Vrije Universiteit Amsterdam

Universiteit van Amsterdam

Master Thesis

Tool Support for Semi-Automated Evaluation of Software Architecture by Leveraging Large Language Models

Author: Anusha Annengala (2853724)

1st supervisor:	Prof. Patricia Lago
daily supervisor:	Iffat Fatima
2nd reader:	Prof. Vincenzo Stoico

A thesis submitted in fulfillment of the requirements for the joint VU-UvA Master of Science degree in Computer Science

June 29, 2025

"I am the master of my fate, I am the captain of my soul"
from Invictus*, by William Ernest Henley*

Tool Support for Semi-Automated Evaluation of Software Architecture by Leveraging Large Language Models

Lappeenranta-Lahti University of Technology LUT
LUT School of Engineering Science
Software Engineers for Green Deal
2025
Anusha Annengala
Examiners: Professor Patricia Lago (Vrije Universiteit Amsterdam)
Professor Vincenzo Stoico (Vrije Universiteit Amsterdam)

ABSTRACT

Lappeenranta-Lahti University of Technology LUT
LUT School of Engineering Science
Software Engineering

Anusha Annengala

Tool Support for Semi-Automated Evaluation of Software Architecture by Leveraging Large Language Models

Master's thesis, 2025

Supervisors: Prof. Patricia Lago, Iffat Fatima
Examiners: Professor Patricia Lago (Vrije Universiteit Amsterdam)
Professor Vincenzo Stoico (Vrije Universiteit Amsterdam)

Keywords: Software Architecture, Sustainability, Large Language Models.

Context. There is a rise in the recognition of sustainability as a key concern in software architecture. There are structured ways to evaluate this, such as the Software Architecture Assessment for Sustainability, but they require manual effort and are often time-consuming.
Goal. The goal of this study is to reduce the time and manual effort required to evaluate the software architecture for sustainability by semi-automating steps that are time-consuming and require manual effort using Large Language Models (LLMs).
Method. We follow a design science research methodology to identify the requirements of the artifact and steps from the Software Architecture Assessment for Sustainability that can be benefited by automation, design, and develop the artifact that semi-automates the Software Architecture Assessment for Sustainability, and evaluate the artifact using quantitative metrics that calculate the semantic similarity between the output of the artifact and manual ground truth.
Results. We provide an artifact to semi-automate the Software Architecture Assessment for Sustainability.
Conclusions. Large Language Models can support the Software Architecture Assessment for Sustainability by providing a semi-automated tool.

DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

Università degli Studi dell'Aquila

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Corso di Laurea in Informatica

Tesi di Laurea Magistrale

Tool Support for Semi-Automated Evaluation of Software Architecture by Leveraging Large Language Models

Relatore

Prof. Patricia Lago

Laureando

Anusha Annengala

Correlatore

Prof. Vincenzo Stoic

Anno Accademico 2024-2025

SE4GD
Software Engineers for Green Deal

With the support of the Erasmus+ Programme of the European Union

UNIVERSITÀ DEGLI STUDI DELL'AQUILA

This thesis has been accepted by partner institutions of the consortium (619839-EPP-1-2020-1-FI-EPPKA1-JMD-MOB). Successful defence of this thesis is obligatory for graduation with the following national diplomas:

- Master of Computer Science (University of L'Aquila)
- Master of Science in Technology (LUT University)
- Master of Computer Science (Vrije Universiteit Amsterdam)

This master's thesis was conducted at Vrije Universiteit Amsterdam. The supervision of this work was carried out by Prof. Patricia Lago and Iffat Fatima, who provided academic guidance and support throughout the research process. This thesis was examined and graded at Vrije Universiteit Amsterdam.

DECLARATION OF AI USAGE

I hereby acknowledge that the research reported in this thesis has been carried out with the responsible and ethical utilization of artificial intelligence (AI) technologies. The AI tool, ChatGPT-4 was utilized to assist in specific tasks, including:

- **Language and Grammar Assistance:** ChatGPT-4 was used to enhance the clarity and coherence of the text, ensuring proper grammar and style.
- **Learning:** AI-powered resources were instrumental in my learning process, helping me to clarify complex concepts, test my understanding through interactive questioning, and explore related research areas.

However, it is important to emphasize that the core ideas, research design, results, interpretations, and conclusions presented in this thesis are solely the product of my own intellectual effort. AI served as a supportive tool, not a substitute for critical thinking and independent research.

I acknowledge the limitations of AI and the potential for biases within its algorithms. Therefore, I have carefully reviewed and validated all AI-generated outputs to ensure their accuracy and alignment with the research objectives.

I accept complete accountability for the material presented in this thesis and assert that it complies with the utmost standards of academic integrity and ethical research practices.

ACKNOWLEDGMENTS

I would like to thank my supervisors Professor Patricia Lago and Iffat Fatima, for their guidance and support throughout this thesis. I am thankful for my friends and family for their continuous support and motivation throughout this journey.

Anusha Annengala
The Netherlands, June 2025

Abstract

Context. There is a rise in the recognition of sustainability as a key concern in software architecture. This includes long-term maintenance, adaptability, and energy efficiency. There are structured ways to evaluate software architecture for sustainability, such as the Software Architecture Assessment for Sustainability, but they require manual effort for tasks such as information extraction from documentation and are often time-consuming.

Goal. The goal of this study is to reduce the time and manual effort required to evaluate the software architecture for sustainability by semi-automating steps that are time-consuming and require manual effort, particularly the extraction of quality attributes and architectural design decisions using Large Language Models (LLMs).

Method. We follow a design science research methodology to identify the requirements of the artifact and steps from the Software Architecture Assessment for Sustainability that can be benefited by automation, design, and develop the artifact that semi-automates the Software Architecture Assessment for Sustainability, and evaluate the artifact using quantitative metrics that calculate the semantic similarity between the output of the artifact and manual ground truth.

Results. We provide an artifact to semi-automate the Software Architecture Assessment for Sustainability. The artifact has strong semantic similarity with the manual ground truth. The evaluation of the artifact resulted in the BERTScore F1 of 0.86 for the extraction of quality attributes and 0.85 for the extraction of architectural design decisions, and manual verification of the evaluation also aligns with this metric. This demonstrates the artifact's ability to support the Software Architecture Assessment for Sustainability while reducing the time consumed and the manual effort.

Conclusions. Large Language Models can support the Software Architecture Assessment for Sustainability by providing a semi-automated tool. Our metrics

show that there is a high semantic similarity between the output of the artifact and the manually extracted results that serve as the ground truth. The tool provided in this study reduces the manual effort and time required to complete the assessment of software architecture for sustainability.

Contents

List of Figures

List of Tables

1

Introduction

1.1 Motivation

The Software Architecture Assessment for Sustainability (1) provides a set of general steps to evaluate software architecture for sustainability. These steps are manual processes, and some of the steps that involve extracting relevant information from documentation are often time consuming. Our motivation is to support software architects following the assessment framework in evaluating software architecture for sustainability by automating some of the time-consuming processes.

To achieve this, we leverage the capabilities of Large Language Models (LLMs). Large Language Models are a class of Artificial Intelligence (AI) systems trained on huge amounts of data. They are capable of understanding and generating human-like language and have demonstrated promising capabilities in tasks that involve information retrieval, summarization, and natural language inference (2). Motivated by these strengths, we propose to leverage LLMs to automate specific steps within the Software Architecture Assessment for Sustainability. Our goal is to support software architects by reducing the time required for manual information extraction.

1.2 Problem Definition

The Software Architecture Assessment for Sustainability (1) provides a set of general steps to evaluate software architecture for sustainability. These steps are manual and time-consuming processes. In a preliminary analysis of the assessment framework, some of the time-consuming steps are extracting relevant information from software architectural documentation. These steps involve identifying quality attributes, architectural design

decisions, and trade-offs from technical requirements and software architectural documentation. We aim to provide tool support by automating some of the time-consuming manual processes for assessment of software architecture by leveraging LLMs.

1.3 Research Question

RQ - How can we support software architects using the Software Architecture Assessment for Sustainability to reduce the time required for sustainability assessments in software architecture?

To answer this research question, we adopt a Design Science Research (DSR) methodology. We develop a semi-automated tool that leverages Large Language Models (LLMs) to automate some of the time-consuming manual processes of the Software Architecture Assessment for Sustainability. Specifically, we automate the extraction of quality attributes and architectural design decisions, and present the output in a structured format that aligns with the sustainability assessment steps proposed by Fatima et al. (1).

