Automatically Assessing Software Architecture Compliance With Green Software Patterns

Naman Ahuja
University College London
London, United Kingdom
namanahuja38@gmail.com

Yile Feng
University College London
London, United Kingdom
fengyileon@gmail.com

Luming Li
University College London
London, United Kingdom
lumingli723@gmail.com

Amisha Malik
University College London
London, United Kingdom
amishamalik23@gmail.com

Thuvaragan Sivayoganathan
University College London
London, United Kingdom
thuvasiva@outlook.com

Navveen Balani
Accenture
Mumbai, India
naveen.balani@accenture.com

Srinivasan Rakhunathan
Microsoft
Hyderabad, India
srrakhun@microsoft.com

Federica Sarro
University College London
London, United Kingdom
f.sarro@ucl.ac.uk

***Abstract*—With increasing awareness of climate change, there is a growing emphasis on the environmental impact of digital solutions. While numerous tools are available to assess softwareenvironmental footprint post-development, few focus on sustainability during the software design phase. To address this gap, we propose EcoDocSense, a framework that supports engineers to evaluate the sustainability of a software design at design time. Utilizing Large Language Models fine-tuned on a catalog of green software patterns, EcoDocSense analyzes software architecture documents to generate sustainability reports, assessing alignment with green software practices to minimize carbon emissions and recommending improvements. As one of the first frameworks targeting sustainability at the design stage, EcoDoc Sense represents a significant advancement, though opportunities remain for further enhancement. In future we plan to extend EcoDocSense's applicability to a variety of architectural types and documents as well as to provide the capability to estimate carbon emissions.**

***Index Terms*—Suistanable Software Architecture, Green Software Patterns**

I. Introduction

As software becomes more complex and widespread, the demand for supporting machines grows, leading to increased energy consumption and environmental consequences. By 2040, the software industry is expected to account for 14% of the global carbon footprint, up from 1.5% in 2007 [1].

Software architectural decisions significantly affect the amount of hardware used, how efficiently a software utilizes that hardware, and the amount of data stored or sent over networks. To make decisions with energy use and carbon emissions in mind, software architects must learn about sustainability concepts and understand how to apply them to their designs. However, the software industry is still at the early stages of the green software development journey [2]–[5]. While green principles are essential, there is limited automated support for their adoption in practice, especially in the early phases of the software lifecycle [3], [4].

To support engineers to integrate sustainability considerations at design phase, we propose EcoDocSense, a framework that analyzes software architecture documents for compliance with a catalog of green software patterns. EcoDocSense takes as input a document describing a software architecture, and leverages automated data extraction, document retrieval and generative models to generate as output a detailed sustainability analysis which not only identifies the patterns that are followed or overlooked in a given architecture, but also provides suggestions for improvements.

To assess the feasibility of such idea we designed and deployed a minimal viable products (MVP) in collaboration with the Green Software Foundation (GSF) [6]. We carried out empirical experiments by using the online open-source catalog of software patterns reviewed and curated by the Green Software Foundation [7] and publicly available documents describing the architectural styles of Instagram, WhatsApp, Dropbox, Uber and Netflix [8]. The results show that EcoSense correctly identifies at least 94% of the patterns depending on the retrieval approach used. Moreover, fine-tuning greatly benefits the generative model performance with the best fine-tuned model achieving a 70% F1-score on the architectural documents used as benchmark (a 19% improvement over the baseline's score of 51%) and a reduced generation time to about 6 minutes per document.

The EcoDocSense's source code [9] and a demo-video [10] are available on-line.

II. EcoDocSense

The EcoDocSense framework is inspired by Retrieval-Augmented Generation (RAG), which combines document retrieval with generative modeling, allowing the model to generate more comprehensive and contextually accurate responses. This approach is particularly effective for addressing complex or open-ended questions.[1]

[1]Initially, we considered Extractive QA Models for this task. These models are designed to identify and extract specific answers directly from a given text. However, they are limited by their dependence on the context provided and cannot generate content beyond what is explicitly stated in the text [11]. We found that the effectiveness of this solution was constrained by its reliance on the supplied context, often leading to irrelevant or incomplete responses when the context lacked sufficient details.

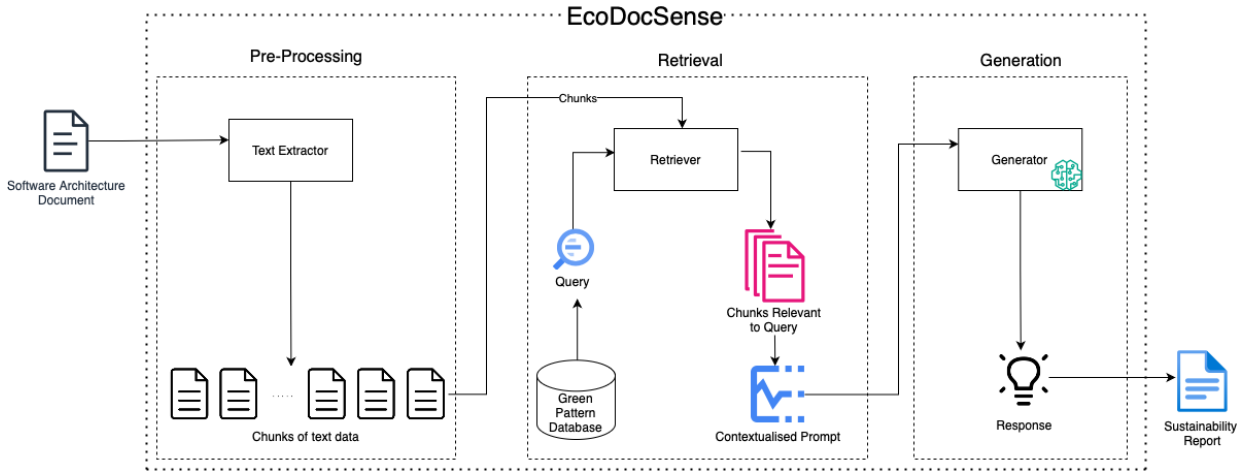


Fig. 1: EcoDocSense Workflow

The EcoDocSense workflow, depicted in Figure 1, consists of three phases:

