



12.06.2026

Template-Based Report Generation from LLM-Extracted Data

Open-Source LLM-Based Information Extraction and
Report Generation from Organised Business Documents

Authors

Claudio Digion David Bürge Tareq Kattit

Supervisor

Prof. Dr. Thomas Bocek

thomas.bocek@ost.ch

Eastern Switzerland
University of Applied Sciences
Oberseestrasse 10
CH-8640 Rapperswil
Postfach 1475



in co-operation with

Aiger Group AG
<https://www.aiger.ch>

AI.GER

Bachelor thesis for the academic degree of Bachelor of Science

Programme	Computer Science
Submission date	12.06.2026

Abstract

Aiger Group AG, the industry partner of this thesis, provides a virtual data room to manage the lifecycle of sensitive corporate documents with high security, privacy, and integrity. However, verifying these documents for completeness and compliance during due diligence remains a highly manual, error-prone, and time-consuming process. Investors and management therefore require greater automation and standardisation when analysing and reporting on customer data. Large Language Models offer a promising approach to extract information from large volumes of unstructured text. However, the use of proprietary cloud-based models raises significant data privacy concerns. Furthermore, the risk of hallucinations poses a critical challenge; any practical solution must maximise extraction accuracy and offer source grounding for extracted information. This thesis addresses these challenges by developing a privacy-preserving system that automates the extraction of predefined information using LLMs and generates standardised reports through a templating engine.

The proposed solution is an end-to-end platform comprising a .NET backend, a Python extraction worker, and a SvelteKit frontend, allowing users to dynamically define, execute, and version extractions and reports. Scriban templates transform the extracted data into standardised, human-readable reports. To process heterogeneous business documents, the system includes a converter that transforms them into Markdown, a format suitable for LLM processing. Both LLM-based and dedicated conversion tools were evaluated to determine the most reliable conversion approach. The solution further exploits Aiger Vault's standardised directory structure to narrow the contextual scope of each extraction. The extraction pipeline operates on document-level chunks and uses either LangExtract or Instructor for extraction. LangExtract performs grounded span extraction, requiring each extracted value to map to a specific span of the source text, while Instructor performs structured extraction without that constraint, allowing the LLM to paraphrase or interpret the source. To preserve privacy, the pipeline relies exclusively on open-source models, selected through an evaluation of extraction quality tested on-premises.

During quantitative analysis, the Gemma-4 model containing 31 billion parameters was used to test both extraction types on a set of documents from our industry partner. Grounded span extraction via LangExtract achieved a 98.24% accuracy rate,

with rare failures caused by minor typographic string validation discrepancies rather than data fabrication. Instructor achieved a 95.88% accuracy rate, where isolated rejections by an LLM Judge, also powered by Gemma-4, resulted from the model occasionally missing small details or providing slightly too much context, and not from fabricated or incorrect information. The solution was further evaluated using an Executive Summary developed in cooperation with Aiger Group AG, covering a total of 89 data extractions. The majority of extractions were accurate, but some limitations remain. Each extraction is tied to a single year to constrain context size, which prevents cross-year reasoning, and output quality depends on the business documents being placed in the correct directories. The results demonstrate that open-source LLMs can support reliable, source-grounded information extraction in privacy-sensitive due diligence environments, which can reduce manual effort.

Keywords: LangExtract, Instructor, Grounded Span Extraction, Structured Extraction, Scriban, Open-Source Models, Gemma-4, Markdown Conversion, Templating Engine, LLM

Table of Contents

- LIST OF ABBREVIATIONS I
- LIST OF FIGURES II
- LIST OF TABLES II
- 1 INTRODUCTION 1
 - 1.1 Problem Definition 1
 - 1.2 Research Objectives 2
 - 1.3 LLM Framework Conditions 2
 - 1.4 Contributions 3
 - 1.5 Methodology & Tooling 4
 - 1.6 Thesis Structure 5
- 2 REQUIREMENTS 6
 - 2.1 Actors and Goals 6
 - 2.2 Use Case Diagram 6
 - 2.3 Use Cases 7
 - 2.4 Functional Requirements 9
 - 2.5 Non-Functional Requirements 12
- 3 ANALYSIS 14
 - 3.1 Document Conversion Target Format 14
 - 3.1.1 Rationale for Markdown 14
 - 3.1.2 Comparison with Alternative Formats 14
 - 3.2 OCR and File Conversion Tool Benchmarks 15

3.3 Extraction Methods	16
3.3.1 LangExtract	16
3.3.2 LangStruct	17
3.3.3 Instructor	17
3.3.4 GLiNER2	18
3.3.5 Chosen Solution: Hybrid Pipeline	19
3.4 LLM Security Considerations	19
3.4.1 Security Against Hallucination	20
3.4.2 Security Against User Manipulation	21
3.5 Cost Efficiency Optimisations	22
3.5.1 Multi-Value Schema Structuring	22
3.5.2 Context Caching	23
3.5.3 Multi-Model Routing	24
3.6 Confidence Scores and Extraction Validation	25
3.6.1 Self-Reported Confidence	25
3.6.2 Token-Level Log Probabilities	25
3.7 Accuracy degradation with context size	26
4 ARCHITECTURE & IMPLEMENTATION	27
4.1 Domain Model	27
4.2 C4 Model	28
4.2.1 Context Diagram	28
4.2.2 Container Diagram	29
4.2.3 Component Diagram	30
4.3 Backend Implementation	34
4.3.1 Extraction Definition	34
4.3.2 Report Definition	35
4.3.3 Versioning Model	35

4.3.4	Extraction Orchestration	35
4.3.5	Templating	37
4.4	Message Queue	37
4.5	Message Contract	37
4.5.1	Request Message	38
4.5.2	Response Message	39
4.6	Extractor Implementation	40
4.6.1	Resource Consumption Monitoring	40
4.6.1.1	<i>The Token Tracking Limitation in LangExtract</i>	40
4.6.1.2	<i>Interception via Dynamic Patching</i>	41
4.6.2	Pipeline	42
4.6.3	Document Retrieval	43
4.6.4	Chunking	43
4.6.5	Grounded Extraction with LangExtract	43
4.6.6	Generative Extraction with Instructor	43
4.6.7	Merging	44
4.6.8	Results	44
4.7	Deployment	45
4.8	Scalability	45
4.8.1	Message Queue	45
4.8.2	Extractor	45
4.8.3	Backend	46
4.8.4	Frontend	46
5	EVALUATION	47
5.1	Evaluation Concept	47
5.2	OCR Conversion Method Evaluation	48
5.2.1	Selected Tools and Models	48

5.2.2 Document Corpus	49
5.2.3 OCR Conversion Speed Evaluation	49
5.2.4 OCR Conversion Quality Evaluation	49
5.2.4.1 <i>Quality Criteria</i>	50
5.3 LLM Evaluation	56
5.3.1 Benchmark Design	56
5.3.1.1 <i>Document Corpus</i>	56
5.3.1.2 <i>Task Structure</i>	56
5.3.1.3 <i>Evaluation Criteria</i>	57
5.3.2 Evaluation Pipeline	58
5.3.2.1 <i>Generation Phase</i>	58
5.3.2.2 <i>Judge-Based Evaluation</i>	58
5.3.3 Evaluated Model Configurations	59
5.4 LLM Extraction Testing	60
5.4.1 Document Corpus	60
5.4.2 Extraction Types and Pass Criteria	60
5.4.2.1 <i>Grounded Span Extraction (LangExtract Section 3.3.1)</i>	60
5.4.2.2 <i>Interpreted/Paraphrased Extraction</i> <i>(Instructor Section 3.3.2)</i>	61
5.4.3 Reference Model Selection	61
5.4.4 Test Case Schema	61
5.4.5 Testing Criteria	62
5.4.5.1 <i>Consistency Metric</i>	63
5.5 Automated Testing	64
5.6 Test Results	65
5.6.1 OCR Evaluation	65
5.6.1.1 <i>OCR Conversion Speed Evaluation Results</i>	65
5.6.1.2 <i>OCR Conversion Quality Evaluation Results</i>	67
5.6.2 LLM Evaluation	68

5.6.2.1	<i>Quantitative Performance Summary</i>	68
5.6.2.2	<i>Category-Level Insights</i>	69
5.6.3	LLM Extraction Testing	70
5.6.3.1	<i>Quantitative Performance Summary</i>	70
5.6.3.2	<i>Analysis of Empirical Findings</i>	70
5.6.4	Functional Requirements	72
5.6.5	Automated Test Coverage and Success Rate	81
5.7	Proof of Concept Report: Executive Summary	82
5.7.1	Evaluation Objective	82
5.7.2	Procedure	83
5.7.3	Report Artefact	83
5.7.4	Template Logic	85
5.7.5	Extraction Coverage	85
5.7.6	Evaluation Result	85
5.7.7	Limitations	86
6	DISCUSSION	87
6.1	Extraction Pipeline Testing and Requirement Alignment	87
6.1.1	Functional Verification	87
6.1.2	Extraction Types: LangExtract versus Instructor	87
6.1.3	Test Limitations and Production Realism	88
6.2	Evaluation Insights	89
6.2.1	Upstream OCR Bottlenecks	89
6.2.2	Splitting Extractions Across Multiple Models	89
6.3	Scope and Systemic Limitations	90
6.3.1	Representativeness of the AIGER Dataset	90
6.3.2	Single Year Extraction Limitations	90
7	CONCLUSION	91

8	OUTLOOK	92
8.1	Handover and integration	92
8.2	Extraction and reporting features	92
8.3	Performance	93
9	REFERENCES	94

List of Abbreviations

CER	Character Error Rate
LLM	Large language model
OCR	Optical character recognition
RAG	Retrieval-augmented generation
RPC	Remote Procedure Call
TLS	Transport Layer Security
WER	Word Error Rate

List of Figures

Figure 2.1	Use case diagram	6
Figure 4.1	Domain model diagram	27
Figure 4.2	Context diagram	28
Figure 4.3	Container diagram	29
Figure 4.4	Component diagram	30
Figure 4.5	Report orchestration sequence	36
Figure 4.6	Extraction flowchart	42
Figure 5.1	Excerpt of the generated Executive Summary	84

List of Tables

Table 4.1	Extraction Definition Properties	34
Table 4.2	Report Definition Properties	35
Table 4.3	Extraction Request Message	38
Table 4.4	Extraction Response Message	39
Table 4.5	Merge strategy by extraction mode.	44
Table 4.6	Conditions for each result code	44
Table 5.1	OCR Conversion - Testing Criteria	50
Table 5.2	OCR Conversion - Text Accuracy Evaluation Criteria	51
Table 5.3	OCR Conversion - Layout Preservation Evaluation Criteria	52

Table 5.4	OCR Conversion - Table Extraction Evaluation Criteria	53
Table 5.5	OCR Conversion - Markdown Syntax Evaluation Criteria	55
Table 5.6	LLM Evaluation Criteria	57
Table 5.7	LLM Extraction Testing Quality Dimensions	62
Table 5.8	Automated Unit Testing	64
Table 5.9	OCR Conversion Methods Speed Evaluation Results	66
Table 5.10	OCR Conversion Quality Evaluation Results	67
Table 5.11	LLM Evaluation Results	68
Table 5.12	LLM Extraction Test Run Metrics Summary	70
Table 5.13	Automated unit and component test results	81

1 Introduction

The lifecycle of sensitive documents, especially regarding their creation and management, such as corporate governance or financial data, poses a significant burden on organisations of any size. **Aiger Group AG**, a Swiss intelligence platform provider, addresses this through their virtual dataroom solution, **Aiger Vault**. This platform serves as a digital foundation for companies to store sensitive data, such as shareholder agreements, financial recordings, and tax documentation, using double encryption and blockchain-based hashing for high-integrity authorisation. Their goal is to provide management, board directors, and investors with a secure “Golden Record” of corporate truth.

1.1 Problem Definition

Maintaining an appropriate level of documentation quality is essential for every business. Aiger Group AG offers a solution, Aiger Vault, for managing sensitive corporate documents and supports the verification of these documents for completeness and compliance, a process often associated with due diligence. This verification can be highly manual, error-prone, and resource-intensive, as it requires extracting a wide range of information from numerous documents that may be distributed across complex folder structures. Investors and management, therefore, require more automation and standardisation to reduce the effort spent searching for information and to free up resources for actual analysis.

The recent rise of Large Language Model technology has made it possible to analyse large volumes of text at comparatively low cost. However, relying on cloud-based models operated by major providers raises significant privacy concerns, especially when handling sensitive customer data that must not leave controlled environments.

Furthermore, the risk of LLM hallucinations presents a critical challenge. Any viable system must therefore support high-fidelity, meaningful responses, provide source grounding, and explicitly indicate when the available data is insufficient instead of generating incorrect or unsupported information.

1.2 Research Objectives

The primary objective of this thesis is to evaluate the feasibility of a reporting system designed for a secure document vault environment. This is achieved by implementing a prototype and producing a proof-of-concept report developed with support from Aiger Group AG. The system shall process PDF, DOC, DOCX, XLSX, and text-based files and extract predefined information by leveraging the capabilities of Large Language Models.

Specific goals include:

1. **Data Extraction:** Develop a data extraction system that enables users to extract predefined information from documents by leveraging LLM technology.
2. **Reporting:** Develop a reporting system that generates reports based on template documents and extracted information.
3. **Extraction Accuracy:** Research and implement strategies to improve the accuracy and correctness of LLM-based data extraction.
4. **Model Evaluation:** Benchmark different open-source LLMs regarding their functional and non-functional suitability for the proposed solution.

1.3 LLM Framework Conditions

The solution must securely handle sensitive input data. To ensure this, the following restrictions regarding LLM use must be enforced:

Any LLM model used must be free and open-source, but not necessarily open-weight.

→ Only models that meet this criterion are considered for benchmarking and later use.

Only trusted LLM providers may be used.

→ During development, two separate and privately hosted LLM providers were used. Both were provided through the Eastern Switzerland University of Applied Sciences, the institution hosting this thesis.

1.4 Contributions

This thesis contributes a prototype that combines Large Language Models, a templating engine, and a web-based GUI to support scalable information extraction and report generation in a secure document vault environment.

The solution enables users to create, refine, and manage a large number of data extraction definitions and their corresponding results. Instead of extracting broad textual summaries, the system focuses on concise and granular information units. This granularity supports more targeted extraction, as each extraction can be performed with a narrower and more relevant context. At the same time, the management of extraction definitions and results helps keep the approach scalable, even when the number of required extractions increases.

In addition to data extraction, the thesis provides a reporting system based on Scriban as the templating engine. Extracted information is made available during report generation, allowing structured reports to be generated dynamically from templates. To demonstrate the practical applicability of this approach, an executive summary was selected as a realistic proof-of-concept report and generated using Markdown and HTML as output formats. However, the representation format is exchangeable and could be replaced by any alternatives, i.e., Typst, YAML, or CSV, provided that the corresponding implementation is available.

The main contribution of this thesis is therefore the design and implementation of a flexible system that simplifies LLM-based information extraction and combines it with template-driven report generation. It demonstrates how extracted information from sensitive business documents can be transformed into structured, reusable report content while maintaining flexibility in both extraction design and output representation.

1.5 Methodology & Tooling

General

Project management methodology	Rational Unified Process
Development	iterative, agile, pragmatic
Internet search tools	common search engines public LLM providers
Research and Review AI Assistant	ChatGPT, Gemini, Claude
Table Viewer	Microsoft Excel
Communication and Collaboration	Microsoft Teams, Whatsapp, Zoom, Element (Matrix), Outlook

Programming

IDE for product code	Visual Studio Code, Visual Studio
DevOps	Gitlab, Docker
AI Coding Assistant	ChatGPT, Gemini, Claude

Documentation

IDE for documentation code	Typst App
AI Documentation Assistant	ChatGPT, Gemini, Claude
Text editing	Sublime
Visualisation	Mermaid.js, draw.io, Structurizr DSL, Paint.net, Typst

1.6 Thesis Structure

The thesis begins by introducing the problem context, objectives, and methodological approach. It then defines the relevant stakeholders, use cases, and requirements that guided the development of the system.

The following sections analyse key design decisions, including document conversion, extraction methods, security considerations, and validation approaches. Based on this analysis, the architecture and implementation of the developed solution are presented.

The solution is then evaluated through conversion benchmarks, LLM extraction tests, automated tests, and a proof-of-concept executive summary developed with Aiger Group AG. The thesis closes with a discussion of the results, limitations, conclusion, and outlook for future work.

2 Requirements

2.1 Actors and Goals

Aiger Advisor Achieve high quality, accuracy, efficiency and intelligence in gaining and providing insight into each customers data.