1.4 Scientific Contribution

This research supports the Software Architecture Assessment for Sustainability (1) by providing tool support for semi-automated assessment of Software Architecture by Leveraging Large Language Models. The study domain lies at the intersection of Software Engineering, Sustainability, and LLM driven automation. This study provides tool support to automate the Software Architecture Assessment for Sustainability following a design science research methodology.

2

Background

In this section, we provide an overview of the necessary background of Software Architecture Assessment for Sustainability, Large Language Models, and LLM Automation in Software Engineering. The related work in this area of research is discussed after this section.

2.1 Software Architecture Evaluation for Sustainability

The Software Architecture Assessment for Sustainability (1), provides a general framework to evaluate software architecture with respect to sustainability. The method consists of the following general steps:

2.1.1 Preparation

In this phase, the context of the evaluation is defined. We identify the software system that is to be evaluated, collect all relevant architectural documentation, and communicate with stakeholders about the scope and purpose of the evaluation. This part ensures that our assessment aligns with the context of the project.

2.1.2 Requirement Identification

In this phase, we identify the requirements for the assessment of the software architecture. The quality attributes of the Software Architecture are identified from technical requirements documentation and validated by experts.

2.1.3 Goal Identification

In this phase, the objectives of the assessment of the software architecture are identified.

2.1.4 Method Presentation

The assessment process is explained to stakeholders to match their expectations and clarify all the steps and the roles that different participants (e.g., architects, developers) will have in the evaluation process.

2.1.5 Architecture Presentation

The architecture of the software system is presented in detail by the stakeholders. They often include architectural views, architectural design decisions, and technical requirements.

2.1.6 Prioritization

The identified quality attributes are ranked in order by stakeholders, prioritizing the identified quality attributes. This ordering helps to understand the importance of each quality attribute in assessing sustainability and is helpful for future quantitative assessments.

2.1.7 Identification of Architectural Approaches

By analyzing the software architectural documentation, architectural design decisions are extracted from the documentation.

2.1.8 Generation of Data for Analysis

In this phase, the extracted architectural design decisions and quality attributes are documented that act as data for evaluation in the next phase.

2.1.9 Evaluation of Obtained Data

For each architectural design decision, we evaluate how it affects the prioritized quality attributes. These evaluations are captured in decision matrices that show whether the decision has a positive, negative, or no effect on each quality attribute. Using these decision matrices and knowing the order of importance of quality attributes, we calculate the sustainability impact score (SIS) for all identified architectural design decisions. SIS provides the weighted impact of the design decision on quality attributes and provides a quantitative comparison between alternative decisions.

2.1.10 Improve and Take Action

Based on SIS and qualitative analysis, trade-offs are identified. It is possible to reconsider design decisions that negatively impact sustainability, and suggestions for improvements are made.

2.1.11 Presentation of Results

Finally, the outcomes of the assessment are documented and reported to stakeholders. The rationale behind design decisions, trade-offs made, and recommended actions are clearly documented. Although the assessment method provides general steps to evaluate the software architecture for sustainability, these general steps are time-consuming and manual processes. In particular, steps that involve the extraction of relevant information from documentation consume a lot of time. Our study explores the possibility of using Large Language Models (LLMs) to partially automate the evaluation process. By leveraging the capabilities of LLMs, we aim to reduce the time and manual effort involved in identifying relevant information that helps the assessment process.

2.2 Large Language Models

Large Language Models (LLMs) represent advanced forms of artificial intelligence that have been trained on substantial data through deep learning methods. These models can understand complex linguistic and semantic patterns. This enables them to carry out a variety of tasks related to text generation, such as creating summaries, responding to queries, understanding natural languages, extracting information, and producing code. Some instances include GPT models from OpenAI (3, 4), LLaMA models from Meta (5), Google's Flan-T5 (6) and Mistral model (7).

In the field of software engineering, Large Language Models (LLMs) have recently become notable due to their ability to assist with intricate tasks that demand substantial knowledge. They have been applied in areas such as requirements engineering (**?**), analyzing information for software developers (8), and extracting architectural information (9). Their primary advantage is handling unstructured or semi-structured textual data, which are usually found in architectural documents. When combined with techniques such as prompt engineering and retrieval-augmented generation (RAG), LLM can be guided to achieve structured results (8, 9).

Recent empirical studies (9) show that LLMs can generate architectural design decisions when given sufficient context. In our study, we investigate the use of LLMs to automate some steps in the Software Architecture Assessment for Sustainability (1). Our goal is to reduce the time and effort required for manual extraction of necessary information while still keeping traceability to source documents and allowing human-in-the-loop validation.

2.3 LLM Automation in Software Engineering

Studies in Large Language Models (LLMs) have created new opportunities for automating tasks in software engineering that usually needed a lot of human effort. LLMs like OpenAI's GPT (4), Meta's LLaMA (5) and Google's Flan-T5 (6) are pre-trained using huge amounts of data, providing good abilities in understanding natural language, reasoning, and generation of information. These qualities make them good at dealing with unstructured or semi-structured documents found in the software documentations.

In the field of software engineering, LLMs have been effectively used for various tasks such as analyzing requirements and creating user stories (**?**), summarizing software documentation (10), generating architectural knowledge (9), and producing design rationale (11). These tasks usually require the understanding of technical documents written in natural language. This is a strong point for LLMs as they can generalize information and understand context.

A main benefit of LLMs in this context is their ability to carry out few-shot and zero-shot inference, which means that automation is possible without the need for retraining specific to a domain. In addition, combining LLM with methods such as prompt engineering and retrieval-augmented generation (RAG) helps create more grounded output that is task-focused (3, 12, 13). These abilities tackle major hindrances in software engineering processes such as manually extracting information from software documentation while maintaining consistency and uniformity.

Current research continues to explore how Large Language Models (LLMs) can be methodically combined with software engineering tools for providing useful and accurate automation. Our work builds on this path by developing a semi-automated tool that leverages LLMs to support Software Architecture Assessment for Sustainability.

3

Related Work

Dhar et al. (9) provides an empirical exploratory study on determining if Large Language Models (LLMs) can be used to generate architectural design decisions. They consider GPT and T5 based models with 0-shot, few-shot, and fine-tuning approaches for the evaluation. GPT-4 outperforms T5 based models in 0-shot and few-shot prompt approaches but achieves comparable results with fine tuning the model. The evaluation metrics used in this study are NLP metrics, ROUGE-1, BLEU, METEOR, and BERTScore. The findings suggest that Large Language Models can be utilized to generate architectural design decisions, although there needs to be further research to generate architectural design decisions at a human level. The focus of our work is to extract relevant information from architectural documentation to support Software Architecture Assessment for Sustainability. We do not explore generating architectural design decisions or architectural decision records based on context but rather focus on extracting architectural design decisions from software architectural documents which could include architectural decision records. We also extract quality attributes from technical requirement documents. Both studies utilize Large Language Models in the context of architectural design decisions and NLP metrics for evaluation, but the objectives of the study differ from each other.

Oswal et al. (14) provides an implementation of GPT-3.5 to convert unstructured software requirements into structured user stories to help agile development methods. They describe an application that uses the few-shot prompt engineering using GPT-3.5 to automate the process of turning requirement texts into regular user story formats, this greatly cuts down on manual effort. They considered zero-shot prompting as well but achieved better structured results with the use of few-shot prompt engineering. Their application results in user stories in JSON format displayed on a user interface. While Oswal et al. (14) aims to support the agile development methods by converting software requirements

into structured user stories, we aim to automate some of the general steps of the Software Architecture Assessment for Sustainability in order to reduce manual effort required. Our study uses few-shot prompt engineering along with retrieval augmented generation to extract architectural design decisions and quality attributes in JSON format to display on a user interface, but the context of our study differs as we provide tool support to automate the Software Architecture Assessment for Sustainability.

Abrahamyan and Fard (8) present StackRAG, a retrieval-augmented multiagent generation tool based on LLMs that acts as a question-answer agent for software developers. The aggregate data from Stack Overflow posts to enhance the reliability of the answers of the tool. The motivation behind this tool is that software developers spend a lot of time finding information relevant to their questions related to software development. The provide this tool which uses the data aggregated from Stack Overflow posts to provide reliable answers to Software Developers. This tool is evaluated using accuracy, correctness, relevance, and usefulness metrics with a score between 1-5. Our study leverages the capabilities of Large Language Models to extract relevant information to support the Software Architecture Assessment for Sustainability. While StackRAG provides a retrieval-augmented multiagent generation tool to enhance the reliability of the answers of the tool, we utilize retrieval-augmented generation to extract reliable information from context documents. The context of the study differs with different input and output requirements, StackRAG helps software developers by answering their questions, and in our study, we aim to support the Software Architecture Assessment for Sustainability.

