASSESSMENT



Figure 3.6: Dataset Inspection Table: AND Combination Node (Valid Viennese Address)

highlighting is applied, and the result is instead displayed on the node itself. Figure 3.7 illustrates this for the final Aggregation node of the running example.



Figure 3.7: Dataset Inspection Table: Columns affected by AVG Aggregation.

Node Result Table

The *Node Result Table* appears beneath the Node Detail Panel (see Figure 3.8) only when a node is selected. Unlike the Dataset Inspection Table, which shows raw input, this table represents the processed output of the specific node. For example, for the DQ metrics Validity, Correctness, and Completeness, it displays the vector of calculated values for the selected node, indexed by row. For Combination Nodes, this table is particularly useful for debugging, as it shows not only the result vector but also the intermediate values from its child nodes, allowing the user to trace how dependencies influenced the final row outcome. For Aggregation Nodes, which yield a single scalar result, the table displays one row with the computed value, where the column header corresponds to the selected operator. Figure 3.9 shows the *Node Result Table* of the *Valid Address* Combination Nodes of the running example.

3.3 UI/UX Design and Interaction

The GUI is designed with a focus on immediate visual feedback to support the task of DAG construction. To assist with spatial orientation on the canvas, the graph background features a subtle dot grid pattern. As described in the previous section, the system employs different state visualisations: currently selected nodes are visually distinct to

Index	Valid Street Entry	Austrian City	AND
1	1	1	1
2	1	1	1
3	1	1	1
4	0	1	0
5	1	0	0
6	1	0	0
7	1	0	0
8	1	1	1
9	1	1	1

Figure 3.8: Edit Interface of the Node Detail Panel Figure 3.9: Node Result Table for the Valid Address Combination Node

guide the user’s focus, while asynchronous states are communicated through pulsing grey animations (loading) or red backgrounds (error).

The GUI follows the concept of perceived affordances, i.e., clear cues, constraints, and feedback, so that users can immediately identify and perform the intended actions [Nor99]. Every interactive button is paired with a semantic icon to reinforce its function. To maintain consistency, dropdowns and select elements for child nodes are coloured to match their respective node types (blue, orange, pink). The tabs for switching between Metric, Combination and Aggregation node creation follow the same semantic colour scheme, as do their corresponding “Create” buttons. This colour coding is also reflected in the detail view, where the list of child nodes retains the same type-specific colours (compare the Edit GUI in Figure 3.8). When the workspace is idle, the system provides passive textual cues (e.g., “Click on any node in the graph for details”) to guide exploration. Furthermore, the application supports a read-only mode. This mode disables modification controls while allowing inspection, making it suitable for sharing quality reports with external stakeholders or management. Navigation between the detailed workspace and the project overview is handled via a “Files” link in the header, though standard browser navigation is also supported.

3.3.1 Detail View Page layout

The layout strategy of the detail view page evolved through several iterations to prioritise data visibility. Figure 3.10 shows the page layout of the GUI. The final design follows a vertical flow intended to support users’ mental-model construction and sensemaking [SN93].

1. **Data Layer (Top):** The view begins with the *Dataset Inspection Table* (1). Placing the raw data at the top ensures that users can analyse the dataset’s structure and

content before engaging with the graph.

2. **Logic Layer (Middle):** Directly below the data is the primary workspace, split between the interactive DAG (2), the *Node Detail Panel* (3), as well as the *Node Result Table* (4). This side-by-side arrangement is intended to keep the overall structure and the currently selected node's details visible at the same time. The goal is to allow users to modify logic and see results without switching tabs.
3. **Control Layer (Bottom):** The *Node Creation Interface* (5) is located at the bottom of the page. By default, the GUI chat for the AI Assistant (see Chapter 4) is disabled and therefore not shown here; if enabled, it appears directly above the Node Creation Interface.

While this layout organises the workflow logically, it requires vertical scrolling to access the creation controls. Consequently, the application is optimised for wide-screen monitors, where the horizontal space allows the graph and detail panels to be rendered side by side without compromising the visualisation.

3.3.2 Adjusting the Tree

Modifying the DAG is the core interaction of the workspace. Users can adjust the tree structure through two primary mechanisms: manual node creation via the GUI or direct modification of existing nodes within the graph. Furthermore, the following Chapter 4 discusses alternative node creation via the AI Assistant.

Adding New Nodes

The node creation component is located at the footer of the workspace. It features a tabbed interface corresponding to the three node types. Upon submission, the system calculates the new node's state and automatically inserts it into the graph with the correct edge connections. While the backend of the application performs the calculation, the node is in a loading state. The new node is set as the "Selected Node" to reveal the detail panel, allowing the user to verify the result without an additional click.

Modifying Existing Nodes

Figure 3.8 shows how existing nodes can be managed via the *Node Detail Panel*. To ensure data integrity and graph stability, the system enforces specific constraints on modification and deletion. *Concurrency Locking:* Modification actions are disabled whenever the graph is in a processing state. This prevents race conditions where a user might attempt to edit a dependency chain that is currently being processed. *Deletion Integrity:* To preserve the structural validity of the DAG, a node cannot be deleted if it is currently assigned as a child to another node (i.e., if it has parents). This restriction prevents the creation of broken links. Users must first update the parent node to remove the dependency before the child node can be safely deleted.

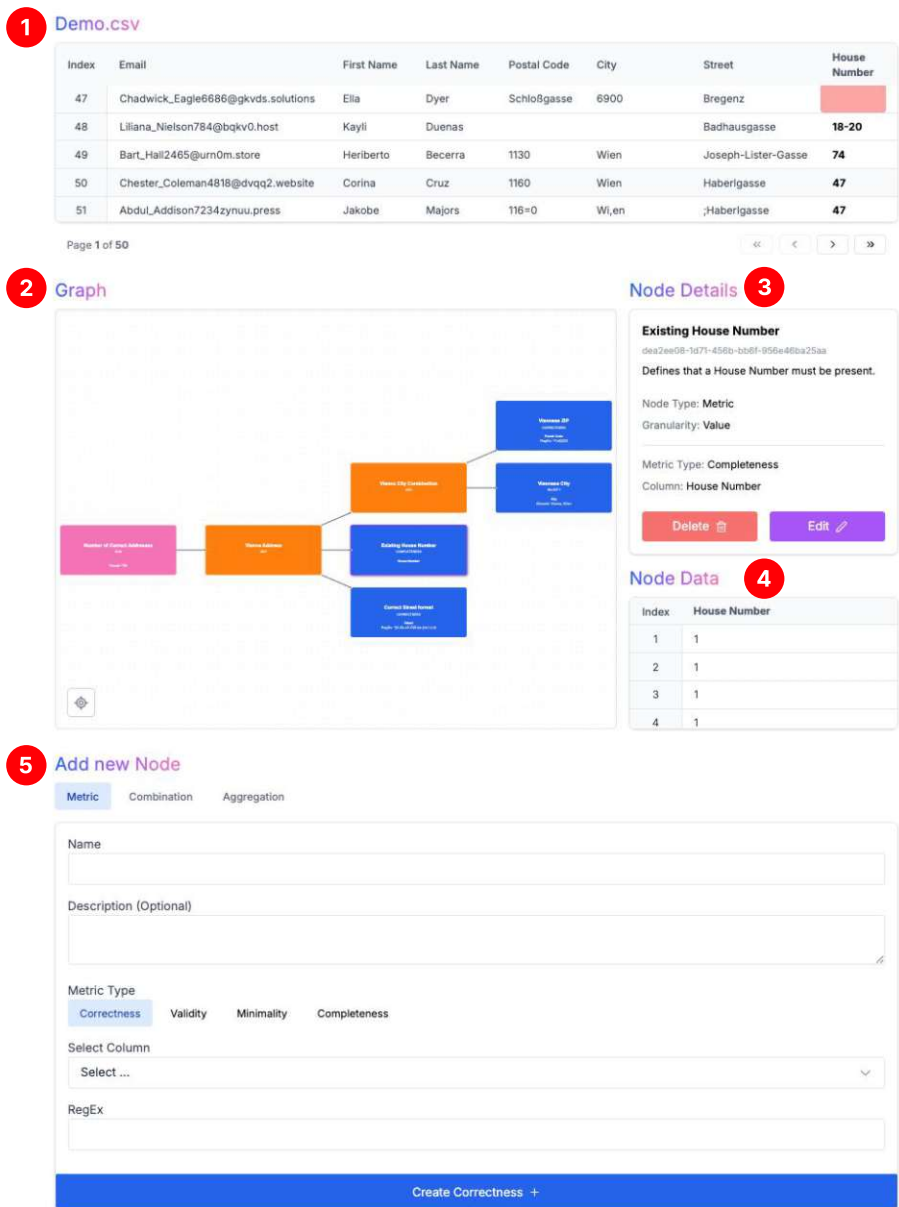


Figure 3.10: Graphical User Interface of the Detail View page (Main component of the prototype).

The editing capabilities are also scoped based on the node’s definition:

- **Editable Attributes:** Users can freely modify the name and description for documentation purposes. For logic adjustments, users can update the children list (for Combination Nodes and Aggregation Nodes) or specific parameters like RegEx

patterns or allowed value lists (for Metric Nodes).

- Immutable Attributes:** Fundamental structural properties, specifically the node type, the assigned operator, the target column, and the internal UUID, cannot be changed after creation. If these attributes require adjustment, the user must delete the node and recreate it.

3.4 System Architecture

The architectural design of the application is a containerised client-server system. Technically, this work builds upon the foundational infrastructure developed by Hoschek [Hos24]. Hoschek’s work established the frontend setup, as well as a pipeline for file ingestion, the object storage configuration, and the containerised backend environment for asynchronous data manipulation. However, the original system was not designed for hierarchical DQ assessment. Rather, it focused on interactive assessment that enables sampling-based inspection and aggregation strategies to measure the DQ metrics minimality and completeness through a GUI.

This system follows a standard client-server model, separated into frontend and backend, communicating via a RESTful Application Programming Interfaces (APIs). Figure 3.11 illustrates the architecture through the request path of the system.

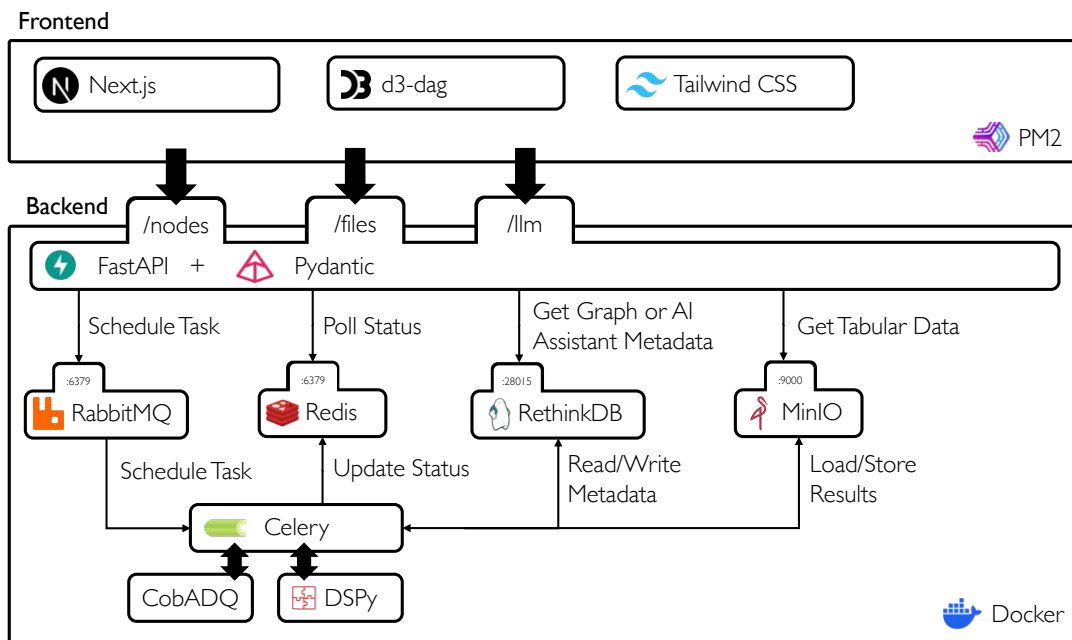


Figure 3.11: High-level architecture overview of the prototype (request path).

3.4.1 Frontend Implementation

The GUI is built using Next.js², a React³ framework. Styling is managed via Tailwind CSS⁴, a CSS framework.

A challenge in the frontend development was selecting a library capable of rendering the DAG. Inspected graph libraries default to force-directed layouts, which are unsuitable for showing hierarchical dependencies as they lack a clear directionality [Kob12]. After evaluating several candidates, we select the library d3-dag⁵. Built on top of D3.js⁶, d3-dag is specifically optimised for DAGs. It natively implements the Sugiyama method as a layout algorithm, which allows for the desired layer-based positioning.

3.4.2 Backend Services

The backend logic is implemented in Python⁷, chosen for its dominance in the data science ecosystem and its rich library support for data manipulation [CBST23]. The core web server utilises FastAPI⁸ as a web framework. FastAPI was selected for its native support of asynchronous programming and its tight integration with Pydantic⁹, which enforces data validation and type checking on all incoming requests.

Asynchronous Task Processing

DQ assessments on large datasets are computationally intensive and potentially long-running. To prevent blocking the main thread, the architecture implements an asynchronous task queue using Celery¹⁰. The workflow is as follows:

- **Broker:** RabbitMQ¹¹ acts as the message broker, receiving task requests from the API and distributing them to available workers.
- **Execution:** Celery workers pick up these tasks and execute them in parallel. These workers integrate Schrott's CobADQ library [Sch24] to perform the actual metric calculations and aggregations. Additionally, they handle the interaction with the multi-agent AI Assistant (see Chapter 4).
- **State Management:** The status of ongoing tasks and their intermediate results are cached in Redis¹², allowing the frontend to poll for progress updates in real-time.

²<https://nextjs.org/>, Version 14.1.3

³<https://react.dev/>

⁴<https://tailwindcss.com/>, Version 3.3.0

⁵<https://github.com/erikbrinkman/d3-dag>, Version 1.1.0

⁶<https://d3js.org/>, Version 7.9.0

⁷<https://python.org>, Version 3.12.11

⁸<https://fastapi.tiangolo.com/>, Version 0.110.0

⁹<https://docs.pydantic.dev/2.6/>

¹⁰<https://docs.celeryq.dev/en/main/index.html>, Version 5.4.0

¹¹<https://rabbitmq.com/>, Image: rabbitmq:3-management-alpine. Version as of January 30 2026

¹²<https://redis.io/>, Image: redis:6.2-alpine. Version as of January 30 2026

3.4.3 Data Persistence and Storage

The system employs two distinct database technologies optimised for different data types:

Object Storage MinIO¹³, an object storage server, is used for handling large binary blobs. Note that an older image version was chosen, which supports a GUI for managing access keys. It stores the initial tabular CSV datasets uploaded by users. Furthermore, to maintain system performance, the potentially large vectors resulting from node calculations are serialised and stored in MinIO rather than the document database.

Document Storage RethinkDB¹⁴ serves as the primary operational database. It stores the metadata of the application, including the graph topology (nodes and edges), file descriptions, and user configurations. Additionally, RethinkDB is used to persist the interaction history and metadata for the AI Assistant, ensuring that the conversational context is preserved across sessions.

3.4.4 LLM Integration

To support the AI-assisted features of the platform, the backend incorporates the framework DSPy¹⁵ (an abbreviation for *Declarative Self-improving Python*). Unlike traditional prompt engineering, DSPy allows for the programmatic definition and optimisation of language model pipelines or agent loops [KSM⁺24, KSL⁺22]. A detailed discussion of the LLM integration is provided in Chapter 4.

3.4.5 Deployment

The backend services (including Celery Workers, RabbitMQ, Redis, MinIO, and RethinkDB) are fully containerised using Docker¹⁶. Multiple `docker-compose` configurations are maintained to support different environments (Debug, Development, and Production). The frontend, serving as the entry point, is built and hosted using PM2¹⁷, a production process manager for Node.js¹⁸.