Aiger Engineer Efficiently manage extractions and reports.

2.2 Use Case Diagram

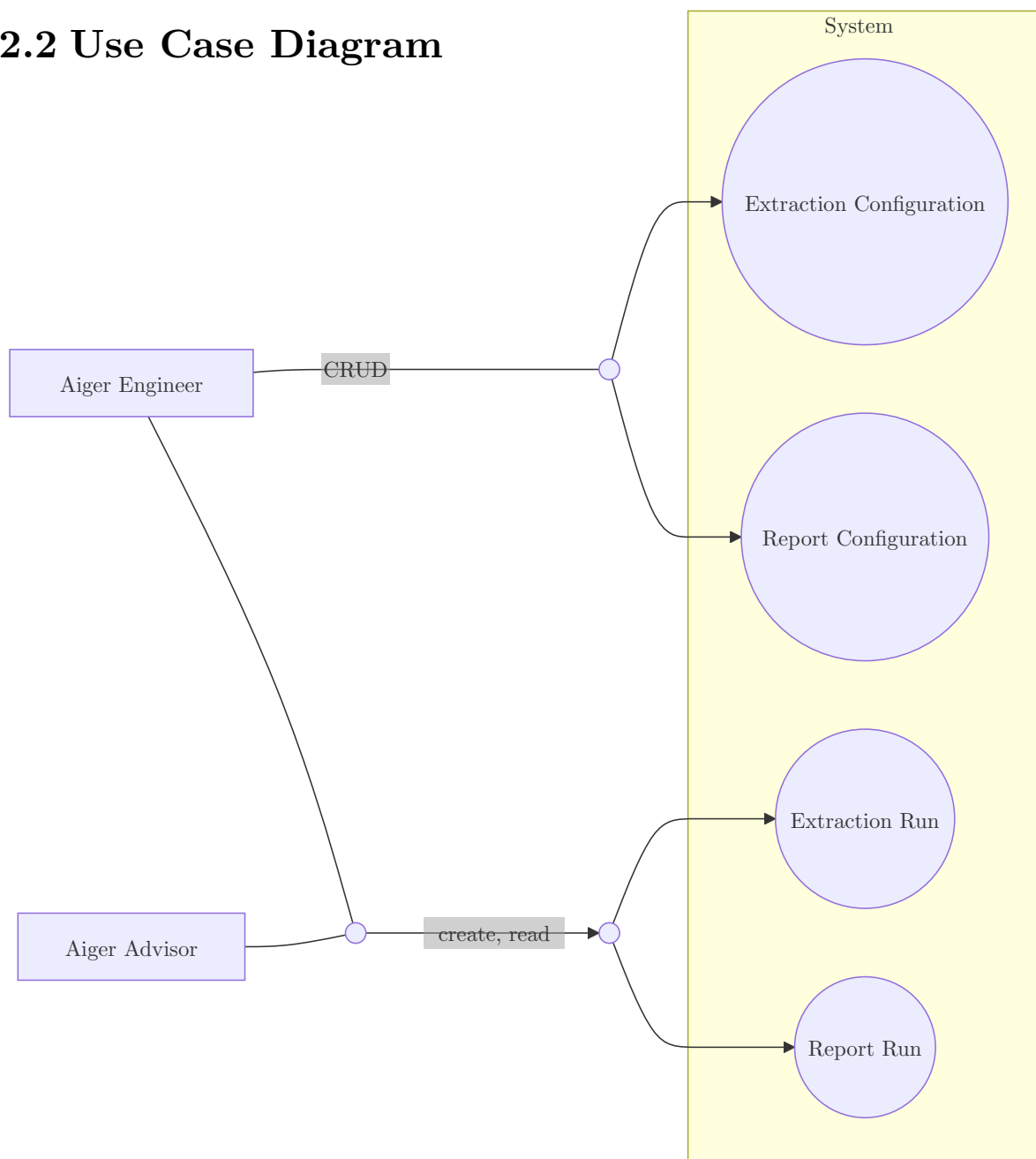


Figure 2.1: Use case diagram

2.3 Use Cases

UC1a - Reports Overview

As an AIGER Engineer, I want to get an overview over existing reports.

- There is a "report GUI section" which gives an overview over existing reports.

UC1b - Create Report

As an AIGER Engineer, I want to create a new report.

- Through the report GUI section a new report can be created.

UC1c - Search Report

As an AIGER Engineer, I want to find a specific report.

- Through the report GUI section a specific report configuration can be searched by name.

UC1d - Access Report

As an AIGER Engineer, I want to access more information on a specific report.

- Through the report GUI section a specific report configuration can be accessed.

UC1e - Delete Report

As an AIGER Engineer, I want to access more information on a specific report.

- Through the report GUI section a specific report configuration can be deleted.

UC2a - Extractions Overview

As an AIGER Engineer, I want to get an overview of existing extractions.

- There is an "extraction GUI section" which gives an overview of existing extractions.

UC2b - Create Extraction

As an AIGER Engineer, I want to create a new extraction.

- Through the extractions GUI section a new report can be created.

UC2c - Search Extraction

As an AIGER Engineer, I want to get find a specific extraction.

- Through the extractions GUI section a specific report configuration can be searched by name.

UC2d - Access Extraction

As an AIGER Engineer, I want to access more information on a specific extraction.

- Through the extractions GUI section a specific report configuration can be accessed.

UC2e - Delete Extraction

As an AIGER Engineer, I want to access more information on a specific extraction.

- Through the extractions GUI section a specific report configuration can be deleted.

UC3a - Define and run extractions

As an AIGER Engineer, I want to define and run an extraction using extracted information.

- Through the extractions GUI section I can create multiple extractions for different realistic possible use cases.
- I can configure these extractions for the desired outcome.
- I can run these extractions for multiple years and review their results

UC3b - Create a template

As an AIGER Engineer, I want to define a report using extracted information.

- Through the template editor I can create a Scriban template.
- I can import predefined extractions and use them in the Scriban template
- I can create a simple showcase document using the extractions and markdown-, HTML- as well as Scriban-syntax
- I can configure the years accordingly for the extraction variables.

UC3c - Run a report

As an AIGER Engineer or Advisor, I want to run a report and view its results.

- Through the report GUI page I can start a report for a specified year.
- The report result will be made available with its metadata in the respective result page.
- I am able to view the report document content including source information for extractions.

2.4 Functional Requirements

FR1.1 - Report Configuration Overview - Create report configuration

The user shall be able to create a report configuration.

FR1.2 - Report Configuration Overview - Search report configurations

The user shall be able to search for report configurations.

FR1.3 - Report Configuration Overview - View report configuration

The user shall be able to access and view a report configuration.

FR1.4 - Report Configuration Overview - Delete report configuration

The user shall be able to delete a report configuration.

FR2.1 - Extraction Configuration Overview - Create extraction configuration

The user shall be able to create an extraction configuration.

FR2.2 - Extraction Configuration Overview - Search extraction configurations

The user shall be able to search for extraction configurations.

FR2.3 - Extraction Configuration Overview - View extraction configuration

The user shall be able to access and view an extraction configuration.

FR2.4 - Extraction Configuration Overview - Delete extraction configuration

The user shall be able to delete an extraction configuration.

FR3.1 - Report Template - Report versioning

Report configurations shall be versioned, and each version shall be retrievable.

FR3.2 - Report Template - Template editing

The system shall provide a template editor with syntax highlighting for the templating engine syntax.

FR3.3 - Report Template - Draft management

The user shall be able to draft report configurations before releasing a new version.

FR3.4 - Report Template - Import variables

The user shall be able to specify which extractions he wants to use in the report template.

FR3.6 - Report Template - Extraction offset

The user shall be able to tell which year relative from the target year of the report the extraction should be from.

FR3.7 - Report Result - Report execution

The user shall be able to execute versioned report configurations for a target year.

FR3.8 - Report Result - Report Draft execution

The user shall be able to execute report configuration drafts.

FR3.9 - Report Result - Report execution model selection

The system shall allow the user to select an LLM for report execution.

FR3.10 - Report Result - Report result presentation

The system shall present report results together with suitable metadata.

FR3.11 - Report Result - Report execution status

The system shall inform about the current status of a report execution.

FR4.1 - Extraction Configuration - Extraction configuration editing

The user shall be able to modify all relevant attributes of an extraction configuration.

FR4.2 - Extraction Result - Source grounding

The system shall indicate the source from which extracted information was obtained.

FR4.3 - Extraction Result - Schema validation

The system shall validate extraction results against the configured schema.

FR4.4 - Extraction Configuration - Natural language input

Extraction configurations shall support natural language input.

FR4.5 - Extraction Configuration - Schema-based configuration

Extraction configurations shall be configurable using a schema.

FR4.6 - Extraction Configuration - Generative output

Extractions shall support LLM-generated output.

FR4.7 - Extraction Configuration - Source path restriction

Extractions shall support restricting input data based on defined source paths.

FR4.8 - Extraction Result - Target year specification

Extractions shall be executable for a specified target year.

FR4.9 - Extraction Configuration - Few-shot examples

Extraction configurations shall allow the user to add examples to the prompt.

FR4.10 - Extraction Result - Extraction result presentation

The system shall present extraction results together with suitable metadata.

FR4.11 - Extraction Configuration - Extraction versioning

Extraction configurations shall be versioned, and each version shall be retrievable.

FR4.12 - Extraction Configuration - Draft management

The user shall be able to draft extraction configurations before releasing a new version.

FR4.13 - Extraction Result - Draft execution

The user shall be able to execute extraction configuration drafts.

FR4.14 - Extraction Result - Extraction execution model selection

The system shall allow the user to select an LLM for extraction execution.

FR4.15 - Extraction Result - Extraction execution status

The system shall inform about the current status of an extraction execution.

FR5.1 - Document Processing - Supported file types

The converter shall be able to convert DOCX, PDF, XLSX, and text-based files to Markdown.

2.5 Non-Functional Requirements

NFR1.1 - Extraction Accuracy - LLM Extraction Quality

Category	Accuracy
Description	The LLM extraction must meet a minimum accuracy threshold when evaluated against expected outputs.
Goal	Minimal: 90% Target: 95% Optimal: 98%
Verification Method	Quantitative evaluation benchmark comparing extracted values against a predefined ground truth dataset.
Stakeholders & interests	high

NFR1.2 - Extraction Consistency - LLM Output Stability

Category	Reliability
Description	The LLM extraction pipeline must produce consistent results when run multiple times on the same input.
Goal	Minimal: 85% Target: 90% Optimal: 95%
Verification Method	Run each test case 5 consecutive times and compute the global consistency metric, defined as the sum of the most frequent identical output per test case divided by the total number of runs.
Stakeholders & interests	high

NFR2.1 - Test Coverage - Unit & Component Tests

Category	Reliability
Description	The system must maintain sufficient unit and component test coverage across all core modules.
Goal	Minimal: 60% Target: 75% Optimal: 85%
Verification Method	Code coverage report generated by the test runner across all unit and component tests.
Stakeholders & interests	medium

3 Analysis

3.1 Document Conversion Target Format

Before selecting specific OCR and document conversion tools, a common target format for converted documents had to be defined. Markdown was selected because it provides a practical balance between preserving document structure and maintaining token efficiency.

3.1.1 Rationale for Markdown

- **Structure without Excessive Overhead**

Markdown preserves basic document structures such as headings, lists, and tables using lightweight syntax. This helps prevent converted documents from becoming unstructured text blocks while avoiding unnecessary formatting overhead.

- **Token Efficiency**

Compared to markup languages such as HTML or XML, Markdown requires fewer syntactic characters. This reduces token consumption and allows more relevant document content to fit into the available context window.

- **Model Familiarity**

LLMs are commonly exposed to Markdown-formatted content, particularly through documentation, source code repositories, and web-based text. As a result, Markdown provides a format that LLMs can process consistently, including common structures such as lists and tables.

3.1.2 Comparison with Alternative Formats

When deciding on the target format, several alternatives were considered but rejected as the primary universal representation format:

- **Plain Text (.txt):** Plain text is highly lightweight, but it removes most structural information. Losing table structures, headings, and list hierarchies can make it more difficult for the LLM to identify relationships between values, especially in financial or semi-structured documents.

- **CSV (.csv):** CSV is well-suited for tabular data, particularly financial spreadsheets. However, due diligence folders also contain legal contracts, narrative descriptions, and mixed document layouts. Therefore, CSV is not suitable as a universal representation format for all supported document types.
- **HTML / XML:** HTML and XML can preserve structure in great detail, but they introduce substantial syntactic overhead. Tags such as `<table>`, `<tr>`, and `<td>` consume context window space that could otherwise be used for actual document content.

By standardising converted documents into Markdown, the pipeline can transform diverse file types such as PDFs, Word documents, Excel files, and text-based files into a uniform representation that can be processed consistently by the LLM.

3.2 OCR and File Conversion Tool Benchmarks

Choosing the right tools for OCR presents one with the difficulty of choosing from a vast number of options. As OCR benchmarking is a complex issue in itself, with multiple papers already written around the topic, it is not the focus of this issue. Therefore, a pragmatic approach was used to narrow down the number of solutions of interest to a reasonable number. On the one hand, at least one model from each big provider was chosen, and on the other hand, the results of a few different benchmarks were included. A compatible open-source licence and model sizes were also eliminating factors, as this thesis searches for cost-efficient solutions.

The following benchmarks were considered:

- OmniDocBench [1]
- OCRBench v2 [2]
- Datalab Benchmark [3]
- olmOCR-Bench [4]
- OCR Arena [5]

3.3 Extraction Methods

To enable automatic processing of raw documents into structured outputs and reports, different frameworks were evaluated. Their purpose is to read unstructured text, extract relevant values, and support the interpretation of the content so that relevant information can be identified and described in a structured form for report generation.

3.3.1 LangExtract

LangExtract [6] is a prompt-based extraction framework built on large language models. Its core idea is to treat extraction as a constrained generation problem: the model is instructed to copy information directly from the input text according to a predefined schema.

In practice, this means the system builds a structured prompt that tells the model exactly what fields to extract and how to format them. The important part is that it explicitly restricts the model to selecting spans that exist in the source text. After generation, the framework performs an alignment step that links each extracted value back to its original position in the document.

This design makes LangExtract very strong at one-to-one extraction tasks. If a value is explicitly present in the text, such as a name, date, identifier, or measurement, it can be extracted reliably and traced back to its origin. This traceability is one of its main strengths, because every output can be verified against the source.

The trade-off is that this strict grounding comes at the cost of higher-level understanding. Since the model is pushed toward copying rather than interpreting, it performs poorly on tasks that require reasoning across sentences or summarising content. It does not try to infer meaning beyond what is explicitly stated, which makes it unsuitable for analytical or summarisation tasks.

3.3.2 LangStruct

LangStruct [7] takes a different approach by putting the schema at the centre of the system. Instead of manually designing prompts, the user defines a structured output schema, and the framework generates the necessary prompts automatically.

Internally, LangStruct constructs prompts from the schema and uses them to guide the language model. To achieve high structural reliability, it integrates with framework optimisers like DSPy [8] to automate prompt engineering.

Similar to LangExtract, LangStruct requires the language model to provide a source span for each property within the schema. This means that while an extracted entity can be spread out across different parts of a source text, every single property must correspond to a specific, verifiable span.

However, LangStruct introduces significant operational overhead regarding performance and cost. Because the framework relies on automatic prompt optimisation, iterative refinement loops, and “Best-of-N” candidate generation strategies, it operates via a multi-pass pipeline. This means processing a single document often requires executing a series of consecutive LLM calls to generate, validate, and judge candidate responses. Consequently, this multi-step architecture makes the extraction process substantially slower (frequently scaling execution time by more than 5x) and exponentially increases API token consumption, making it an expensive choice for high-volume pipelines.

3.3.3 Instructor

Instructor [9] is a structured extraction library that builds a type-safe bridge between Large Language Models and Python data structures by utilising Pydantic [10], a Python library for data validation and parsing. Instead of relying on manual prompt formatting or internal template generation, Instructor leverages the native tool-calling and function-calling interfaces of modern LLMs to enforce structural constraints.

- **Automatic Validation and Self-Correction**

Because it natively integrates with Pydantic, Instructor can validate the model’s output the moment it is generated. If the LLM misses a required field or returns an incorrect data type, the framework can automatically pass the validation error back to the model, instructing it to self-correct and re-generate the output dynamically.

- **Dynamic Schema Generation**

Instructor is highly adaptable to runtime changes. Using Pydantic’s dynamic model creation tools, schemas can be constructed and modified on the fly based on incoming user requests, allowing the system to handle varied document structures without code redeployment.