Seyedi and Avgeriou (15) present DRMiner, an approach that extracts design rationales from Jira issue logs by leveraging large language models (LLMs) along with prompt engineering and custom heuristics. It leverages LLM based extraction and not traditional methods of classification and achieves improvements over prior Machine Learning and Deep Learning baselines. DRMiner extracts design rationales as pairs of solutions and it's corresponding rationale. Our study extracts architectural design decisions from software architectural documents, and the context and methodology of extraction differ from DRMiner since the source of extraction is different. We extract architectural design decisions and quality attributes from technical requirement documents and software architectural documents unlike DRMiner, which extracts rationale from Jira issue logs. We also focus on extracting architectural design decisions rather than just the design rationales. Our study aims to support the Software Architecture Assessment for Sustainability by automating some of the manual processes.

Miskell et al. (16) provides a framework that extracts functional software requirements directly from Java source code using LLM-based techniques as well as NLP techniques. The context for this study is to support software developers by extracting functional requirements from source code and hence streaming the software development process. This aim is to bridge the gap between actual implementation and documentation. They utilize GPT based large language models to convert Java source code into software requirement statements which can reduce the overhead of manual documentation. Although our study deals with extracting quality attributes from technical requirements documents, it targets a different layer of abstraction. We extract quality attributes from technical requirement documents and do not consider the actual source code for the extraction process. Our study also provides tool support to semi-automate the software architecture assessment for sustainability.

To the best of our knowledge, there is no prior work addressing automation in software architecture assessment for sustainability. We document some of the prior work in leveraging large language models to extract information relevant to software architecture above, since it closely aligns with the automation tasks addressed in our study. Our study automates some of the steps in the software architecture assessment for sustainability by leveraging large language models.

4

Design

The design science research methodology is a structured process for designing, developing, and validating artifacts. The artifact in our study supports the Software Architecture Assessment for Sustainability (1). Our study attempts to solve an artifact-centric research problem, and hence design science research methodology is followed as it allows for a structured artifact development. This study supports the Software Architecture Assessment for Sustainability (1) by providing tool support for semi-automated evaluation of software architecture by using large language models.

In order to solve our research question, we follow a design science research methodology 4.1 as proposed by Peffers et al. (17). The strategy is divided into separate phases that is, Problem and Requirement Identification, Design and Develop Artifact, and Evaluate Artifact. In the Problem and Requirement Identification phase, we review the general steps from the Software Architecture Assessment for Sustainability and classify them into steps that require manual effort by software architects who could be benefited by leveraging large language models to automate these steps, steps that could be automated by a simple Python script and do not need additional efforts of LLM automation, and the steps that need human input or validation. Based on these requirements, we proceed to the Design and Develop Artifact phase, where we review existing literature on LLM models and LLM approaches to identify and select appropriate LLM models and LLM approaches to develop the artifact. Based on these design choices, we then develop the artifact. Finally, we evaluate the artifact in the Evaluate Artifact phase. We use open-source data from GitHub as our evaluation data. We compare manually extracted quality attributes and architectural design decisions with the output of the artifact through quantitative metrics.

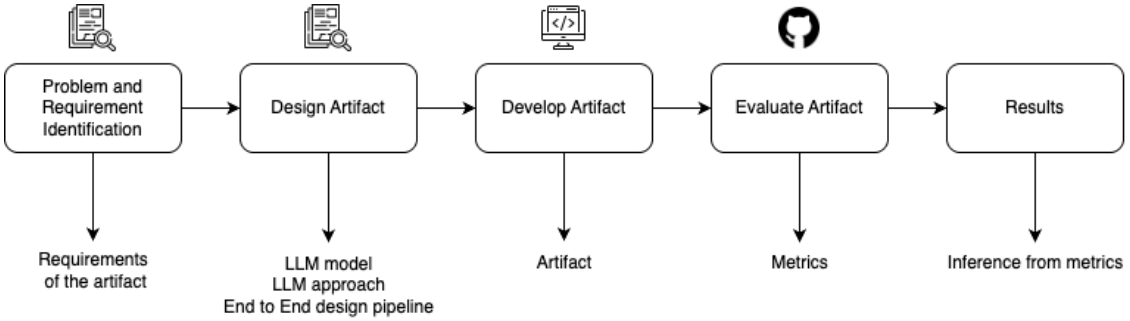

Figure 4.1: Study Design

4.1 Problem and Requirement Identification

In this phase, we define the requirements of the artifact that supports the framework, Software Architecture Assessment for Sustainability (1). We review each of the general steps in the assessment framework and classify them into three categories.

- Steps that require manual effort, which could be reduced by automation via LLM.
- Steps that require manual effort, which could be reduced by simple Python scripting.
- Steps that require human input or validation and cannot be fully automated.

This decision is taken considering each of the general steps individually and is based on the time and manual effort needed and if it could be reduced by LLM automation. For the steps that were not considered for LLM automation, we classify them as ones that would require human efforts or validation, and the steps that could be automated by a simple Python script. The result of this phase is a set of requirements for the artifact to be designed, developed, and evaluated in the next phases of our study.

4.2 Design and Develop Artifact

In the design phase, we review recent literature on LLMs and their applications in tasks related to the extraction of information from documents. We use the search query ("Large Language Models" OR "LLM") AND ("Software Architecture" OR "Software Documentation") AND ("Information Extraction" OR "Information Retrieval" OR "Data Extraction" OR "Data Retrieval") and include papers relevant to the use of LLM for information or data extraction. The main design goal in the context of our study included selecting an

LLM model with high accuracy and reasoning capabilities, while also considering privacy and local deployment needs.

Since the artifact takes architectural documents as input, we consider data privacy to be a key factor when selecting LLM models. Similarly, LLM approaches are selected based on the requirements of the artifact and approaches reviewed from literature which are best suited for the requirements.

We then design an input pipeline and specify the expected output of the artifact. Finally, we integrate the framework, Software Architecture Assessment for Sustainability, and develop a semi-automation tool along with a user-friendly interface that will allow stakeholders to interact with the tool.

4.3 Evaluate Artifact

In this phase, we evaluate the artifact using open-source data collected from GitHub as our evaluation data. Our evaluation strategy consists of two approaches: (i) automated evaluation using standard Natural Language Processing (NLP) metrics, and (ii) manual verification using precision, recall, and the F1 score.

We first collect open-source data from GitHub as our evaluation data. We use the search terms "Requirement Documents" and "Architectural Design Records" to find relevant documents that could be used to evaluate our artifact. We sort the results by the number of stars and skim through the first 20 results. In this way, we identify several relevant repositories. For the evaluation phase, we select 20 architectural decision records and 17 requirement documents from GitHub repositories [1] [2].

Once we collect the evaluation data, our evaluation strategy is to first manually identify quality attributes and architectural design decisions from the evaluation documents. Our approach involves analyzing the evaluation data to extract relevant quality attributes and architectural design decisions. We use these terminologies of quality attribute and architectural design decisions for the rest of the study and to derive quality attributes and architectural design decisions manually from the evaluation data. An architectural design decision is a description of the set of architectural additions, subtractions and modifications to the software architecture, the rationale, and the design rules, design constraints, and additional requirements that (partially) realize one or more requirements on a given architecture. The reasons behind an architectural design decision are the rationale of an

[1] https://github.com/arachne-framework/architecture
[2] https://github.com/SoftengPoliTo/Lab2

architectural design decision. It describes why a change is made to the software architecture. Rules are mandatory guidelines, prescriptions for further design decisions. Design constraints describe the opposite side of design rules. They describe what is not allowed in the future of the design, i.e. they prohibit certain behaviors. A design decision may result in additional requirements to be satisfied by the architecture. These new requirements need to be addressed by additional design decisions (18). A quality attribute (QA) is a measurable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders beyond the basic function of the system (19). A common form is proposed by Bass et al. (19) to specify all quality attribute requirements as scenarios which has the following parts, not all of these fields necessary to extract quality attributes but this structure helps us identify and map quality attributes in the input or evaluation documents.

- **Stimulus:** We use the term "stimulus" to describe an event arriving at the system or the project. The stimulus can be an event to the performance community, a user operation to the usability community, or an attack to the security community, and so forth. We use the same term to describe a motivating action for developmental qualities. Thus a stimulus for modifiability is a request for a modification; a stimulus for testability is the completion of a unit of development (19).

- **Stimulus Source:** A stimulus must have a source—it must come from somewhere. Some entity (a human, a computer system, or any other actor) must have generated the stimulus. The source of the stimulus may affect how it is treated by the system. A request from a trusted user will not undergo the same scrutiny as a request by an untrusted user (19).