1) **Pre-Processing**: In this phase, the textual information contained in the architectural document is automatically extracted, chunked in smaller pieces and converted into a format that can be interpreted by both the retriever and the generative model;
2) **Retrieval**: In this phase, a number of queries are built based on the database of green software patterns (i.e., one query per pattern). These queries are used as input to a Retriever which is able to find from the input document those text chunks (i.e., architectural information) that best matches each of the queries (i.e., green design patterns). This retrieved text (which we refer to as text chunks relevant to the query, or in short relevant chunks), is then used as a context for the generative model's prompt in the next phase.
3) **Generation**: In this phase, a fine-tuned Large Language Model (LLM) is used to answer each prompt built in the previous phase. Moreover, the answer provided by the LLM is further parsed to output a comprehensive sustainability analysis containing both detailed pattern compliance assessments and actionable recommendations, presented through a combination of graphs and structured textual analysis.

In the following subsections we describe each of the phases in more detail, as well as reflect on the design and technological choices we had to make for each phase.

A. Pre-Processing

This phase takes as input a software architecture document in PDF format, typically containing text, tables, and images. Our project primarily focused on extracting and processing textual content. Text extraction was accomplished using the pymupdf4llm [12] Python library. After extracting text, we faced the challenge of handling large documents due to the context window limitations of generative models, which measure input capacity in tokens (words or parts of words). For example, the Phi-3 model has a context window of 4,000 tokens, which may be insufficient for a complete design document. To address this, the text needs to be divided into manageable chunks. To this end, we considered different existing chunking strategies including Fixed Size Chunking, Recursive Chunking, and Chunking Using Markdown [13].[2] We ultimately selected Recursive Chunking, which divides the text into progressively smaller chunks in a hierarchical and iterative manner, using a set of separators. If the initial split does not yield the desired chunk size, the process is repeated recursively with alternative separators. We found recursive chunking more effective due to its adaptability to varied data sizes and structures, its usefulness in handling hierarchical or nested data, and its capability to dynamically adjust and finely control chunk sizes [14].

The text extracted from the architecture document is often unstructured, making machine processing and analysis challenging. To address this, we converted the text into word embeddings [15], numerical representations that convert words or phrases into vectors to capture semantic meaning and preserve vital information. This numerical format simplifies similarity computations, essential for context retrieval. The vectors are stored in a *vector database*, a specialized persistent storage that enables efficient lookup of nearest neighbours [16]. Persistently storing vectors avoids the need to regenerate embeddings for previously processed documents and ensures fast insert and retrieval operations, enhancing overall pipeline speed and performance.

B. Retrieval

This phase works with two primary inputs: the *queries*, which we derive from a green software pattern database, and the *vector database* containing embedded text chunks, which is obtained in the pre-processing phase.

[2]Chunking refers to dividing a large text corpus into smaller, manageable pieces or segments, each acting as a standalone unit of information for indexing and retrieval.

TABLE I: Number of Green Patterns per Category.

Category	Number of Patterns
Resource Optimization	19
Data Efficiency	6
Performance Management	12
Security	7
User Impact	1
Total	**45**

Queries are essentially questions formulated based on existing green software patterns. For instance, if the green software pattern is "Optimize Storage Utilization", the corresponding query might be "Is there any mention of optimizing storage utilization?"

The current implementation of EcoDocSense makes use of the online open-source database of software patterns reviewed and curated by the Green Software Foundation [17] to derive queries.[3] Specifically, the database used for EcoDocSense encompasses both Web and Cloud patterns derived from the GSF database [17], for a total of 45 green patterns. Table I summarises these patterns grouped under five categories (namely resource optimization, data efficency, performance managment, security and user impact). The description of these patterns can be found in the EcoDocSense's GitHub.

To facilitate future extension of queries to encompass additional green patterns, the queries are stored in JSON format. Each query contains the following details:

- Type: The architectural type to which the green pattern applies (e.g., Web, Cloud).
- Category: The category of the green pattern (e.g., Resource Optimization, see Table I).
- Pattern: The specific green pattern (e.g., Optimize storage utilization);
- Query: A question derived from the green pattern to be included in the prompt (e.g., Is there any mention of optimizing storage utilization?).

The Retriever component uses a query to identify relevant information from the vector database (representing the original architectural document) that can address the query. The Retriever specifically uses the query to identify chunks containing information relevant to that query from the vector database, where vector distances are increasingly utilized to measure text similarity [18]. These vectors are represented in a high-dimensional space, where shorter distances indicate greater similarity between statements, and longer distances indicate lesser similarity. To balance reliability and precision, the top five texts with the shortest vector distances are selected and used as *context* to construct a comprehensive output. Such output forms the so called *contextualized prompt*, which is provided, alongside the query, as input to the generator in the next phase.