- **Interpretative Flexibility**

The model is not forced to rigidly copy-paste exact text fragments. It has the freedom to synthesise, summarise, and transform information across multi-sentence contexts to fit the requested schema fields.

However, Instructor has clear limitations regarding precision and grounding. Because it relies entirely on the underlying language model to populate a Pydantic schema based on prompt instructions, it lacks native token-level alignment and source tracing.

3.3.4 GLiNER2

GLiNER2 [11] is a lightweight named entity recognition model designed for efficient local inference. Unlike LangExtract or LangStruct, it does not rely on prompting a language model. Instead, it uses a transformer encoder to process the input text and directly classify tokens into entity labels.

The process is straightforward: the text is tokenised, contextual embeddings are computed for each token, and a classification layer assigns entity types. Tokens belonging to the same entity are then grouped into spans.

A key advantage of GLiNER2 is efficiency. It runs locally, requires no external API, and is significantly faster and cheaper to execute than LLM-based approaches. It is therefore attractive from a performance perspective.

However, our tests showed that its extraction quality was clearly poorer than LangExtract. It often failed to identify even simple entities like the company name reliably and struggled with ambiguous or context-dependent cases. This limitation comes from the fact that it relies on learned token patterns rather than explicit reasoning or instruction following.

Because of this weakness, GLiNER2 is ultimately not suitable for the final pipeline, despite its strong performance characteristics. Therefore, it did not meet the required level of extraction reliability needed for the use case.

3.3.5 Chosen Solution: Hybrid Pipeline

Based on the evaluation of the individual frameworks, the final architecture implements a hybrid pipeline combining both LangExtract and Instructor. Rather than relying on a single tool, this approach pairs the strengths of both frameworks to handle different stages of the document processing workflow.

LangExtract is deployed for strict, verifiable data extraction. It is used exclusively for single-value fields where data integrity is paramount, such as extracting dates, identifiers, numbers, and names. Because LangExtract enforces verbatim token-level alignment, these critical data points can be reliably traced back to their exact coordinates in the source text, ensuring strict audit trails.

Instructor is utilised subsequently to handle analytical and synthesis-driven tasks that LangExtract struggles with. It leverages its interpretative flexibility to summarise lengthy sections, cross-reference data across paragraphs, and populate complex fields requiring contextual reasoning.

By eliminating LangStruct due to its severe performance overhead and rejecting GLiNER2 due to poor extraction quality, this hybrid configuration delivers the optimal balance for the production pipeline: absolute grounding precision where compliance matters and flexible linguistic processing where interpretation is required.

3.4 LLM Security Considerations

Since the system processes unstructured user-provided documents using large language models, two main security aspects must be considered: incorrect model outputs caused by hallucination and deliberate manipulation of either prompts or document contents.

A complete elimination of these risks is currently not possible, as modern LLM systems fundamentally cannot fully distinguish between instructions and data within the context window. Current research, therefore, focuses on layered mitigation strategies rather than absolute prevention [12], [13].

3.4.1 Security Against Hallucination

One important mitigation strategy is grounding generated outputs in the source material. Instead of allowing unrestricted report generation, extracted information should remain traceable to the original document content. Frameworks such as LangExtract already support this through span-grounded extraction, where extracted values can be linked back to their source text. This reduces the risk of fabricated information because generated statements can be verified against the original document.

Additionally, the implemented extraction pipeline follows a two-stage architecture. In the first stage, relevant values are extracted from the documents. In the second stage, these extracted values are used to generate reports. This separation reduces the amount of unverified raw text, which reduces the amount of possible hallucinations.

A further mitigation is reducing generative freedom through schema-guided extraction. Both LangExtract and LangStruct take a predefined schema and pass it to the model as part of the prompt. The returned output is then parsed and validated against that schema. This reduces the likelihood of hallucinated values and improves output consistency.

Potential future improvements include introducing confidence measurements or validation layers that compare generated outputs against extracted evidence and detect inconsistencies between generated conclusions and extracted values.

3.4.2 Security Against User Manipulation

Besides model hallucination, deliberate manipulation of uploaded documents must also be considered. A user could attempt to influence generated reports either by modifying document contents or by embedding hidden instructions intended to manipulate the language model.

In practice, however, this risk is considered comparatively low within our project context. According to Aiger Group AG, customer integrity is assumed, and uploaded documents are already protected through versioning and audit trails. This makes document modifications traceable and reduces the likelihood of unnoticed tampering. Therefore, this thesis assumes that security mitigations regarding malicious user manipulations are out of scope for this solution.

Nevertheless, several technical mitigations could reduce the attack surface further:

Separated Instructions

One important measure is separating instructions from document content. Uploaded documents should always be treated as unverified data rather than executable instructions. Prompt structures, therefore, explicitly separate:

- system instructions
- user requests
- document contents

This reduces the risk of prompt injection attacks in which hidden instructions inside documents attempt to override system behaviour [14].

Preemptive Filtering and Document Sanitisation

Additional protection can be achieved by filtering suspicious instructions before processing uploaded documents. Potential indicators could include phrases such as the following:

- “ignore previous instructions”
- “system prompt”
- “do not mention”
- “pretend”

Research has shown that indirect prompt injection through retrieved or uploaded documents represents a major attack vector in RAG-based systems [12].

Another possible mitigation is document sanitisation before ingestion. Recent work has demonstrated that malicious instructions may be hidden inside:

- invisible Unicode characters
- HTML comments
- hidden PDF text layers
- white-on-white text

The OCR-based preprocessing pipeline already reduces some of these risks because documents are transformed into extracted Markdown before processing.

Cross-document Consistency Checks

Cross-document consistency checks can further help identify manipulated statements. If one section contradicts other extracted evidence, the system can flag the inconsistency for manual review instead of directly generating a conclusion.

3.5 Cost Efficiency Optimisations

There are substantial operational expenses in a document extraction pipeline. Because both LangExtract and Instructor frameworks heavily query upstream Large Language Models, optimising the data flow can be helpful in ensuring the system remains resource-efficient.

Several architectural strategies can be implemented to minimise token consumption and reduce API latency.

3.5.1 Multi-Value Schema Structuring

The most straightforward driver of high API costs is a highly fragmented query pipeline. If the system is designed to extract values sequentially by sending an individual API request for each required field using the same documents as context, it creates massive token redundancy. For example, extracting five distinct fields from a ten-page document means sending the exact same ten pages of text to the model five separate times, forcing the system to pay for the input document context repeatedly.

To maximise cost efficiency, the pipeline should group all required fields into a single, comprehensive extraction schema. In practice, this means passing a unified Pydantic model with multiple properties to Instructor, or defining a single, multi-target prompt configuration within LangExtract. By restructuring the request this way, the raw document text is sent to the LLM only once. The model reads the context a single time and returns all extracted values simultaneously within one structured response. This single change eliminates duplicate input data, which significantly reduces the total input token overhead and shortens processing times.

However, forcing a model to search for too many variables simultaneously can dilute its attention and degrade extraction accuracy. If a single request asks for dozens of unrelated data points, the model is more likely to miss details or confuse context boundaries. To balance financial savings with data quality, fields should not be lumped into a single massive schema. Instead, they can be grouped into smaller, logically related batches of three to five related properties, which keeps the model focused on a narrow task context while still eliminating the bulk of the redundant document token overhead.

3.5.2 Context Caching

Running separate sequential extractions on the same documents forces the LLM to reprocess the same context for every call, creating unnecessary computational overhead.

Implementing **semantic context caching** significantly mitigates this redundancy. Modern LLM providers support API-level prefix caching, which retains highly repetitive prompt segments, such as massive raw document context, active within server memory for subsequent execution calls. By structuring prompts so that the raw document content remains at the beginning of the LLM call, the system can achieve near-total input-token reuse [15]. This drastically lowers the cost of processing multi-stage pipelines and dramatically accelerates how fast the model returns an answer.

However, this optimisation was not implemented due to fundamental framework constraints regarding LangExtract and Instructor:

- **Prefix Cache Invalidation**

API-level context caching strictly requires the shared tokens to occupy the absolute beginning (the prefix) of the prompt string. Because the system utilises a pipeline where instructions, few-shot examples, and schemas change dynamically between every extraction call, the prompt prefix is constantly altered. This sequence natively invalidates the cache before the downstream document context can be evaluated.

- **Framework Abstraction Limitations**

Both LangExtract and Instructor operate as high-level abstraction layers over the underlying LLM clients. Both LangExtract and Instructor completely control the internal payload compilation and formatting. Neither tool exposes the granular control necessary to rearrange the prompt structure into a document-first layout.

3.5.3 Multi-Model Routing

Not all extraction tasks require the analytical capabilities of premium, expensive frontier models. While complex requests benefit from high-tier models, capturing standard metadata fields (such as dates and values) can often be cleanly executed by smaller, lightweight models, which can reduce the baseline cost of document processing [16].

However, because our architecture already utilises open-source models with our maximum size at 31 billion parameters, which is a resource constraint of this thesis, the theoretical benefits of introducing an even smaller model tier are minor. Integrating a smaller model offers negligible processing or cost savings compared to our current configuration. Instead, it unnecessarily increases the risk of extraction errors across those fields.

3.6 Confidence Scores and Extraction Validation

If our pipeline could reliably measure whether a generated field is correct, marking out low-confidence extractions would be a highly valuable feature. In language modelling literature, tracking generation certainty typically relies on two primary methodologies: self-reported numerical confidence or token-level log probabilities. However, evaluating these metrics reveals that neither provides a dependable proxy for factual truth. Because neither approach is truly reliable, we made the deliberate design decision to omit confidence-based validation mechanisms from our architecture entirely.

3.6.1 Self-Reported Confidence

The most straightforward way to check certainty is to prompt the model to add a numerical confidence score directly inside the output JSON schema. However, these verbalised self-assessments are untrustworthy. Large language models suffer from a severe suggestibility bias when prompted to rate their own accuracy [17].

Without an external mechanism to verify its own output, the model tends to be heavily overconfident. It frequently assigns certainty scores of 90% or 100% even when the extracted text is completely hallucinated or factually incorrect [17]. This makes self-reported metrics practically useless for structural filtering, as the model cannot reliably identify when it has made a mistake or extracted the wrong data.

3.6.2 Token-Level Log Probabilities

An alternative approach is to look at the raw token-level log probabilities (logprobs) generated by the model while it is writing the response. Logprobs show how confident the model is during the decoding phase, but they do not measure actual factual correctness. Instead, a logprob strictly measures linguistic predictability. It simply tells us how statistically certain the model is about which specific word should come next based on grammar patterns and the text it just processed.

In a structured extraction pipeline, this creates a major flaw when handling wrong answers. If the model misinterprets a complex prompt and starts generating a response from the wrong section of a document, it will still output that incorrect data with near 100% token probability. It does this simply because the specific words it is writing form the most grammatically fluent and natural sequence to complete that sentence. Conversely, a perfectly accurate numeric value, such as a unique ID or financial figure, might receive a very low logprob score simply because that specific sequence of random digits is statistically rare and unpredictable to the model’s vocabulary matrix. Therefore, token-level log probabilities provide no reliable metric for verifying whether an extracted value is correct or not.

3.7 Accuracy degradation with context size

A substantial body of work has shown that LLMs lose accuracy on extraction-style (“needle-in-a-haystack”) tasks as the input context increases, well before their advertised context limits are reached [18], [19]. The exact point at which accuracy drops is highly dependent on model and task, and many of the studies were conducted on already outdated models, so their token figures do not transfer directly. The consistent and transferable finding is: a model’s effective context length, meaning the length over which it maintains reliable accuracy, is shorter than its advertised context length, and accuracy tends to degrade progressively rather than only at the window’s edge. Notably, the NoLiMa [19] benchmark/paper reports that many of the previous generation models (GPT-4o or Gemini 2.0) already fall below half of their short-context (1k tokens) accuracy at around 32k tokens.

The models evaluated in this work support context lengths of 128k tokens or more; operating near those limits would risk working in this degraded region. Since we work with more recent models, we capped the input context length at roughly 16k tokens. This is not a proven safe threshold, but it is a deliberately conservative trade-off. It stays well below every model’s advertised window. Because we don’t exactly know how many characters equal one token, we are using the standard estimate of four characters per token and setting the threshold at 64k characters.