- **Response:** The response is the activity that occurs as the result of the arrival of the stimulus. The response is something the architect undertakes to satisfy. It consists of the responsibilities that the system (for runtime qualities) or the developers (for development- time qualities) should perform in response to the stimulus. For example, in a performance scenario, an event arrives (the stimulus) and the system should process that event and generate a response. In a modifiability scenario, a request for a modification arrives (the stimulus) and the developers should implement the modification—without side effects—and then test and deploy the modification (19).

- **Response Measure:** When the response occurs, it should be measurable in some fashion so that the scenario can be tested—that is, so that we can determine if

the architect achieved it. For performance, this could be a measure of latency or throughput; for modifiability, it could be the labor or wall clock time required to make, test, and deploy the modification (19).

- **Environment:** The environment is the set of circumstances in which the scenario takes place. Often this refers to a runtime state: The system may be in an overload condition or in normal operation, or some other relevant state. For many systems, "normal" operation can refer to one of a number of modes. For these kinds of systems, the environment should specify in which mode the system is executing. But the environment can also refer to states in which the system is not running at all: when it is in development, or testing, or refreshing its data, or recharging its battery between runs. The environment sets the context for the rest of the scenario. For example, a request for a modification that arrives after the code has been frozen for a release may be treated differently than one that arrives before the freeze. The fifth successive failure of a component may be treated differently than the first failure of that component (19).

- **Artifact:** The stimulus arrives at some target. This is often captured as just the system or project itself, but it's helpful to be more precise if possible. The artifact may be a collection of systems, the whole system, or one or more pieces of the system. A failure or a change request may affect just a small portion of the system. A failure in a data store may be treated differently than a failure in the metadata store. Modifications to the user interface may have faster response times than modifications to the middleware (19).

The manual extraction process begins with reading the evaluation data, that is, the technical requirement document and architectural decision records from GitHub. We manually map the information present in these documents to quality attributes and architectural design decisions following the terminologies mentioned above. If any of the information fields mentioned above are not present in the input or evaluation documents, we leave them blank.

We then input the evaluation data into our artifact and document the results. Both manual extraction and output of the artifact is documented in a structured manner based on the terminologies mentioned above. We compare the manually extracted data with the artifact output using quantitative metrics.

4.3.1 NLP Metrics

To evaluate the quality of generated text, we used the following NLP metrics, taking inspiration from (9):

- **ROUGE-1:** Recall-Oriented Understudy for Gisting Evaluation measures unigram overlap between the artifact's output and the manually extracted reference. It captures recall-oriented matching between the reference and system generated text (20).
- **BLEU:** Bilingual Evaluation Understudy is a precision-based metric often used in machine translation. It evaluates how closely the generated text matches the reference text in n-gram overlap (21).
- **METEOR:** Metric for Evaluation of Translation with Explicit ORdering accounts for synonyms and paraphrasing by aligning chunks and considering precision, recall, and semantic matching (22).
- **BERTScore:** Bidirectional Encoder Representations from Transformers uses contextual embeddings from BERT to evaluate semantic similarity (23). We calculate BERTScore Precision, Recall, and F1 to capture semantic similarity of the output.

These metrics help assess the linguistic and semantic similarity between the artifact's output and the manually labeled reference text. We consider BERTScore as our main metric as it calculates the semantic similarity between the reference and system-generated text and not just textual overlap.

4.3.2 Manual Verification

Inspired by Zhou et al. (11), in the manual verification step, we read and label each instance of the extracted quality attributes and architectural design decisions, both from manual extraction and from the output of the artifact. To classify the results, we used the following labels:

True Positive (TP): A quality attribute or an architectural design decision identified by both the manual extraction process and the artifact.

False Positive (FP): A quality attribute or an architectural design decision extracted by the artifact but not present in the manual extraction.

False Negative (FN): A quality attribute or an architectural design decision missed by the artifact but present in the manual extraction.

True Negative (TN): A quality attribute or an architectural design not present in both the artifact and the manual extraction, not included in the evaluation as they do not exist.

From these classifications, we compute:

- **Precision** = $\frac{\text{TP}}{\text{TP+FP}}$
- **Recall** = $\frac{\text{TP}}{\text{TP+FN}}$
- **F1 Score** = $2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision+Recall}}$

This gives us a grounded estimate of how reliable the artifact is and acts as an additional verification to the calculated NLP metrics.

5

Artifact

This section describes the artifact developed in this study, guided by the Design Science Research Methodology. It includes requirements, design, development, and evaluation of the artifact. These are the results of the design phases mentioned in Section 4. The artifact implements tool support to semi-automate the Software Architecture Assessment for Sustainability using Large Language Models (LLMs) [1].

5.1 Requirements

The requirements for our artifact were derived by analyzing the general steps of the Software Architecture Assessment for Sustainability (1). The evaluation method provides a set of general steps to assess the software architecture for sustainability. The overview of the general steps is mentioned in Section 2.1. These steps are manual processes and can be time consuming. To reduce the time required and manual effort, we classified the general steps into the following categories:

- Steps that require manual effort, which could be reduced by automation via LLM.
- Steps that require manual effort, which could be reduced by simple Python scripting.
- Steps that require human input or validation and cannot be fully automated.

5.1.1 Classification of Steps for Automation

These steps are manual processes and can be time consuming. To reduce the time required and manual effort, we classified the general steps into the following categories as mentioned above.

[1] https://github.com/anusha2009/replication-package/tree/main/src

- **LLM Automation Steps:** These are the steps that require manual effort, which could be reduced by automation via LLM. These steps involve extracting or interpreting information present in input documents (e.g., technical requirement documents, software architectural documents). These steps could benefit from the capabilities of Large Language Models, at the same time reducing the time and manual effort required to execute these steps.
  - Identification of quality attributes from technical requirement documents.
  - Generation of Sustainability Quality (SQ) model.
  - Extraction of Architectural Design Decisions from architectural documents.
  - We initially considered the generation of trade-offs as an automation step under this category but during development and evaluation phase, our open-source data did not have documented trade-offs, and hence we were unable to develop this feature in our artifact. This can be considered as a future improvement step.
- **Script Automation Steps:** These are the steps that do not need the capabilities of Large Language Models for automation and can be automated by a simple Python script. We considered steps based on numerical calculations for this category.
  - Generation of Sustainability Impact Scores (SIS)
- **Human in the Loop Steps:** These are the steps that could benefit from human validation and judgment or steps that cannot be fully automated. Hence, they are not considered as requirements of the artifact.
  - Collection of documents
  - Presentation of the subject system
  - Prioritization and validation of identified quality attributes
  - Validation of extracted design decisions.
  - Creation of DMatrices.
  - Presentation of quality attribute trade-offs and sustainability impact scores.

This classification provided a set of initial requirements that guided our thought process in identifying which parts of the assessment method would be automated by the artifact.

5.1.2 Derived Artifact Requirements

Based on the above classification and considering the possible needs of potential stakeholders using the artifact, we derived the following technical and usability requirements for our artifact.

- **R1:** The artifact must allow users to upload documents.
- **R2:** The artifact must support the automated extraction of quality attributes along with impacted sustainability dimension and architectural design decisions from uploaded documents.
- **R3:** The artifact must calculate sustainability impact scores (SIS), allowing users to input impact values and quality attribute priorities.
- **R4:** The artifact must allow users to review and validate the quality attributes and architectural decisions extracted.
- **R5:** The system must provide a user-friendly, easy-to-use web interface to upload documents, view extracted elements, and calculate SIS values.
- **R6:** The system must support local deployment options for organizations with privacy-sensitive architectural documentation.
- **R7:** The software must be modular and extensible to integrate future improvements.

5.2 Design of the Artifact

In this section, we describe the main design choices of the artifact to semi-automate the Software Architecture Assessment for Sustainability. We describe the choices for the LLM model and LLM approach and explain the end-to-end design pipeline and modular components.

5.2.1 Choice of LLM Model

Our main factors to consider for the LLM model were the accuracy and privacy of the LLM model. We chose the LLM model to be used to develop the artifact in a structured manner, and we used the following criteria to guide the selection.

- Strong performance in comprehending documents and extracting structured information, particularly within the software engineering domain.

- Feasibility of local deployment to ensure privacy when processing potentially sensitive technical documentation.

Initially, we reviewed recent literature that uses the capabilities of large language models (LLMs) to extract relevant information from documentation. Relevant sources (6, 13, 14, 24, 25, 26, 27, 28, 29) present how models such as GPT, LLaMA and Mistral, Flan-T5 are used to extract and summarize knowledge.