[3]As defined by the GSF: "*A green software pattern is a specific example of how to apply one or more green principles in a real-world example. Whereas principles describe the theory that underpins green software, patterns are the practical advice software practitioners can use in their software applications today. Patterns are vendor-neutral.*" The goal of each such software green pattern is that its use will reduce software emissions.

To realize the retriever component, we explored various retrieval strategies but found that not all align with the project's specific application scenario. For instance, the Contextual Compression Retriever employs a large language model to summarize text chunks for more accurate searches when dealing with excessively long texts. However, given that we are working with a relatively small dataset consisting of only one architectural document at a time, the resulting text chunks are concise, rendering this retriever unsuitable for our purposes. Thus, three retrievers were chosen for comparison to empirically assess their suitability for EcoDocSense, all of which use vector distance to calculate text similarity:

- The **Chroma Retriever enhanced with Maximal Marginal Relevance (MMR)** [4], which balances query relevance and result diversity by using vector distances between text segments based on their embedding vectors. It prioritizes segments most similar to the query while diversifying results by considering the independence of selected segments, reducing redundancy and offering varied perspectives [19].
- The **Multiquery Retriever**, which employs the LLM model; in our case, we chose Llama2 [20], to broaden the query's semantic scope by generating 3-5 variations of the original query, capturing a wider range of related information [21]. Each variation is assessed using vector distance measurements to ensure comprehensive results that closely match the user's informational needs.
- The **Ensemble Retriever**, which combines the strengths of two distinct retrievers to enhance retrieval performance; in our solution, we utilize BM25 [22] and Faiss [23]. BM25 excels in keyword-based searches by analyzing word frequency and term importance, while Faiss retrieves semantically similar content based on vector similarity, even without exact keyword matches. Integrating these methods provides a more precise and robust search experience through both keyword precision and semantic understanding.

The design and results of our empirical comparison of the above retrievers are discussed in Section III.

C. Generation

The generation phase is responsible for answering a given query (i.e., identifying a given pattern) given the contextualized prompt retrieved in the previous phase (i.e., given the most relevant information to the query as retrieved from the architectural document). This ultimately enables us to determine whether specific green software patterns are being considered in the architectural document.

This phase involves three key components: selecting an appropriate language model; optimizing the language model's performance with fine-tuning; and engineering effective prompts.

[4]MMR considers the similarity of keywords/keyphrases with the document, along with the similarity of already selected keywords and keyphrases. This results in a selection of keywords that maximize their diversity with respect to the document.

To identify a suitable LLM model, we focus on five open-source models to experiment with: LLaMA2 [20], LLaMA3 [24], LLaMA3-ChatQA [25], Mistral [26], and Phi-3 [27]. These models were chosen for their lightweight nature, ease of deployment, and fine-tuning capabilities.

We conducted a manual review of the responses generated by these models. Mistral was inadequate in making relevant judgments. LLaMA2 provided more detailed explanations but frequently affirmed the presence of a pattern even when it was not explicitly mentioned in the document. For example, it incorrectly suggested that the "cache static data" pattern was being followed, even though the context merely mentioned storing data in a database without any reference to caching. LLaMA2's responses often implied that if a feature could potentially be integrated into the system, it should be considered present. In contrast, LLaMA3 demonstrated strong writing skills but was slower in processing, and LLaMA3-ChatQA struggled with analyzing lengthy documents. Phi-3, however, delivered high-quality answers quickly, with responses that were more concise and context-aware than those of LLaMA2. Unlike LLaMA2, which was overly optimistic about identifying green patterns, Phi-3 provided more cautious and precise evaluations, making it more suitable for our task. Moreover, we experimented the use of Phi-3 with and without fine-tuning. The design and results of our experiments are discussed in Section III.

Prompt engineering was also crucial. Prompt engineering is the process of structuring an instruction that can be interpreted and understood by a generative AI model. We found that the structure and content of the prompts significantly affected the quality of the output, leading us to experiment with different prompts. Due to space constraints, we provide a description of the prompts we compared in our on-line repository [9] We ultimately selected the prompt *"Act as a professional assistant in software development and use this as a context* $< context >$ *to answer this question* $< query >$*"*, which delivered promising results both before and after fine-tuning. This process involved considerable trial and error. Although there may be better prompts available, this one proved sufficient for our prototype by providing consistent results. .

D. Web Interface and Sustainability Report

EcoDocSense features a modern web interface where engineers can upload their document and analyses the results by using intuitive and interactive charts (see Figure 2) as well as a comprehensive sustainability report containing both detailed pattern compliance assessments and actionable recommendations (see Figure 3). The report categorizes green patterns and offers a clear overview of which patterns are being implemented, which are not, and which are irrelevant to a specific architectural style. The findings are visually presented using bar charts and pie charts. Additionally, the report includes detailed recommendations on which patterns should be adopted and how they can be integrated into the software development lifecycle to help reduce the environmental carbon footprint. An example of report can be found on GitHub.

TABLE II: Fleiss Kappa and Agreement Results

Architecture Document	Fleiss Kappa	No. of Queries with 100% Agreement Among 5 Raters
Uber	0.945	52/54
Netflix	0.959	43/45
Instagram	0.964	47/48
Dropbox	0.985	45/46
Whatsapp	0.984	45/46

III. Evaluation

To assess the feasibility of our solution, we formulated three research questions designed to evaluate the practicality and effectiveness of different approaches for the retrieval and generator phases, allowing us to identify the most fruitful strategies.