4 Architecture & Implementation

4.1 Domain Model

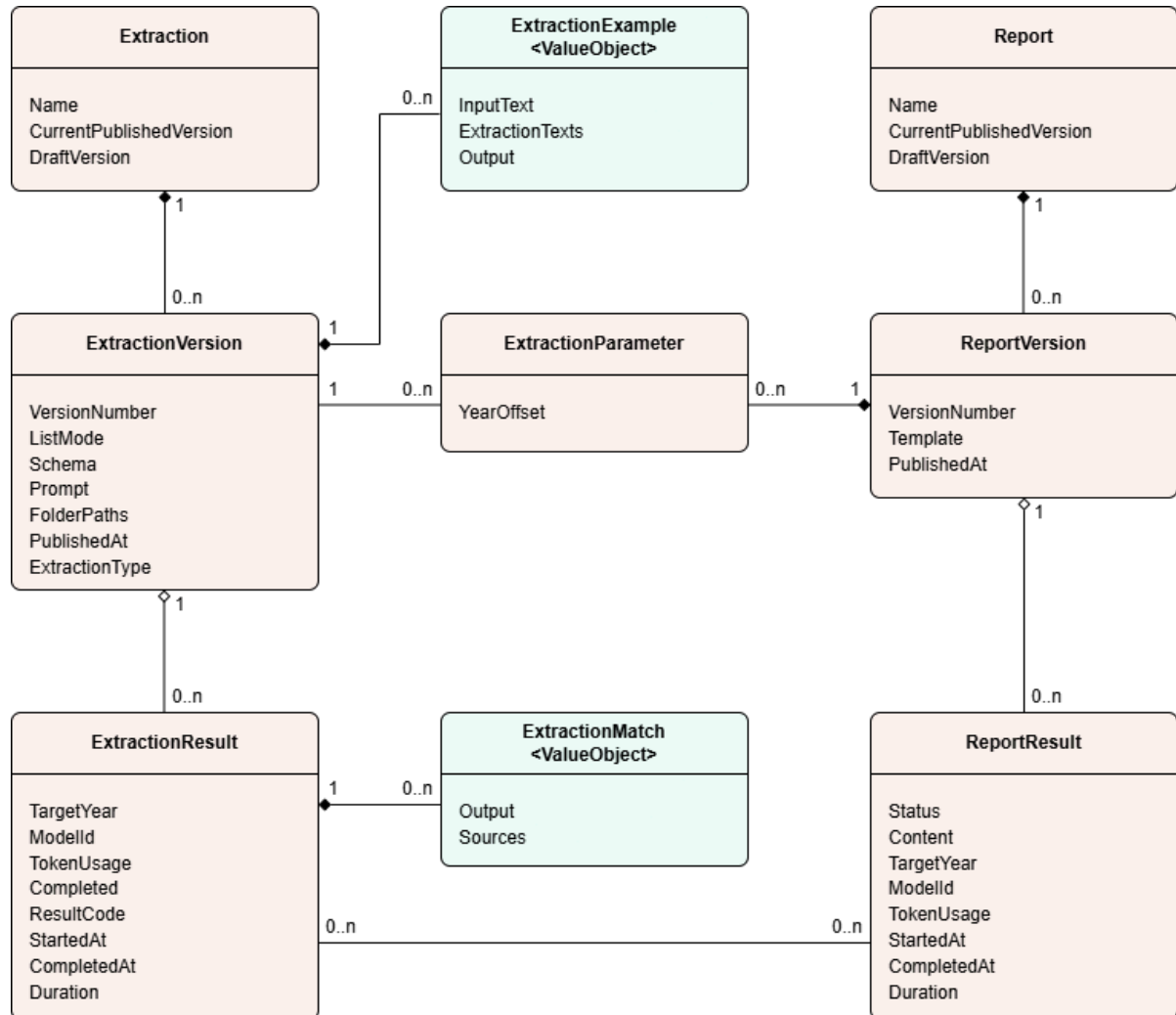


Figure 4.1: Domain model diagram

4.2 C4 Model

4.2.1 Context Diagram

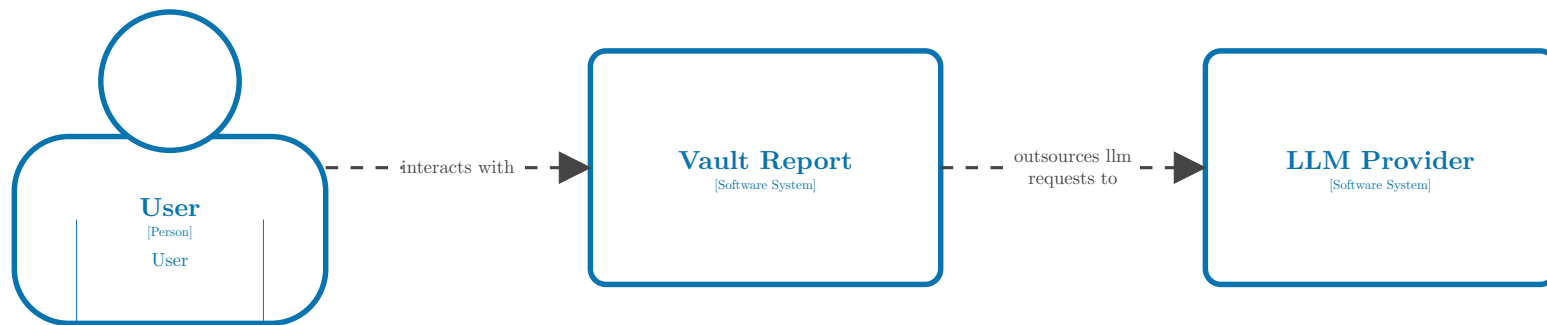


Figure 4.2: Context diagram

4.2.2 Container Diagram

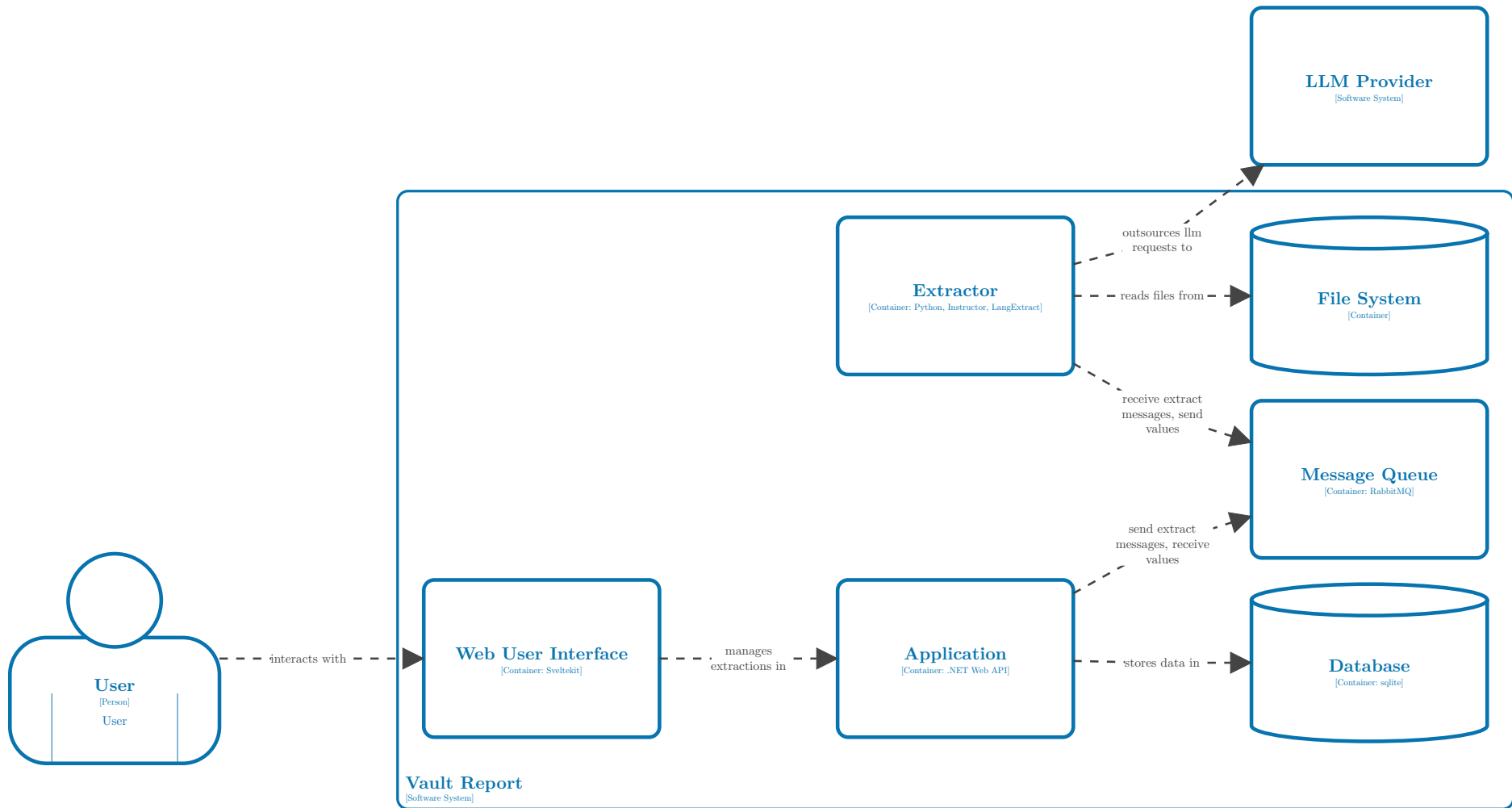


Figure 4.3: Container diagram

4.2.3 Component Diagram

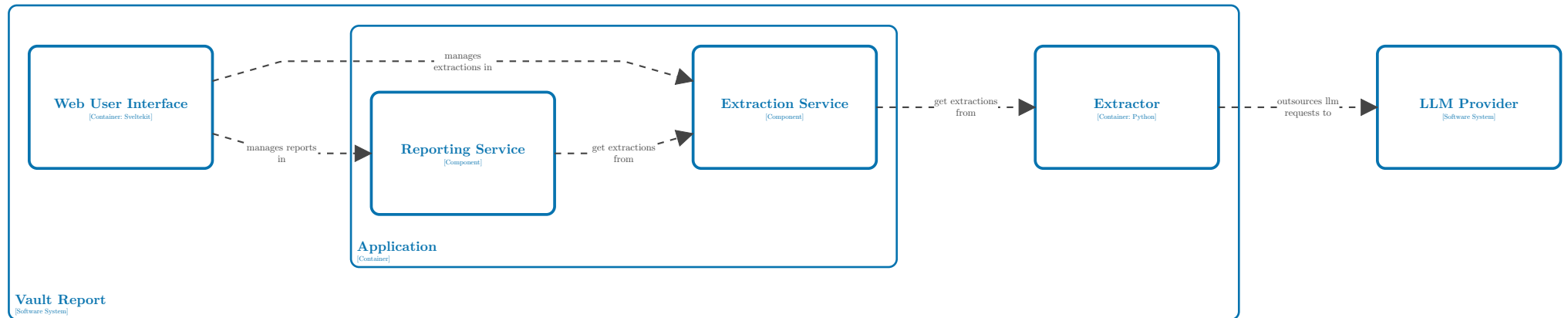


Figure 4.4: Component diagram

ADR ID Y Statement

- ADR-1** In the context of how documents are supplied to the language model for extraction, facing the choice between passing each document whole or splitting it for retrieval, we decided to chunk at the document level (pass the entire document) and neglected a retrieval-augmented generation (RAG) approach to achieve simpler processing and full-document reasoning, since the documents fit within the model’s context windows and the model can therefore see the whole document at once rather than isolated fragments, accepting that this requires models capable of handling larger contexts and may degrade accuracy as document and context length grow.
- ADR-2** In the context of choosing a ground architecture for the software, facing a decrease in complexity during the project based on user feedback during vision refinement meetings, we decided on a configurable extraction pipeline and neglected a node-based system to achieve less complexity overall, accepting the exclusion of the idea of a rule-based “red flag detection system” because the vision of the partner changed during the project, which greatly altered the product requirements.
- ADR-3** In the context of providing a user interface for demonstration purposes, facing the need to build a working UI with minimal effort given the team’s existing skill set, we decided on SvelteKit and neglected Blazor to achieve lower development effort by leveraging the team’s stronger experience with JavaScript/TypeScript-based UI frameworks, accepting that the demo UI diverges from the partner’s Blazor stack, which is acceptable since the partner will implement their own UI regardless.

ADR ID	Y Statement
--------	-------------

- | | |
|--------------|---|
| ADR-4 | In the context of the communication between the application and the Python extraction worker, facing the need to handle long-running extraction tasks that should not block the application, we decided on a message queue and neglected synchronous RPC / direct calls to achieve asynchronous processing, guaranteed delivery if the worker crashes, and request buffering when many extraction requests arrive simultaneously, accepting that we introduce an additional infrastructure component (the message broker) that has to be selected, deployed, operated, and monitored. |
| ADR-5 | In the context of the main application's framework and language, facing the need to choose a technology stack that the team can build on effectively and that integrates well with the partner's systems, we decided on .NET and neglected Python to achieve faster development through the team's existing .NET experience and smoother integration with the partner, who also runs .NET, accepting that the extraction logic, which relies on the Python libraries, had to be split out into a separate Python worker container rather than living in the main application. |
| ADR-6 | In the context of the application's data storage, facing the need for a lightweight database without committing to production-grade infrastructure, we decided on SQLite and neglected PostgreSQL to achieve a simple, zero-setup datastore that is easy to integrate through EF Core (which both we and the partner use), accepting that SQLite is suited to lighter workloads rather than production load, since the partner will host their own database anyway. |

ADR ID	Y Statement
--------	-------------

ADR-7	In the context of data access within the application, facing the choice of whether to introduce a repository abstraction over EF Core, we decided to use EF Core directly in the application layer and neglected the repository (and unit-of-work) pattern to achieve less boilerplate and full access to EF Core’s querying capabilities without a layer that largely duplicates what the framework already provides, accepting that the application is more tightly coupled to EF Core and that swapping the data access technology later would require more change.
--------------	--

ADR-8	In the context of how source documents are accessed, facing the choice between implementing document/file access in the backend or confining it to the extractor, we decided to keep all file access in the extractor and to omit it from the backend entirely, neglecting a backend file abstraction, to achieve a smaller backend scope and avoid building an interface that the partner’s integration will replace anyway, since the partner already has their own file abstraction and the production integration will handle file access differently, accepting that the backend cannot reason about files, so file-content-based features such as <code>ExtractionResult</code> invalidation on file changes are not possible in the current system and are left to the integration.
--------------	--

4.3 Backend Implementation

The backend is an ASP.NET Core Web API and is the only component that accesses the database. It is structured following the principles of clean architecture, so that the domain and application logic do not depend on infrastructure.

4.3.1 Extraction Definition

An extraction definition describes what should be extracted and how.

Property	Purpose
Name	Name that captures the identity of the entity to be extracted. The name is also passed on to the extractors to be used by the LLMs.
ListMode	Defines if a list or a single value is to be extracted.
Schema	A JSON Schema describing a single entity. It allows for validation and also for defining descriptions of properties that the LLMs can use.
ExtractionType	An extraction can either be of the grounded or generative type. This toggle decides if langextract for grounded extractions or instructor for more interpretive extractions that need paraphrasing is used.
Prompt	A prompt specific to the extraction to add more domain context can be defined, although in most cases, it's not necessary.
FolderPaths	This defines the list of folder paths used to collect the source documents for the extraction.
Examples	Examples that guide the extraction. They include an input text, an output JSON string, and a list of extraction texts in case the extraction is grounded.

Table 4.1: Extraction Definition Properties

4.3.2 Report Definition

A report definition pairs a template with the extractions needed to fill it.

Property	Purpose
Name	Name of the report.
Template	A markdown/HTML template including <code>Scriban</code> syntax. The template is used together with the extraction results to render the final report.
ExtractionParameters	The list of referenced extractions, each with a year offset, that supply the values the template renders.

Table 4.2: Report Definition Properties

4.3.3 Versioning Model

Both extractions and reports are versioned. An extraction holds a list of versions, a reference to its currently published version, and a draft. The published version may be empty when nothing has been published yet, whereas a draft always exists. Publishing a draft clones it into a new published version, after which editing continues on a fresh draft. Reports follow the same model.

4.3.4 Extraction Orchestration

An extraction can be run on its own or as part of a report. When a report is run for a target year, the system processes every referenced extraction, reusing an existing result where possible instead of recomputing. A stored result is reused when it was produced from the same extraction version, with the same LLM and target year, and completed without error. Each referenced extraction also carries a year offset, so its effective target year is the report’s target year plus that offset, which lets a single report combine extractions from different years.

Reports are completed asynchronously: extractions without a reusable result are dispatched to the queue, and the report is finalised once the last of them returns. If every result is already available, the report is rendered immediately.

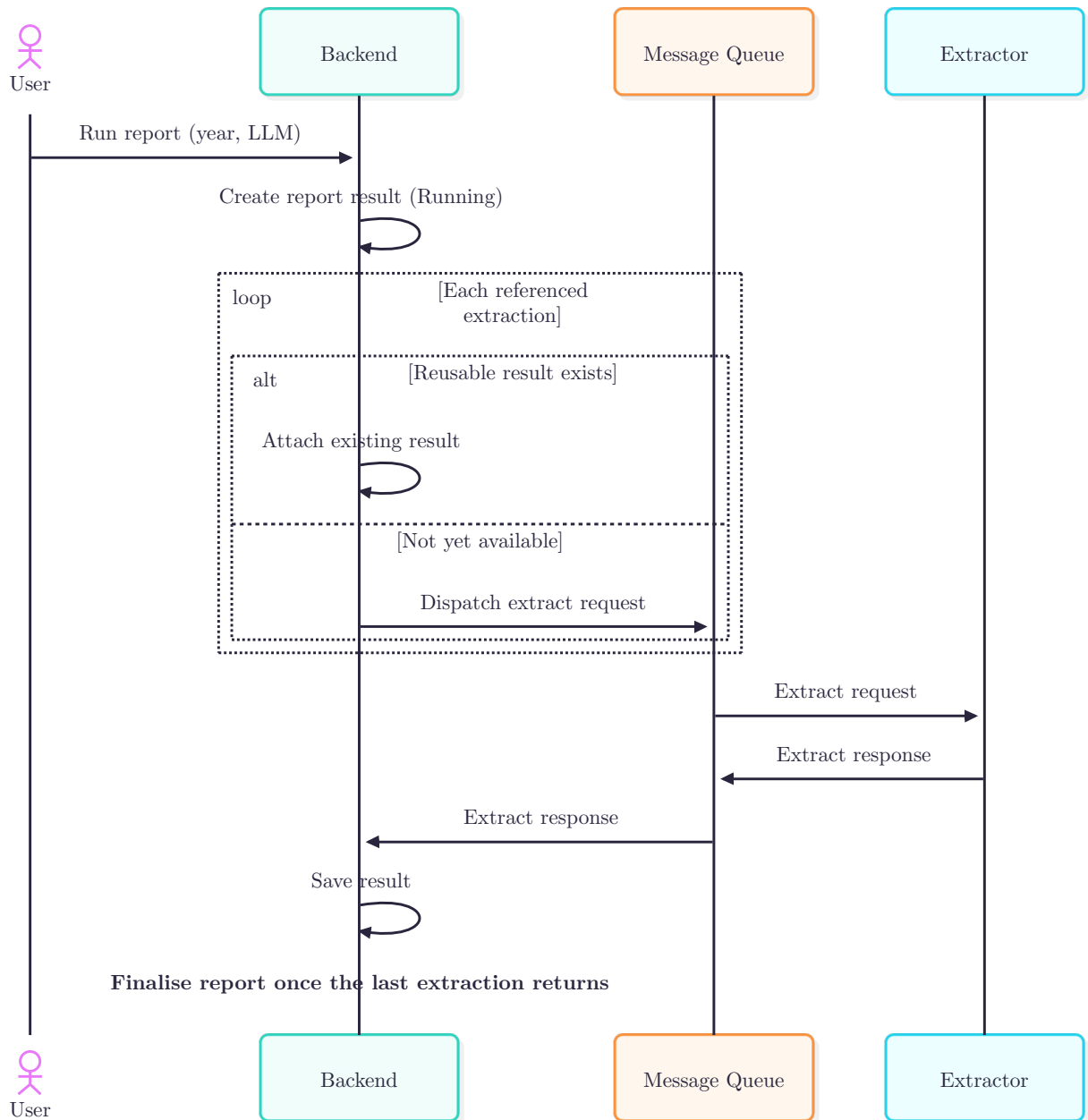


Figure 4.5: Report orchestration sequence

A change to the underlying source files does not currently invalidate a stored result, because the backend has no access to the files, only the extractor reads documents. File-aware invalidation is therefore deferred to the production integration, where document access is handled by the partner's existing file abstraction.