Based on this, we considered four candidate models:

- **OpenAI GPT-4**: This is a proprietary model that shows great results in tasks requiring good reasoning (4, 9, 24). However, it cannot be used on local machines, which restricts its suitability to handle sensitive software architectural documents.
- **Meta LLama 3**: It is an open-source LLM model and has been trained on a larger dataset compared to Llama 2, and its ability to reason has been enhanced. Shows good performance in information extraction tasks (5, 15, 26, 30).
- **Mistral**: This is an open-source LLM model (7, 31) that has been used for information extraction and summarization tasks.
- **Google Flan-T5**: This is an open-source LLM model and has demonstrated good performance in tasks such as text classification and resource extraction (6). Recent research (9) shows that when fine-tuning Flan-T5 models, it can achieve results of a similar quality to those of GPT models.

We chose Meta's LLaMA 3 (8B) as the main support for our pipeline for these reasons:

- It is open source and supports local deployment, which means it satisfies our need for privacy when processing sensitive technical documentation (5, 13).
- It achieves good results on tests such as MMLU and NaturalQuestions and outperforms Mistral 7B and Flan-T5. This shows that it can generalize well even when there are only a few or no examples given (5, 6).
- It is used in previous studies in tasks related to extracting knowledge from software documentation (15, 24, 26).
- Ease of API access via Nebula [1], which ensures privacy as the physical data store is present at Vrije Universiteit Amsterdam and used for research projects.

[1] https://networkinstitute.org/nebula/

We chose LLama 3 (8B) and access the model via API access provided by Nebula [1] and this choice is based on a balance between model capability, deployment feasibility, privacy and suitability for information extraction tasks.

5.2.2 Choice of LLM Approach

The choice of the LLM approach to develop the artifact to semi-automate the Software Architecture Assessment for Sustainability is important when utilizing LLMs to extract Quality Attributes and Architectural Design Decisions from software documentation. We reviewed the literature to identify feasible approaches and strategies that can be used to develop the artifact.

Initial Strategies Considered We initially considered three LLM approaches: zero-shot prompting, few-shot prompting, and retrieval-augmented generation (RAG).

- **Zero-shot prompting** This is type of prompt engineering technique where the prompt contains general instructions without prior examples. It is a simple technique, but might produce varying outputs when applied to domain-specific documents (14).
- **Few-shot prompting** This is a type of prompt engineering technique where the prompt contains general instructions as well as some examples that guide the LLM model towards accurate responses. Improves reliability by extracting relevant information from the software documentation (14, 24). Studies (14, 24) show that this approach has been effective in tasks such as extracting software requirements and design rationales.
- **Retrieval-Augmented Generation (RAG)** This technique allows us to combine the capabilities of LLMs with vector-based retrieval, which guides the model to provide responses based on semantically relevant input chunks. Studies have used this technique to extract relevant information to fix code and query technical documentation (13, 26, 27). This technique helps to improve the relevance of the responses with respect to the provided context.

We decided to implement the artifact with a hybrid approach of a few-shot prompt engineering technique and retrieval augmented generation to extract quality attributes and architectural design decisions. This decision was taken based on the reliability of

[1] https://networkinstitute.org/nebula/

extracting relevant information when using few-shot prompt engineering over zero-shot prompt engineering technique as well as the ability to guide the LLM model based on semantically relevant chunks.

5.2.3 End-to-End Pipeline

We design a modular and scalable end-to-end pipeline to semi-automate the Software Architecture Assessment for Sustainability. We follow a sequence of phases, and each of them is designed to support a specific task in the artifact.

1. **Document Ingestion:** Users can upload documents through a web-based interface. Documents are stored in local storage as well as parsed and segmented into chunks based on semantic boundaries. We use sentence embedding-based segmentation (32) and store them in a local vector storage.

2. **Chunk Embedding and Storage:** Once the uploaded document is parsed and segmented into chunks, each chunk is embedded using a transformer based embedding model. In our artifact, we utilize the Sentence Transformers library (32). This gives us a compact and effective embedding representation along with semantic accuracy. The embeddings are saved in a local vector database together with metadata like chunk ID, document source and token count. This helps in quick fetching in retrieval tasks.

3. **LLM-based Extraction:** We use a hybrid approach of few-shot prompt engineering technique and retrieval augmented generation to query the LLM model. The prompts are designed following a task-specific structure (33) to ensure good quality responses. The semantic chunks from the previous phases are also considered when querying the LLM model. Nebula provides API access to the LLM model and the outputs are stored in local storage.

4. **Script Automation:** We calculate the Sustainability Impact Score (SIS) using a python script that is based on the numerical formula provided by Software Architecture Assessment for Sustainability (1). We provide this feature as a SIS calculator using a web-interface where users can provide inputs and the artifact displays the calculated SIS score.

5. **Results:** The extracted information and results of the LLM extraction are displayed using a web interface build using React.

This ensures that the design is modular and scalable, which supports extensibility enabling future iterations of the artifact.

5.3 Development of the Artifact

In this section, we describe the results of the development of the artifact to semi-automate the Software Architecture Assessment for Sustainability. We give an overview of the components of the artifact, user flows, document processing strategy, prompt engineering strategy, and an overview of the user interface of the artifact.

5.3.1 Components of the Artifact

The artifact includes the following main components:

- **Ingestion:** This module is responsible for parsing uploaded documents and segmenting the document into semantic chunks.
- **Embedding:** This module is responsible for indexing chunks and storing them with metadata for retrieval tasks.
- **LLM Client:** This module queries the LLM model via the APIs provided by Nebula [1].
- **Query Engine:** This module is responsible for retrieving relevant segments and querying the LLM client along with the designed prompts.
- **SIS Calculator:** This module is responsible for calculating SIS scores using the quality attribute priorities and impact values provided by the user.
- **Web Interface:** We use React and Flask framework to provide users with a web-interface to upload documents, trigger extraction tasks and calculate SIS scores.

5.3.2 System Overview

Figure 5.1 illustrates the component diagram of the artifact, using a three-level view, the presentation layer, the application layer, and the knowledge base. The separation of concerns and multi-layer component ensures that the system is modular and extensible.

[1] https://networkinstitute.org/nebula/

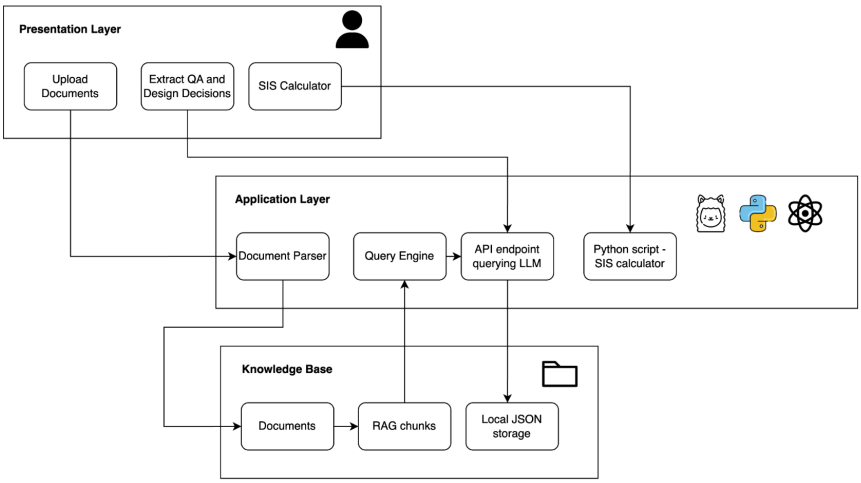

Figure 5.1: Component Diagram

1. Presentation Layer The presentation layer represents the user interface of the artifact. It consists of three components:

- **Upload Documents**: Users can upload software architectural documents such as technical requirement document, architectural decision records. This action triggers the Document Parser component in the Application Layer.
- **Extract QA and Design Decisions**: This module allows users to trigger extraction of Quality Attributes and Architectural Design Decisions. This action triggers the back-end module responsible for querying the LLM model.
- **SIS Calculator**: We provide a web interface where users can input Quality Attribute priorities and impact values to view the Sustainability Impact Score (SIS) for each architectural design decision. We provide a simple form-based user interface where users can select the architectural design decision and quality attribute and fill in the input values to calculate SIS. This action triggers the Python script to run and perform numerical calculations in the application layer.

2. Application Layer The application layer consists of the main logic responsible for parsing documents, querying the LLM model with relevant document chunks and prompts, accessing the LLM model via Nebula APIs, and automating the sustainability impact score (SIS) numerical calculation.