3.4.6 Implementation of Node Creation and Processing Pipeline

To demonstrate the flow of architectural components, this section details the application's core functionality: the creation and processing of a new quality task node. The creation of a new node is an asynchronous operation designed to maintain GUI responsiveness while handling expensive data calculations. The process can be divided into five phases:

¹³<https://www.min.io/>, Image: minio/minio:RELEASE.2025-04-22T22-12-26Z

¹⁴<https://rethinkdb.com/>, Image: rethinkdb:2

¹⁵<https://dspy.ai/>, Version 2.6.27

¹⁶<https://www.docker.com/>, Docker Engine (server) Version 28.5.2 (linux/amd64)

¹⁷<https://pm2.keymetrics.io/>, Version 6.0.13

¹⁸<https://nodejs.org/en>

1. Request Initiation The workflow begins when the user configures a node in the frontend and submits the form. The client dispatches a POST request to the backend API containing the type of the node, the necessary configuration parameters (e.g., RegEx patterns, target columns), and the UUID of the dataset context.

2. Task Scheduling and Optimistic UI Update Upon receiving the request, the FastAPI backend does not immediately perform the calculation. Instead, it performs two actions:

1. It creates a database entry for the new node with a status flag set to `PROCESSING`.
2. It pushes a calculation job to the message broker.

The API then immediately responds to the client with the updated graph structure, including the new node. This allows the frontend to render the new node without waiting for the calculation to complete.

3. Frontend State Management (Polling) Once the frontend receives the response, it enters a processing state. Visually, the new node is rendered in a pulsing grey colour to indicate active computation. Functionally, the application enforces a lock on the graph, disabling modifications to prevent race conditions. Simultaneously, the client initiates a polling mechanism, querying the backend status endpoint to check for the task's completion.

4. Worker Execution and Persistence In the background, a Celery worker picks up the task. It invokes the CobADQ library to calculate node data. Because the resulting data can be substantial (e.g., a binary vector of length N , where N is the dataset size), it is not stored in the document database. Instead, the worker serialises the result and saves it as a CSV object in MinIO. The returned object storage UUID is then updated in the node's metadata entry in RethinkDB, which links the graph node to its data result.

5. Finalisation and Rendering Once the worker completes the task, the database status is updated to `COMPLETED`. During the next polling cycle, the frontend receives this status change. The polling loop terminates, the modification lock is released, and the node's visual representation is updated from grey to its semantic node type colour. This signals to the user that the result is ready for inspection.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Multi-Agent AI Assistant

By enabling natural language, LLMs can democratise access to DQ assessment for non-technical users while simultaneously increasing efficiency for technical users by automating tasks. To realise this potential, this thesis implements a *Multi-Agent AI Assistant*, abbreviated as “AI Assistant” in the following. Unlike LLM chatbots, this system utilises a set of specialised AI agents (each agent being implemented as one or more LLM calls) designed to plan, validate, and execute complex DQ assessments within the proposed DAG-based prototype.

4.1 AI-Assisted DQ Assessment Process

The AI Assistant shifts the workflow from manual construction to intent-based generation. Instead of manually defining DAG nodes, the user may describe their DQ task in natural language.

The process is designed as a collaborative, human-in-the-loop workflow consisting of three distinct stages: *Intent Definition*, *Plan Review*, and *Execution*. Internally, the AI Assistant is implemented as a two-phase system, each phase with a subset of agents.

1. Intent Definition (Phase 1 Agents) The workflow begins when the user enters a natural language query into a chat-like interface, which is located in the Control Layer of the Detail View Page (compare Section 3.3.1). This query can range from high-level questions (e.g., “*What is the number of valid addresses?*”) to specific constraints (e.g., “*Check if the email column holds valid email addresses*”). Crucially, the user is not required to know the system’s internal taxonomy (e.g, the difference between a Metric Node and Combination Node) or specific configuration parameters. Figure 4.1 shows the AI Assistant GUI, after the user prompt was submitted. The “processing” state of this phase is visualised through three jumping dots as well as a glowing purple “Thinking ...” button. Next to that is a “Cancel” button to interrupt the current process.

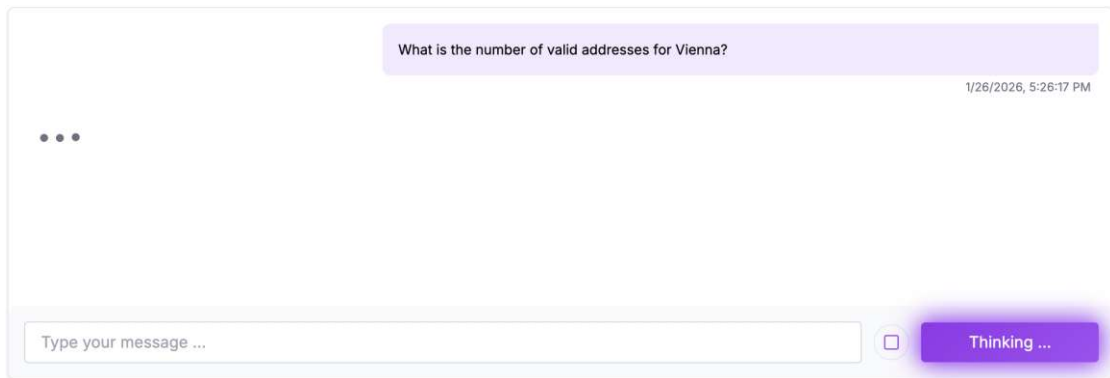


Figure 4.1: AI Assistant GUI: Intent definition phase.

2. Plan Review (Human-in-the-Loop) Upon receiving the prompt, the AI Assistant analyses the request and generates a proposed *Execution Plan*, i.e., a list of tasks that it intends to perform. Before any action is taken on the graph, this plan is presented to the user for validation and can be modified as follows:

- **Refine Intent:** Modify the textual description of individual tasks.
- **Adjust Types:** Override the inferred node type (e.g., changing a Metric Node to a Combination Node).
- **Reorder Logic:** Adjust the execution sequence to ensure dependencies are met (e.g., ensuring child nodes are created before their parents).
- **Add/Delete:** Remove redundant steps or insert new tasks that the AI Assistant might have missed.
- **Resolve Ambiguity:** If the AI flags a task type as “UNKNOWN”, the user is forced to manually assign a valid type before proceeding.

Figure 4.2 reveals the answer of the AI Assistant, presenting the proposed Execution Plan based on the initial request above, as well as the GUI. Note how *City*, *Postal Code*, *Street*, *House Number* and *Door Number* were correctly identified as required columns for this DQ task.

3. Execution and Result (Phase 2 Agents) Once the user submits the *Human Reviewed Plan*, the AI Assistant begins the creation of the DAG by inserting nodes. Figure 4.3 shows the state of this GUI during this phase. It reveals the adjusted task list from the user, as well as a “processing” state similar to Phase 2. Note that the user slightly adjusted the “Postal Code” check to further specify a Viennese format.

The user can observe the execution node by node as the AI Assistant follows a topological order: Metric Nodes are created first, followed by Combination Nodes, and finally the

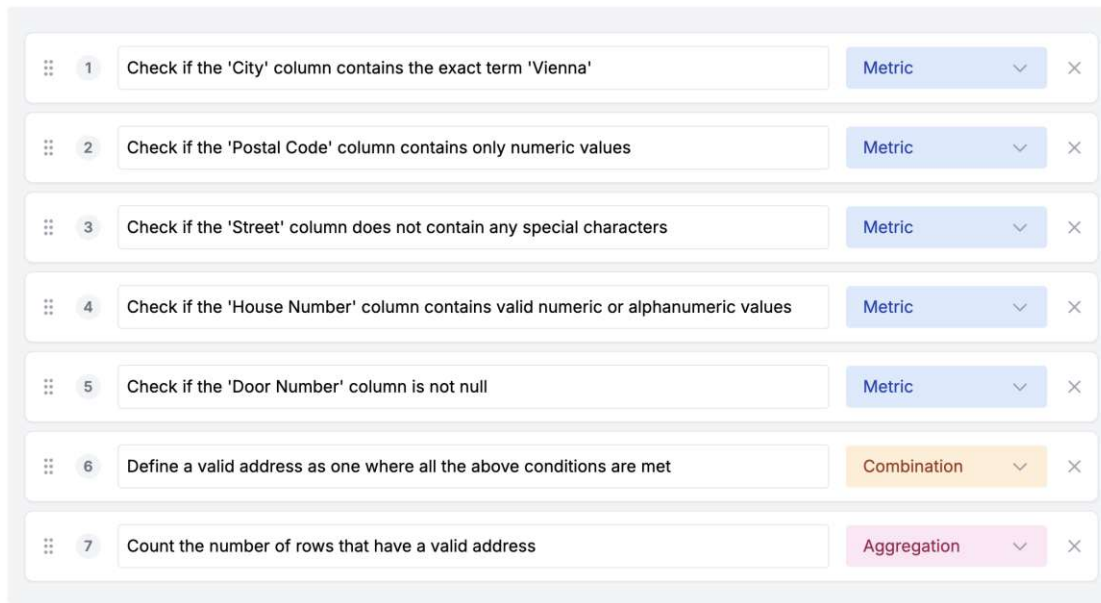


Figure 4.2: AI Assistant GUI: Execution Plan with proposed DQ tasks.

Aggregation Nodes. Figure 4.4 shows the final constructed DAG in this example, which was created based on the Human Reviewed Plan.

Notice how the Metric Nodes types and parameters are correctly inferred. For “City”, the AI Assistant used the correct value “Wien”, observed from the dataset. The “Postal Code” RegEx format was created as described in natural language, while the “House Number” and “Street” formats were created by observing the existing values, given their instruction. The *Vienna Address* Combination Node then correctly connects all Metric Nodes, which is used by the *Amount of Correct Addresses* Aggregation Node to provide the amount of all valid entries.

After the graph is fully constructed and processed, the AI Assistant observes the final state of the nodes and generates a summary response that answers the initial user prompt as a response in the GUI, as seen in Figure 4.5.

At this point, the DQ task is considered complete. The user can perform further manual adjustments or start a new request. Notably, if the user cancels the process during node creation, any nodes created up to that point are preserved.

4.1.1 Architectural Decision: Multi- vs. Single-Agent

Initial experiments for this thesis utilise a single-agent approach. While this setup performs adequately for trivial requests (e.g., creating a single Metric Node), it struggles significantly with complex, multi-step quality assessments. A single agent tasked with interpreting intent, enforcing system constraints, and executing graph logic, produces ambiguous or *AI Hallucination* results [MNB20]. This limitation is due to the “Lost

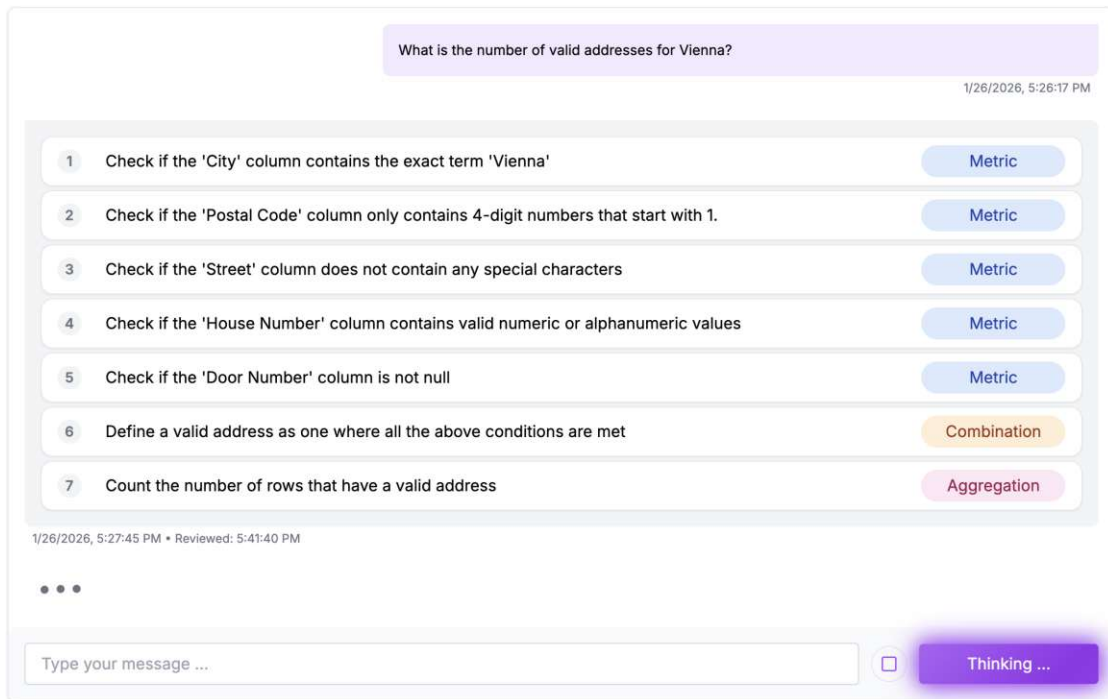


Figure 4.3: AI Assistant GUI: Execution phase.

in the Middle” effect, where LLMs struggle to prioritise task-relevant instructions near the middle of the input within large context windows [LLH⁺24].

To address this, the architecture was refactored into a multi-agent system. By splitting the DQ task into smaller, isolated sub-tasks, each agent can be optimised for a specific function (e.g., purely “Requirements Gathering” or purely “Node Creation”). This approach not only improves the accuracy and determinism of the results but also enables debugging and fine-tuning. This architectural shift aligns with findings in software engineering research, such as the *MAGIS* framework [TZW⁺24], which demonstrates that multi-agent systems significantly outperform single-agent models in complex problem-solving domains, like GitHub issue resolution.

4.1.2 AI Limitations

Despite the multi-agent optimisation, the system remains a prototype with limitations:

- **Interaction Scope:** The AI Assistant is reserved to creating DAG. It cannot modify existing graph structures (e.g., “Rename this node” or “Delete the last node”) once the execution phase is finished.
- **Performance:** The sequential nature of the agent chain prioritises accuracy over speed. This leads to noticeable latency during requests.

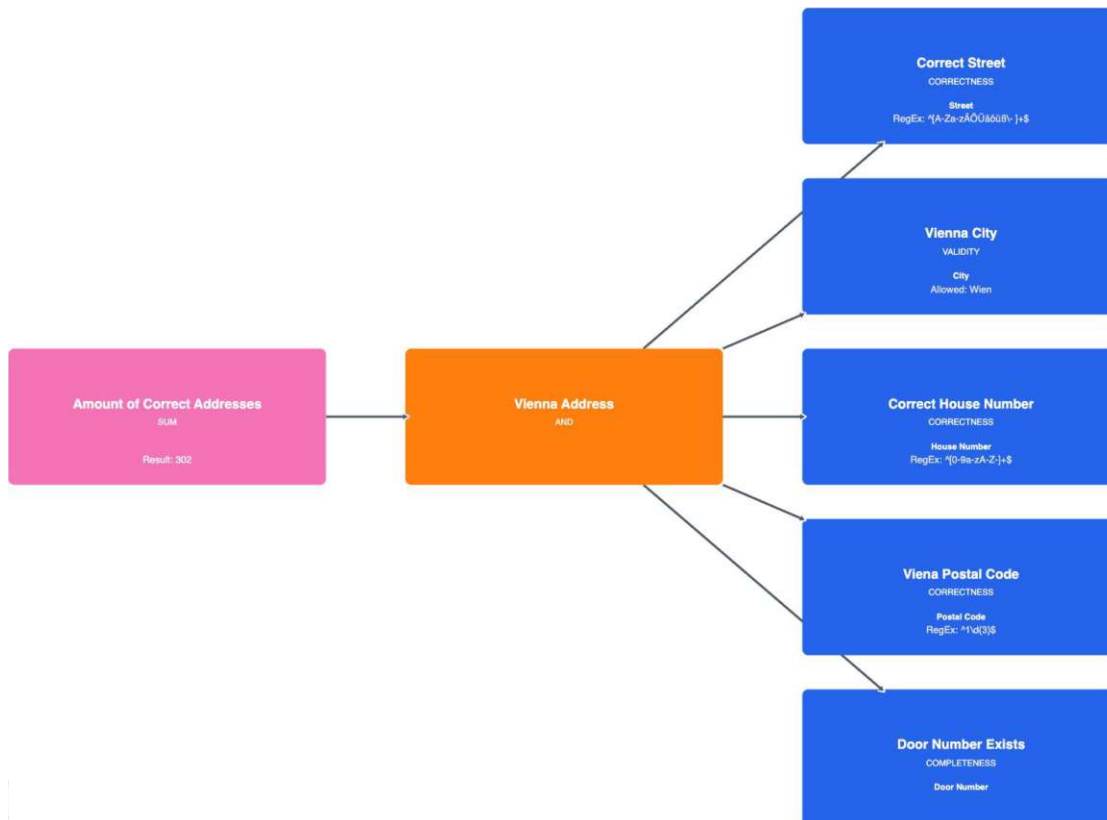


Figure 4.4: Final DAG generated by the AI Assistant.