RQ1: Effectiveness of Retrieval Strategies: What is the effectiveness of the different retrieval strategies within our framework? To evaluate this, we assess the performance of each retrieval strategy described in Section II-B, thus identifying which strategy delivers optimal results.

RQ2: Impact of Fine-Tuning: To what extent does fine-tuning the language model enhance the performance of the generator? This is evaluated by comparing the outcome of the Phi3 fine-tuned model with the outcome of the base model (i.e., Phi3 without fine-tuning).

RQ3: EcoDocSense Overall Performance: What combination of retrieval and generator strategies does provide the best performance for EcoDocSense? To measure the overall performance of EcoDocSense, we analyzed the combination of the three retrieval strategies analyzed in RQ2 with the best Phi3 large language model from RQ2 (tuned and non tuned) and empirically assessed their effectiveness to accurately interpret and convey sustainability-related information in software architectural documents.

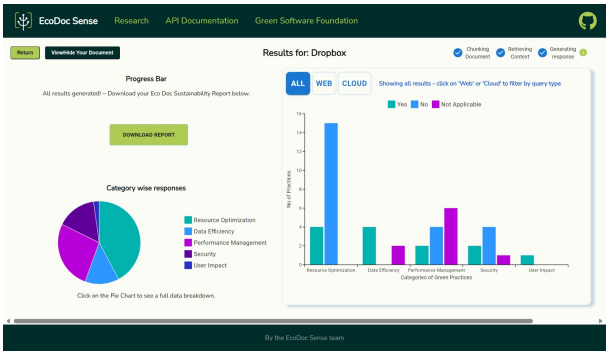
A. Benchmark Documents

To answer the above RQs, we require software architecture documents to serve as a benchmark. To this end, we identified five documents describing the architectural styles of Instagram, WhatsApp, Dropbox, Uber, and Netflix. These documents are publicly available from GeeksforGeeks [8].[5]

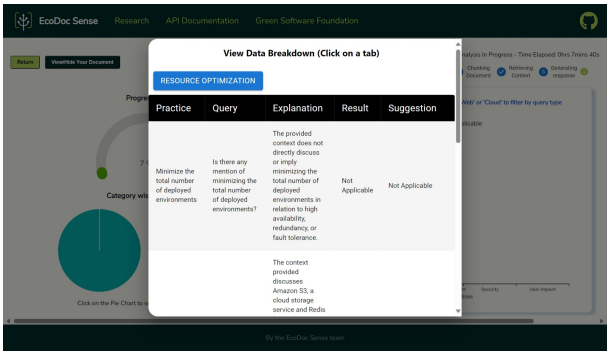
Initially, we considered using official software documentation from Apache projects. However, these documents were too large and complex, making them impractical for processing with our LLM due to the manual effort required to review and analyze content related to green software patterns. Consequently, we opted for these five documents from GeeksforGeeks, which were more concise and easier to understand, enabling us to manually curate a reliable ground truth dataset.

In order to curate the ground truth data, five authors consulted each of the five documents and provided an answer for each of the green software patterns (i.e., queries) in our study. Precisely, each query was answered by each of the five authors with "yes," "no," or "not applicable," depending on whether the pattern described in the query was applicable to

[5]This usage complies with the terms of use of GeeksforGeeks [28].

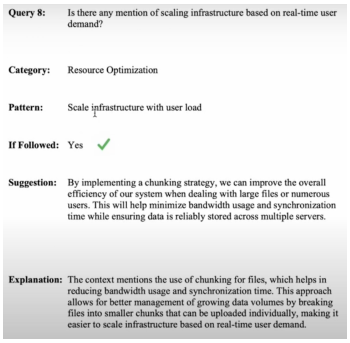


(a) Sustainability Analysis Overview



(b) Detailed results

Fig. 2: EcoDocSense Web Interface



Query 8: Is there any mention of scaling infrastructure based on real-time user demand?

Category: Resource Optimization

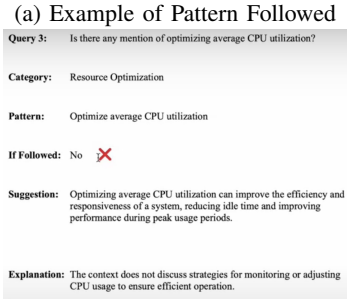
Pattern: Scale infrastructure with user load

If Followed: Yes ✓

Suggestion: By implementing a chunking strategy, we can improve the overall efficiency of our system when dealing with large files or numerous users. This will help minimize bandwidth usage and synchronization time while ensuring data is reliably stored across multiple servers.

Explanation: The context mentions the use of chunking for files, which helps in reducing bandwidth usage and synchronization time. This approach allows for better management of growing data volumes by breaking files into smaller chunks that can be uploaded individually, making it easier to scale infrastructure based on real-time user demand.

(a) Example of Pattern Followed



Query 3: Is there any mention of optimizing average CPU utilization?

Category: Resource Optimization

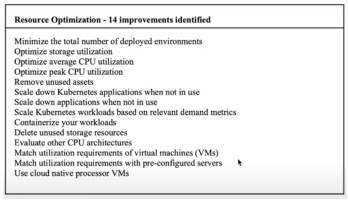
Pattern: Optimize average CPU utilization

If Followed: No ✗

Suggestion: Optimizing average CPU utilization can improve the efficiency and responsiveness of a system, reducing idle time and improving performance during peak usage periods.