4.3.5 Templating

Reports are rendered with “Scriban”. Before rendering, each extraction result attached to the report is converted into an object that the template addresses by the identifier configured on the report’s extraction parameter. A single-value extraction becomes one object, and a list extraction becomes an array.

For every scalar field of an extracted value, a sibling field with the suffix `_source` is added. It holds an HTML link back to the originating extraction result, using the URI scheme `vault://<result-id>`, so a template can render a value together with a link to its source. Empty or missing values are replaced with the placeholder `N/F` (not found) or `N/A` (not applicable), depending on the result code.

In addition to the extracted values, a report object exposes metadata about the run those templates, then can use.

4.4 Message Queue

The backend and the extractor communicate exclusively through a message broker, implemented with `RabbitMQ`. There are two queues: one for extraction requests and one for extraction responses. Queues and messages are configured as durable and persistent, so they survive a broker restart. A message that cannot be processed is routed to a dead-letter queue, where it is retained for inspection rather than discarded.

4.5 Message Contract

The two components exchange two message types over the two queues. Both carry a shared `Id`, the identifier of the extraction result created when the run is requested. This `Id` is the correlation key: the extractor echoes it back in the response so the result can be matched to its originating request and its database row. It is the only piece of state the otherwise stateless extractor needs to retain and return.

4.5.1 Request Message

Property	Purpose
Id	Identifier of the <code>ExtractionResult</code> this request was created for. It is echoed back unchanged in the response and serves as the correlation key.
Name	Name of the entity.
ListMode	A boolean to define if a list or a single match is to be extracted.
ExtractionType	Defines whether the extraction is grounded (<code>LangExtract</code>) or generative (<code>Instructor</code>).
Prompt	The extraction-specific prompt.
Schema	<code>JsonSchema</code> describing an entity.
TargetYear	The year the extraction targets; only files from that year are used as context.
LLM	An object with the capability information of the LLM to be used. Defines which LLM is used, the maximum context size for chunking, and whether <code>Instructor</code> uses tool functions for extractions.
FolderPaths	A list of folders from which the documents are taken as sources.
ExtractionExamples	A list of example extractions to guide the model.

Table 4.3: Extraction Request Message

4.5.2 Response Message

Property	Purpose
Id	The same Id as the request. Identifies which <code>ExtractionResult</code> the backend updates when the response is consumed.
ResultCode	Result of the extraction: Success / NotFound / NotApplicable / Error.
Matches	A list of the matches found, each with its source. In single mode, the list contains only one match. For grounded extractions, the source references the exact span in the document; for generative (Instructor) extractions, it references the source document the match was derived from.
TokenUsage	The number of input, completion, and total tokens consumed for the extraction.

Table 4.4: Extraction Response Message

4.6 Extractor Implementation

The extractor is a stateless Python worker. It consumes a request from the queue, runs a fixed pipeline, and publishes a response. Because it holds no state beyond the request it is processing, it neither reads from nor writes to the database.

4.6.1 Resource Consumption Monitoring

In production-grade data extraction pipelines, monitoring computational resource consumption is critical for budgeting, predictability, and auditing. This consumption is primarily measured via token usage.

4.6.1.1 The Token Tracking Limitation in LangExtract

During the development of our pipeline, we encountered a critical limitation in the LangExtract framework. While LangExtract abstracts the orchestration of LLM calls, its internal parsing and response-handling mechanisms omit the underlying resource metadata. When LangExtract yields the final extracted data structure, it strips away the native OpenAI API response object, thereby discarding the `usage` payload.

Without access to this data, the system cannot natively audit how many tokens a specific extraction run consumed. Because the framework’s core abstraction layer lacked a native hook to pass this metadata upward, a standard implementation was unfeasible.

4.6.1.2 Interception via Dynamic Patching

To resolve this limitation without maintaining a custom, forked version of the `LangExtract` library, we implemented an architectural workaround using dynamic runtime modification, commonly referred to as a **monkey patch**.

Instead of waiting for `LangExtract` to return the filtered response, we intercepted the network communication at the source: the `openai` client library itself. By capturing the original API completion method and wrapping it in a tracking layer, we injected stateful logging directly into the execution lifecycle.

```
_original_create = openai.resources.chat.completions.Completions.create

def _patched_create(*args, **kwargs):
    response = original_create(*args, **kwargs)
    if hasattr(response, 'usage') and response.usage:
        u.PromptTokens += usage.prompt_tokens
        u.CompletionTokens += usage.completion_tokens
        u.TotalTokens += usage.total_tokens
    return response

openai.resources.chat.completions.Completions.create = _patched_create
```

Listing 4.1: OpenAi token tracking monkey patch

This approach provides a clean separation of concerns. `LangExtract` continues to handle schema enforcement and extraction logic natively, completely unaware that its underlying dependency is being audited. Every time the framework initiates a chat completion, our `patched_create` function intercepts the raw response, increments the global `token_usage` state, and returns the unaltered object to the framework.

4.6.2 Pipeline

The pipeline retrieves the source documents, splits them into batches that fit the model's context window, runs the extraction on each batch, and merges the per-batch results into a single response. The diagram below shows the full flow.

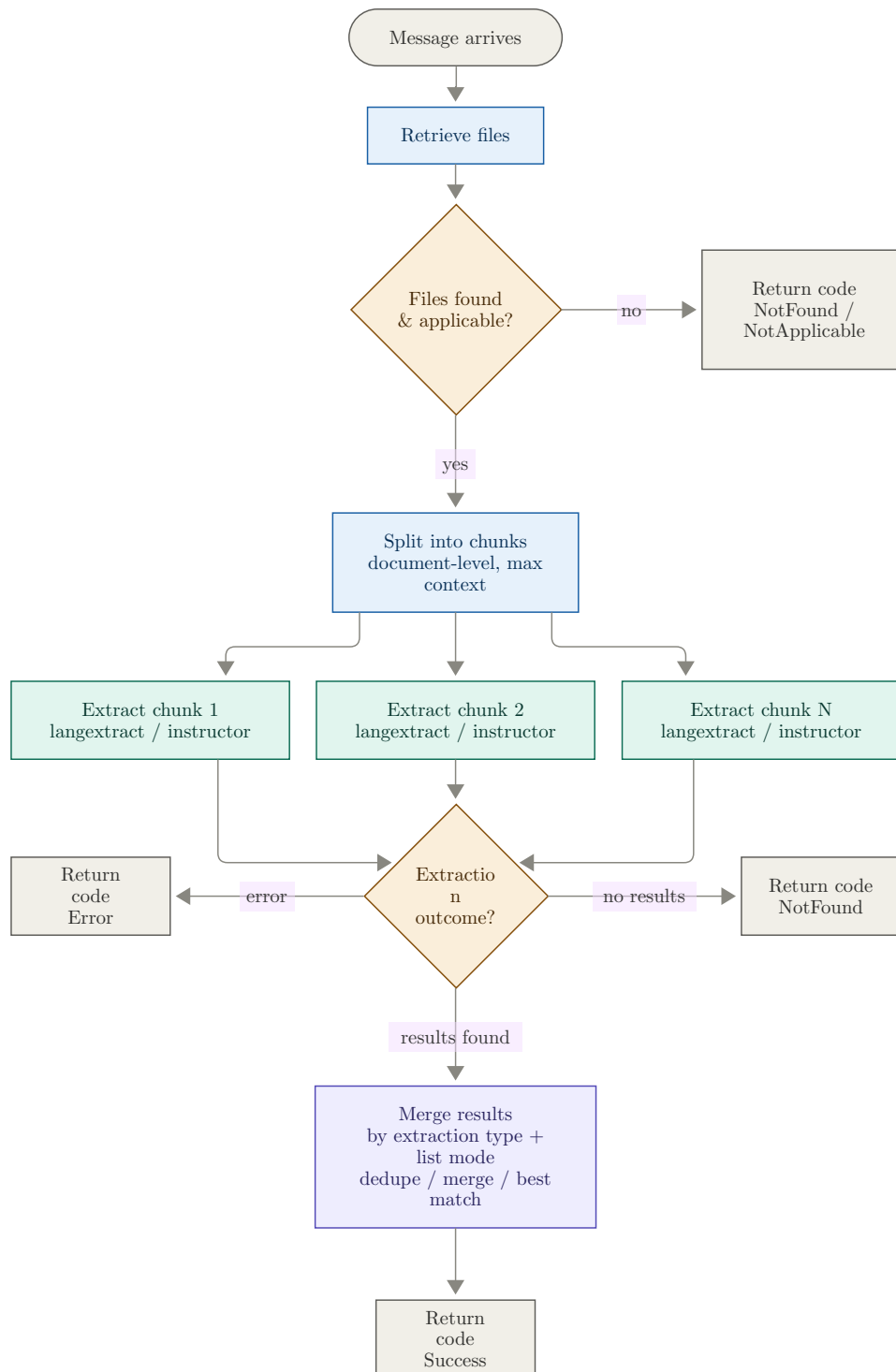


Figure 4.6: Extraction flowchart

4.6.3 Document Retrieval

Documents are read from the filesystem based on the request's folder paths and target year; only documents from the target year are used. As a special case, a folder may contain a marker file that flags the extraction as not applicable. If every retrieved document has this marker, the pipeline stops and returns a not-applicable result without calling the model.

4.6.4 Chunking

Documents are grouped by folder and packed into batches up to the model's context window, so documents from the same folder are kept together in a batch wherever possible. A single document larger than the context window is placed in its own batch. Chunking at the document level is deliberate: it keeps whole documents together so the model reasons over complete documents rather than isolated fragments. The rationale for this approach over retrieval-augmented generation is recorded in ADR-1.

4.6.5 Grounded Extraction with LangExtract

Grounded extractions use LangExtract. The JSON Schema is turned into a Pydantic model that parses the result, the extraction examples, and the property descriptions from the schema are passed to the library. Because documents are already chunked at the document level, LangExtract's own chunking threshold is set high, so its internal chunking only takes effect in edge cases.

4.6.6 Generative Extraction with Instructor

Generative extractions use Instructor. A Pydantic model is generated from the JSON Schema with the examples embedded in its configuration. Instructor then calls the model and parses the response directly into that model.

4.6.7 Merging

When an extraction spans more than one batch, the per-batch matches are reconciled with a further LLM call through Instructor, using the requested model. The strategy depends on the extraction mode.

ListMode	ExtractionType	Merge strategy
true	any	The model is instructed to deduplicate the matches
false	Grounded	The model is instructed to select the best-supported match.
false	Generative	The model is instructed to merge the matches into one consolidated result.

Table 4.5: Merge strategy by extraction mode.

4.6.8 Results

The response carries a result code that summarises the outcome

Property	Purpose
Success	At least one match was found.
NotFound	No matches were found.
NotApplicable	All retrieved source documents were the not-applicable marker.
Error	A batch raised an error during extraction.

Table 4.6: Conditions for each result code

4.7 Deployment

The system is packaged as a set of containers orchestrated with Docker Compose: the SvelteKit frontend, the .NET backend, the Python extractor, the RabbitMQ broker, and a Caddy reverse proxy that terminates TLS and routes external traffic to the services. The backend and the extractor both wait for the broker to report healthy before starting, so the system comes up in a defined order rather than racing the broker connection. State that must outlive a container is kept on mounted volumes: the SQLite database, the application logs, and the broker's durable queues.

4.8 Scalability

The system consists of the frontend, the backend, the extractor (worker), and the message queue between the backend and the worker. Each part scales differently. This section describes how far each can scale and where the limiting factor lies.

4.8.1 Message Queue

The message queue decouples the backend from the extractor: the backend enqueues a request and returns immediately instead of waiting for the extraction to finish. This lets the two scale independently and keeps the backend responsive regardless of how much extraction work is outstanding, since bursts accumulate in the queue rather than overwhelming the worker.

4.8.2 Extractor

The worker can already scale vertically by configuring the concurrency setting, which defines how many messages from the message queue are pulled and processed in parallel. Beyond that, since the worker is stateless, and requests are independent, additional instances can be started using Docker Compose, and each instance pulls messages at its own configured concurrency, while the queue distributes the work between them.

The practical limit on scaling is the LLM provider, not local resources. The extractor spends most of its time waiting for the LLM requests to complete, so throughput is bounded by what the provider allows (rate and cost limits) rather than by local CPU or memory.

4.8.3 Backend

Even on a single instance, concurrent access is not fully safeguarded. The backend has no optimistic concurrency control, so two requests that load and modify the same entity can overwrite each other and cause lost updates. Horizontally scaling the backend would additionally require switching from the embedded, file-based SQLite database to a client-server database such as PostgreSQL. Multiple backend instances cannot safely share a single SQLite file, whereas a client-server database runs as a separate service that all instances connect to and is built for concurrent access.

4.8.4 Frontend

The SvelteKit frontend is stateless, so multiple instances can be started without issue.

5 Evaluation

Due to the probabilistic nature of large language models, the testing approach combines traditional software testing with data-driven evaluation methods. The goal is to ensure reliability, consistency, and business usefulness of the generated outputs.

5.1 Evaluation Concept

The evaluation concept covers:

- **OCR Evaluation:** Evaluation of different tools and LLMs for OCR.
- **LLM Evaluation:** Evaluation of different LLMs for extraction tasks.
- **Unit/Component Testing:** Verification of the behaviour of individual deterministic components.
- **Functional Requirements Testing:** Evaluation of whether the solution meets the functional requirements.
- **LLM Testing:** Assessment of the quality and reliability of information extracted by the LLM using LangExtract and instructor.
- **Proof of Concept Report - Executive Summary:** Scriban template developed in cooperation with Aiger Group AG to showcase the system capabilities to non-technical stakeholders.

5.2 OCR Conversion Method Evaluation

This chapter presents the evaluation of the OCR system. It situates our work within existing benchmarks and demonstrates how the system performs in practical scenarios. Both quantitative and qualitative analyses are used to assess its accuracy, robustness, and overall utility.

5.2.1 Selected Tools and Models

We selected these specific OCR models and document parsing frameworks because they consistently ranked at the top of recent online benchmarks for text extraction accuracy and were available for us to evaluate.

- GLM-OCR [20]
- Docling Tesseract [21], [22]
- Paddle OCR VL [23]
- Deepseek OCR [24]
- Llama-3.1-Nemotron-Nano-VL-8B-V1 [25]
- Qwen3.5 A35-A3B [26]

5.2.2 Document Corpus

The evaluation used a selected corpus of converted markdown documents derived from authentic business documents of our industry partner. The dataset includes:

- Commercial register excerpt (complex table structure)
- Consultancy Agreement (text based)
- Tax statement (tables with lots of numbers)
- Employee CV (important use case)

5.2.3 OCR Conversion Speed Evaluation

The conversion speed was evaluated by measuring the time each selected model and tool took to convert the document corpus.

Due to technical limitations, the speed tests were executed in different environments. Docling, PaddleOCR-VL, DeepSeek-OCR, and Nemotron were run on a machine with an RTX 4080 Super using the Python libraries and transformers, while Qwen3.5 and GLM-OCR were run on a Mac Mini using llama.cpp. This makes the results not a direct ranking: differences within roughly an order of magnitude cannot be attributed to the models alone, since the hardware and runtime also differ. Differences of seconds versus minutes, however, exceed what the environment differences could plausibly account for and therefore still indicate a real gap in conversion speed.

5.2.4 OCR Conversion Quality Evaluation

This chapter defines the evaluation framework used to assess the OCR system. Beyond standard metrics like CER and WER, the criteria include numeric accuracy, layout preservation, table extraction, Markdown syntax correctness, and processing performance. Each criterion is weighted to reflect its relative importance for practical OCR usability. Together, these criteria support a comprehensive, quantitative, and qualitative assessment of OCR performance.

5.2.4.1 Quality Criteria

Our evaluation extends existing benchmarks by introducing new criteria such as numeric accuracy, where all numeric values including decimals, IDs, totals, and quantities are tested separately, and Markdown syntax evaluation to ensure correct rendering and formatting. While benchmarks like OmniDocBench [1] rely on automated metrics such as TEDS Section 5.2.4.1.3.1 for table structure and component-level evaluation of layout, recognition, and parsing, our approach also provides fine-grained, interpretable scores across text accuracy, layout preservation, table extraction, and Markdown correctness. This enables a more practical assessment of OCR output quality for structured Markdown documents, capturing errors in numeric values and formatting that are often overlooked in other evaluations.