The number of valid addresses for Vienna is 302.

1/26/2026, 5:43:39 PM

Figure 4.5: AI Assistant GUI: Final response

- **AI Hallucination Propagation:** While specialised agents exist to critique and validate plans, AI hallucination can still occur. An error at an early stage can be passed through the chain. This may result in a malformed graph structure.
- **Context Retention:** The AI Assistant treats each assessment task as an isolated event. It does not retain memory of previous conversations once a task is completed.

4.2 Implementation Details

4.2.1 DSPy Framework

The LLM interactions within the multi-agent system are managed through DSPy [KSM⁺24, KSL⁺22]. The framework abstracts raw string prompting away by

treating language models as programmable modules. It enforces strict type signatures on both inputs and outputs, ensuring that the agents return structured objects (e.g., JSON or Pydantic models) that can be processed by the backend code, rather than unstructured text. Furthermore, it allows for function calls within the prototype’s system.

The system is built and tested with OpenAI’s GPT-4o¹ model. This model is selected for its state-of-the-art reasoning capabilities and cost-performance ratio at the time. However, the architecture is model-agnostic. The abstraction layers of DSPy allow for the integration of alternative providers (e.g., Anthropic Claude², Google Gemini³) or locally hosted models (e.g., Ollama⁴).

4.2.2 DSPy Agent Types

The system uses two primary module types provided by DSPy [KSM⁺24, KSL⁺22].

- **ChainOfThought**: This module forces the model to generate reasoning steps (“rationale”) before producing the final answer. This is used for logic-heavy tasks, such as interpreting user intent or validating rules, where intermediate reasoning improves accuracy.
- **ReAct (Reasoning and Acting)**: This paradigm enables agents to interact with external tools. In the ReAct loop, the model reasons about its current state, decides to call a provided function, observes the output, and iterates until the task is complete. This is essential for agents that need to inspect the dataset before making decisions, as well as for creating nodes.

4.2.3 Linking AI Assistant to the Prototype

The AI Assistant is not a standalone service but integrated into the existing backend API. It utilises the same service layer functions as the standard REST endpoints but accesses them programmatically. To enable the “Acting” capability of the ReAct agents, a specific set of read-only and write functions was exposed as “Tools”:

- **observe_column_sample_values**: Allows agents to peek at data to verify formats (e.g., checking if “Country” contains “AT” or “Austria”).
- **get_graph_nodes**: Provides context on the current state of the DAG, by providing a list of nodes and their relations.
- **insert_[metric/combination/aggregation]_node**:
The primitive actions to modify the graph structure.

¹<https://platform.openai.com/docs/models/gpt-4o>, Usage between June 2025 – December 2025

²<https://claude.ai/>

³<https://gemini.google/>

⁴<https://ollama.com/>

4.3 Agent System Design

The multi-agent architecture is divided into two phases separated by the human-in-the-loop verification step. Figure 4.6 illustrates the system's flow and human interaction with the agents. The *Interpretation Agents* are responsible for understanding the problem and proposing a plan, while the *Creation Agents* are responsible for executing that plan in the system and answering the initial request.

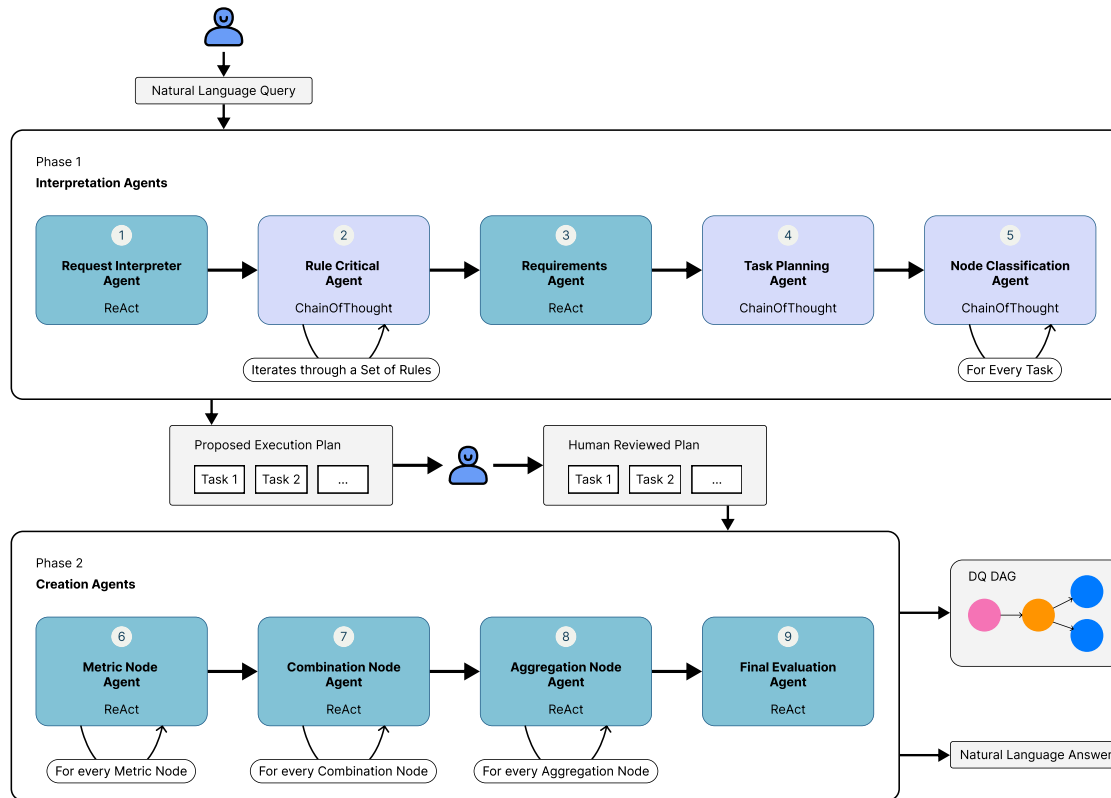


Figure 4.6: Multi-agent architecture flow with human-in-the-loop verification

4.3.1 Interpretation Agents (Phase 1)

This group of agents transform a raw string request into a structured Execution Plan. This plan is a list of tasks, where each task corresponds to a potential node in the DAG.

The process begins with the *Request Interpreter Agent* (Figure 4.7), which reinterprets the user's potentially ambiguous prompt. This agent's role is to provide an improved, explicit request for the downstream agents. It accepts the raw user string, the file ID, and the list of available columns as input. Crucially, it utilises the tool `observe_column_sample_values` to ground the user's natural language in the dataset's actual schema. For instance, it determines that a request to check Contactability

within a dataset implies validating specific columns such as “Phone” or “Email” based on the available data headers.

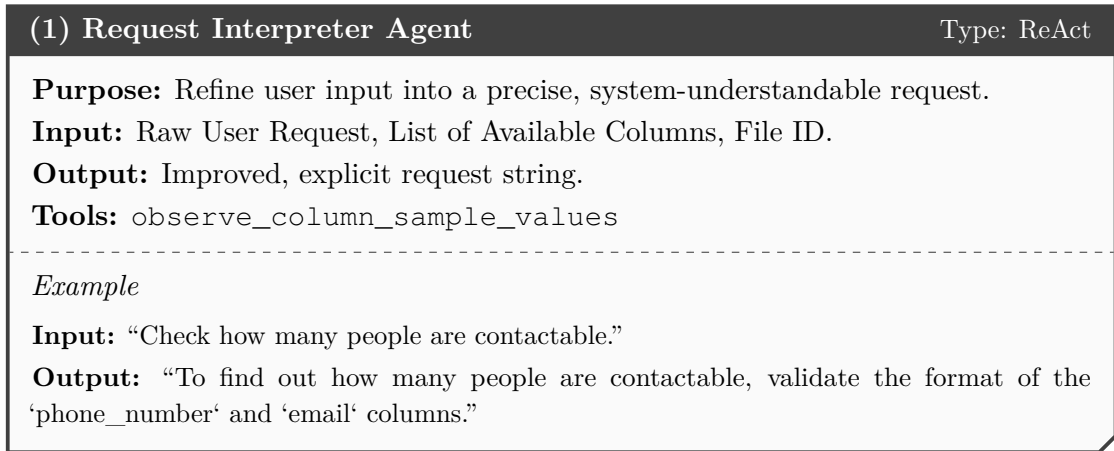


Figure 4.7: Specification of the Request Interpreter Agent.

Once the user’s request was interpreted, the *Rule Critical Agent* (Figure 4.8) acts as a guardrail by iterating through subsets of rules. The ChainOfThought approach evaluates the request against defined system limitations.

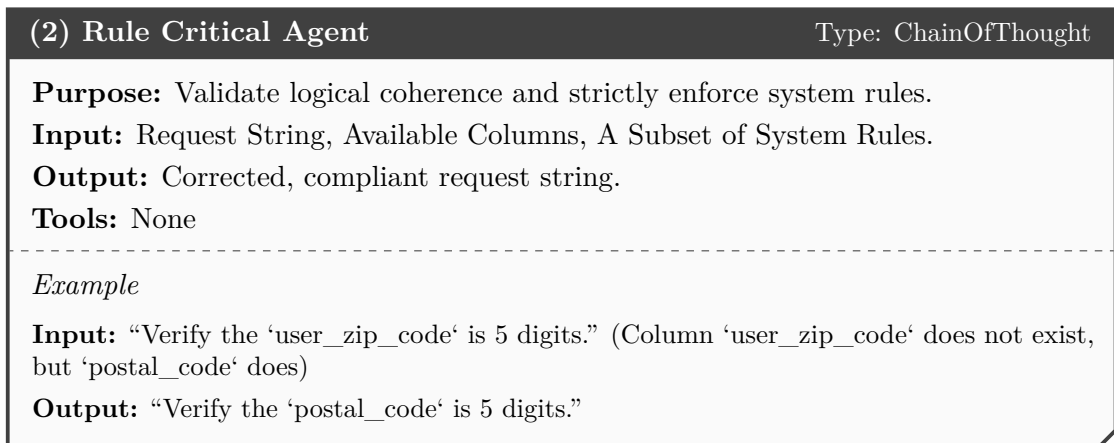


Figure 4.8: Specification of the Rule Critical Agent.

The agent is therefore forced to adhere to the following rules:

- **Referential Integrity:** Verify that combining and aggregating tasks only reference valid and existing children that are logically part of the request.
- **Self-Contained Validation:** The request must only use the provided `available_columns`. Do not use external data, APIs, or services. Invalid:

“Check if the IBAN exists.” (Requires an external service). Valid: “Check if the IBAN follows the standard format”.

- **Column Existence:** All columns referenced in the request must be present in the `available_columns` list. Invalid: “Verify the ‘user_zip_code’ is 5 digits.” (if ‘user_zip_code’ is not in the list). Corrected: “Verify the ‘postal_code’ is 5 digits.” (if ‘postal_code’ is in the list).
- **Logical Coherence:** Use your real-world knowledge to fix illogical requests. Invalid: “Ensure ‘Phone’ is a German number with 30 digits.” (German phone numbers are not that long). Corrected: “Ensure ‘Phone’ follows a valid German phone number format.”
- **Implicit Dependencies:** Identify and add conditions for related columns. Request: “Ensure ‘Phone’ is not null.” Corrected: “Ensure that if ‘Phone’ is not null, then ‘Phone_prefix’ must also not be null.”
- **Critically analyse the task:** Check if other columns need to be validated as well to fulfil the general request. Invalid: “Ensure ‘Phone’ is not null.” (If checking if a person is contactable) Corrected: “Ensure ‘Phone’ is not null OR Ensure ‘Email’ is not null.”
- **Conditional and Contextual Validation:** Acknowledge that certain fields only become mandatory based on a category or type selected in another field. Invalid: “Validate that `payment_method`, `credit_card_number`, and `paypal_email` are all provided. Valid: “If `payment_method` is ‘Credit Card’, then `credit_card_number` must not be null. If `payment_method` is ‘PayPal’, then `paypal_email` must not be null.”

This ensures that AI hallucinations or technically invalid instructions are eliminated before they reach the next agents.

With a valid request established, the *Requirements Agent* (Figure 4.9) decomposes the text into sub-tasks of work. It breaks down logic into individual verifiable conditions to create the initial draft of the Execution Plan. For example, a request to “Check if Category is Electronics or Books” is split into three distinct tasks: checking for Electronics, checking for Books, and a final task to combine those results via a logical OR. This approach ensures that every condition is verifiable and that implicit dependencies are made explicit in the final plan.

A list of requirements is insufficient without logical ordering. The *Task Planning Agent* (Figure 4.10) organises the steps into a strict topological dependency order (Metric Node → Combination Node → Aggregation Node). It analyses the unordered steps provided by the Requirements Agent and ensures that child nodes are created before the parent nodes that consume them. It may also insert missing intermediate steps if the logical flow is incomplete.

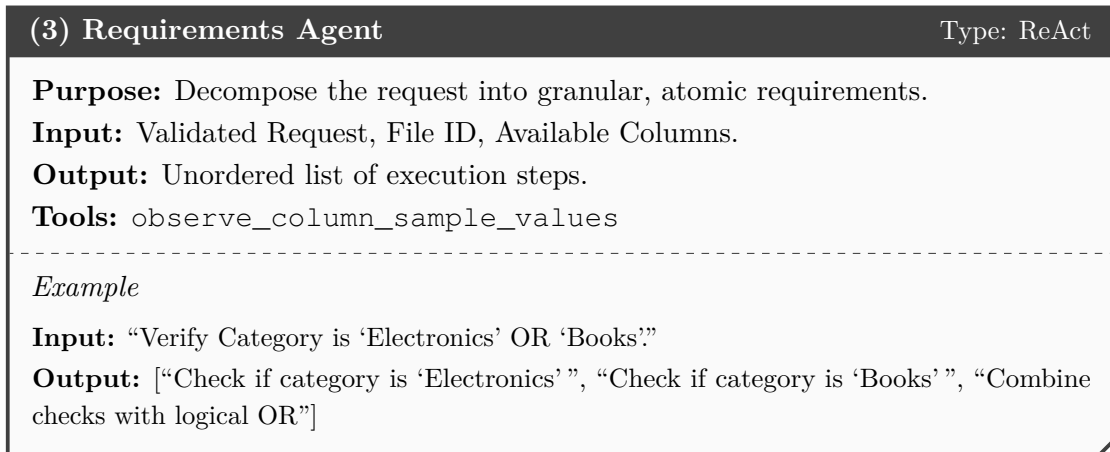


Figure 4.9: Specification of the Requirements Agent.

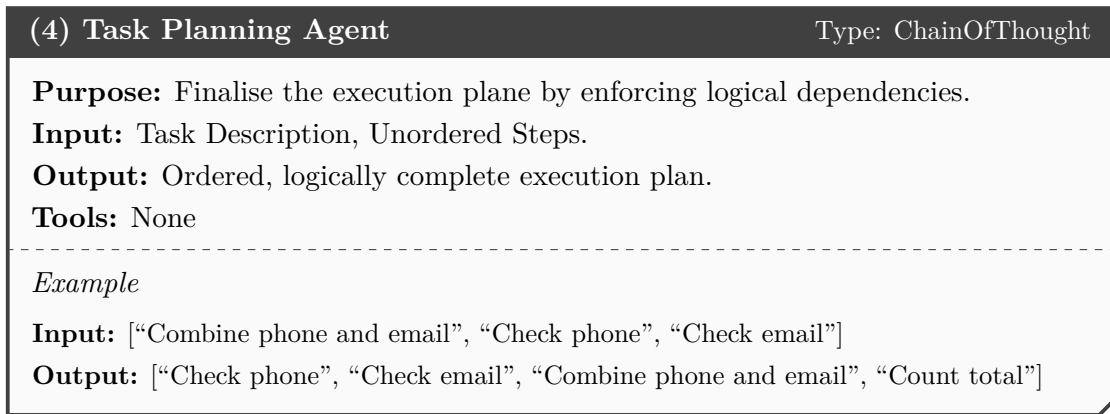


Figure 4.10: Specification of the Task Planning Agent.