Explanation: The context does not discuss strategies for monitoring or adjusting CPU usage to ensure efficient operation.

(b) Example of Pattern Not Followed



Resource Optimization - 14 improvements identified

Minimize the total number of deployed environments
Optimize storage utilization
Optimize average CPU utilization
Optimize peak CPU utilization
Remove unused assets
Scale down Kubernetes applications when not in use
Scale down applications when not in use
Scale Kubernetes workloads based on relevant demand metrics
Containerize your workloads
Delete unused storage resources
Evaluate other CPU architectures
Match utilization requirements of virtual machines (VMs)
Match utilization requirements with pre-configured servers
Use cloud native processor VMs

(c) Example of Improvements Identified

Fig. 3: EcoDocSense Sustainability Report

the architecture described in a given document and, if so, whether it was being followed or not. For each of the five documents, the final response to each query was determined through majority voting among the five independent reviewers. The inter-rater reliability was quantified using Fleiss' Kappa, which is a statistical measure evaluates the consistency among different raters [29]. The results are shown in Table II. We can observe that the inter-rater agreement for each benchmark document is very high, with kappa values ranging from 0.94 to 0.98, indicating almost perfect agreement.

B. RQ1: Effectiveness of Retrieval Strategies

To evaluate the three retrieval strategies outlined in Section II-B we documented the chunks retrieved by each of the methods. Two authors conducted a manual review of all retrieved chunks to assess whether they contained the relevant information necessary to accurately answer the queries, based on the ground truth data described III-A.

Table III presents the comparative performance of the retrieval strategies applied to the ground truth dataset. We can observe that the results are closely aligned, likely due to the limited number of documents analyzed. Although the Multi-query Retriever slightly exceeded the others in the number of correct retrievals, it often returned verbose contexts containing irrelevant information. Moreover, during our experiment, we discovered large differences in runtime performance among the retrievers. The Ensemble Retriever, requiring the loading of two separate retrievers, took approximately 142.4 seconds just to create an instance. Additionally, the Multiquery Retriever had a query processing time of around 22.6 seconds per query, which was nearly 10 times slower than the Chroma Retriever with MMR. In fact, Chroma Retriever with MMR demonstrated superior performance, completing the entire process in just 2.4 seconds. Given that Chroma with MMR strikes the best accuracy-runtime trade off, it is the most desirable retriever for EcoDocSense.

C. RQ2: Impact of Fine-Tuning

We experimented the use of Phi-3 with and without fine-tuning. To this end, we compare the baseline model (Origin Phi-3) with 8 versions of fine-tuned models different in the number of epochs or learning rate. In particular, we explore the following parameters:

- Temperature - Set to 0.8 to balance creativity and consistency, based on satisfactory results given by it.

TABLE III: RQ1. Number of correct retrievals for all the queries.

Retriever	Netflix	Uber	Instagram	Dropbox	Whatsapp	Total
Chroma Retriever with MMR	43/44	51/53	46/47	41/45	39/45	220/234
Multiquery Retriever	43/44	51/53	47/47	44/45	43/45	228/234
Ensemble Retriever	41/44	51/53	46/47	43/45	40/45	221/234

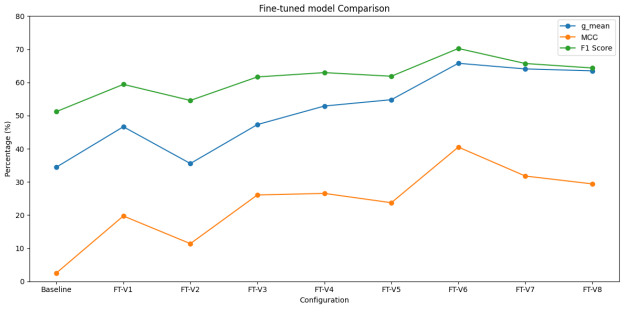


Fig. 4: Comparison of Phi-3 Fine-tuned Models VS. Base in terms of F1, G-mean and MCC.

- Epochs - We tested 1-3 epochs initially, increasing to 10 and 30, which decreased training loss[6], indicating improved performance.
- Learning rate[7] - Started at 2e-4 (0.0002) for fewer epochs, adjusted to 2e-5, which is 10 times slower for stability with more epochs.

Since no existing dataset was available to carry out fine tuning, we created a synthetic dataset by using ChatGPT-4, producing 1,600 entries for training [30]. This approach allowed us to improve Phi-3 without the need for complex data collection. Although this dataset is relatively small compared to what is typically used for fine-tuning models, it was sufficient for exploring the viability of synthetic data generation under time constraints.

We evaluated the performance of these models by comparing each model's responses to the ground truth based on three measures: F1 score, G-Mean and Matthews Correlation Coefficient (MCC) [31]. These measures are particularly suitable for imbalanced datasets. The G-Mean balances sensitivity (true positive rate) and specificity (true negative rate), offering a comprehensive view of model performance across all classes. In contrast, MCC provides a single summary statistic that reflects the quality of predictions, considering both true and false positives and negatives. The F1 score, which balances precision and recall, was chosen for its ability to account for both true positives and false negatives, critical in identifying green patterns in architecture documents [31]. These peculiarities make them suitable for cases of class imbalance, evident in our dataset where non-followed patterns outnumbered adhered ones, aligning with our pipeline's objectives.