- **Document Parser**: The input of this module is the uploaded files from the user in the presentation layer. First, it stores the uploaded file in local storage in the Knowledge Base layer. It is responsible for parsing the files and segmenting the document into semantic chunks. These chunks are also stored in local storage in the Knowledge Base layer.
- **Query Engine**: This module is responsible for formatting the query, with both the designed prompt and relevant document chunks from the Knowledge Base layer. The output of this module is used as the final query to the LLM model.
- **API Endpoint Querying LLM**: This module is responsible for querying and accessing the LLM model via APIs provided by Nebula [1].
- **Python Script - SIS Calculator**: This module is responsible for performing numerical calculations to compute the Sustainability Impact Score (SIS).

3. Knowledge Base The Knowledge Base consists of all the data uploaded by users and persistent data which is the output of the Application Layer.

- **Documents**: This represents the user-uploaded documents that are stored in local storage.
- **RAG Chunks**: This represents the segmented document chunks by the Document Parser in the Application Layer stored in vector database in local storage.
- **Local JSON Storage**: This contains the output of the LLM automation tasks, that is, the quality attributes and architectural design decisions, which are stored in local storage.

Data Flow

- Documents are first uploaded via the user interface and then parsed by the Document Parser.
- The content that is parsed is stored in the Knowledge Base as document chunks and sent to the Query Engine for formatting the final query to the LLM model.

[1] https://networkinstitute.org/nebula/

- The Query Engine guides data to the LLM through the API endpoint. The quality attributes and architectural design decisions that are extracted are stored in the Knowledge Base.
- When the SIS calculator is triggered, numerical calculations are performed to compute the sustainability impact score.

Technological Stack

- The LLM is queried using the API provided by the Nebula platform with LLaMA 3 (8B).
- We use a Python-based Flask framework to develop the Application Layer including segmenting documents, SIS calculation, querying the LLM model via API.
- We parse and segment the documents using custom logic in the Application Layer. Once the documents are segmented, it is embedded using Sentence Transformers (32).
- We use React framework to build the front-end component for responsiveness and interactivity.

5.3.3 User Flows

This section describes the user flows when interacting with the artifact.

1. **Upload Documents:** The user uploads documents and we display a success message on the user interface when the document upload is successful.
2. **Extract QA and Design Decisions:** Once the user uploads documents, they can trigger the extraction of quality attributes and architectural design decisions. A success message is displayed on the user interface when the extraction is successful, users can view the extracted data on the user interface.
3. **Manual Validation:** Users can view and edit the extracted quality attributes and architectural design decisions, these changes are reflected in the user interface and the local storage in the Knowledge Base layer.

4. **SIS Calculation:** The user provides selects the architectural design decision and quality attributes in a form-based user interface and provides the priorities of the quality attribute and impact values which results in the Sustainability Impact Score (SIS).

5.3.4 Document Parsing Strategy

The documents that are uploaded are parsed in the Document Parser component in the Application Layer. It uses custom logic to parse the document and segment the document into semantic chunks, which is further used in the Query Engine.

The logic to segment the document into chunks prioritizes semantic boundaries over fixed size windows. The document is split into sections based on headings, ensuring that each chunk corresponds to a unit of content in the documentation. This method of splitting or segmenting the document aligns with the semantic meaning and flow of the document and avoids arbitrarily splitting, since these chunks are used for retrieval tasks.

Segmenting the documentation into chunks is necessary because:

- LLM models have token limits and it is not best practice to input documents directly which may exceed these token limits.
- Segmenting the document considering the semantic boundaries helps to ensure efficient retrieval as only relevant segments are passed to the model.

Each chunk or segment is stored in a vector database in local storage using an embedding model from the Sentence-Transformers (32) library. The helps in efficient retrieval when formatting the final query in the Query Engine module in the Application Layer.

5.3.5 Prompt and Query Strategy

To extract relevant information, that is, quality attributes and architectural design decisions from documents uploaded by users, prompt engineering plays a key role, in matching the output of the artifact with the expected output data.

To ensure that the output of the artifact is semantically in line with the ground truth and is reliable, we use a structured prompt and query strategy using task-specific instructions and a retrieval-augmented approach for the following tasks:

- Quality Attribute extraction
- Architectural Design Decision extraction

The prompts are structured by combining a fixed task instruction and the retrieved chunks from the user-uploaded documentation. Task instruction consists of specific task instruction related to quality attribute extraction or architectural design decision extraction along with their definitions as mentioned in 4 and examples following the structure of the definitions. We iterated over the prompt structure during the development phase, which resulted in this final structure that provided reliable output following the prompt structure and instruction.

- **Task instruction:** Instruction as to extract quality attributes or architectural design decisions
- **Definition:** Along with the task instruction, we provide the definitions mentioned in section 4 to instruct the model to provide output following this format.
- **Examples:** Along with the task instruction, we provide the example output following the structure of the definitions mentioned in section 4.
- **Retrieved chunks:** We also provide context to the model to support the extraction process, which are the semantically relevant chunks of the documents uploaded by the user.

The prompt structure provides explicit instruction and relevant context which enables the LLM model to respond in a structured format in line with the requirements of the tool.

Retrieval-Augmented Generation (RAG) We used a RAG approach along with prompt engineering so as to increase the relevance and accuracy of the LLM responses.

1. Documents uploaded to the artifact are segmented semantically and embedded using Sentence Transformers.
2. These embeddings are stored in local vector storage.
3. When the extraction tasks are triggered by the user in the Presentation Layer, the Application Layer retrieves the semantically relevant chunks from the Knowledge Base based on the type of extraction, that is, quality attributes or architectural design decisions.
4. These retrieved chunks are added to the prompt as mentioned above and used as a query to the LLM model.

5. The LLM model performs the extraction tasks and provides results following the defined structure.

This query strategy allows us to combine the RAG and prompt engineering technique to increase the relevancy and accuracy of the output of the LLM model.

5.3.6 User Interface

The web-based user interface allows users to upload documents, trigger the extraction of quality attributes and architectural design decisions, and provides a SIS calculator.

- We provide a document upload tab in the user interface where users can upload documents.
- We provide separate tabs to view the results of quality attributes and the extraction of architectural design decisions. Users can view the results, as well as edit the results which are reflected in the local storage in the Knowledge Base layer.
- We provide a SIS calculator, where users can select architectural design decision and quality attributes and input the quality attribute priorities and impact values. The backend script is triggered, and we display the SIS score which is the output of the script.

We provide a simple, user-friendly interface which allows users to interact with the core logic of the artifact.

5.4 Evaluation

To evaluate the artifact, we used open source data from GitHub as our evaluation data and utilized two strategies to calculate quantitative metrics that help to assess the artifact.

5.4.1 Evaluation Data

We searched GitHub to find open source evaluation data as explained in Section 4. Our evaluation data consists of:

- 20 Architecture Decision Records from open source repository [1].
- 17 technical requirement documents from open source repository [2].

[1] https://github.com/arachne-framework/architecture
[2] https://github.com/SoftengPoliTo/Lab2

5.4.2 Evaluation Strategy

Our evaluation consisted of the following steps:

1. **NLP Metrics:** The artifact was run on the same documents and its outputs were compared to the results extracted manually using natural language processing (NLP) metrics. This allowed us to measure the textual similarity and quality of the generated text.

2. **Manual Verification:** We manually extracted quality attributes and architectural design decisions from the evaluation data. This was compared with artifact production by labeling the results as explained in Section 4. This allowed us to manually verify the correctness of the NLP metrics.

. The overview of the process and the metrics is given in Section 4.

We give an instance of the manually extracted data along with the result of the artifact considering the above mentioned evaluation data, all the results of both manual extraction and output of the artifact are provided in the replication package [1].

1. **Manual extraction of Quality Attribute:**

   - Stimulus source: User using the application
   - Stimulus: Any function call in the app
   - Environment: Normal Operation
   - Artifact: The EZGas application
   - Response: Function completes execution
   - Response Measure: Execution time $<$ 1 sec
   - Quality Attribute: Performance
   - Description: Technical, Economic
   - Sustainability Dimension: All functions should complete in $<$ 1 sec

2. **Artifact Output:**

   - Stimulus source: The user is interacting with the application through a web or mobile interface.
   - Stimulus: A user requests a service from the application.