Criteria	Weight	Description
Text Accuracy	35	CER, WER, Semantic Correctness (Dates, Currency, Percentages, Numbers)
Layout Preservation	20	Correct headings, Correct paragraphs, Correct lists, correct handling of Multi-column layouts
Table Extraction	25	correct table structure (Markdown-Syntax), correct headers, correct interpretation of merge-cells
Markdown Syntax	15	correct markdown syntax, nothing breaking
Performance	5	Duration of how long the model needs to run per page

Table 5.1: OCR Conversion - Testing Criteria

5.2.4.1.1 Text Accuracy

Sub-Criterion	Description	Weight	Scoring
Numeric Accuracy	Correct extraction of all numeric values including decimals, IDs, totals, and quantities.	15	5 = no numeric errors 4 = 1 minor numeric deviation 3 = 2 numeric errors 2 = 3 numeric errors 1 = >3 numeric errors
Semantic Correctness	Correct interpretation of structured semantic values such as dates, currencies, and percentages.	10	5 = no semantic errors 4 = 1 semantic error 3 = 2 semantic errors 2 = 3 semantic errors 1 = >3 semantic errors
Character Error Rate (CER)	Percentage of incorrectly recognized characters compared to the original text.	6	5 = $\leq 1\%$ CER 4 = 1–3% 3 = 3–6% 2 = 6–10% 1 = >10%
Word Error Rate (WER)	Percentage of incorrect words including substitutions, insertions, or deletions.	4	5 = $\leq 2\%$ WER 4 = 2–5% 3 = 5–8% 2 = 8–12% 1 = >12%

Table 5.2: OCR Conversion - Text Accuracy Evaluation Criteria

5.2.4.1.2 Layout Preservation

Sub-Criterion	Description	Weight	Scoring
Heading Hierarchy	Correct reconstruction of heading levels (#, ##, ###) and logical document structure.	6	5: perfect • 4: 1 shift • 3: several shifts • 2: mostly wrong • 1: unusable
Multi-column Linearisation	Correct reading order when converting multi-column layouts into linear text.	6	5: perfect order • 4: 1 misplaced • 3: several misplaced • 2: mixed columns • 1: lost
Paragraph Separation	Correct detection of paragraph boundaries and logical text grouping.	4	5: perfect • 4: 1 minor issue • 3: several issues • 2: many errors • 1: lost
List Formatting	Preservation of ordered and unordered lists including indentation and bullet structure.	4	5: preserved • 4: minor change • 3: flattened • 2: partly lost • 1: lost

Table 5.3: OCR Conversion - Layout Preservation Evaluation Criteria

5.2.4.1.3 Table Extraction

Sub-Criterion	Description	Weight	Scoring
Tree-Edit-Distance-based Similarity (TEDS Score)	Measures how well the table preserves merged cells, row/column spans, and cell contents compared to ground truth. Continuous score 0–1; higher is better.	9	5 = ≥ 0.9 4 = 0.75-0.9 3 = 0.5-0.75 2 = 0.25-0.5 1 = < 0.25
Table Structure (Rows/Columns)	Correct detection of table rows, columns, and overall structural grid.	7	5: correct • 4: 1 issue • 3: several issues • 2: major loss • 1: collapse
Column Alignment	Correct horizontal alignment of table values without column shifting.	5	5: none • 4: 1 shift • 3: several shifts • 2: frequent shifts • 1: unusable
Header Identification	Correct recognition of table header rows and column labels.	4	5: correct • 4: minor issue • 3: partial • 2: mostly data • 1: lost

Table 5.4: OCR Conversion - Table Extraction Evaluation Criteria

5.2.4.1.3.1 Tree Edit Distance-based Similarity (TEDS)

To evaluate how well an extraction framework preserves tables, including rows, columns, merged cells, and text, the Tree Edit Distance-based Similarity (TEDS) metric is used.

Formula and Calculation

TEDS converts both the ground-truth table (T_{gt}) and the predicted table (T_{pred}) into HTML tree structures. It measures the minimum number of changes (insertions, deletions, or substitutions) needed to turn the predicted tree into the correct tree, resulting in a score between 0 and 1:

$$\text{TEDS}(T_{\text{pred}}, T_{\text{gt}}) = 1 - \frac{d(T_{\text{pred}}, T_{\text{gt}})}{\max(|T_{\text{pred}}|, |T_{\text{gt}}|)}$$

Where:

- $d(T_{\text{pred}}, T_{\text{gt}})$ is the total cost of structural and text edits required.
- $|T|$ is the total size (number of HTML elements) of the table.

When comparing individual cells, the substitution cost checks both structural properties (matching `colspan` and `rowspan` attributes) and text accuracy using character-level string distance. The less differences there are, the closer the score is to 1.

Limitations

Because TEDS relies purely on the HTML structure, it is highly sensitive to alternative markup. If the framework uses different HTML tags to represent the exact same visual layout, such as omitting a `<thead>` block, the score drops significantly even if the extracted data is completely correct.

5.2.4.1.4 Markdown Syntax

Sub-Criterion	Description	Weight	Scoring
Rendering-Breaking Issues	Presence of syntax errors that prevent Markdown rendering or break document display.	8	5: valid • 4: tiny issues • 3: minor glitches • 2: several breaks • 1: fails
Table Markdown Syntax	Correct use of Markdown table syntax including separators and column alignment.	4	5: correct • 4: 1 issue • 3: several issues • 2: partly invalid • 1: invalid
Heading/List Syntax	Correct Markdown syntax for headings and lists.	2	5: correct • 4: 1 issue • 3: several issues • 2: many errors • 1: broken
Escaping Issues	Incorrect escaping of Markdown special characters such as *, __, or #.	1	5: none • 4: 1 • 3: 2 • 2: 3 • 1: >3

Table 5.5: OCR Conversion - Markdown Syntax Evaluation Criteria

5.3 LLM Evaluation

The purpose of this evaluation is to measure how effectively different large language models can extract, combine, and interpret structured information from heterogeneous corporate documents. The benchmark reflects realistic enterprise-style tasks over legal, financial, and operational records.

The evaluation focuses on document understanding under strict output constraints. Each task requires identifying relevant information from one or multiple sources and returning structured, verifiable outputs.

5.3.1 Benchmark Design

5.3.1.1 Document Corpus

The benchmark corpus consists of converted markdown documents derived from corporate records. It includes:

- Commercial register excerpts
- Articles of association and charter documents
- Financial statements
- Shareholder and capital structure documents
- Organisational and operational records

The dataset combines structured, semi-structured, and unstructured text, including tables, legal clauses, and accounting statements.

5.3.1.2 Task Structure

Each benchmark case contains:

- A natural-language query
- One or more source documents
- A reference answer
- A judge-based evaluation output

Tasks vary in complexity. Some require simple field extraction, while others require cross-document consistency checks or interpretation of accounting/legal conventions.

5.3.1.3 Evaluation Criteria

Tasks are classified into three categories. Each case belongs to exactly one category.

Category	Definition	Core requirement
Direct Extraction	Retrieve explicit facts or structured fields from one or more documents without computation	Accurate retrieval of values, names, or fields
Structured Reasoning (Maths / Linking / Validation)	Requires calculation, reconciliation, comparison, or cross-document consistency checks	Correct arithmetic, correct line-item selection, consistent multi-document linking
Financial and Legal Interpretation	Requires classification, explanation, or semantic judgment over financial/legal meaning	Correct interpretation of accounting/legal meaning and correct categorisation

Table 5.6: LLM Evaluation Criteria

The taxonomy is defined as follows:

Direct Extraction Includes retrieval of explicit values such as:

- Registered addresses
- Share capital
- Organisational roles
- Personnel expenses (when directly stated)
- CV and register fields

Key difficulty: locating correct fields in heterogeneous document layouts.

Structured Reasoning (Maths / Linking / Validation) Includes tasks requiring:

- Arithmetic computation (ratios, totals, reconciliation)
- Cross-document consistency checks
- Alignment of financial statement components

Key difficulty: correct selection of line items and maintaining numerical consistency across documents.

Financial / Legal Interpretation Includes higher-level semantic judgments such as:

- Going concern assessment
- Financing classification (conditional vs unconditional)
- Legal interpretation of shareholder rights or capital structure
- Risk and liquidity assessments

Key difficulty: correct domain interpretation without introducing unsupported assumptions.

5.3.2 Evaluation Pipeline

5.3.2.1 *Generation Phase*

Each model receives:

- The task query
- Relevant document excerpts
- Output formatting instructions (JSON or structured text)

The model output is stored together with metadata including model identifier and source documents.

5.3.2.2 *Judge-Based Evaluation*

A separate judge model evaluates outputs against expected answers.

The judge determines:

- Whether the response is correct (pass/fail)
- Missing required elements
- Extra or unsupported elements
- Semantic equivalence where formatting differs

Minor formatting differences are tolerated when semantic correctness is preserved.

Example judge output:

```

{
  "case_id": "balance_sheet_totals_2023",
  "passed": true,
  "notes": "Semantic equivalence accepted despite formatting differences."
}

```

Listing 5.1: Judge output example

5.3.3 Evaluated Model Configurations

We evaluated four available open-weights models under the following local deployment configurations:

- **Gemma-4 (31B):** Dense architecture, 31 billion parameter pool. Evaluated using a 64k token context window and Q4_K_XL (4-bit) quantisation [27].
- **GLM-4.7-Flash (30A3):** Mixture of Experts (MoE) architecture, 30 billion total parameters with 3 billion active parameters per token. Evaluated using a 165k token context window and Q4_K_XL (4-bit) quantisation [28].
- **Ministral-3 (14B):** Dense architecture, 14 billion parameter pool with native vision support. Evaluated using a 120k token context window and Q6_K_XL (6-bit) quantisation [29].
- **Qwen3.6 (27B):** Dense architecture, 27 billion parameter pool with native reasoning capabilities. Evaluated using a 128k token context window and unquantised weights [30].

5.4 LLM Extraction Testing

The evaluation of LLM-based extraction is designed to address the unique challenges posed by large language models. Unlike traditional software testing, where outputs are deterministic and easily verifiable, LLM outputs can vary and require a more nuanced, two-layered assessment framework.

Testing is executed through python scripts over a total of 68 benchmark scenarios designed to pull data out of single files or across multi-document sets.

5.4.1 Document Corpus

The testing framework utilises a standardised corpus of converted markdown documents derived from authentic business documents of our industry partner. The dataset includes:

- Commercial register excerpts (e.g., Zurich and Zug cantonal registries)
- Articles of association and statutory charter documents
- Statutory financial statements (balances, income statements, and notes)
- Shareholder lists and capital structure records
- Operational records and employee CVs

This corpus combines structured data tables, semi-structured accounting layouts, and unstructured narrative legal clauses.

5.4.2 Extraction Types and Pass Criteria

Testing is strictly split into two specialised types depending on the nature of the target data. Each type contains exactly 34 test cases.

5.4.2.1 Grounded Span Extraction (LangExtract Section 3.3.1)

This extraction type targets precise, isolated fields (such as email addresses, exact dates, monetary values, or registration IDs).

- **Scope:** 34 test cases (5 runs each, totalling 170 runs)
- **Pass Criteria:** Direct string verification. A test run passes only if the extracted text matches the predefined ground truth exactly.

5.4.2.2 *Interpreted/Paraphrased Extraction*

(Instructor Section 3.3.2)

This extraction type deals with complex semantic tasks where data is scattered across one or multiple documents and must be returned as synthesised text summaries or structured narratives.

- **Scope:** 34 test cases (5 runs each, totalling 170 runs)
- **Pass Criteria:** Judge-based verification. Because exact text matching is impossible for synthetic summaries, an autonomous LLM Judge evaluates the output. A run passes only if the judge verifies that all requested information is present, no required elements are missing, and no additional un-requested or hallucinated information is added.

5.4.3 Reference Model Selection

All 68 test cases across both suites are executed using **Gemma-4** with 31 billion parameters. This model was selected because it achieved the best overall results during the evaluation phase Section 5.3, establishing it as our production baseline.

5.4.4 Test Case Schema

Each test case is modelled as a self-contained JSON configuration. The following snippet illustrates a sample case for a deterministic span extraction:

```
{
  "name": "CV_Email",
  "call": {
    "extraction_class": "Email",
    "prompt": "Extract the email address. Return only the email.",
    "examples": [
      { "text": "E-Mail test@example.com", "extraction_text":
"test@example.com" }
    ],
    "markdown_path": "Employees/John_Doe_CV.md"
  },
  "expected": [
    { "extraction_text": "john.doe@mail.com" }
  ]
}
```

Code Fragment 5.1: span extraction test example

This test case evaluates the system’s data extraction accuracy by targeting a specific field (Email) within a source file (markdown_path). The call object defines the execution parameters, including the structural category (extraction_class), the precise instruction (prompt), and a few-shot example (examples) to guide the model’s formatting. The pipeline executes this call and verifies the output against the ground-truth value defined in the expected array.

5.4.5 Testing Criteria

Assessment Criterion	Grounded Span Extraction (LangExtract)	Interpreted/ Paraphrased Extraction (Instructor)
Accuracy	Verified via exact string matching against the specific ground-truth value.	Verified via semantic equivalence checked by an autonomous LLM Judge.
Consistency	Evaluated across 5 identical iterations by checking for strict text parity of the mode response.	Evaluated across 5 identical iterations by checking for semantic parity of the mode response via the LLM Judge.
Hallucination / Missing Data	The system must explicitly return null or leave an empty string if the requested structural field is absent from the file.	The system must explicitly state that there is no information about the topic if it cannot be found in the documents.

Table 5.7: LLM Extraction Testing Quality Dimensions

5.4.5.1 Consistency Metric

To ensure the system remains reliable under non-deterministic conditions, every individual test case is run 5 consecutive times. We define system stability using a global consistency metric, which aggregates the frequency of the most dominant identical output value against the total number of test iterations.

The metric is formally evaluated as:

$$\text{Consistency} = \frac{\sum \text{top_count}}{\sum N}$$

Where `top_count` represents the frequency of the most common identical output variation per test case, and `N` is the total number of runs allocated to that case. A high global consistency percentage confirms that the extraction pipeline produces predictable, reliable results across subsequent enterprise runs.

5.5 Automated Testing

Test Category	Automated Unit and Component Tests
Test Level	Unit / Component Level
Test Framework	Pytest, xUnit.net and Vitest
Scope	Deterministic code logic including preprocessing, variable schema handling, template rendering, and validation utilities.
Execution	Executed on each commit and merge request within the CI pipeline.
Purpose	Ensure correctness of isolated business logic components independent of LLM behavior and external document retrieval.

Table 5.8: Automated Unit Testing

5.6 Test Results

5.6.1 OCR Evaluation

The primary objective of this evaluation is to determine which parsing and OCR framework is the most effective for converting heterogeneous business documents into clean Markdown text. Identifying the optimal model is critical, as the accuracy of the downstream LLM extraction pipeline relies entirely on the structural and textual fidelity of this initial conversion.

5.6.1.1 OCR Conversion Speed Evaluation Results

Test	Method	Total Runtime [Seconds]	Average Runtime per page [Seconds]
Agreement [7 pages]	GLM OCR	18.11	2.59
	Docling Tesseract	21.01	3.00
	Paddle OCR VL	103.88	14.84
	Deepseek OCR	353.57	50.51
	Qwen 3.5	779.15	111.31
	Nemotron	2080.06	297.15
CV [2 pages]	Docling Tesseract	5.80	2.90
	GLM OCR	13.38	6.69
	Paddle OCR VL	37.20	17.60
	Deepseek OCR	112.14	56.07
	Qwen 3.5	246.66	123.33
	Nemotron	584.00	292.00

Test	Method	Total Runtime [Seconds]	Average Runtime per page [Seconds]
HR Excerpt [1 page]	Docling Tesseract	5.07	5.07
	GLM OCR	15.32	15.32
	Paddle OCR VL	36.26	36.26
	Deepseek OCR	72.00	72.00
	Qwen 3.5	306.81	306.81
	Nemotron	576.69	576.69
Tax Declaration [8 pages]	Docling Tesseract	21.95	2.74
	GLM OCR	25.29	3.16
	Paddle OCR VL	172.22	21.53
	Deepseek OCR	395.68	49.46
	Qwen 3.5	1503.90	187.99
	Nemotron	3480.54	435.07

Table 5.9: OCR Conversion Methods Speed Evaluation Results

5.6.1.2 OCR Conversion Quality Evaluation Results

	Weight (%)	Criteria	Deepseek OCR	Docling Tesseract	GLM OCR	Nemotron	Paddle OCR VL	Qwen 3.5
Text Accuracy	10	Numeric Accuracy	3	2	2	5	1	5
	10	Semantic Correctness	2	3	4	4	3	5
	8	Character Error Rate (%)	3 (4.81)	4 (2.82)	1 (16.20)	2 (9.76)	3 (4.46)	5 (0.92)
	7	Word Error Rate (%)	3 (6.81)	2 (9.83)	1 (12.83)	1 (11.31)	2 (7.25)	5 (1.60)
Layout Preservation	6	Heading Hierarchy	5	5	4	2	4	3
	6	Multi-column Linearization	5	2	5	2	4	4
	4	Paragraph Separation	5	5	5	5	5	5
	4	List Formatting	4	5	5	4	4	5
Table Extraction	9	TEDS Score	4 (0.857)	1 (0.019)	5 (0.971)	2 (0.263)	5 (0.913)	5 (0.945)
	7	Table Structure	4	1	4	3	3	4
	5	Column Alignment	5	2	4	5	4	4
	4	Header Identification	5	3	3	2	3	5
Markdown Syntax	8	Rendering Breaks	5	5	5	5	5	5
	4	Table Syntax	5	1	5	5	5	5
	2	Heading/List Syntax	4	4	5	5	5	5
	1	Escaping Issues	3	3	4	4	4	4
	5	Performance (seconds)	4 (57)	5 (3.4)	5 (7)	1 (400)	4 (22.6)	2 (182.4)
100		Total	3,91	2,96	3,73	3,24	3,54	4,54

Table 5.10: OCR Conversion Quality Evaluation Results

These results show that the best OCR model in our evaluation is Qwen 3.5 [30]. Furthermore, several recurring issues were observed across the evaluated OCR systems:

- **Docling** struggled with German-language text, particularly special characters such as “ä”, “ö”, and “ü”. Additionally, it failed to correctly load the table structure in the HR document.
- **Nemotron** was unable to fully process the HR file; the output was truncated and incomplete.
- **All systems** showed at least some difficulties with complex **tax return tables**, indicating consistent challenges in extracting multi-level financial tables with unclear borders.
- **Handwritten content and signatures** were almost universally not recognized, which aligns with expectations given the systems’ design limitations.
- **TEDS Score** was only applied on the HR file; it was not applicable on the Tax file, since these tables there are unclear and thus support multiple interpretations.