Finally, the *Node Classification Agent* (Figure 4.11) iterates through the ordered plan and assigns a specific node type to each step. Note that while the system internally recognises only METRIC, COMBINATION, and AGGREGATION nodes, this agent is prompted to use the terms VALIDATION (for Metric Nodes) and LOGIC (for Combination Nodes). Empirical testing revealed that LLMs consistently misclassify nodes when using the abstract graph terms “Metric” or “Combination”. However, using the semantic terms “Validation” and “Logic” yields significantly higher accuracy.

This categorisation is used in the frontend GUI, allowing the user to visually distinguish between metric checks, combination operations and final aggregations, during the review phase. The Node Classification Agent also appends a confidence score for each task. If the score is below 0.85, the type is marked as UNKNOWN, flagging it for human review.

At the conclusion of Phase 1, the system halts. The final structured Execution Plan is

(5) Node Classification Agent	Type: ChainOfThought
<p>Purpose: Classify each task step into a specific graph node type.</p> <p>Input: Single Task Step.</p> <p>Output: Node Type or “Unknown”, Confidence Score.</p> <p>Tools: None</p> <hr style="border-top: 1px dashed #000;"/> <p><i>Example</i></p> <p>Input: “Ensure phone_number is not null.”</p> <p>Output (Node Type): VALIDATION (Mapped to METRIC)</p> <p>Output (Confidence): 0.99</p>	

Figure 4.11: Specification of the Node Classification Agent.

presented to the user. The user may refine, reorder, or delete tasks and then submit a Human Reviewed Plan to Phase 2.

4.3.2 Creation Agents (Phase 2)

While Phase 1 focused on text processing, Phase 2 focuses on graph creation. The *Creation Agents* translate the approved Human Reviewed Plan plan into actual nodes within the DAG.

The *Metric Node Agent* (Figure 4.12) handles DQ checks, creating nodes of type Validity, Correctness, or Completeness. Before creating a node, it checks if a similar node already exists to allow reuse. This agent utilises observed data context to parametrise the node correctly. For instance, when creating a validity check for a “Country” column, it will sample the data distribution to generate a precise `allowed_values` list (e.g., “Austria”, “AT”), while excluding misspellings or outliers based on DQ rules.

The *Combination Node Agent* (Figure 4.13) constructs the combination layer of the graph. It is responsible for linking previously created Metric Nodes via logical or statistical operators to represent logic. Its goal is to fulfil the requirement by either identifying a matching existing node (one with the same purpose, operator, and children) or creating a new Combination Nodes.

The *Aggregation Node Agent* (Figure 4.14) applies table-value operations to the Combination Node or Metric Node to calculate the final DQ score. Like the other Creation Agents, it first attempts to identify an existing Aggregation Node that matches the requested operator and child node to avoid redundancy, inserting a new node only if no exact match is found.

The pipeline concludes with the *Final Evaluation Agent* (Figure 4.15). This agent’s purpose is to determine if the executed sequence of tasks successfully fulfils the initial

(6) Metric Node Agent	Type: ReAct
<p>Purpose: Create Metric Node of type Validity, Correctness, or Completeness by enforcing data quality rules and utilising observed data context.</p> <p>Input: Task Step, Available Columns, File ID.</p> <p>Output: Metric Node creation, if it doesn't already exist.</p> <p>Tools:</p> <ul style="list-style-type: none"> • observe_column_sample_values • get_graph_nodes • insert_metric_node 	
<hr style="border-top: 1px dashed #000;"/> <p><i>Example</i></p> <p>Input: "Ensure the 'Country' column only contains valid variations of Austria."</p> <p>Observation: 'Country' column values: ["Austria", "AT", "Australia", "Poland"]</p> <p>Final Action:</p> <pre>insert_metric_node(column="Country", type="VALIDITY", allowed_values=["Austria", "AT"])</pre>	

Figure 4.12: Specification of the Metric Node Agent.

user request. It reviews the list of performed tasks and analyses the final state of the DQ graph. Verifying that the correct nodes are present and correctly linked returns a final text answer and a confidence score indicating the operation's success.

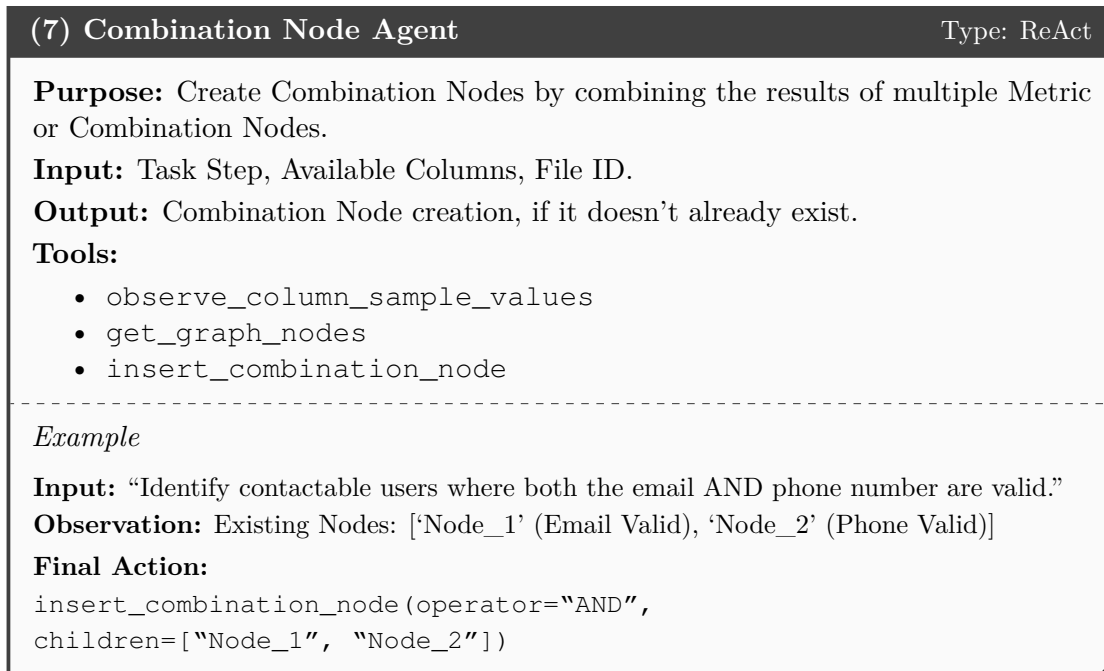


Figure 4.13: Specification of the Combination Node Agent.

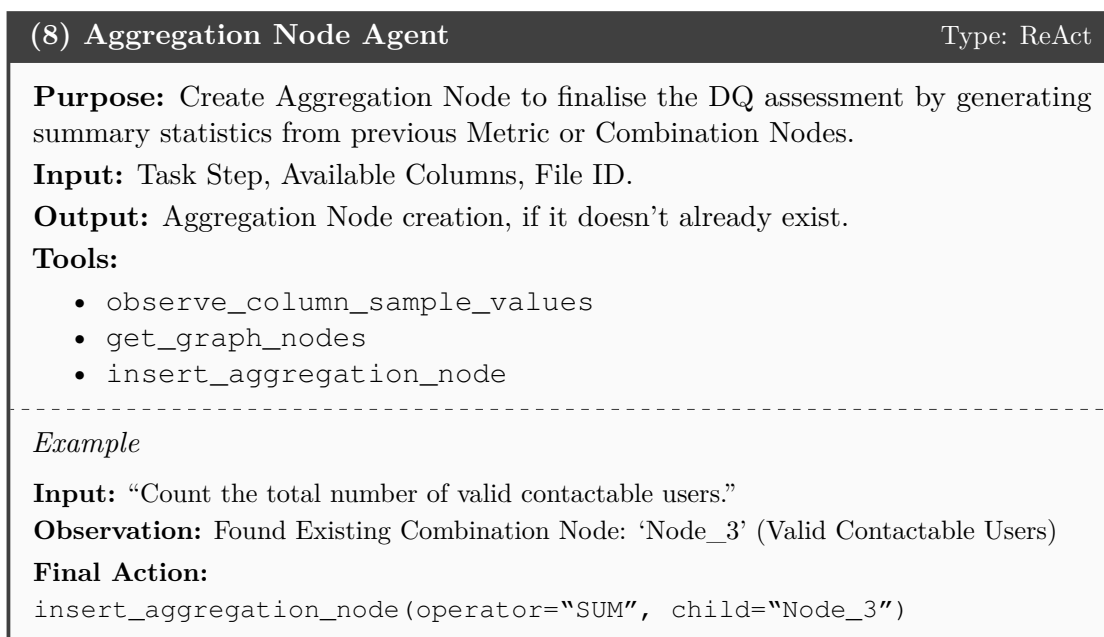


Figure 4.14: Specification of the Aggregation Node Agent.

(9) Final Evaluation Agent	Type: ReAct
Purpose: Verify the final graph state against the initial request and output a final result for the underlying DQ assessment.	
Input: Initial Request, Performed Tasks, File ID.	
Output: Evaluation, Final Answer, Confidence.	
Tools:	
<ul style="list-style-type: none">• observe_column_sample_values• get_graph_nodes	
<hr/>	
<i>Example</i>	
Input (Initial Request): “Count how many rows have a valid AT phone number.”	
Input (Performed Tasks): [‘Task 1’, ‘Task 2’, ...]	
Observation: Graph contains: Aggregation Node (Operator: ‘SUM’, Result: 250).	
Output (Evaluation): “The graph successfully includes a Metric Node for AT phone formats and an Aggregation Node to count valid results.” (Confidence: 1.0)	
Output (Confidence): 1.0	
Output (Final Answer): “250 rows hold valid AT phone numbers.”	

Figure 4.15: Specification of the Final Evaluation Agent.

Evaluation & Results

This chapter assesses the prototype against the challenges outlined in Chapter 1 by addressing the Research Questions through both qualitative and quantitative evidence.

The objective is to determine whether the system operates as intended and is genuinely useful in its intended context: facilitating DQ assessment in a manner that is transparent, efficient, and accessible to various user groups.

We employ a mixed evaluation approach to capture the prototype's practical value under realistic conditions. This includes examining how it supports DQ assessment tasks, the transparency and trustworthiness of the interaction, and how users can efficiently obtain accurate results.

Our evaluation centres around the three core Research Questions:

- First, we compare the prototype with existing approaches and tools to clarify what current solutions cover and identify the gaps that the prototype addresses.
- Second, we engage domain experts in a Focus Group to test the core interaction concepts. This step validates design decisions and identifies conceptual misunderstandings during the implementation of the prototype. It is crucial to assess whether the system employs transparent reasoning about quality, rather than just providing a result.
- Third, we evaluate the prototype through an empirical User Study. Participants complete representative tasks in both a manual and a hybrid AI-assisted workflow, allowing for a comparison between the two approaches. This study quantifies practical impact in terms of task efficiency, accuracy, and user experience, while also testing whether the LLM integration reduces effort and lowers the technical barrier for non-technical users.

The following sections outline the methodology and findings for each phase of the evaluation.

5.1 Literature Analysis

To contextualise the contributions of the developed prototype, this section analyses existing DQ solutions through the lens of Research Question 1:

“What are the functional and usability limitations of existing data quality metric tools, and in which aspects do they hinder domain experts from independently assessing data quality?”

Building upon the analysis of Chapter 2, this evaluation assesses current approaches regarding their ability to support non-technical experts in defining, combining, and verifying DQ constraints.

5.1.1 Categorisation of Existing Solutions

To systematically identify functional gaps, the analysis classifies DQ tools into three categories. It must be noted that this classification is not exhaustive. Due to the proprietary nature of commercial platforms, access to paid enterprise models is not feasible within the scope of this research. Furthermore, given the vast and rapidly evolving market, it is not possible to analyse every available tool in detail. However, to the best of our knowledge, the following groups represent the main paradigms used in practice, identified in the state-of-the-art review:

1. **Frameworks and CLI Tools:** This category holds code-centric solutions. It includes Python/SQL-based solutions (e.g., dbt Core¹ or Great Expectations²), constraint-based aggregation frameworks (e.g., Schrott’s CobADQ [Sch24]) or custom coding solutions. This category offers broad expressiveness, but requires technical expertise.
2. **Visual Solutions:** This group comprises visual solutions that offer GUIs, such as open-source projects, like MetricDoc [BGK⁺18] or MobyDQ³ or enterprise solutions (e.g., Ataccama One⁴ or Informatica⁵). These solutions focus on stability, broad connectivity, and high-level dashboarding for management reporting. It is important to acknowledge that modern iterations of these suites increasingly incorporate AI capabilities, such as Informatica’s CLAIRE⁶. However, due to the cost and restricted access associated with these features, this analysis treats them

¹<https://github.com/dbt-labs/dbt-core>

²github.com/great-expectations/great_expectations

³<https://ubisoft.github.io/mobydq/>

⁴<https://www.ataccama.com/platform>

⁵<https://www.informatica.com/>

⁶<https://www.informatica.com/platform/claire-ai.html>

primarily as components of traditional visualisation solutions. A deep hands-on evaluation of their AI modules is outside the constraints of time and budget for this thesis.

3. **Generative AI Tools and LLM Interfaces:** This category includes tools that leverage LLMs to facilitate HDI. This ranges from specific tools like Soda⁷ to AI extensions in data wrangling tools like OpenRefine⁸, as well as the ad-hoc usage of general-purpose models (e.g., ChatGPT’s GUI for data analysis).

5.1.2 Analysis of Current Approaches

Based on the review of existing solutions above, three primary limitations hinder independent assessment by domain experts.

The Technical Barrier and Coding Dependency

The most significant hurdle for domain experts is the requirement for programming skills. Frameworks that offer deep logical capabilities operate via CLI or require Python/SQL scripting. While tools such as Schrott’s framework enable the implementation of complex logic [Sch24], they exclude stakeholders who understand the semantics of the data but lack the syntax knowledge to implement it. Consequently, experts must rely on data engineers to translate their requirements into code, creating a bottleneck that prevents rapid iteration on Fitness for Use definitions.

Flat Visualisation and Lack of Visualised Aggregation and Combination

GUIs in the enterprise space rely on flat or simple visualisation techniques. Tools present DQ as a list of isolated pass/fail checks or aggregate them into single scores [EW22]. There is a distinct lack of tools that visualise the *hierarchical dependencies* between metrics. For example, complex Fitness for Use scenarios, composed of specific “Email” and “Phone” checks combined via an OR logic, are not visually represented. Without a DAG or other forms of visualisations, users cannot mentally model how low-level data errors propagate up to affect high-level results.

The Black Box Problem of Automation

The rise of AI-assisted tools causes a Black Box effect. While generative AI tools like Soda allow for natural language input and address the accessibility barrier, the execution logic remains opaque. If a user asks an AI to “check for valid addresses”, the system might generate complex SQL code or internal rules in the background, but if the user cannot read that code, they are forced to blindly trust the result. Studies indicate that this widens the Gulf of Evaluation, as the expert cannot verify if the tool’s definition of quality matches their domain-specific expectations without reviewing generated code [GSA⁺24, XZX⁺24].

⁷<https://soda.io/product/soda-ai>

⁸<https://github.com/sunilnatraj/llm-extension>

5.1.3 Feature Comparison

Table 5.1 compares the capabilities of the developed prototype against the three solution categories defined above.

Table 5.1: Comparison of functional and usability features across data quality approaches.

Feature / Capability	Frameworks & CLI Tools	Visual Solutions	GenAI Tools & LLMs	Our Prototype
<i>Interaction & Usability</i>				
No Programming Required	×	~	✓	✓
Natural Language Input	×	×	✓	✓
DQ Combination & Aggregation Visualisation	×	×	×	✓
Transparent Verification	✓ ¹	~	×	✓
<i>Production Readiness</i>				
Pre-defined Metric Catalogue	✓	✓	~ ²	×
Continuous Monitoring	✓	✓	✓	×

Legend: ✓ = Fully Supported; ~ = Partially Supported; × = Not Supported / Uncommon.

¹ Transparent only to those who can read the code.

² LLMs possess knowledge of metrics but do not offer a GUI-based catalogue.

The Table highlights a balance between production readiness and interaction. While script-based and enterprise solutions offer robust features for long-term operations, such as predefined metric catalogues and monitoring capabilities, they lack the accessibility required for exploration by domain experts. Conversely, while the developed prototype in this thesis lacks the breadth of a production library (e.g., no pre-defined buttons for common checks) or the infrastructure for continuous monitoring, it fills the gap in visualising combination and aggregation techniques and transparent verification. This comparison highlights that the prototype is not designed to replace production solutions, but rather to address the specific representation (Verifiability by Design) and verification (Gulf of Evaluation) problems identified in the research gap analysis.