[1] https://github.com/anusha2009/replication-package/tree/main/data

- Environment: Normal operation of the application.
- Artifact: The EZGas application.
- Response: The response to this stimulus is that all functions should complete in less than one second.
- Response Measure: Time taken by functions to complete (measured in seconds).
- Quality Attribute: Performance
- Description: The application should be designed to respond quickly to user requests, providing a fast and responsive experience.
- Sustainability Dimension: Economic, Technical

3. **Manual extraction of Architectural Design Decision:**

   - Design Decisions: The top-level entity in an Arachne system is a reified Arachne Runtime object.
   - Description: This object contains both the Component system object, and the configuration value upon which the runtime is based. It satisfies the Lifecycle protocol itself; when it is started or stopped, all of the component objects it contains are started or stopped in the appropriate order.
   - Rationale: Arachne should provide an orderly, structured startup (and shutdown) procedure, and make it possible for modules and application authors to hook into it to ensure that their own code initializes, starts and stops as desired. Additionally, it must be possible for different system components to have dependencies on eachother, such that when starting, services start after the services upon which they depend. Stopping should occur in reverse-dependency order, such that a service is never in a state where it is running but one of its dependencies is stopped.
   - Rules: Only component entities that are actually used will be instantiated; unused component entities defined in the config will be ignored.

4. **Artifact Output:**

   - Design Decisions: Create an Arachne Runtime object
   - Description: The top-level entity in an Arachne system is a reified Arachne Runtime object. This object contains both the Component system object, and the configuration value upon which the runtime is based.

- Rationale: The rationale behind this decision is to provide a central point for managing the startup and shutdown of components.
- Rules: The constructor function for a Runtime takes a configuration value and some number of "roots"; entity IDs or lookup refs of Component entities in the config. Only root components and their transitive dependencies will be instantiated or added to the Component system.

5.4.3 Evaluation Metrics

Tables 5.1 and 5.2 provide the results of the evaluation using NLP metrics.

Metrics	**ROUGE-1**	**BLEU**	**METEOR**
QA	0.430	0.073	0.241
Architectural Design Decisions	0.317	0.062	0.351

Table 5.1: NLP Evaluation Metrics for Quality Attributes and Architectural Design Decision Extraction

Metrics	**BERTScore Precision**	**BERTScore Recall**	**BERTScore F1**
QA	0.894	0.841	0.866
Architectural Design Decisions	0.842	0.867	0.854

Table 5.2: NLP Evaluation Metrics for Quality Attributes and Architectural Design Decision Extraction

The BERTScore values indicate that the output of the artifact has high semantic similarity with the manually extracted data that serves as the ground truth for the evaluation phase. Low ROUGUE-1, BLEU, METEOR indicates that there is textual overlap between the output of the artifact and the manually extracted data.

To further verify the results of these quantitative metrics, we manually labeled the data as explained in 4 and calculated the precision, recall and F1 values. These results are shown in Table 5.3.

Metrics	**Precision**	**Recall**	**F1**
QA	0.95	0.83	0.88
Architectural Design Decisions	0.88	0.88	0.88

Table 5.3: Manual Evaluation Metrics for Quality Attributes and Architectural Design Decision Extraction

These results align with our primary NLP metric (BERTScore) and indicate that the artifact is capable of extracting relevant information to semi-automate the Software Architecture Assessment for Sustainability.

6

Discussion

In this section, we reflect on the results of our evaluation, examine what they mean related to the initial research question, and discuss the possible impacts for practitioners.

6.1 Addressing the Research Question

The primary research question that guided this study was the following:

> **RQ:** How can we support software architects using the Software Architecture Assessment for Sustainability to reduce the time required for sustainability assessments in software architecture?

The results of the evaluation phase show that the tool can achieve high semantic similarity between the output of the artifact and manually extracted data. This result is further verified by manually labeling the data and calculating the precision, recall, and F1 scores. This shows that Large Language Models (LLMs) can be used in reducing the manual effort required for sustainability assessments using the Software Architecture Assessment for Sustainability by automating some of the general steps of the framework that require manual effort. We provide tool support that automates the extraction of quality attributes along with impacted sustainability dimensions, architectural design decisions, and SIS calculation. The tool uses Llama 3 (8B) accessed via a nebula and a few-shot prompt engineering along with a retrieval-augmented query approach. Although the tool achieves high semantic similarity with the manual ground truth, it is not a replacement for human validation. The tool is only proposed as a support for the assessment process, and hence human input and validation are still required to successfully assess software architecture for sustainability.

6.2 Implications for Practitioners

This study has some possible impact for practitioners, such as software architects and sustainability practitioners.

- **Reduction in manual effort:** Steps that require manual effort, such as manual reading and annotation of documents, can now be assisted by automation through the semi-automated tool.
- **Enhanced consistency:** The tool can ensure that the extracted data follow a defined structure to identify and document relevant information from documentation.
- **Interactive workflows:** The tool provides the ability to review and edit the extracted information to support a human-in-the-loop process. This ensures that practitioners remain in control while benefiting from automation.

Finally, the modular and extensible tool allows local deployment, ensuring privacy of information and uploaded documents. This also enables organizations and practitioners to use the tool to support them in sustainability assessments using the Software Architecture Assessment for Sustainability.

6.3 Reflections on Tool Performance

The evaluation phase provides quantitative metrics of the performance of the tool. It shows both the strengths and limitations of the semi-automated tool. There is a high semantic similarity that shows that the extracted information is relevant and matches with the manually extracted ground truth. But it also shows how the textual overlap between the output of the tool and the manually extracted information is low. This indicates that the results of the extraction process match well with the manual ground truth in meaning, but do not use the same words from the documents.

6.4 Future Research and Tool Development

Future work and iterating on tool development could enhance the tool and increase research contributions.

- We can extend the tool by incorporating an additional LLM approach by fine-tuning the LLM model. This could increase the accuracy of the extracted information.

- We can extend the tool to support trade-off analysis of architectural design decisions by training on documents that document trade-offs of architectural design decisions with explanations.

To summarize, this study provides a tool that semi-automates the Software Architecture Assessment for Sustainability. Some of the manual processes in the evaluation method is automated and the results show high semantic similarity with the manual ground truth. While, this is not a complete replacement of human input in the evaluation process, the tool reduces the manual effort required and supports the evaluation process.

7

Threats To Validity

In this section, we discuss the main threats to validity of our study.

7.1 Internal Validity

The manual labeling and extraction process in the evaluation phase of our study introduces a threat to validity since it is performed by a single researcher without external validation. As explained in section 4.1, we extracted manual information in accordance with the terminologies so as to maintain consistency in the extracted information. But some subjectivity might still persist. We first extracted manual information and later documented the output of the artifact to mitigate the threat to validity.

7.2 External Validity

Since there is no stakeholder in our study, in the evaluation phase, we considered open source documents from GitHub as our evaluation data. These documents may not fully represent industrial documentation, and this could introduce a threat to the validity of our study. In an attempt to mitigate this, we sorted the GitHub repositories found during the search process by the number of stars to find the most relevant repository for the evaluation phase.

7.3 Construct Validity

In the evaluation phase of our study, we used NLP metrics as well manual verification through Precision, Recall, and F1 values. Not all NLP metrics capture the semantic similarity between the manual ground truth and the output of the artifact. ROUGUE-1,

BLEU and METEOR do not capture the semantic similarity in detail, and to mitigate this threat to validity, we consider BERTScore and further manual verification as our main metrics, since we value semantic similarity over textual overlap.

7.4 Conclusion Validity

The conclusion of this study is based on the quantitative metrics in the evaluation phase of our study. These metrics indicate that there is a high semantic similarity between the manual ground truth and the output of the artifact. Since the evaluation data consist of a limited set of documents, this could introduce some threat to validity. Further studies with larger number evaluation documents along with industrial relevance could strengthen the conclusions of the study.

8

Conclusion

In our study, we develop a tool to support the Software Architecture Assessment for Sustainability framework by automating some of the general steps that require manual effort by leveraging large language models. The Software Architecture Assessment for Sustainability (1) provides a set of general steps that are manual processes and often time-consuming. Our goal of this study was to provide tool support to automate some of the manual processes to reduce the time and manual effort required in sustainability assessments. The main contribution of this study is the developed tool support and the evaluation metrics that include both NLP metrics and further manual verification.

We initially classified the general steps of the assessment framework into three categories. Steps to be automated by LLM, Steps to be automated by a script and human-in-the-loop tasks. Based on this classification, we derived the requirements of the tool, designed and developed the semi-automation tool that supports the extraction of quality attributes, architectural design decisions, SIS calculations and provides a user-friendly interface.

The evaluation phase of our study shows the feasibility of leveraging LLMs to semi-automate the assessment framework as well as the performance of the developed tool. We used open-source data from GitHub for evaluation, and the results show that there is a high semantic similarity between the manual ground truth and the output of the artifact. These results confirm the feasibility of leveraging LLMs to reduce the time and manual effort required in sustainability assessments.