5.6.2 LLM Evaluation

The primary objective of this evaluation is to identify which large language model is the most suitable and effective for the platform’s data extraction tasks. The evaluation covers 52 cases across three task categories.

5.6.2.1 Quantitative Performance Summary

Model	Direct Extraction (17)	Structured Reasoning (16)	Interpretation (19)	Overall (52)
Gemma 4	15 (88.2%)	16 (100%)	17 (89.5%)	48 (92.3%)
Qwen 3.6	13 (76.5%)	13 (81.2%)	18 (94.7%)	44 (84.6%)
GLM 4.7 Flash	16 (94.1%)	12 (75.0%)	15 (78.9%)	43 (82.7%)
Ministral 3	14 (82.4%)	13 (81.2%)	15 (78.9%)	42 (80.8%)

Table 5.11: LLM Evaluation Results

5.6.2.2 Category-Level Insights

Since Gemma 4 has the overall best success-rate, it is thus the winner of this evaluation. However, the results across individual categories reveal a more nuanced picture:

GLM 4.7 Flash demonstrated the highest precision in raw data retrieval, achieving 94.1% accuracy by minimising omitted field attributes. Across all models, failures in this category were primarily structural rather than contextual. The main error profiles involved the missing collection of secondary attributes in multi-attribute objects, incomplete mapping of deeply nested lists, and the occasional omission of critical units of measure such as currency markers and role designations.

In the structured reasoning category Gemma 4 achieved perfect accuracy (100.0%) by correctly resolving arithmetic constraints and calculating the correct totals from individual entries. For the remaining models, failures typically originated from incorrectly matching entries to their corresponding rows or making basic arithmetic mistakes. Ministral 3 and Qwen 3.6 tied at 81.2% accuracy, displaying similar structural compliance behaviours.

In the interpretation category Qwen 3.6 led at 94.7% accuracy, successfully resolving ambiguous clauses without injecting external assumptions. In contrast, weaker iterations regularly introduced unprompted assumptions or over-interpreted baseline statements, particularly when processing complex financing structures and liability conditions.

5.6.3 LLM Extraction Testing

The evaluation pipeline processed a total of 340 individual executions across both testing types (68 unique cases multiplied by 5 iterations each). The performance metrics highlight the operational capabilities of the reference **Gemma-4 (31B)** model under strict production constraints. Detailed test configurations can be found in Section 5.4.

5.6.3.1 Quantitative Performance Summary

The performance of each extraction type is summarised across two core quality dimensions: accuracy (the rate of passing runs) and consistency (the architectural stability of outputs over sequential runs).

Extraction Type	Total Iterations	Succeeded Iterations	Accuracy	Consistency
Value Extraction (LangExtract)	170	167	98.24%	98.82%
Information Extraction (Instructor)	170	163	95.88%	98.82%

Table 5.12: LLM Extraction Test Run Metrics Summary

5.6.3.2 Analysis of Empirical Findings

5.6.3.2.1 Span-Value Extraction

The deterministic value extraction framework achieved an accuracy of 98.24% with only 3 out of 170 individual runs failing. Error analysis shows that the failures were entirely concentrated within a single case: `Fin_Liabilities_TradePayables_2023`.

The model successfully located the correct financial metric but failed strict string validation by dropping the Swiss standard typographic apostrophes from the numeric formatting (predicting `120931.29` instead of the expected `120'931.29`). Because LangExtract enforces rigid schemas, the model maintained an exceptional global consistency score of 98.82%, showing zero structural drift and zero instances of false data fabrication.

5.6.3.2.2 Interpretative/Summarising Extraction

The narrative-driven information extraction suite achieved a pass rate of 95.88%, with 7 out of 170 runs rejected by the autonomous LLM Judge. These failures were confined to three specific scenarios and stem from strict scope boundaries rather than factual fabrication:

- **Company_Officers (5 failures):** All 5 iterations failed because the model over-included operational staff and advisers under the definition of “corporate officers”. The judge strictly classified these additional roles as out of scope.
- **Company_and_Financials_Overview (1 failure):** A single iteration (run 5) omitted a required cash figure, which the judge flagged as a missing critical element.
- **Personnel_Profile_Daniel_HR (1 failure):** One iteration (run 3) omitted the specific location of the registered office.

Furthermore, the consistency matched the value extraction pipeline at 98.82%. The error footprint confirms that the system is highly resistant to hallucination; failures were strictly caused by minor omissions of requested numeric details or the over-inclusion of closely related context, rather than the introduction of unsupported facts.

5.6.4 Functional Requirements

T1 - Create Report Configuration

UC / FR	UC1b, FR1.1
Action	Create a new report configuration through the report configuration overview.
Expected Result	The report configuration is created successfully and appears in the report configuration overview.
Status	Successfully tested

T2 - Search Report Configurations

UC / FR	UC1c, FR1.2
Action	Search for an existing report configuration by name in the report configuration overview.
Expected Result	The matching report configuration is displayed in the search results.
Status	Successfully tested

T3 - View Report Configuration

UC / FR	UC1d, FR1.3
Action	Open an existing report configuration from the report configuration overview.
Expected Result	The selected report configuration is opened and its details are displayed.
Status	Successfully tested

T4 - Delete Report Configuration

UC / FR	UC1e, FR1.4
Action	Delete an existing report configuration from the report configuration overview.
Expected Result	The report configuration is deleted and no longer appears in the overview.
Status	Successfully tested

T5 - Create Extraction Configuration

UC / FR	UC2b, FR2.1
Action	Create a new extraction configuration through the extraction configuration overview.
Expected Result	The extraction configuration is created successfully and appears in the extraction configuration overview.
Status	Successfully tested

T6 - Search Extraction Configurations

UC / FR	UC2c, FR2.2
Action	Search for an existing extraction configuration by name in the extraction configuration overview.
Expected Result	The matching extraction configuration is displayed in the search results.
Status	Successfully tested

T7 - View Extraction Configuration

UC / FR	UC2d, FR2.3
Action	Open an existing extraction configuration from the extraction configuration overview.
Expected Result	The selected extraction configuration is opened and its details are displayed.
Status	Successfully tested

T8 - Delete Extraction Configuration

UC / FR	UC2e, FR2.4
Action	Delete an existing extraction configuration from the extraction configuration overview.
Expected Result	The extraction configuration is deleted and no longer appears in the overview.
Status	Successfully tested

T9 - Report Versioning

UC / FR	UC3b, FR3.1
Action	Release multiple versions of a report configuration and retrieve a previous version.
Expected Result	Each released report configuration receives a version, and previous versions remain retrievable.
Status	Successfully tested

T10 - Template Editing

UC / FR	UC3b, FR3.2
Action	Edit a report template using the integrated template editor and add Scriban syntax.
Expected Result	The template can be edited, and the templating engine syntax is highlighted in the editor.
Status	Successfully tested

T11 - Report Draft Management

UC / FR	UC3b, FR3.3
Action	Create and edit a draft report configuration before releasing it as a new version.
Expected Result	The draft can be modified without replacing the released report configuration version.
Status	Successfully tested

T12 - Import Variables into Report Template

UC / FR	UC3b, FR3.4
Action	Select extraction variables that should be available inside the report template.
Expected Result	The selected extraction variables are imported and can be used in the report template.
Status	Successfully tested

T13 - Configure Extraction Offset

UC / FR	UC3b, FR3.6
Action	Configure the relative year offset for an extraction variable used in a report template.
Expected Result	The selected offset is stored and applied relative to the report target year.
Status	Successfully tested

T14 - Execute Versioned Report Configuration

UC / FR	UC3c, FR3.7
Action	Execute a released report configuration for a selected target year.
Expected Result	The system executes the selected report configuration version for the specified target year.
Status	Successfully tested

T15 - Execute Report Draft

UC / FR	UC3c, FR3.8
Action	Execute a draft report configuration before releasing it as a new version.
Expected Result	The system executes the report draft and generates a report result.
Status	Successfully tested

T16 - Select Report Execution Model

UC / FR	UC3c, FR3.9
Action	Select an LLM before executing a report configuration.
Expected Result	The report is executed using the selected LLM.
Status	Successfully tested

T17 - Present Report Result

UC / FR	UC3c, FR3.10
Action	Open a generated report result after report execution.
Expected Result	The report result is displayed together with suitable metadata.
Status	Successfully tested

T18 - Edit Extraction Configuration

UC / FR	UC3a, FR4.1
Action	Modify the relevant attributes of an existing extraction configuration.
Expected Result	The changed extraction configuration attributes are stored correctly.
Status	Successfully tested

T19 - Source Grounding

UC / FR	UC3a, FR4.2
Action	Execute an extraction and inspect the generated extraction result.
Expected Result	The system indicates the source from which the extracted information was obtained.
Status	Successfully tested

T20 - Schema Validation

UC / FR	UC3a, FR4.3
Action	Execute an extraction configuration with a defined result schema.
Expected Result	The extraction result is validated against the configured schema.
Status	Successfully tested

T21 - Natural Language Input

UC / FR	UC3a, FR4.4
Action	Create or edit an extraction configuration using natural language instructions.
Expected Result	The natural language input is stored and used as part of the extraction configuration.
Status	Successfully tested

T22 - Schema-Based Extraction Configuration

UC / FR	UC3a, FR4.5
Action	Configure an extraction using a defined schema.
Expected Result	The schema is stored as part of the extraction configuration and used to structure the result.
Status	Successfully tested

T23 - Generative Extraction Output

UC / FR	UC3a, FR4.6
Action	Configure and execute an extraction that generates textual output.
Expected Result	The extraction produces LLM-generated output according to the configuration.
Status	Successfully tested

T24 - Source Path Restriction

UC / FR	UC3a, FR4.7
Action	Configure an extraction with a restriction to selected source paths.
Expected Result	The extraction only uses documents from the configured source paths.
Status	Successfully tested

T25 - Target Year Specification

UC / FR	UC3a, FR4.8
Action	Execute an extraction for a specified target year.
Expected Result	The extraction is executed using the selected target year.
Status	Successfully tested

T26 - Few-Shot Examples

UC / FR	UC3a, FR4.9
Action	Add few-shot examples to an extraction configuration.
Expected Result	The examples are stored and included in the extraction prompt.
Status	Successfully tested

T27 - Present Extraction Result

UC / FR	UC3a, FR4.10
Action	Open a generated extraction result after extraction execution.
Expected Result	The extraction result is displayed together with suitable metadata.
Status	Successfully tested

T28 - Extraction Versioning

UC / FR	UC3a, FR4.11
Action	Release multiple versions of an extraction configuration and retrieve a previous version.
Expected Result	Each released extraction configuration receives a version, and previous versions remain retrievable.
Status	Successfully tested

T29 - Extraction Draft Management

UC / FR	UC3a, FR4.12
Action	Create and edit a draft extraction configuration before releasing it as a new version.
Expected Result	The draft can be modified without replacing the released extraction configuration version.
Status	Successfully tested

T30 - Execute Extraction Draft

UC / FR	UC3a, FR4.13
Action	Execute a draft extraction configuration before releasing it as a new version.
Expected Result	The system executes the draft extraction configuration and generates an extraction result.
Status	Successfully tested

T31 - Select Extraction Execution Model

UC / FR	UC3a, FR4.14
Action	Select an LLM before executing an extraction configuration.
Expected Result	The extraction is executed using the selected LLM.
Status	Successfully tested

T32 - Converter - Supported File Types

UC / FR	FR5.1
Action	Input a PDF, XLSX, DOC and DOCX to the converter
Expected Result	All input files are converted to a markdown representation.
Status	Successfully tested

T33 - Report Execution Status

UC / FR	FR3.11
Action	Execute a report.
Expected Result	The report status shows that it is running. Completed reports show that they are completed.
Status	Successfully tested

T34 - Extraction Execution Status

UC / FR	FR4.15
Action	Execute an extraction.
Expected Result	The extraction status shows the statuses for it being running, having failed.
Status	Successfully tested

5.6.5 Automated Test Coverage and Success Rate

Test Type	Number of Tests	Success Rate	Code Coverage
VaultReport.Application Unit Tests	35	100%	78%
VaultReport.WebUI Component / Unit Tests	93	100%	88%
VaultReport.Extractor Unit Tests	20	100%	78%

Table 5.13: Automated unit and component test results

5.7 Proof of Concept Report: Executive Summary

This part of the evaluation describes the development and use of a proof of concept report in the form of an executive summary. The purpose of this proof of concept was not to perform a formal system test of individual functions, but to evaluate whether the developed system could be applied to a realistic reporting scenario defined together with Aiger Group AG.

The executive summary was selected because it represents a realistic and stakeholder-oriented report format. It combines company master data, management information, market positioning, products and services, financial information, and financing requirements into one coherent document. This made it suitable for evaluating whether structured LLM-based extraction results could be transformed into a readable business report using a template-driven reporting approach.

5.7.1 Evaluation Objective

The proof of concept was used to evaluate the practical feasibility of the reporting workflow. More specifically, it was intended to show whether the system can:

- extract a defined set of business-relevant data points from organised company documents,
- make the extraction results available to a report template,
- generate a coherent executive summary from these results,
- preserve source grounding for extracted information,
- represent missing or not applicable information explicitly,
- and support stakeholder review through an understandable report artefact.

Unlike a classical test case, this proof of concept does not validate one isolated requirement at a time. Instead, it evaluates whether the developed system works as an integrated workflow from extraction configuration to generated report output.

5.7.2 Procedure

The report was developed iteratively and in close exchange with Aiger Group AG. At the beginning, there were no clear set of fixed requirements defining the exact structure, level of detail, or formatting of the executive summary. Therefore, an exploratory approach was chosen.

The development followed a pragmatic cycle:

1. An initial report structure was drafted based on typical information needs in an executive summary.
2. Required information elements were translated into extraction definitions.
3. The extraction results were collected and referred in the Scriban report template.
4. Generated report versions were reviewed together with Aiger Group AG.
5. The structure, wording, extracted fields, and presentation logic were refined based on feedback.

This iterative approach allowed the proof of concept to evolve from a rough reporting idea into a concrete report.. It also helped identify which information could be represented as direct structured values and which information required concise summarisation or interpretation.

5.7.3 Report Artefact

The final proof of concept report was implemented as a Scriban template. Scriban was used to define the static structure of the report while dynamically inserting extracted values during report generation.

The report contains the following main sections:

- company master data,
- mission,
- company and management,
- shareholder information,
- products and services,
- market,
- competitive advantages,
- financials,
- and financing requirements.

The template combines different presentation formats depending on the type of information. Short values are rendered inline, explanatory content is rendered as paragraphs, and repeated structured values are rendered as tables or lists. This made it possible to generate a report that is both structured and readable.