5.1.4 Limitations of Existing DQ Tools

Based on the analysis of the related work and the functional comparison above, it is possible to derive an answer to Research Question 1. Existing tools hinder domain experts primarily through two mechanisms: high technical barriers and low verifiability.

Functional Limitations Existing tools lack visual aggregation and combination capabilities. They treat DQ as a flat list of isolated constraints rather than a hierarchical system of dependencies. This prevents users from modelling complex Fitness for Use scenarios visually, where multiple metrics must be combined to form a meaningful assessment.

Usability Limitations The reliance on programming syntax or opaque automation creates a dependency loop. Domain experts find themselves hindered from working independently because they must either rely on engineers to write the code or blindly trust a Black Box system. They lack a transparent, visual medium (a Glass Box) to verify that the implemented logic matches their domain intent.

The prototype developed in this thesis addresses these limitations by introducing a visual DAG-based workflow. This workflow uses natural language for generation and interactive table highlighting for verification, while consciously omitting production-grade features to focus on the interaction paradigm.

5.2 Focus Group

Following the identification of the functional and usability limitations of DQ tooling in Section 5.1, a moderated Focus Group evaluates whether the developed prototype effectively bridges these gaps. This session had two purposes. First, to gather qualitative insights regarding the impact of visual aggregation techniques by contrasting the prototype against the baseline established in the Literature Analysis. And second, to evaluate the system’s operational status and readiness for the User Study. Therefore, the session took place prior to the User Study, allowing feedback on the final version. It involved two data management experts and lasted approximately 75 minutes. The agenda comprised a presentation of the prototype, a collaborative task involving visual DQ assessment, a demonstration of the multi-agent AI Assistant integration, and a concluding discussion on how the tool aligns with or enhances the current state of practice.

Consequently, the primary objective of this session is to answer Research Question 2:

“To what extent do visual aggregation and combination techniques improve the efficiency and accuracy of data quality assessments for both technical and non-technical users?”

The following sections outline the experts’ feedback on improvements over existing tools, identify limitations, and present an analysis of the system’s impact.

5.2.1 Improvements to the Current Status Quo

Based on the comparison with the three solution categories from the Literature Analysis, the experts explicitly state that the prototype is a clear addition to the current state-of-the-art. They confirm that the system addresses the “Flat Visualisation” problem of enterprise dashboards. A key benefit identified is the ability to see exactly which columns and rows are affected by a specific Metric Node. Unlike Black Box solutions, the prototype’s Glass Box visual highlighting allows for rapid result identification in the metric or combination logic. This feature supports validation along *syntactic validity* and *semantic consistency*, both central to the thesis framing. As a result, the visual feedback reduces the need for manual checks.

Furthermore, the Focus Group participants emphasise the value of the DAG structure for building complex logic. They note that constructing nested conditions using Boolean operators without writing code is a “massive efficiency boost”, particularly when augmented by the AI Assistant as a Drafting Engine. This visual abstraction of technical details, such as applying RegEx, lowers the barrier to entry, allowing non-technical users to define constraints that previously required engineering support.

Beyond the initial creation of rules, the experts point out the system’s benefits regarding reusability and adaptability. The visual representation allows stakeholders to face the Black Box problem and quickly understand the logical flow of a quality assessment without reading complex code. This read-only capability is particularly valuable for auditors who need to verify the solution independently. By keeping the human-in-the-loop during the graph construction, the system mitigates the risk of blind trust associated with pure AI automation. The user remains involved in verifying the structure, ensuring that the final solution is both understood and trusted.

5.2.2 Critique and Limitations

Despite the positive reception, the experts identify several areas for improvement regarding the GUI and the AI integration. A primary concern involves the handling of natural language prompts, which occasionally result in incorrect node suggestions. The participants observe logical errors in the output of the AI Assistant, such as wrongly applying a LEN operator for Aggregation Nodes to calculate an average. This reinforces the necessity of the human-in-the-loop approach to catch such AI hallucinations.

Regarding the GUI, the experts note that the extensive scrolling required to navigate between the graph visualisation and the chat interface hindered the workflow. Closely related to this navigation problem, experts highlight a disconnect between the human-approved tasks and the visual graph. Participants note that the absence of this feature forces them to rely on guesswork to trace the origin of graph elements. They recommend implementing a linking mechanism where selecting a specific task in the chat highlights the corresponding node, and vice versa.

A discussion focuses on the naming convention for result columns in the Data Result

Table, which currently display the respective node names. While one expert suggests renaming these to specific quality dimensions (e.g., Validity) to improve readability, the other argues against it, noting that unique node names are essential for distinguishing between multiple validity checks in complex graphs.

Furthermore, the experts note the absence of visual indicators for specific node types. They suggest that distinguishing metric, combination, and aggregation nodes via distinct colours in the GUI elements would improve the speed at which users can scan and parse the graph structure.

Functionally, the lack of a pre-defined catalogue of common DQ checks (e.g., email or phone validation) was identified as a missing feature. Additionally, participants point out the absence of an edit functionality for existing nodes, which forces them to recreate nodes entirely if errors occur.

5.2.3 Summary of Insights and Development Status

A key outcome of the Focus Group is a set of actionable insights regarding the tool’s usability and feature set. As this evaluation was conducted before the User Study, the feedback is categorised based on the urgency of implementation:

- **Implemented:** Features that were added to the prototype before the User Study. These are prioritised because of usability blockers that would have otherwise invalidated the experimental results.
- **Mitigated:** Issues that are partially addressed or where expert consensus is mixed, resulting in the retention of the current design.
- **Future Work:** Non-critical enhancements or “nice-to-have” features that, while valuable, are outside the scope and time constraints of this thesis.

Decisions on which features to implement are driven by the need to ensure a viable experimental setup. For instance, the lack of an edit function is deemed a “showstopper” that would penalise users during the study tasks and require implementation up front. Conversely, a GUI polish, while beneficial, was considered manageable for a controlled study environment and was therefore deferred. Table 5.2 summarises these findings and their respective statuses.

Based on this review, the Focus Group concludes with a formal assessment of the system’s maturity. With the critical “Edit Functionality” implemented and the strategies for mitigating AI hallucinations in place, the experts confirm that the prototype is ready for the User Study.

Table 5.2: Categorisation of Focus Group feedback and implementation status.

Identified Issue	Description	Status
Lack of Edit Functionality	Experts note that recreating nodes upon making a mistake is tedious.	Implemented
Visual Node Indicators	Lack of distinct colours/indicators for different node types in GUI.	Implemented
Exhausting Page Navigation	Extensive scrolling between the GUI and the Graph causes friction.	Future Work GUI Overhaul
Task-Node Traceability	Linking tasks to resulting nodes to visually trace element origin and reduce guessing.	Future Work GUI Overhaul
Missing Pre-Defined Metric Catalogue	Lack of pre-sets for standard checks (e.g., email, phone) slows down standard manual tasks.	Future Work Out of Scope
Result Column Naming	Experts disagree on whether to use generic unique names or specific dimension names.	Mitigated No Consensus
AI Assistant Logical Errors	The AI Assistant occasionally applies functions incorrectly.	Mitigated Human-in-the-Loop

5.2.4 Effects of Visual Combination and Aggregation Techniques

The Focus Group findings suggest that visual aggregation techniques significantly enhance the assessment process, although the benefits vary with users' technical expertise.

For technical users and data experts, the primary improvement lies in efficiency and documentation. The interactive DAG serves as self-documenting logic, allowing for faster management and customisation of complex rule systems compared to maintaining flat lists of constraints. The ability to verify metric impacts instantly through table highlighting streamlines the debugging process.

For non-technical users, the tool successfully lowers the barrier to entry. The combination of natural language interaction and graphical representation allows them to understand and construct complex quality logic that was previously inaccessible. The report-style visualisation enables management stakeholders to see results without needing to understand underlying formulas or query languages.

However, the evaluation also reveals that a certain level of conceptual understanding is

required. Users must understand the basic concepts of tree or graph structures to utilise the tool effectively. While the AI Assistant helps generate nodes (and therefore helps overcome the Blank Canvas problem), the user must still possess the logical ability to structure the dependencies between metrics. Therefore, while the tool bridges the gap between raw data and complex logic, it functions best where AI suggestions are verified through visual evidence.

Following this analysis, the Focus Group provides the necessary expert validation to interpret the efficiency and accuracy gains in order to derive an answer to Research Question 2.

The findings indicate that visual aggregation improves the process for both groups, but through different mechanisms:

Impact on Technical Users (Experts)

For data engineers, the primary improvement is operational efficiency and transparency.

- **Self-Documenting Logic:** Unlike flat SQL scripts, code repositories, or complex GUIs, the interactive DAG serves as a sort of self documentation. Experts report that this structure makes it significantly faster to understand, adapt, and to reuse existing rule sets.
- **Verifiability by Design:** The system eliminates the need for blind trust, common in Black Box automation. By providing visual feedback (highlighting affected rows), experts can verify complex logic.

Impact on Non-Technical Users

For domain experts and management, the primary improvement is accessibility and independence.

- **Lowering the Entry Barrier:** The abstraction of technical complexity (e.g., RegEx or logical operators) into visual nodes allows users to define Fitness for Use without programming knowledge.
- **Bridging the Gulf of Evaluation:** The combination of using natural language and receiving visual feedback allows non-experts to not only generate rules but also verify them. They can immediately see if the system's understanding of quality matches their domain knowledge, a task that was previously limited without engineering support.

Synthesis

In conclusion, visual aggregation techniques transform DQ assessment from a technical engineering task into an accessible workflow. However, it is important to note that a

conceptual barrier remains. While the tool removes the need for coding syntax, users must still possess the logical ability to understand hierarchical dependencies to utilise it to its full potential.

5.3 User Study

To evaluate the practical effectiveness of the proposed system and to address the third research question, a User Study was conducted involving 14 participants.

The primary objective of this study is to investigate Research Question 3:

“Which further improvements does the integration of large language models offer to a data quality tool in terms of enhancing user interaction and enabling natural language query construction?”

5.3.1 Study Design and Procedure

In the study, each participant solves two distinct DQ assessment tasks. One task requires a manual approach, constructing the graph node-by-node using the Node Creation Interface. The other utilises a hybrid approach, leveraging the multi-agent AI Assistant. In this hybrid mode, users prompt the system using natural language, after which the AI Assistant generates a proposed graph structure. Crucially, this workflow maintains the human-in-the-loop paradigm. Users are explicitly encouraged to analyse, refine, and extend the AI-generated results before and after execution. To mitigate potential learning biases, such as the carry-over effect where the second task becomes easier due to familiarity with the interface, the order of these tasks is iterated across participants [Gre76].

The evaluation is conducted as a guided User Study with a moderator present throughout the session. Depending on the participant’s availability, sessions are held either remotely using a deployed version of the application or in person on a local machine. The moderator provides an introduction and remains available to answer clarifying questions regarding the GUI or task definitions. This guided approach ensures that technical hurdles do not prevent the observation of core interaction patterns.

Dataset Description

Participants perform their assessments on a synthetic dataset designed to mimic customer data. The dataset contains 4995 rows and ten columns, including *Email*, *First Name*, *Last Name*, *Postal Code*, *City*, *Street*, *House Number*, *Door Number*, *Date of Birth*, and *Phone Number*. To simulate a realistic DQ scenario, the dataset includes various quality issues. All columns contain null values, and specific fields feature malformed entries, such as forbidden email formats (e.g., `Thea_Pond6857nanoff.info`), typographic errors in street names (e.g., `Fliedergass+e`), or non-existent postal codes.

Task Design and Complexity

Task complexity varies deliberately between the two tasks. The *Manual Task* is smaller and requires the construction of 4 nodes, whereas the *Hybrid Task* is larger and requires 6 nodes. This asymmetry tests the scalability of the AI Assistant. For small tasks, the overhead of prompting an AI may exceed the effort of manual creation. By assigning the more complex logic to the Hybrid Task, the study examines whether the AI reduces cognitive load for larger graph structures.

Task steps are well explained, so the focus stays on system interaction (GUI usage, AI Assistant conversation, graph comprehension) rather than problem-solving.

Task A: Manual Graph Construction In this task, participants identify customers based on Contactability. They manually create metric nodes for phone and email validation, combine them using a logical OR operator, and aggregate results using SUM. Figure 5.1 shows the provided instructions, and Figure 5.2 shows the target solution.

Manual Graph Construction

In this task, you'll answer the following question: *How many entries are either contactable via phone, email or both?*

To answer this question, you should perform a data quality assessment task by manually constructing a graph. Start by creating metric nodes to check for valid 'Phone Number' and 'Email'. Then, combine these metrics using a combination node (OR). Finally, create an aggregation node (SUM) to compute the final amount.

Please inform your instructor if you have completed the task or need assistance.

Hints

- You'll need 4 nodes (2 metric, 1 combination, 1 aggregation) in order to produce a correct result.
- For 'Phone Number' you may simply create a COMPLETENESS node that checks if the field is not empty.
- For 'Email' you may use a CORRECTNESS node that checks if emails correctly matches a RegEx pattern. Use the following RegEx:

```
^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$
```

Figure 5.1: Manual Task: Participant provided task description.



Figure 5.2: Manual Task: Target graph solution (4 Nodes).

Task B: Hybrid AI-Assisted Construction In this task, participants analyse personal information. The task requires a more complex structure with four distinct metric checks, a logical AND combination, and an AVG aggregation to determine the percentage of valid rows. Unlike the Manual Task, the instructions remain intentionally underspecified with respect to validation rules. For instance, the instructions provide no concrete RegEx or format constraints for the “Email” or “Date of Birth” columns. This design choice requires reliance on the AI Assistant to propose technical details for high-level requirements. The aim is for a participant’s shift to a human-in-the-loop verifier who validates the AI’s proposal by inspecting the results in the Dataset Inspection Table. Figure 5.3 shows the provided instructions, and Figure 5.4 shows the target solution.

Measurements

Quantitative and qualitative data are recorded throughout the sessions. Regarding task performance, the study measures *Total Time on Task*, *Correctness of the Solution* (based on the final calculated value), and *Optimal Solution*, i.e., node count. To specifically evaluate the performance of the AI Assistant in the Hybrid Task, logs capture the Execution Plan and the Human Reviewed Plan. Capturing these two states allows to calculate semantic alignment metrics, specifically *Precision*, *Recall*, *F₁ Score*, and *Similarity*. Qualitatively, observational notes regarding participant behaviour focus on prompting strategies, errors in tool usage, and verbalised confusion. Following the tasks, participants complete a survey about their experience with the tool and conduct a System Usability Scale (SUS) assessment.

5.3.2 Participants

The study includes a total of 14 participants ($N = 14$), consisting of 5 female and 9 male subjects. To analyse the impact of the tool across different expertise levels, two groups are formed:

- **Technical Group** ($n = 7$): Participants active in the fields of Data Science or Computer Science, or holding a relevant academic degree.

Hybrid AI-Assisted Graph Construction

In this task, you'll answer the following question: *What percentage of entries have their First Name, Last Name, Date of Birth and Email correctly filled?*

To answer this question, you should perform a data quality assessment task by constructing a graph with the AI Assistant. Start by using the chat to prompt your task. After a few minutes, the assistant will propose a list of nodes to create. Analyse the proposed nodes and adjust them as needed (rewrite, reorder, delete, etc.). Once submitted, watch as your graph gets built. Analyse the meaning and result of the graph. You're allowed to make adjustments if needed. You can also start over at any time and try to use different prompts to get different graphs.

Please inform your instructor if you have completed the task or need assistance.

Hints

You'll need 6 nodes (4 metric, 1 combination, 1 aggregation) in order to produce a correct result. A graph is typically constructed in the following order:

- First, all metric nodes are defined
- Then, a combination node (here: *AND* operation) combines metrics
- Finally, an aggregation node (here: *AVG* operation) aggregates the result of a combination node to produce the final score

Figure 5.3: Hybrid Task: Participant provided task description.

- **Non-Technical Group** ($n = 7$): Participants with no professional background in data engineering or programming.

Regarding the usage of LLMs, the majority of participants are experienced users. 50% report the use of LLMs daily, while six participants use them several times a week. Only one participant reports rare usage.

5.4 User Study Results

The results of the study fall into three areas: task performance, the performance of the AI Assistant integration, and the subjective user perception collected via the surveys.