This study has potential impacts on practitioners using the assessment framework. The developed tool can be deployed locally to support the assessment process while maintaining privacy when dealing with sensitive documents and the consistency of the extracted data. Future work could include iterations on the tool in order to increase the quality and accuracy of the extracted data by fine-tuning the LLM model. The tool and the output of

the tool will only support the practitioner in assessing the software architecture and not completely replace human validation or human input.

Overall, this study contributes to semi-automating the Software Architecture Assessment for Sustainability providing tool support and practical impact for sustainability assessments.

References

[1] Iffat Fatima and Patricia Lago. **Software Architecture Assessment for Sustainability: A Case Study**. In *European Conference on Software Architecture*, pages 233–249. Springer, 2024. 1, 2, 3, 6, 10, 11, 17, 22, 39

[2] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M. Zhang. **Large Language Models for Software Engineering: Survey and Open Problems**. In *Proceedings of the IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*. IEEE, 2023. 1

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. **Language Models are Few-Shot Learners**. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, **33**, pages 1877–1901. Curran Associates, Inc., 2020. 5, 6

[4] OpenAI. **GPT-4 Technical Report**, 2023. Available at `https://openai.com/research/gpt-4`. 5, 6, 20

[5] Abhinav Jauhri Abhinav Pandey Abhishek Kadian Ahmad Al-Dahle Aiesha Letman Akhil Mathur Alan Schelten Alex Vaughan Amy Yang

Angela Fan Anirudh Goyal Anthony Hartshorn Aobo Yang Archi Mitra Archie Sravankumar Artem Korenev Arthur Hinsvark Arun Rao Aston Zhang Aurelien Rodriguez Austen Gregerson Ava Spataru Baptiste Roziere Bethany Biron Binh Tang Bobbie Chern Charlotte Caucheteux Chaya Nayak Chloe Bi Chris Marra Chris McConnell Christian Keller Christophe Touret Chunyang Wu Corinne Wong Cristian Canton Ferrer Cyrus Nikolaidis Damien Allonsius Daniel Song Danielle Pintz Danny Livshits Danny Wyatt David Esiobu Dhruv Choudhary Dhruv Mahajan et al Aaron Grattafiori, Abhimanyu Dubey. **The Llama 3 Herd of Models**, 2024. 5, 6, 20

[6] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. **Scaling Instruction-Finetuned Language Models**. *Journal of Machine Learning Research*, **25**(70):1–53, 2024. 5, 6, 20

[7] Mistral AI team. **Mistral 7B**, 2023. `https://mistral.ai/news/announcing-mistral-7b/`. 5, 20

[8] Davit Abrahamyan and Fatemeh H Fard. **StackRAG Agent: Improving Developer Answers with Retrieval-Augmented Generation**. In *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 893–897. IEEE, 2024. 5, 8

[9] Rudra Dhar, Karthik Vaidhyanathan, and Vasudeva Varma. **Can llms generate architectural design decisions?-an exploratory empirical study**. In *2024 IEEE 21st International Conference on Software Architecture (ICSA)*, pages 79–89. IEEE, 2024. 5, 6, 7, 15, 20

[10] Zibin Zheng, Kaiwen Ning, Qingyuan Zhong, Jiachi Chen, Wenqing Chen, Lianghong Guo, Weicheng Wang, and Yanlin Wang. **Towards an Under-**

standing of Large Language Models in Software Engineering Tasks. *Empirical Software Engineering*, **30**:50, 2025. 6

[11] Xiyu Zhou, Ruiyin Li, Peng Liang, Beiqi Zhang, Mojtaba Shahin, Zengyang Li, and Chen Yang. **Using LLMs in Generating Design Rationale for Software Architecture Decisions**. *arXiv preprint arXiv:2504.20781*, 2025. 6, 15

[12] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. **Retrieval-augmented generation for knowledge-intensive nlp tasks**. *Advances in neural information processing systems*, **33**:9459–9474, 2020. 6

[13] Zhuo-Fan Shi, Kun Liu, Shan Bai, Yun-Tao Jiang, Tong Huo, Xiang Jing, Rui-Zhi Li, and Xin-Jian Ma. **Meta data retrieval for data infrastructure via RAG**. In *2024 IEEE International Conference on Web Services (ICWS)*, pages 100–107. IEEE, 2024. 6, 20, 21

[14] Jay U. Oswal, Harshil T. Kanakia, and Devvrat Suktel. **Transforming Software Requirements into User Stories with GPT-3.5 – An AI-Powered Approach**. In *Proceedings of the 2nd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT-2024)*. IEEE, 2024. 7, 20, 21

[15] Seyed Mostafa Seyedi and Paris Avgeriou. **DRMiner: An NLP-based Tool for Extracting Design Rationale from Software Architecture Documentation**. In *Proceedings of the 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 61–70. IEEE/ACM, 2023. 8, 20

[16] Cameron Miskell, Richard Diaz, Parth Ganeriwala, Khaled Slhoub, and Fitzroy Nembhard. **Automated Framework to Extract Software Requirements from Source Code**. In *Proceedings of the 2023 7th International Conference on Natural Language Processing and Information Retrieval (NLPIR)*, pages 130–134. ACM, 2023. 9

[17] Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee. **A Design Science Research Methodology for Information Systems Research**. *Journal of Management Information Systems*, **24**(3):45–77, 2007. 10

[18] Anton Jansen and Jan Bosch. **Software Architecture as a Set of Architectural Design Decisions**. In *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 109–120. IEEE, 2005. 13

[19] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 4th edition, 2021. 13, 14

[20] Chin-Yew Lin. **ROUGE: A Package for Automatic Evaluation of Summaries**. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 74–81, Barcelona, Spain, 2004. Association for Computational Linguistics. 15

[21] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. **BLEU: A Method for Automatic Evaluation of Machine Translation**. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL)*, pages 311–318. Association for Computational Linguistics, 2002. 15

[22] Alon Lavie and Abhaya Agarwal. **METEOR: An Automatic Metric for MT Evaluation with High Levels of Correlation with Human Judgments**. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 228–231. Association for Computational Linguistics, 2007. 15

[23] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Weinberger, and Yoav Artzi. **BERTScore: Evaluating Text Generation with BERT**. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020. 15

[24] Xiyu Zhou, Ruiyin Li, Peng Liang, Beiqi Zhang, Mojtaba Shahin, Zengyang Li, and Chen Yang. **Using LLMs in Generating Design Rationale for Software Architecture Decisions**. *arXiv preprint arXiv:2504.20781*, 2025. 20, 21

[25] Rudra Dhar, Karthik Vaidhyanathan, and Vasudeva Varma. **Leveraging Generative AI for Architecture Knowledge Management**. In *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*, pages 163–166. IEEE, 2024. 20

[26] Elijah Mansur, Johnson Chen, Muhammad Anas Raza, and Mohammad Wardat. **RAGFix: Enhancing LLM Code Repair Using RAG and Stack Overflow Posts**. In *2024 IEEE International Conference on Big Data (BigData)*, pages 7491–7496. IEEE, 2024. 20, 21

[27] Alireza Marefat, Abbaas Alif Mohamed Nishar, and Ashwin Ashok. **Text2Net: Transforming Plain Text into Dynamic, Interactive Network Simulations**. In *2024 23rd ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 315–316. IEEE, 2024. 20, 21

[28] Raghav Joshi, Yash Bubna, M Sahana, and A Shruthiba. **An Approach to Intelligent Information Extraction and Utilization from Diverse Documents**. In *2024 8th International Conference on Computational System and Information Technology for Sustainable Solutions (CSITSS)*, pages 1–5. IEEE, 2024. 20

[29] Nataliia Klievtsova, Juergen Mangler, Timotheus Kampik, and Stefanie Rinderle-Ma. **Utilizing Process Models in the Requirements Engineering Process Through Model2Text Transformation**. In *2024 IEEE 32nd International Requirements Engineering Conference (RE)*, pages 205–217. IEEE, 2024. 20

[30] Quanjun Zhang, Chunrong Fang, Yang Xie, Yaxin Zhang, Yun Yang, Weisong Sun, Shengcheng Yu, and Zhenyu Chen. **A survey on large language models for software engineering**. *arXiv preprint arXiv:2312.15223*, 2023. 20

[31] Md Shahidul Salim, Sk Imran Hossain, Tanim Jalal, Dhiman Kumer Bose, and Mohammad Jahid Ibna Basher. **LLM based QA chatbot builder: A generative AI-based chatbot builder for question answering**. *SoftwareX*, **29**:102029, 2025. 20

[32] Nils Reimers and Iryna Gurevych. **Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks**. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019. 22, 26, 27

[33] Laria Reynolds and Kyle McDonell. **Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm**, 2021. arXiv:2102.07350. 22