Figure 4 compares models using the F1 score, configuration labels in the diagram represent the baseline model (Origin Phi-3) and 8 fine-tuned models. Among all models, FT-V6 (Version 6 of Phi-3 fine-tuned for 30 epochs) was chosen for its efficiency and effectiveness. The best model achieved a 70% F1 score, a 19% improvement over the baseline Phi-3 score of 51%, and also reduced generation time to about 6 minutes per document.

[6]Training loss measures how well the model fits the training data, indicating learning effectiveness. A decreasing training loss suggests better model performance on the training set.

[7]The learning rate is a tuning parameter dictating the step size in minimizing the loss function.

D. RQ3: EcoDocSense Performance

In the previous RQs, we independently tested different retrievers (RQ1) and fine-tuning strategies (RQ2). This RQ3 explores whether their combined use could enhance overall performance, aiming at identifying the best combination to be used for EcoDocSense. To this end, we considered six different configurations by pairing each retriever with both the non-tuned Phi3 generator (which we refer to as Base) and the best fine-tuned Phi3 generator found in RQ2 (namely FT_V6). We use the same benchmark data used for the previous RQs.

Table IV reports the results of each configuration in terms of G-mean, MCC and F1 measures; we also include sensitivity and specificity for completeness.

We can observe that the baseline models frequently exhibited low sensitivity (ranging from 0% to 16.67%) and high specificity (ranging from 89.02% to 100%), thus resulting in lower G-Mean and MCC values. The worst configuration, namely the MultiQuery_Base model achieves 0% sensitivity, an MCC of -2.67 and an F1 of 46.23, indicating a significant failure to correctly identify positive instances. Such deficiencies are critical in applications like software architecture documentation, where the accurate identification of key patterns is essential.

Fine-tuned models, on the other hand, generally performed better, showing enhanced sensitivity (ranging from 26.67% to 40%), which coupled with a high specificity (ranging from 76.96% to 92.67%) positively impacts the other measures. For example, the Chroma_Fine-Tuned configuration achieved a G-Mean of 65%, an MCC of 40% and a F1 of 64.35%. This represents a substantial improvement over the base models. These results highlight that fine-tuning enhances the model's ability to detect true positives while maintaining high specificity. The improved G-Mean and MCC values demonstrate a more balanced and effective model, underscoring the importance of using a set of diverse metrics in evaluating performance, especially in the context of imbalanced data [31].

E. Limitations

Our primary goal was to develop a minimal viable product in collaboration with the Green Software Foundation in order to assess the proposed idea and product's viability within under four months. While this first prototype of EcoDocSense demonstrates positive results, it also has several limitations

TABLE IV: RQ3. Sensitivity, Specificity, G-mean, MCC and F1 for different configurations

Configuration (Retriever_Generator)	Sensitivity (%)	Specificity (%)	G-mean (%)	MCC (%)	F1 (%)
Chroma_Base	13.33	89.01	34.45	2.53	51.20
MultiQuery_Base	0.00	99.48	0.00	-2.67	46.23
Ensemble_Base	16.67	100.00	40.82	38.39	61.21
Chroma_Fine-Tuned	46.67	92.67	65.76	40.51	64.35
MultiQuery_Fine-Tuned	26.67	80.63	46.37	6.20	52.63
Ensemble_Fine-Tuned	40.00	76.96	55.48	13.36	55.25

resulting from the given time constraints. Aware of these, we prioritized completing the system end-to-end with all planned functionalities and documented the shortcomings.

A significant threat to the generalizability of our results is the evaluation based on only five architectural documents, which was mainly due to the large manual effort required to create the ground truth dataset. The use of a fine-tuning dataset synthetically generated by using LLMs, is also affecting the internal validity of our study. This issue remains unless the data is manually validated by software practitioners.

Another key limitation is the focus on extracting and processing textual content from software architecture documents, with limited attention to image-based information. These documents often include technical diagrams that visually summarize software aspects and may contain annotations about specific technologies not mentioned in the text, making it important to capture such visual details. To tackle this, we experimented with an image extraction component using the LLaVA model [32] to generate visual summaries. However, initial results showed hallucinations, with the model incorrectly referencing irrelevant technologies or layers. Due to these inaccuracies, further improvements were deferred, and this feature has been made optional in EcoDocSense for engineers to enable or disable as needed.

Additionally, the system's suggestions highlight green patterns' benefits but lack guidance on implementing these practices. This is because the model was fine-tuned to emphasize importance rather than practical implementation, requiring further training for more specific guidance.

IV. Related Work

A. Sustainability in the Software Development Lifecycle

Research in green IT has mainly focused on hardware energy efficiency [33], yet delivering sustainable solutions across the entire software lifecycle is becoming more and more prominent [2], [4]. Agarwal et al. [34] highlight the importance of integrating sustainability in software development and Rashid and Khan [33] identify agile practices that enhance sustainability in software development. Both studies suggest verifying green practices during the design phase as a way to improve software sustainability. Moreover, researchers have devised different ways to derive green design patterns. For example, Shanbhag et al. [35] synthesize eight energy patterns for deep learning projects from an analysis of Stack Overflow posts. The patterns were subsequently validated with a questionnaire survey with 14 practitioners. Subsequently, Järvenpää et al. [36] extended the scope and coverage of their work by including all forms of ML, and deriving green architectural tactics from scientific literature. This ultimately led to the 30 green tactics, which also include the 8 energy patterns from Shanbhag et al. [35].