5.4.1 Task Performance

Time to completion and correctness of the final result serve as measurements for both the Manual Task and Hybrid Task. The quantitative data reveal that the manual approach

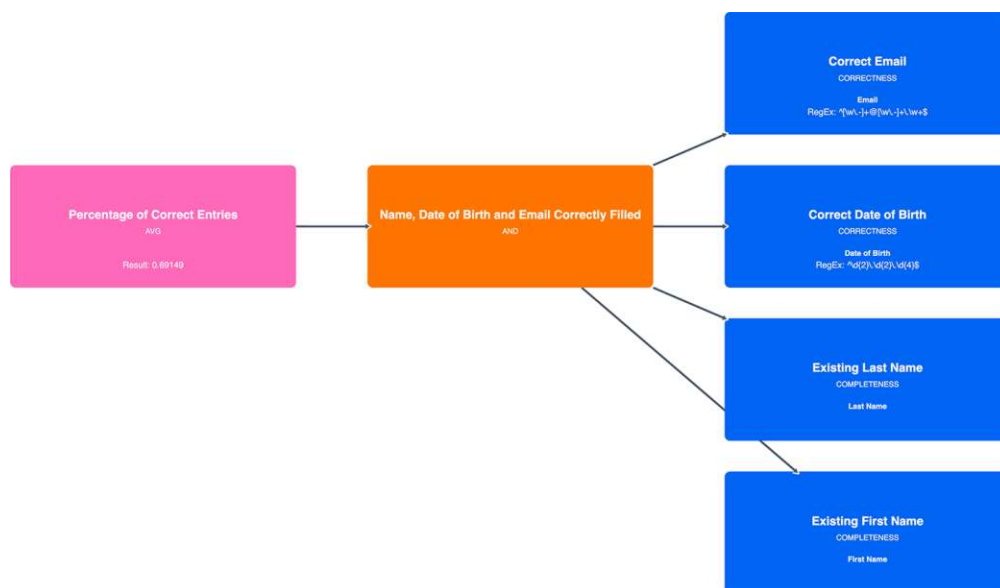


Figure 5.4: Hybrid Task: Target graph solution (6 Nodes).

(4 nodes) is, on average, faster and more accurate for this specific set of tasks than the hybrid approach (6 nodes).

In the Manual Task, participants achieve an average completion time of 04:58 minutes. Notably, 100% of participants construct the correct graph structure (4 nodes) and achieve the correct final result. In contrast, the Hybrid Task has an average completion time of 09:22 minutes. However, this increase is also attributable to the task’s greater complexity. Consequently, even after subtracting the processing time required by the AI Assistant, the net interaction time (08:02 minutes) reflects the additional workload of the larger structure rather than strictly slower performance. While the technical group maintains a high accuracy rate, the non-technical group shows errors in the Hybrid Task. Overall, 92.9% of participants construct the correct number of nodes (one participant creates a more complex, but correct graph with 7 nodes instead of 6), and final result correctness drops to 85.7% (two incorrect results). Table 5.3 summarises these results.

Observations during the study indicate that non-technical users spend significantly more time verifying the AI output and navigating the interface. On average, non-technical participants require over one minute longer for the Manual Task and almost two minutes longer for the Hybrid Task than technical participants.

5.4.2 AI Assistant Results

To evaluate the quality of the AI Assistant’s semantic alignment, we analyse the difference between the tasks generated by the LLM and the tasks reviewed by participants. We adopt an information retrieval approach, utilising Precision, Recall, F_1 Score, and Similarity based on set theory. Note that, similar to information retrieval theory, *Accuracy* is

Table 5.3: Average task performance of User Study participants.

	Technical ($n = 7$)	Non-Technical ($n = 7$)	All ($N = 14$)
Manual task			
Time	04:21 min	05:34 min	04:58 min
Solution Correctness	100%	100%	100%
Optimal Solution	100%	100%	100%
Hybrid task			
Time (incl. processing time)	08:37 min	10:06 min	09:22 min
Time (excl. processing time)	07:12 min	08:52 min	08:02 min
Solution Correctness	100%	71.4%	85.7%
Optimal Solution	100%	85.7%	92.9%
Total User Study Time	25:17 min	30:09 min	27:43 min

ill-suited for generative tasks. In a generative context, the set of True Negatives (valid tasks that the LLM correctly *does not* generate) is theoretically infinite, which renders standard Accuracy calculations meaningless [Man08]. Therefore, the following two sets of tasks are defined:

- O_{AI} (**AI Output**): The initial set of task suggestions generated by the AI Assistant based on the user’s prompt (Execution Plan).
- T_{Human} (**Human Tasks**): The final set of tasks that remains after the user reviews, edits, and filters the AI output (Human Reviewed Plan).

Classification Methodology The classification logic discriminates between three levels of matches for True Positives (TPs) to account for semantic differences:

- **TP1 (Exact Match)**: Strings are identical or differ only by insignificant white space or casing.
- **TP2 (Light Rephrasing)**: The logic and parameters are identical, but the wording differs (e.g., “Check Date Format” vs. “Validate Date”).
- **TP3 (Logic Change/Abstraction)**: The AI provides a correct direction, but the human generalises it (e.g., changing specific fields to “All Metric Nodes”) or adjusts the implementation details (e.g., changing an operator).
- **False Positive (FP)**: A task generated by the glsai that the human deletes entirely (indicating a AI hallucination or redundancy).
- **False Negative (FN)**: A new task from the human that the AI failed to generate.

To ensure consistent and scalable mapping between O_{AI} and T_{Human} , we employ an LLM acting as an automated evaluator with a “Lead QA Analyst” persona. The evaluator provides specific matching rules to classify the relationship between generated and finalised tasks. To ensure the validity of this automated evaluation, the resulting classifications are manually verified and corrected. The exact prompt used for this evaluation is shown in Listing 5.4.2.

```

1  # ROLE
2  You are a Lead QA Analyst for a Data Quality AI. Your job is to compare an "AI Created
   ↳ Pipeline" against a "Human Reviewed Pipeline" to grade the AI's performance.
3
4  # INPUT DATA
5  <created_tasks>
6  {{createdTasks}}
7  </created_tasks>
8
9  <reviewed_tasks>
10 {{reviewedTasks}}
11 </reviewed_tasks>
12
13 # CRITICAL MATCHING RULES
14 1. Filter by `nodeType` first: A "METRIC" node in the AI list can only match a
   ↳ "METRIC" node in the Human list. An "AGGREGATION" cannot match a "COMBINATION".
15 2. Ignore `order`: The integer order value will shift if the human deletes tasks.
   ↳ Do not use `order` to match tasks.
16 3. Redundancy is an FP: If the AI generates two similar checks (e.g., "Check Not
   ↳ Null" AND "Check Not Empty") and the human deletes the weaker one ("Check Not
   ↳ Null"), the deleted task is a False Positive (FP).
17
18 # CLASSIFICATION DEFINITIONS
19 ## TP1: Exact Match
20 * Strings are identical or differ only by insignificant whitespace/casing.
21 * Example: "Check if 'First Name' is not empty" vs "Check if 'First Name' is not
   ↳ empty".
22
23 ## TP2: Light Rephrasing
24 * Same logic, same parameters, but different wording or casing in values.
25 * Example: "Validate 'DoB' format using 'DD.MM.YYYY'" vs "Validate 'DoB' using
   ↳ dd.MM.yyyy regex pattern".
26
27 ## TP3: Logic Change / Abstraction / Implementation
28 * Abstraction: AI lists specific fields ("A, B, C"); Human generalizes ("All Metric
   ↳ Nodes").
29 * Implementation: AI describes the math ("divide valid by total"); Human uses a
   ↳ specific tool operator ("Use AVG operator").
30 * Constraint Change: Human changes the operator or threshold (e.g., changing AND to
   ↳ OR, or changing > 5 to > 10).
31
32 ## FP: False Positive (AI Error)
33 * The AI created a task that the human deleted entirely.
34 * Includes redundant tasks (e.g., "Not Null" checks that were removed in favor of "Not
   ↳ Empty").
35
36 ## FN: False Negative (AI Missed)
37 * The human added a new task that the AI did not generate at all.
38
39 # INSTRUCTIONS
40 1. Iterate through the `createdTasks` list.

```

```

41 2. Attempt to find the best semantic match in `reviewedTasks` (matching `nodeType` is
   ↪ mandatory).
42 3. Assign classification (TP1, TP2, TP3, FP).
43 4. Identify any items in `reviewedTasks` that were never matched (these are FN).
44 5. **Reasoning is mandatory:** Explain why you chose the label (e.g., "Human
   ↪ abstracted specific fields to 'all nodes'").
45
46 # OUTPUT JSON
47 Return a single JSON object with a list "evaluations".
48 {
49   "evaluations": [
50     {
51       "node_type": "METRIC",
52       "ai_task": "Check if 'First Name' is not null",
53       "human_task": null,
54       "classification": "FP",
55       "reasoning": "Human removed redundant 'not null' check in favor of 'not empty'"
56     },
57     ...
58   ]
59 }

```

Listing 1: System prompt used for automated task classification. Model: Gemini 3 Pro, Version as of January 9 2026

Metric Calculation To accurately reflect user effort, the effective TP count ($TP_{effective}$), differently for each strictness level, is calculated as follows:

- **Strict:** $TP_{effective} = TP1$ (Only exact matches count).
- **Lenient:** $TP_{effective} = TP1 + TP2$ (Exact matches and light rephrasing count).
- **Loose:** $TP_{effective} = TP1 + TP2 + TP3$ (All helpful suggestions count).

Any match type not included in $TP_{effective}$ for a given tier is treated as a mismatch (contributing to FP and FN). Using $TP_{effective}$, the final metrics are defined as follows:

Precision Precision is the fraction of suggested tasks that are correct under the chosen matching criterion [Man08]:

$$\text{Precision} = \frac{TP_{effective}}{TP_{effective} + FP} = \frac{TP_{effective}}{|O_{LLM}|} \quad (5.1)$$

In our setting, lower precision corresponds to more FPs suggestions, i.e., incorrect or irrelevant outputs. In LLM contexts, such unsupported or incorrect generations are discussed as AI hallucination [MNBM20].

Recall Recall is the fraction of tasks successfully identified by the AI Assistant under the chosen matching criterion [Man08]:

$$\text{Recall} = \frac{TP_{effective}}{TP_{effective} + FN} = \frac{TP_{effective}}{|T_{Human}|} \quad (5.2)$$

Lower recall indicates more missing tasks (FNs), implying more additional work is needed to reach a correct result.

F-Measure (F_1) The F_1 Score provides a balanced view between Precision and Recall, which is the harmonic mean of the two metrics. Unlike a simple arithmetic average, the harmonic mean penalises large imbalances between Precision and Recall [Man08]. This ensures that the score remains low if either Precision or Recall is poor.

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (5.3)$$

In the context of this study, a high F_1 Score indicates that the AI Assistant is effectively balanced, meaning it is not only accurate, but also comprehensive (few missing tasks).

Similarity (Jaccard Index) For Similarity, the Jaccard Index (JAC) is used, which quantifies the overall overlap between the AI’s generated set of tasks and the final human-verified set as the ratio of intersection to union [Jac12]. In other words, it calculates the valid tasks, $TP_{effective}$, divided by all unique tasks present in either the AI output or the user’s final list.

$$\text{Similarity} = \text{JAC} = \frac{|O_{AI} \cap T_{Human}|}{|O_{AI} \cup T_{Human}|} = \frac{TP_{effective}}{TP_{effective} + FP + FN} \quad (5.4)$$

Similarity provides a single statistic indicating how close the AI’s initial guess was to the final desired state.

Results and Analysis Table 5.4 summarises the performance results, averaged across all 14 participants.

Table 5.4: Average LLM performance metrics.

Metric	Precision	Recall	F_1 Score	Similarity
Strict	0.65	0.86	0.74	0.62
Lenient	0.68	0.90	0.76	0.65
Loose	0.73	0.99	0.83	0.73

The data reveals a trade-off in the AI’s performance profile. Most notably, the LLM achieves a near-perfect Recall of 0.99 in the loose category. This indicates that the Assistant successfully identifies all required steps for the DQ task, ensuring that users do not face a Blank Canvas problem or need to add nodes manually. However, the Precision of 0.73 suggests the presence of noise in the output. Roughly one in four suggestions generated by the AI is redundant or a result of AI hallucination, requiring user intervention to filter the results.

Similar to the Focus Group, the observations during the User Study reveal a significant portion of these errors to a specific, reproducible defect: the LLM consistently fails to utilise the native AVG operator. Instead, it creates AI hallucinations of non-working structures (e.g., combining SUM and LEN Aggregation Nodes) to manually calculate averages for some participants. This indicates that the current score reflects a known, fixable engineering limitation rather than a fundamental reasoning flaw. This suggests that targeted training can significantly mitigate such bugs in future iterations.

Besides the greater complexity of the Hybrid Task, this dynamic also explains the longer task completion times observed in Section 5.4.1. Users spend less time creating, but more time verifying and pruning. Despite this friction, the F_1 (0.74–0.83) and the Similarity metrics (0.62–0.73) confirm that the system offers substantial utility. We therefore characterise the system as a Drafting Engine: it generates a sufficiently accurate output which users can then verify and prune.

5.4.3 Qualitative Feedback and User Perception

Despite the objective data showing longer completion times for the Hybrid Task, user perception paints a different picture. Responses were collected using 5-point Likert scale questions, ranging from “Strongly Disagree” (1) to “Strongly Agree” (5) [Lik32], and generally favoured the AI integration:

- “The LLM-assisted approach allowed me to complete the task faster than the manual approach.”
(Average Score: **3.79**)
- “The LLM-assisted approach was easier to use than the manual approach.”
(Average Score: **4.14**)
- “The LLM-assisted approach required less mental effort.”
(Average Score: **4.14**)
- “I feel the LLM-assisted approach requires less technical expertise.”
(Average Score: **4.00**)

In terms of general usability, the AI Assistant is the preferred approach. The high ratings (≥ 4.00) for ease of use, reduced mental effort, and technical accessibility indicate that participants perceive the hybrid workflow better than the manual.

Interestingly, participants rated the AI-assisted approach as faster (Score: 3.79), appearing to contradict the measured time logs. However, a direct comparison is misleading, as the Hybrid Task has greater complexity. We conclude that users feel that the AI accelerates the process, regardless of the absolute time.

Preference Rationale Open-ended survey questions provide a deeper understanding of the quantitative results. Participants are asked to justify their preferred approach for solving the underlying tasks and to identify missing features. This qualitative data

reveals that the preference for the hybrid approach is driven by three primary factors: *Cognitive Load Reduction*, *Technical Abstraction*, and *Iterative Workflow*.

12 of 14 participants preferred the hybrid approach in the survey. The justifications heavily emphasise the reduction of mental effort. Participants describe the AI as a mechanism to bypass the Blank Canvas problem, stating it is “[...] *less work to drop my question into the AI assistant than having to think everything through*” and that they “[D]on’t need to come up with general structure [...]”. One participant explicitly notes the benefit of the system acting as a second opinion: “*It provides an extra step where you can reconsider your approach. Maybe the [LLM] has a better idea than the one you had in your mind*”.

A second major theme is the abstraction of technical syntax. Even for technical users, defining RegEx or specific operator syntax is seen as a tedious task best delegated to the AI. One participant remarks, “[...] *sometimes I don’t want to use my brain [...]*”, highlighting that the value of the AI Assistant lies in handling technical details rather than high-level logic.

Conversely, the two participants preferring the manual-only approach cited “Control” and “Overhead” as their primary reasons. They argue that for simpler tasks, the waiting time for the AI generates friction, noting that “*more control over the results [leads to] less checking*”.

Identified Limitations and Feature Requests Furthermore, participants provide criticism regarding the tool’s limitations. The most frequently cited usability issue is the inability to edit specific node properties (e.g., changing an aggregation operator) after creation. Users report that they “*had to recreate [nodes], which was a bit of extra effort*”, since some modifications are not possible within the current edit functionalities. Additionally, as already established in the Literature Analysis and Focus Group, users express a desire for a “Pre-defined Check Catalogue” (e.g., standard checks for email or phone numbers) to further speed up DQ assessments.

Regarding the AI integration, users request better traceability between the chat interface and the visual graph. Similar to the feedback of the Focus group (see Section 5.2.2), one user suggests a “*link between Chat and Nodes*” to better understand which tasks of the Human Reviewed Plan generated which node of the graph. Another user humorously suggests a “*mini-game to pass the time*” during LLM generation. This highlights that latency remains a friction point in Human–AI Interaction.