Figure 5.1: Excerpt of the generated Executive Summary

5.7.4 Template Logic

The Scriban template contains additional logic to make the report more robust and transparent. Helper functions are used to check whether values exist, whether an extraction was successful, whether information was not found, and whether information was not applicable. This prevents missing information from being silently ignored or replaced by unsupported assumptions.

The report also includes formatting helpers, for example for number formatting and textual list joining. Source links are inserted where available, allowing extracted values to be traced back to their underlying source. In addition, the template supports redaction logic for sensitive information such as customer names or sales pipeline names.

5.7.5 Extraction Coverage

The proof of concept required a broad set of extraction definitions, in total 89, to provide the information needed by the executive summary. These extractions covered company identity, management, governance, market, products and services, competitive positioning, financials, and financing-related information.

5.7.6 Evaluation Result

The proof of concept demonstrated that the developed system can generate a stakeholder-oriented report from structured extraction results. The executive summary showed that LLM-based extraction and template-driven report generation can be combined in a practical workflow.

The most important result was not only that a report could be generated, but that the report could be generated from a completed set of defined extractions. This showed that the system is capable of connecting extraction configuration, extraction execution, structured results, template rendering, and final report generation.

The proof of concept also showed that a template-based approach is suitable for this use case because it gives control over the final report structure. Instead of asking the LLM to generate an entire report freely, the system uses the LLM primarily for targeted extraction and controlled summarisation. The report template then determines how the information is presented. This reduces the risk of uncontrolled report generation and makes the output easier to review.

5.7.7 Limitations

The proof of concept should not be interpreted as a full validation of all possible report types. It focused on one representative report format, the executive summary. Other report types may require different extraction definitions, different presentation logic, or additional stakeholder review.

Furthermore, the development of the report was exploratory. This means that the report structure was refined during the process rather than derived from a complete set of predefined reporting requirements. While this was appropriate for the proof of concept, a production-ready reporting solution would require more formal agreement on report structure, terminology, review responsibilities, and acceptance criteria for the template as well as for the extractions.

Another limitation is that the quality of the generated report depends on the quality and completeness of the underlying source documents and extraction definitions. If relevant information is missing from the document set, the system can indicate that the information was not found, but it cannot replace missing business data or too vaguely defined requirements as well as the lack of fine-tuning through the addition of more real-life examples in extraction prompts.

6 Discussion

6.1 Extraction Pipeline Testing and Requirement Alignment

The empirical results demonstrate that the platform successfully satisfies the project’s engineering and performance criteria.

6.1.1 Functional Verification

The pipeline successfully enforced source grounding (FR 4.2) and schema validation (FR 4.3). Across all 340 test executions, zero instances of data fabrication or hallucination occurred. The system demonstrated a safe failure mode: errors were strictly confined to typographic discrepancies or minor context omissions, which were intercepted by the autonomous LLM Judge or schema validations.

6.1.2 Extraction Types: LangExtract versus Instructor

The quantitative results show that both pipelines met our goals for accuracy (NFR 1.1) and consistency (NFR 1.2). But instead of just comparing which percentage is higher, the real takeaway is how these two tools actually handle data:

- **Source Grounding via Span Extraction:** LangExtract is best for exact numbers and specific financial figures. Because it forces the model to link every value to an exact snippet of the original text, it prevents the model from hallucinating or altering information. This physical grounding mechanism is the main reason it is so reliable for fixed fields.
- **Structured Semantic Synthesis:** Instructor is meant for situations where the model needs to paraphrase or summarise instead of copying word-for-word. This gives it the flexibility needed for text summaries, but having that freedom sometimes means it pulls in extra, unnecessary context. To maintain precision, the prompt instructions must define the exact scope of the requested data.

6.1.3 Test Limitations and Production Realism

While the quantitative metrics demonstrate exceptional baseline capabilities, the benchmark was conducted under controlled testing conditions that do not fully mirror our end-to-end production environment. This highlights a few reasons why these test results might not translate perfectly to a live production environment:

- **Context Volume and Scaling**

Testing was performed primarily on isolated, individual files rather than full multi-document folders. Consequently, the pipeline was not systematically evaluated at its upper operational limit of around 64,000 characters. In production, processing maximum-capacity contexts may introduce higher cognitive load on the LLM, potentially increasing retrieval degradation or context dilution.

- **Bypassing of Pipeline Mechanics**

Because the evaluation focused on direct document-to-model extraction, the upstream chunking and merging algorithms were omitted from the quantitative benchmark. In a live environment, improper chunk boundaries or poor merging logic could truncate critical data spans, introducing a layer of operational error not captured in these tests.

- **Robustness to Absent Data**

On a positive note, the evaluation included negative test cases where the requested information was entirely absent from the source documents. The system accurately handled these scenarios by returning null values without generating false information, confirming that the baseline model is highly resilient against hallucination when data is missing.

Therefore, while the system’s ability to handle negative cases is verified, the reported accuracy rates should still be interpreted as an optimal baseline of the model’s extraction competence under isolated conditions, rather than a definitive guarantee of end-to-end production accuracy.

6.2 Evaluation Insights

6.2.1 Upstream OCR Bottlenecks

The consistent failure of systems like Docling with German special characters (ä, ö, ü) and the structural collapse of all tested models when processing complex tables pose a major operational risk. In a due diligence context, financial audits depend entirely on multi-level tax declarations and signed agreements. Because handwritten signatures and blurred table borders are not always correctly recognised, the downstream LLM pipeline could frequently be forced to operate on corrupted or incomplete structural text. While Qwen 3.6 emerged as the most viable component, this bottleneck indicates that human verification is still required, either at the conversion gate before extraction or as a final review of the generated report to ensure the data is indeed correct.

6.2.2 Splitting Extractions Across Multiple Models

The category-level insights from the LLM evaluation reveal that open-source models display distinct operational strengths rather than uniform capability. While Gemma-4 (31B) secured the highest overall success rate due to its perfect performance in structured mathematical reasoning, it was outpaced in precision by GLM-4.7-Flash for raw data retrieval and by Qwen 3.6 for high-level legal interpretation.

This non-uniform performance suggests that relying on a single monolithic model for all due diligence tasks may be architecturally suboptimal. A valuable architectural evolution would be the implementation of a dynamic routing gateway. By automatically assigning data retrieval tasks to GLM, mathematical reconciliations to Gemma and semantic clause interpretations to either Gemma or Qwen, the platform could leverage the specialised accuracy of each open-source model while maintaining a strictly on-premises, privacy-preserving footprint.

Lastly, to achieve a more definitive evaluation, testing would need to expand beyond just the 52 unique test cases evaluated as part of this thesis. A larger, more diverse dataset is required to confirm whether these specific operational specialisations remain consistent across a broader range of corporate edge cases.

6.3 Scope and Systemic Limitations

6.3.1 Representativeness of the AIGER Dataset

Evaluating the system solely on data from Aiger Group AG provides a strong baseline, but it relies on their specific document structures and formatting styles. While the folder structure is guaranteed to remain the same, real-world datasets from other companies will not mirror the Aiger dataset perfectly. Because corporate files vary dramatically between organisations, deploying the system on data from a different company could cause unexpected drops in extraction accuracy or require adjustments to the prompt boundaries.

6.3.2 Single Year Extraction Limitations

Extractions are restricted to a single target year to conserve token capacity, preventing native cross-year reasoning within the pipeline. Multi-year trend analysis must instead be handled within the Scriban templates using separate historical variables.

This single-year constraint introduces a major operational problem: it completely misses historical documents that are still active. For instance, a long-term contract, lease agreement, or historical policy might only be uploaded once during a past year’s audit. Even though that document remains completely relevant to the current year’s due diligence, the pipeline will entirely ignore it because it only looks at files assigned to the current target year. To capture this information, users are forced to manually re-upload the same historical documents every single year.

7 Conclusion

This thesis successfully delivered an end-to-end automation platform that transforms a traditionally manual corporate due diligence workflow into an automatic data pipeline. The testing phase confirmed that the system satisfies the target thresholds for both extraction accuracy and extraction consistency, proving the viability of open-source LLMs in a privacy-preserving, on-premises environment.

Furthermore, the executive summary proof of concept provided concrete evidence of the system’s practical applicability. By demonstrating the feasibility of combining LLM-based data extraction with Scriban-based report generation, the prototype proved that raw, unstructured data can be successfully transformed into coherent report artefacts. This evaluation showed that the system is not limited to isolated extraction tasks; rather, it demonstrates how extracted information can be organised within a stakeholder-oriented reporting workflow, supported by source references, missing value handling, and controlled template-based output.

8 Outlook

This chapter outlines directions in which the system could be developed further, both as part of the handover to Aiger Group AG and as longer-term improvements.

8.1 Handover and integration

The system is intended as a foundation that Aiger Group AG will take into production. Integrating it into their existing application and their user and company context is a substantial effort in its own right and lies with Aiger Group AG. Beyond that broader effort, a few specific changes follow directly from the prototype built here. The user interface is one such case. The SvelteKit frontend developed in this work serves mainly to demonstrate the system and is not meant for production use, so Aiger Group AG will provide their own. Two further changes also belong to this stage. The embedded SQLite database would be replaced by a client-server database suited to production use, as discussed in the scalability section. In addition, a file abstraction would be integrated so that extraction results can be invalidated when their source documents change, which the current system cannot detect because the backend has no access to the files.

8.2 Extraction and reporting features

Several extensions would broaden what can be extracted and how reports are produced. Extractions could be made to span multiple years rather than a single target year, which requires managing the larger context this entails so that it still fits within the model. Results could carry a change history, so that any later edit remains traceable back to the original extraction.

On the reporting side, PDF export is the most prominent missing output. It requires pagination and a dedicated render path that adds a bibliography and replaces inline source links with references to it, instead of the inline links used in the current rendering. Templates could also be made more approachable, so that less template code is needed to produce a report and missing data is handled implicitly rather than by the template author.

Finally, the extraction itself could move away from LangExtract and Instructor toward a tool tailored to the problem. The two libraries were a deliberate choice for this work, but once the requirements have stabilised, a purpose-built extractor could fit the domain more closely.

8.3 Performance

The system currently processes each extraction independently, even when several extractions within a report draw on the same source documents. Processing such extractions together, so that shared documents are retrieved and handled only once, would reduce redundant work and the load placed on the LLM provider.

9 References

- [1] L. Ouyang *et al.*, ‘OmniDocBench: Benchmarking Diverse PDF Document Parsing with Comprehensive Annotations’. Accessed: Feb. 24, 2026. [Online]. Available: <https://opendatalab.com/omnidocbench>
- [2] L. Fu *et al.*, ‘OCRBench v2: An Improved Benchmark for Evaluating Large Multimodal Models on Visual Text Localization and Reasoning’. Accessed: Feb. 24, 2026. [Online]. Available: https://99franklin.github.io/ocrbench_v2/
- [3] I. Endless Labs, ‘Datalab Benchmark’. Accessed: Feb. 24, 2026. [Online]. Available: <https://www.datalab.to/benchmark/overall>
- [4] T. A. I. for Artificial Intelligence, ‘olmOCR-Bench’. Accessed: Feb. 24, 2026. [Online]. Available: <https://github.com/allenai/olmocr/tree/main/olmocr/bench>
- [5] E. AI, ‘OCR Arena’. Accessed: Feb. 24, 2026. [Online]. Available: <https://www.ocrarena.ai/leaderboard>
- [6] G. Research, ‘LangExtract: Structured Information Extraction with Language Models’. Accessed: May 09, 2026. [Online]. Available: <https://github.com/google/langextract>
- [7] L. Team, ‘LangStruct: Schema-Driven LLM Structured Extraction Framework’. Accessed: May 09, 2026. [Online]. Available: <https://langstruct.dev/>
- [8] S. NLP, ‘DSPy: The framework for programming - not prompting - language models’.
- [9] J. Liu, ‘Instructor: Structured LLM Outputs’. Accessed: May 20, 2026. [Online]. Available: <https://github.com/jxnl/instructor>
- [10] S. Colvin and others, ‘Pydantic: Data Validation and Settings Management Using Python Type Hints’. Accessed: June 11, 2026. [Online]. Available: <https://docs.pydantic.dev/>
- [11] G. Authors, ‘GLiNER2: Lightweight Generalist Named Entity Recognition Model’. Accessed: May 09, 2026. [Online]. Available: <https://github.com/fastino-ai/GLiNER2>

- [12] H. Guo and Z. Wei, ‘Hidden-in-Plain-Text: A Benchmark for Social-Web Indirect Prompt Injection in RAG’. Accessed: May 14, 2026. [Online]. Available: <https://arxiv.org/abs/2601.10923>
- [13] A. A. Masoud, M. Arazzi, and A. Nocera, ‘SD-RAG: A Prompt-Injection-Resilient Framework for Selective Disclosure in Retrieval-Augmented Generation’. Accessed: May 14, 2026. [Online]. Available: <https://arxiv.org/abs/2601.11199>
- [14] B. Ramakrishnan and A. Balaji, ‘Securing AI Agents Against Prompt Injection Attacks’. Accessed: May 14, 2026. [Online]. Available: <https://arxiv.org/abs/2511.15759>
- [15] OpenAI, ‘Prompt Caching’. 2024.
- [16] X. Chu, H. Wang, L. Zhang, Y. Zhou, and J. Shen, ‘Leveraging LLMs for Structured Information Extraction and Analysis from Cloud Incident Reports’, *arXiv preprint arXiv:2603.16818*, Accessed: June 06, 2026. [Online]. Available: <https://arxiv.org/abs/2603.16818>
- [17] V. Wang and E. Stengel-Eskin, ‘Calibrating Verbalized Confidence with Self-Generated Distractors’, in *Proceedings of the International Conference on Learning Representations (ICLR)*, Accessed: June 06, 2026. [Online]. Available: <https://arxiv.org/abs/2509.25532>
- [18] J. H. A. P. M. B. F. P. P. L. Nelson F. Liu Kevin Lin, ‘Lost in the Middle: How Language Models Use Long Contexts’. Accessed: June 01, 2026. [Online]. Available: <https://arxiv.org/abs/2307.03172>
- [19] F. D. T. B. R. A. R. S. Y. H. S. Ali Modarressi Hanieh Deilamsalehy, ‘NoLiMa: Long-Context Evaluation Beyond Literal Matching’. Accessed: June 01, 2026. [Online]. Available: <https://arxiv.org/abs/2502.05167>
- [20] Z. Org, ‘GLM-OCR’. Accessed: June 06, 2026. [Online]. Available: <https://huggingface.co/zai-org/GLM-OCR>
- [21] C. Auer *et al.*, ‘Docling Technical Report’. Accessed: June 06, 2026. [Online]. Available: <https://docling-project.github.io/docling/>
- [22] T. Team, ‘Tesseract OCR’. Accessed: June 06, 2026. [Online]. Available: <https://github.com/tesseract-ocr/tesseract>

- [23] P. Team, ‘PaddleOCR-VL’. Accessed: June 06, 2026. [Online]. Available: <https://huggingface.co/PaddlePaddle/PaddleOCR-VL>
- [24] D. AI, ‘DeepSeek-OCR’. Accessed: June 06, 2026. [Online]. Available: <https://huggingface.co/deepseek-ai/DeepSeek-OCR>
- [25] NVIDIA, ‘Llama-3.1-Nemotron-Nano-VL-8B-V1’. Accessed: June 06, 2026. [Online]. Available: <https://huggingface.co/nvidia/Llama-3.1-Nemotron-Nano-VL-8B-V1>
- [26] Q. Team, ‘Qwen3.5-35B-A3B’. Accessed: June 06, 2026. [Online]. Available: <https://huggingface.co/Qwen/Qwen3.5-35B-A3B>
- [27] Gemma Team, Google DeepMind, ‘Gemma 4: Open Multimodal Models with Configurable Reasoning’. Accessed: May 24, 2026. [Online]. Available: <https://huggingface.co/google/gemma-4-31b>
- [28] Zhipu AI Team, ‘GLM-4.7-Flash: Lightweight Open-Source Models for Long-Context Agentic Tasks’. Accessed: May 24, 2026. [Online]. Available: <https://huggingface.co/zai-org/GLM-4.7-Flash>
- [29] Mistral AI Team, ‘Minstral 3 14B: High-Efficiency Edge Multimodal Transformers’. Accessed: May 24, 2026. [Online]. Available: <https://huggingface.co/mistralai/Minstral-3-14B-Instruct-2512-GGUF>
- [30] Qwen Team, Alibaba Cloud, ‘Qwen3.6: Multimodal Hybrid-Thinking Architecture and Open Weights Series’. Accessed: May 24, 2026. [Online]. Available: <https://huggingface.co/Qwen/Qwen3.6-27B>