B. Tools Computing Software Environmental Impacts

Tools like the AWS Carbon Footprint Tool [37], the Microsoft Sustainability Calculator [38], and the Cloud Carbon Footprint [39] assess environmental impacts by monitoring resource usage and carbon emissions post-development. The Green Software Foundation has developed the "Impact Framework" [40], which simplifies the calculation of a software's environmental impact by using custom manifest files.[8] However, these tools do not enable users to make an assessment of the environmental impact or compute carbon footprints at the design stage. The AWS Well-Architected Tool [41] is the only one handling environmental impact at the design stage, as it aims at guiding architects and developers on best practices for designing AWS workloads across six pillars, namely 'Operational excellence,' 'Security,' 'Reliability,' 'Performance efficiency,' 'Cost optimization,' and 'Sustainability.' Each pillar has specific practices to evaluate a workload's adherence by asking users predefined questions, assessing the risks of not implementing practices, and prioritizing them accordingly. A limitation of this tool is its reliance on users to manually identify practices rather than automating this process. If practices are missing, an improvement plan is created. Our proposed framework, instead, enable software architects and engineers to automatically assess the compliance of their software architecture with respect to a catalog of green software patterns in order to evaluate their software environmental impact at design stage.

V. Conclusion and Future Work

In this paper we proposed EcoDocSense, a framework that can be used to integrate a sustainability perspective into software architecture documents at design phase. We realised EcoDocSense as an end-to-end mechanism, which serves as a proof of concept, demonstrating the feasibility of our idea. While EcoDocSense has some limitations and requires further work to be ready for a broader adoption, its status as the first initiative of its kind, combined with its open-source nature and the development of a fully functional MVP, supported by the Green Software Foundation, has laid a foundation for future

[8]Manifest files are YAML files which are specially structured to hold user inputs and generated outputs, making it easier to communicate the software's environmental impact.

advancements. Next steps involve the integration of EcoDocSense with the the Impact Framework [40]; this enhancement would allow one to obtain a quantitative evaluation of the carbon emissions associated with different architectural choices [42]. Moreover, EcoDocSense currently focuses solely on web and cloud green patterns yet, thanks to its extensible queries design, it could be expanded to incorporate additional architectural types, for example targeting AI-driven applications [4], [35], [43]. By extending EcoDocSense's knowledge base, one can increase the number of green patterns it recognizes, thereby covering a broader spectrum of sustainable practices. Future work will also focus on enhancing the extraction of information. At present, the framework primarily handles text data extracted from PDF software architecture documents. We aim to extend its capabilities to process other types of documents and data, including, for example, tabular data and technical diagrams. This integration will lead to a more comprehensive analysis of software architecture documents.

ACKNOWLEDGMENTS

This project was carried out for the MSc Software Systems Engineering degree at UCL in collaboration with the Green Software Foundation under the Industry Exchange Network.

REFERENCES

[1] S. Podder, A. Burden, S. K. Singh, and R. Maruca, "How green is your software?" 2020. [Online]. Available: https://hbr.org/2020/09/how-green-is-your-software

[2] A. Fonseca, R. Kazman, and P. Lago, "A manifesto for energy-aware software," *IEEE Software*, vol. 36, no. 6, pp. 79–82, 2019.

[3] R. Verdecchia, J. Sallou, and L. Cruz, "A systematic review of green ai," *WIREs Data Mining and Knowledge Discovery*, vol. 13, no. 4, p. e1507, 2023. [Online]. Available: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1507

[4] T. Menzies and B. Johnson, "Powering down: An interview with Federica Sarro on tackling energy consumption in ai-powered software systems," *IEEE Software*, vol. 41, no. 5, pp. 89–92, 2024.

[5] M. Freed, S. Bielinska, C. Buckley, A. Coptu, M. Yilmaz, R. Messnarz, and P. M. Clarke, "An investigation of green software engineering," in *Systems, Software and Services Process Improvement*. Springer Nature, 2023, pp. 124–137.

[6] G. S. Foundation, "Gsf website," 2024. [Online]. Available: https://greensoftware.foundation/

[7] GSF, "Green software patterns - catalog," 2024. [Online]. Available: https://patterns.greensoftware.foundation/catalog/

[8] GeeksforGeeks, "System design tutorial," 2024. [Online]. Available: https://www.geeksforgeeks.org/system-design-tutorial/

[9] N. et al., "Ecodocsense - github repository," 2024. [Online]. Available: https://github.com/KungfuNaman/green-software-foundation

[10] ——, "Ecodocsense - github repository," 2024. [Online]. Available: https://www.youtube.com/watch?v=BybWQCZ-pXo

[11] H. Touvron, L. Martin, K. Stone *et al.*, "Llama 2: Open foundation and fine-tuned chat models," 2023. [Online]. Available: https://arxiv.org/abs/2307.09288

[12] Artifex, "Pymupdf4llm," 2024. [Online]. Available: https://pymupdf.readthedocs.io/en/latest/pymupdf4llm/

[13] R. Schwaber-Cohen, "Chunking strategies for llm applications," 2023. [Online]. Available: https://www.pinecone.io/learn/chunking-strategies/

[14] M. C. Rillig, M. Ågerstrand, M. Bi, K. A. Gould, and U. Sauerland, "Risks and benefits of large language models for the environment," *Environmental Science Technology*, vol. 57, pp. 3464–3466, 2023. [Online]. Available: https://pubs.acs.org/doi/epdf/10.1021/acs.est.3c01106