Table 5.5 summarises the aggregated feedback themes.

Observational Findings

Besides the quantitative metrics, direct observation of participants reveals distinct behavioural patterns that correlate strongly with technical background. These differences are particularly seen in prompting strategies and error recovery.

Table 5.5: Categorisation of qualitative participant feedback.

Category	Representative Participant Quotes & Insights
Cognitive Load	<p>“Less cognitive effort in terms of understanding the task in the first place.”</p> <p>“Feels like less work [...] than having to start from scratch.”</p>
Technical Barrier	<p>“Don’t need to come up with details like RegEx myself.”</p> <p>“My preference is still the LLM because it is faster (e.g., defining regex expressions).”</p>
Control (Manual)	<p>“More control over the results, less checking.”</p> <p>“For daily/easier use cases, the LLM could generate too much overhead.”</p>
Feature Requests	<p>Editability: “Editing Operator on Aggregation node wasn’t possible.”</p> <p>Traceability: “Link between Chat and Nodes.”</p> <p>Visualisation : “Tab to switch between raw data and filtered one.”</p>

Analysis of Prompting Strategies The two user groups instruct the AI Assistant in different ways. Technical participants use an efficiency-driven strategy, blindly relying on the model’s ability to infer context from the input. In four out of seven cases, these users simply copy-paste the entire task description or the specific question directly into the chat interface, rather than formulating a custom request. Conversely, non-technical participants use a more instructional prompting style. They guide the system with explicit constraints and instructions and also reformulate the underlying DQ task. For instance, one non-technical user constructs a prompt by combining the introduction, the specific question, and the hints. Others try to enforce strict output formats, such as explicitly requesting a specific number of nodes or describing the visual appearance of the desired graph.

This behaviour suggests that non-experts feel a greater need to explain the task logic to the AI in natural language, whereas experts are more comfortable treating the system as a context-aware agent.

Interaction and Error Patterns The types of errors and recovery strategies also differ between groups. As mentioned above, both groups encounter AI hallucination, such as the LLM generating redundant Aggregation Nodes or replacing the intended AVG function with combinations of LEN and SUM operations. However, technical users identify and correct these mismatches by recreating the affected Aggregation Node. In contrast, non-technical users struggle more with the logical translation of requirements. In the Manual Task, one participant misinterprets the natural language phrase “either

or both” as a complex Boolean combination of AND and OR gates, rather than a simple inclusive OR. Additionally, domain-specific constraints prove challenging. For example, one user defines an insufficient RegEx for email validation that allows invalid strings (e.g., “ch!uck”) to pass, while another fails to exclude null values from their metrics.

Verification Strategies Despite these challenges, effective verification strategies are observed across both groups. A notable example involves a non-technical participant who receives a plausibly formatted but logically incorrect result from the AI Assistant. Instead of accepting the Black Box output, the user identifies the result as numerically impossible and immediately backtracks to review the graph structure. This indicates that even without deep technical knowledge, users employ sanity checks to validate the AI-assisted workflow.

5.4.4 System Usability Scale Assessment

The SUS test quantitatively assesses the perceived usability of the prototype. First introduced by Brooke, it has become an industry standard in HCI research due to its robustness and reliability, even with smaller sample sizes (e.g., $N < 20$) [B⁺96, TS⁺04].

Methodology and Calculation The SUS consists of a 10-item questionnaire with alternating positive and negative statements (e.g., “I thought the system was easy to use” vs. “I found the system unnecessarily complex”). Participants rate each item on a 5-point Likert scale.

To calculate the final score, the contributions of each item are normalised to a range of 0 to 4:

- For odd-numbered items (positive sentiment), the contribution is calculated as $(\text{Score} - 1)$.
- For even-numbered items (negative sentiment), the contribution is calculated as $(5 - \text{Score})$.

The sum of these contributions is multiplied by 2.5 to convert the raw total into a standardised scale of 0 to 100:

$$SUS_{Score} = 2.5 \cdot \left(\sum_{n \in \text{odd}} (x_n - 1) + \sum_{n \in \text{even}} (5 - x_n) \right) \quad (5.5)$$

It is important to note that although the SUS produces a score from 0 to 100, it is not a percentage. Instead, the score is most useful for relative judgments when compared against SUS benchmarks (e.g., adjective mappings derived from large collections of SUS studies) [BKM08].

Results and Interpretation The prototype achieves an overall mean SUS score of 83.9 ($SD = 9.49$, $N = 14$). The individual scores range from a minimum of 67.5 to a maximum of 95.0. This indicates generally positive ratings with modest variability across participants. The estimated 95% confidence interval for the mean SUS score is approximately 78.4 to 89.4.

To interpret this result, the analysis references the adjective rating scale developed by Bangor et al. [BKM08], which maps numerical SUS scores to descriptive adjectives (see Table 5.6).

Table 5.6: Adjective rating scale for SUS scores.

SUS Score Range	Adjective Rating
85.59 – 100.00	Best imaginable
72.76 – 85.58	Excellent
52.02 – 72.75	Good
39.18 – 52.01	OK
25.01 – 39.17	Poor
00.00 – 25.00	Worst imaginable

Based on this scale, the prototype’s score of 83.9 falls into the *Excellent* category. This suggests that the visual abstractions through DAGs and natural language reduce the complexity associated with DQ assessment tasks.

A closer look at specific questionnaire items further supports this conclusion. Notably, the statement “*I thought the system was easy to use*” receives an average rating of 4.5 out of 5 across all participants. This high score on the core usability question reinforces the qualitative feedback that the learning curve was manageable, even for users encountering the concept of DAG-based quality rules for the first time.

When grouping the results by expertise, the technical group ($n = 7$) reports a mean SUS score of 86.43 ($SD = 7.89$), while the non-technical group ($n = 7$) reports a mean score of 81.43 ($SD = 10.88$). This corresponds to a 5.00-point higher average in the technical group. Given the small sample size and the observed variability, this difference should be interpreted as a modest trend rather than a robust group effect. Overall, both groups rate the prototype highly, suggesting the GUI is usable for non-technical users while still meeting technical users’ expectations.

5.4.5 Benefits of LLM Integration in DQ Tools

The objective efficiency and subjective preference, combined with the LLM performance profile, provide an answer to Research Question 3.

The study demonstrates that integrating LLMs via a multi-agent AI Assistant improves user interaction not by increasing execution speed, but by shifting the workflow from *imperative specification* to *declarative verification*.

This shift enhances the interaction in three key ways:

1. **Enhancing Interaction through High Recall:** The high Recall, ranging from 0.86 (Strict) to 0.99 (Loose), is critical for user trust. Because the LLM identifies the necessary components for a solution, it closes the gap between user intent and technical implementation. Non-experts no longer face the cognitive barrier of a Blank Canvas problem. As observed in the study, participants successfully use natural language to describe their intent (e.g., validating emails) rather than the implementation (e.g., creating RegEx patterns). They rely on the system’s high Recall to populate the technical graph structure.
2. **Shifting Cognitive Load via Precision Trade-offs:** The Precision scores, ranging from 0.65 to 0.73, require a workflow of pruning and filtering rather than pure creation. While this results in longer task completion times (as users must verify and delete AI hallucination), the overwhelmingly positive SUS score (83.9) and user preference (12 out of 14) indicate that users prefer this interaction mode. They are willing to trade efficiency for a reduction in the cognitive effort. The interaction becomes a review process, which users perceive as “easier” and “less mental effort”.
3. **Enabling Natural Language Construction:** The integration allows users to map natural language directly to executable logic. The observational data show that non-technical users utilise this to bypass technical barriers, formulating queries that combine context, constraints, and formatting instructions. The $F1$ Scores (0.74–0.83) confirm that, despite some noise, the system successfully interprets these prompts into valid DQ assessments, thereby democratising access to complex DQ tools.

In conclusion, the LLM integration enhances user interaction by lowering the barrier to entry for complex logic and transforming the user’s role from a creator of syntax to a reviewer of requirements. This benefit, however, relies heavily on a transparent Glass Box design that allows users to inspect and prune the generated output.

Conclusion

The reliability of modern data-driven systems depends not only on the existence of data but on its Fitness for Use. While the field of DQ has matured, this thesis identified a gap between the complexity of real-world domain requirements and the capabilities of tools. Standard solutions flatten quality into isolated results, failing to capture the hierarchical dependencies that define the fitness. Conversely, script-based frameworks offer the necessary expressiveness but exclude domain experts due to high technical barriers.

This thesis aimed to close this gap and therefore proposed, implemented, and evaluated a novel interaction approach that performs DQ assessment task through an interactive DAG, supported by a human-in-the-loop AI workflow. Consequently, the work delivered three primary contributions to the domain of HDI, HCI and Human–AI Interaction.

1. The thesis established a theoretical and practical critique of the Black Box problem in present DQ tooling. By analysing the state-of-the-art, the research highlighted that the integration of LLMs into data tools introduces verifiability issues. When systems generate opaque programming code, they force users to blindly trust the output. In response, this work defined the requirements for a Glass Box system in which the logic remains visually transparent and editable.
2. The core contribution was the design and implementation of a web-based prototype that renders DQ logic as a visual graph. Unlike traditional visual solutions that hide logic, this system exposes the combinations and aggregations of metrics. A key innovation within this GUI was the implementation of interactive data highlighting. By linking graph nodes directly to the underlying data table, the system provides visual feedback. This allows users to trace how specific constraints (e.g., a RegEx failure) propagate up the hierarchy to affect the final DQ score.
3. The system integrated a multi-agent AI Assistant designed specifically to abstract technical complexity. Rather than replacing human judgment, the AI functioned as

a Drafting Engine. It translates natural-language requirements into a visual graph structure, abstracting syntactic complexities such as RegEx patterns and Boolean logic. Crucially, this integration maintained the human-in-the-loop paradigm. The AI generates the graph structure, but only after the user reviews and submits the proposed assessment logic using a visual interface.

The evaluation of the prototype, conducted through an expert Focus Group and a User Study, provided empirical answers to the defined research questions. The findings underscored the roles that visual structure and AI assistance played in the assessment process.

Regarding the limitations of existing tools (Research Question 1), the analysis confirmed that current solutions hindered domain experts by forcing them to trade off power and verifiability. Tools that offered deep aggregation logic require engineering skills, while those accessible to non-experts lack the ability to model complex, nested relationships through combination and aggregation techniques. This creates a dependency loop where domain experts cannot independently translate their knowledge into executable rules.

The evaluation of the visual combination and aggregation techniques (Research Question 2) demonstrated that representing logic as a DAG benefited both technical and non-technical users through different mechanisms. For experts, the visual graph acts as maintainable and transparent logic, which allows for rapid debugging and use of complex rule sets. For non-experts, the visualisation provided an understandable layer of transparency and usability. The study showed that the combination of the logic graph and interactive data highlighting allowed users to understand *why* a DQ failed. Therefore, this enabled non-experts to examine the system's structure without reading code.

Finally, the investigation into LLM integration (Research Question 3) yielded a significant insight regarding efficiency and user experience. Participants preferred the AI-assisted workflow, awarding it an "Excellent" SUS score of 83.9. The evaluation revealed that the LLM's high Recall effectively solved the Blank Canvas problem. The F_1 Scores from 0.74 to 0.83 reveal that, despite some noise, the system was able to turn natural language into verifiable DQ assessments. This integration turns the user's role from *imperative specification* to *declarative verification*.

Users rated the AI-assisted approach as faster, more usable and less cognitively demanding. Both groups accepted the task of pruning AI hallucinations in exchange for not needing to use complex syntax. This confirmed that, in a transparent Glass Box environment, the primary value of AI lies not in increasing execution speed but in transforming the user's role from a code creator to a requirements reviewer.

6.1 Implications

The findings of this thesis extended beyond the specific implementation of the prototype to the broader design of intelligent data management tools. As AI becomes more present in software engineering, the research demonstrated that the primary design challenge shifted from automation to verification. Trust in AI-driven systems did not derive from

the assumption of perfect accuracy, but from the user’s ability to detect and correct errors.

This highlights the need for Verifiability by Design concepts. Interfaces must be constructed so that the cost of verifying an AI suggestion is lower than the cost of generating the solution manually. The study implies that users do not necessarily seek the fastest tool, but the one that minimises cognitive friction. By solving the Blank Canvas problem with a proposed Execution Plan, the AI removed the mental burden of starting DQ assessments from scratch. Consequently, future Human–AI systems should not strictly optimise for execution speed, but rather for the reduction of the Blank Canvas problem. We argue that high-recall drafts that invite user critique should be prioritised over high-precision constraints that might limit exploration.

The success of the Glass Box approach implies that data tools should prioritise intermediate representations that are understandable by humans. The interactive DAG proved to be a successful medium for this exchange. It was structured enough for the system to execute upon, yet intuitive enough for users to validate. This suggests a shift away from simple chat interfaces that generate code toward interfaces where AI manipulates visible, interactive elements. For domain experts, this democratisation of logic definition means they can take ownership of DQ rules. By removing the dependency on engineering support for every rule change, users can align DQ assessments more closely with dynamic domain requirements.

6.2 Future Work

While the presented prototype successfully validates the core interaction concepts, the transition from a research instrument to a production-ready state requires addressing limitations in scalability, functional expressiveness, and system scope.

A primary area for future investigation is visual scalability and GUI efficiency. Reflecting the feedback from the Focus Group, which identified page navigation friction, a comprehensive GUI overhaul is required to support more complex workflows. Future work should investigate the limits of the visual DAG representation at this scale. To address the experts’ concerns regarding scrolling and context switching, future iterations should transform the workspace into an infinite canvas with floating panels and a dedicated split-view mode for AI interaction. Additionally, to maintain readability, the interface requires collapsible sub-graphs that abstract lower-level logic into single blocks. This would ensure that visual exploration remains efficient for even more complex logic trees. Regarding semantic expressiveness, incorporating all capabilities of Schrott’s CobADQ library [Sch24] would enable advanced features such as weighted aggregation. Such features would allow users to define DQ scores in which critical fields have a greater impact on the result than optional ones. This could be complemented by a plug-in architecture that allows domain experts to import a catalogue of standardised checks (e.g., validated email or phone formats) rather than defining RegEx patterns from scratch. Such a catalogue was explicitly requested by both the Focus Group and User Study participants. Additionally, further graph interaction features, such as changing the

6. CONCLUSION

current left-to-right visualisation, would offer alternative analytical viewpoints. Finally, the insights from the LLM evaluation necessitate an evolution of the AI Assistant. Since the study identified the primary user workflow as “pruning” (high recall, lower precision), future designs should prioritise features that streamline the verification process. Visual confidence indicators (e.g., highlighting nodes where the model’s certainty is low) would allow users to direct their attention to potential AI hallucination. To further improve precision, the interaction model should move beyond relying on a single user prompt. Rather than guessing intent from ambiguous prompts, the AI should be capable of asking questions before generating an Execution Plan or creating the final graph. Lastly, the user corrections observed during the study represent a valuable data source. Future systems could leverage these verification actions (e.g., which nodes users adjusted) to fine-tune the underlying agents and align their outputs with domain-specific requirements.

Overview of Generative AI Tools Used

I acknowledge the use of generative AI tools in the creation of this work. These tools were used for tasks such as code generation, debugging purposes, summarisation, paraphrasing, drafting, and grammar checking throughout the thesis. They have been an important aid in my research and writing process, and their impact on the final product is hereby acknowledged. Nonetheless, all outputs were reviewed critically and, where appropriate, revised, and my creative influence predominates in the final work.