[15] Turing, "A guide on word embeddings in nlp," 2024. [Online]. Available: https://www.turing.com/kb/guide-on-word-embeddings-in-nlp

[16] AWS, "What is a vector database?" 2024. [Online]. Available: https://aws.amazon.com/what-is/vector-databases/

[17] G. S. Foundation, "Green software patterns - catalog," 2024. [Online]. Available: https://patterns.greensoftware.foundation/

[18] Pinecone, "What is a vector database & how does it work?" 2023. [Online]. Available: https://www.pinecone.io/learn/vector-database/

[19] K. Sakkaravarthy Iyyappan and S. Balasundaram, "Phrase embedding based multi document summarization with reduced redundancy using maximal marginal relevance," in *2020 Inter. Conference on Electrical Engineering and Informatics (ICELTICs)*, 2020, pp. 1–5.

[20] Meta, "Llama 2: open source, free for research and commercial use," 2024. [Online]. Available: https://llama.meta.com/llama2/

[21] T. AI, "Enhance retrieval with langchain multi-query retriever for rag," 2023. [Online]. Available: https://www.toolify.ai/ai-news/enhance-retrieval-with-langchain-multiquery-retriever-for-rag-996777/

[22] S. Robertson and H. Zaragoza, "The probabilistic relevance framework: Bm25 and beyond," *Found. Trends Inf. Retr.*, vol. 3, no. 4, p. 333–389, April 2009. [Online]. Available: https://doi.org/10.1561/1500000019

[23] Langchain, "Faiss," 2024. [Online]. Available: https://python.langchain.com/v0.2/docs/integrations/vectorstores/faiss/

[24] Meta, "Llama 3," 2024. [Online]. Available: https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-3/

[25] Nvidia, "Llama3-chatqa-1.5-70b," 2024. [Online]. Available: https://huggingface.co/nvidia/Llama3-ChatQA-1.5-70B

[26] Mistral, "Mistral ai," 2024. [Online]. Available: https://mistral.ai/

[27] Microsoft, "Introducing phi-3: Redefining what's possible with slms," 2024. [Online]. Available: https://azure.microsoft.com/en-us/blog/introducing-phi-3-redefining-whats-possible-with-slms/

[28] GeeksforGeeks, "Terms of use," 2023. [Online]. Available: https://www.geeksforgeeks.org/legal/terms-of-use/

[29] DATAtab, "Fleiss kappa," 2024. [Online]. Available: https://datatab.net/tutorial/fleiss-kappa

[30] Turing, "Synthetic data generation: Definition, types, techniques, and tools," 2024. [Online]. Available: https://www.turing.com/kb/synthetic-data-generation-techniques

[31] R. Moussa and F. Sarro, "On the use of evaluation measures for defect prediction studies," in *Procs. of ISSTA'22*, p. 101–113. [Online]. Available: https://doi.org/10.1145/3533767.3534405

[32] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Llava: Large language and vision assistant," 2023. [Online]. Available: https://llava-vl.github.io/

[33] N. Rashid and S. U. Khan, "Agile practices for global software development vendors in the development of green and sustainable software," 2018. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1002/smr.1964

[34] S. Agarwal, A. Nath, and D. Chowdhury, "Sustainable approaches and good practices in green software engineering," *Inter. Journal of Research and Reviews in Computer Science*, vol. 3, pp. 1425–1428, 2012.

[35] S. Shanbhag, S. Chimalakonda, V. S. Sharma, and V. Kaulgud, "Towards a catalog of energy patterns in deep learning development," in *Procs. of EASE'22*, 2022, p. 150–159. [Online]. Available: https://doi.org/10.1145/3530019.3530035

[36] H. Jarvenpaa, P. Lago, J. Bogner, G. Lewis, H. Muccini, and I. Ozkaya, " A Synthesis of Green Architectural Tactics for ML-Enabled Systems ," Los Alamitos, CA, USA, pp. 130–141, Apr. 2024. [Online]. Available: https://doi.ieeecomputersociety.org/

[37] AWS, "Aws customer carbon footprint tool overview," 2024. [Online]. Available: https://aws.amazon.com/aws-cost-management/aws-customer-carbon-footprint-tool/

[38] Microsoft, "Sustainability calculator." [Online]. Available: https://azure.microsoft.com/en-us/blog/microsoft-sustainability-calculator-helps-enterprises-analyze-the-carbon-emissions-of-t

[39] T. Inc., "Cloud carbon footprint tool," 2024. [Online]. Available: https://www.cloudcarbonfootprint.org/

[40] GSF, "Impact framework," 2024. [Online]. Available: https://if.greensoftware.foundation/?ref

[41] AWS, "Sustainability pillar - aws well-architected framework," 2024. [Online]. Available: https://docs.aws.amazon.com/wellarchitected/latest/sustainability-pillar/sustainability-pillar.html

[42] S. Rakhunathan and N. Balani, "Emission calculations through large language model," 2024. [Online]. Available: https://greensoftware.foundation/articles/emission-calculations-through-large-language-model

[43] S. Georgiou, M. Kechagia, T. Sharma, F. Sarro, and Y. Zou, "Green AI: do deep learning frameworks have different costs?" in *Procs. of ICSE'22*. [Online]. Available: https://doi.org/10.1145/3510003.3510221