AI tools used:

- ChatGPT¹, Models: GPT-4o, GPT-5.1, GPT-5.2, Versions as of July 2024 – January 2026
- Gemini², Models: 2.5 Pro, 3 Pro, Versions as of July 2025 – January 2026
- Perplexity³, Model: Deep Research, Version as of January 2026
- Grammarly⁴, Free and Pro Plan, Versions as of December 2025 – January 2026
- GitHub Copilot⁵, Models: Various underlying LLMs, Versions as of July 2024 – December 2025

¹<https://platform.openai.com/docs/models/>

²<https://ai.google.dev/gemini-api/docs/models>

³<https://www.perplexity.ai/>

⁴<https://www.grammarly.com/ai>

⁵<https://docs.github.com/en/copilot/reference/ai-models/supported-models>



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

3.1	Dataset Inspection Table: Default view (no node selected).	14
3.2	Conceptual DAG for the address-quality running example, showing Metric (blue), Combination (orange), and Aggregation (pink) Nodes.	15
3.3	GUI representation of the address-quality assessment DAG.	19
3.4	Visualisation of incremental graph construction and layout adaptation. . .	20
3.5	Dataset Inspection Table: Viennese ZIP Metric Node anomalies.	21
3.6	Dataset Inspection Table: AND Combination Node (Valid Viennese Address)	22
3.7	Dataset Inspection Table: Columns affected by AVG Aggregation.	22
3.8	Edit Interface of the Node Detail Panel	23
3.9	Node Result Table for the Valid Address Combination Node	23
3.10	Graphical User Interface of the Detail View page (Main component of the prototype).	25
3.11	High-level architecture overview of the prototype (request path).	26
4.1	AI Assistant GUI: Intent definition phase.	32
4.2	AI Assistant GUI: Execution Plan with proposed DQ tasks.	33
4.3	AI Assistant GUI: Execution phase.	34
4.4	Final DAG generated by the AI Assistant.	35
4.5	AI Assistant GUI: Final response	35
4.6	Multi-agent architecture flow with human-in-the-loop verification	37
4.7	Specification of the Request Interpreter Agent.	38
4.8	Specification of the Rule Critical Agent.	38
4.9	Specification of the Requirements Agent.	40
4.10	Specification of the Task Planning Agent.	40
4.11	Specification of the Node Classification Agent.	41
4.12	Specification of the Metric Node Agent.	42
4.13	Specification of the Combination Node Agent.	43
4.14	Specification of the Aggregation Node Agent.	43
4.15	Specification of the Final Evaluation Agent.	44
5.1	Manual Task: Participant provided task description.	55
5.2	Manual Task: Target graph solution (4 Nodes).	56
5.3	Hybrid Task: Participant provided task description.	57
5.4	Hybrid Task: Target graph solution (6 Nodes).	58
		75



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

5.1	Comparison of functional and usability features across data quality approaches.	48
5.2	Categorisation of Focus Group feedback and implementation status. . . .	52
5.3	Average task performance of User Study participants.	59
5.4	Average LLM performance metrics.	62
5.5	Categorisation of qualitative participant feedback.	65
5.6	Adjective rating scale for SUS scores.	67



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Glossary

- Aggregation Node** Level of abstraction in the quality task structure, operating at the table level. It compresses the output vector of a child node into a single scalar value to provide a final answer to a data quality question 14, 15, 17–22, 25, 33, 39, 41, 50, 63, 65
- AI Assistant** A system component designed to improve data quality assessments. It utilises a multi-agent LLM architecture to translate natural language requests into data quality graphs via a Human-in-the-Loop workflow 3–5, 18, 24, 27, 28, 31–36, 49, 50, 53–59, 61–66, 68, 69, 72
- AI Hallucination** A phenomenon where a large language model generates output that is grammatically plausible and authoritative in tone but factually incorrect, nonsensical, or unfaithful to the provided context 33, 35, 39, 50, 51, 59, 61–63, 65, 68, 70, 72
- Black Box** A system or model where the internal logic is hidden from the user, forcing blind trust regarding the correctness of the output 2, 47, 49, 50, 53, 66, 69
- Blank Canvas Problem** A cognitive barrier that arises when users must create complex content from scratch without system support 4, 53, 62, 64, 68, 70, 71
- Combination Node** A structural element that operates at the row/column level by accepting other nodes as input. It combines results using logical, statistical, or arithmetic operators to create a new vector for further processing 14–19, 21, 22, 25, 31–33, 39–41
- Completeness** A data quality metric that evaluates the presence of values within a selected column by marking missing entries as errors and existing values as valid 5, 7, 15, 22, 41
- Contactability** The degree to which a record can be successfully used for contacting a person (digitally and/or offline) 1, 2, 37, 55
- Correctness** A data quality metric that assesses whether values adhere to a specified format by matching them against a regular expression pattern 5, 7, 16, 19, 22, 41

- Drafting Engine** An interaction model where an AI system generates an editable structure to solve the initial creation barrier 3, 4, 11, 50, 63, 70
- Execution Plan** A sequence of atomic data quality tasks generated by the AI Assistant based on a user's request, presented for verification before graph construction 32, 37, 39, 40, 56, 59, 71, 72
- Fitness for Use** The degree to which data is suitable for its intended purpose 1, 4, 5, 7–9, 47, 49, 53, 69
- Glass Box** A system design paradigm in which the internal logic and processing steps are transparent and inspectable, allowing users to understand and verify how an output was generated 4, 8, 12, 49, 50, 68–71
- Gulf of Evaluation** The gap between a system's output and a user's ability to interpret and evaluate that output 4, 10, 47, 48
- Human Reviewed Plan** The validated and potentially modified version of the Execution Plan that has been approved by the user, serving as the instruction set for the AI's creation agents 32, 33, 41, 56, 59, 64
- Human-in-the-Loop** A methodological paradigm in which human judgment is integrated into AI-assisted decision-making processes 4, 11, 31, 37, 50, 54, 56, 69, 70
- Hybrid Task** A higher-complexity experimental task of the User Study, requiring the use of an AI Assistant to generate a 6-node graph from underspecified instructions 55–58, 63
- Manual Task** A lower-complexity experimental task of the User Study requiring the manual construction of a 4-node graph without AI assistance 55–58, 65
- Metric Node** The fundamental building block of a larger quality task that operates at the value level. It evaluates data values against specific conditions and outputs a binary vector indicating pass or failure for each value 14–17, 19–21, 26, 31–33, 39–41, 50
- Sugiyama method** A layered layout approach for directed graphs that assigns nodes to ranks and arranges edges to improve readability 20, 27
- Validity** A data quality metric that verifies whether data values conform to a defined domain of allowed entries 5, 7, 16, 19, 22, 41, 51
- Verifiability by Design** A design principle ensuring that system outputs, particularly from AI components, are presented in a transparent structure that reduces the cognitive effort required for human verification 2, 11, 48, 71

Acronyms

- AI** Artificial Intelligence 1–5, 8–12, 31, 32, 45–47, 50, 53–56, 58, 59, 62–66, 69–72
- API** Application Programming Interface 26, 27, 29, 36, 38
- CLI** Command-Line Interface 9, 11, 47
- CSV** Comma-Separated Values 13, 14, 28, 29
- DAG** Directed Acyclic Graph 2–5, 11, 13, 14, 17–22, 24, 27, 31–34, 36, 37, 41, 47, 49, 50, 52, 53, 67, 69–71
- DQ** Data Quality 1–5, 7–13, 15, 17, 19, 22, 26, 27, 31–34, 41, 42, 45–47, 49, 51, 53, 54, 62, 64, 65, 67–71
- FN** False Negative 59, 61, 62
- FP** False Positive 59, 61
- GUI** Graphical User Interface 9, 13, 18–20, 22–24, 26–28, 31–33, 40, 46, 47, 50, 51, 53–55, 67, 69, 71
- HCI** Human-Computer Interaction 4, 66, 69
- HDI** Human-Data Interaction 4, 5, 47, 69
- LLM** Large Language Model 2, 5, 7, 10, 11, 28, 31, 34, 35, 40, 45, 47, 57–65, 67–70, 72
- RegEx** Regular Expression 1, 4, 16, 20, 25, 29, 33, 50, 53, 56, 64, 66, 68–71
- SQL** Structured Query Language 2, 11, 46, 47, 53
- SUS** System Usability Scale 56, 66–68, 70
- TP** True Positive 59, 61



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

- [AIMS25] Moamin Abughazala, Motunrayo Ibiyo, Henry Muccini, and Mohammad Sharaf. Quality by prompt: Llm-powered transformation of data quality requirements into great expectations. In *Euromicro Conference on Software Engineering and Advanced Applications*, pages 130–147. Springer, 2025.
- [B⁺96] John Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.
- [BBB⁺24] Xianchun Bao, Zian Bao, Bie Binbin, QingSong Duan, Wenfei Fan, Hui Lei, Daji Li, Wei Lin, Peng Liu, Zhicong Lv, et al. Rock: Cleaning data by embedding ml in logic rules. In *Companion of the 2024 International Conference on Management of Data*, pages 106–119, 2024.
- [BGK⁺18] Christian Bors, Theresia Gschwandtner, Simone Kriglstein, Silvia Miksch, and Margit Pohl. Visual interactive creation, customization, and analysis of data quality metrics. *Journal of Data and Information Quality (JDIQ)*, 10(1):1–26, 2018.
- [BKM08] Aaron Bangor, Philip T Kortum, and James T Miller. An empirical evaluation of the system usability scale. *Intl. Journal of Human–Computer Interaction*, 24(6):574–594, 2008.
- [BS06] Carlo Batini and Monica Scannapieca. *Data quality: concepts, methodologies and techniques*. Springer, 2006.
- [CBST23] Oscar Castro, Pierrick Bruneau, Jean-Sébastien Sottet, and Dario Torregrossa. Landscape of high-performance python to develop data science and machine learning applications. *ACM Computing Surveys*, 56(3):1–30, 2023.
- [CCS12] Hsinchun Chen, Roger HL Chiang, and Veda C Storey. Business intelligence and analytics: From big data to big impact. *MIS quarterly*, pages 1165–1188, 2012.
- [EW22] Lisa Ehrlinger and Wolfram Wöß. A survey of data quality measurement and monitoring tools. *Frontiers in big data*, 5:850611, 2022.

- [FHJ⁺24] Kasra Ferdowsi, Ruanqianqian Huang, Michael B James, Nadia Polikarpova, and Sorin Lerner. Validating ai-generated code with live programming. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, pages 1–8, 2024.
- [GKHH11] Philip J Guo, Sean Kandel, Joseph M Hellerstein, and Jeffrey Heer. Proactive wrangling: Mixed-initiative end-user programming of data transformation scripts. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 65–74, 2011.
- [Gre76] Anthony G Greenwald. Within-subjects designs: To use or not to use? *Psychological Bulletin*, 83(2):314, 1976.
- [GSA⁺24] Ken Gu, Ruoxi Shang, Tim Althoff, Chenglong Wang, and Steven M Drucker. How do analysts understand and verify ai-assisted data analyses? In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, pages 1–22, 2024.
- [Ham13] Kelli Ham. Openrefine (version 2.5). <http://openrefine.org>. free, open-source tool for cleaning and transforming data. *Journal of the Medical Library Association: JMLA*, 101(3):233, 2013.
- [HDBL17] Melanie Herschel, Ralf Diestelkämper, and Housseem Ben Lahmar. A survey on provenance: What for? what form? what from? *The VLDB Journal*, 26(6):881–906, 2017.
- [Hos24] Michael Hoschek. An interactive tool for data quality assertion. Bachelor’s thesis, TU Wien, Vienna, Austria, 2024.
- [Inf25] Informatica LLC. Artificial intelligence for the data-driven enterprise. https://www.informatica.com/content/dam/informatica-com/en/collateral/white-paper/artificial-intelligence-for-data-driven-disruption_white-paper_3328en.pdf, 2025. White paper. Accessed: 01-05-2026.
- [Jac12] Paul Jaccard. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2):37–50, 1912.
- [KAH23] Timon Sebastian Klann, Marcel Altendeitering, and Falk Howar. Towards a low-code tool for developing data quality rules. In *DATA*, pages 22–29, 2023.
- [Kob12] Stephen G. Kobourov. Spring embedders and force directed graph drawing algorithms. *CoRR*, abs/1201.3011, 2012.
- [KSL⁺22] Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive NLP. *arXiv preprint arXiv:2212.14024*, 2022.

- [KSM⁺24] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines. 2024.
- [LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
- [LCQ⁺20] Yuyu Luo, Chengliang Chai, Xuedi Qin, Nan Tang, and Guoliang Li. Visclean: Interactive cleaning for progressive visualization. *Proc. VLDB Endow.*, 13(12):2821–2824, 2020.
- [Lik32] Rensis Likert. A technique for the measurement of attitudes. *Archives of psychology*, 1932.
- [LLH⁺24] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.
- [Man08] Christopher D Manning. *Introduction to information retrieval*. Syngress Publishing,, 2008.
- [Mas23] Masschelein, Maarten. Introducing sodagpt. <https://soda.io/blog/introducing-sodagpt>, June 2023. Accessed: 01-24-2026.
- [MNBM20] Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. On faithfulness and factuality in abstractive summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1906–1919, 2020.
- [NFLR21] Arpit Narechania, Adam Fourney, Bongshin Lee, and Gonzalo Ramos. Diy: Assessing the correctness of natural language to sql systems. In *Proceedings of the 26th International Conference on Intelligent User Interfaces*, pages 597–607, 2021.
- [Nor99] Donald A Norman. Affordance, conventions, and design. *interactions*, 6(3):38–43, 1999.
- [NYCP25] Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, and Tomas Pfister. Ds-star: Data science agent via iterative planning and verification. *arXiv preprint arXiv:2509.21825*, 2025.
- [PEG25] Vasileios Papastergios, Lisa Ehrlinger, and Anastasios Gounaris. Unfolding data quality dimensions in practice: A survey. *ACM Journal of Data and Information Quality*, 2025.

- [RAAM25] Fernanda S Rossi, Meredith CB Adams, Gregory Aarons, and Mark P McGovern. From glitter to gold: recommendations for effective dashboards from design through sustainment. *Implementation Science*, 20(1):16, 2025.
- [RCIR17] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. Holoclean: Holistic data repairs with probabilistic inference. *Proceedings of the VLDB Endowment*, 10(11), 2017.
- [Sch24] Johannes Schrott. Constraint-based measurement and aggregation of data quality. Master’s thesis, Johannes Kepler University Linz, Linz, Austria, 2024.
- [Sch25] Alexander Schneider. *Template-Guided Rule Generation and Evaluation for Data Quality using Large Language Models*. PhD thesis, Technische Universität Wien, 2025.
- [SLS⁺18] Sebastian Schelter, Dustin Lange, Philipp Schmidt, Meltem Celikel, Felix Biessmann, and Andreas Grafberger. Automating large-scale data quality verification. *Proceedings of the VLDB Endowment*, 11(12):1781–1794, 2018.
- [SN93] Nancy Staggers and Anthony F. Norcio. Mental models: concepts for human-computer interaction research. *International Journal of Man-machine studies*, 38(4):587–605, 1993.
- [SPMM24] Flavia Serra, Veronika Peralta, Adriana Marotta, and Patrick Marcel. Use of context in data quality management: a systematic literature review. *ACM Journal of Data and Information Quality*, 16(3):1–41, 2024.
- [STT07] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiro Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 2007.
- [TKLZ24] Yuan Tian, Jonathan K Kummerfeld, Toby Jia-Jun Li, and Tianyi Zhang. Sqlucid: Grounding natural language database queries with interactive explanations. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, pages 1–20, 2024.
- [TS⁺04] Thomas S Tullis, Jacqueline N Stetson, et al. A comparison of questionnaires for assessing website usability. In *Usability professional association conference*, volume 1, pages 1–12. Minneapolis, USA, 2004.
- [TZW⁺24] Wei Tao, Yucheng Zhou, Yanlin Wang, Wenqiang Zhang, Hongyu Zhang, and Yu Cheng. Magis: Llm-based multi-agent framework for github issue resolution. *Advances in Neural Information Processing Systems*, 37:51963–51993, 2024.
- [VDA12] Wil Van Der Aalst. Process mining: Overview and opportunities. *ACM Transactions on Management Information Systems (TMIS)*, 3(2):1–17, 2012.

- [WGM⁺08] Lujin Wang, Joachim Giesen, Kevin T McDonnell, Peter Zolliker, and Klaus Mueller. Color design for illustrative visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1739–1754, 2008.
- [WS96] Richard Y Wang and Diane M Strong. Beyond accuracy: What data quality means to data consumers. *Journal of management information systems*, 12(4):5–33, 1996.
- [XCSL25] Shuyang Xie, Hailing Cai, Yaoqin Sun, and Xudong Lv. Llm-dqr: Large language model-based automated generation of data quality rules for electronic health records. *Journal of Biomedical Informatics*, page 104951, 2025.
- [XZX⁺24] Liwenhan Xie, Chengbo Zheng, Haijun Xia, Huamin Qu, and Chen Zhu-Tian. Waitgpt: Monitoring and steering conversational llm agent in data analysis with on-the-fly code visualization. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, pages 1–14, 2024.