



Master's thesis
Master's Programme in Data Science

Comparative Assessment of Large Language Model-Driven Recommendation Systems in Smart Spaces

Sergei Panarin

May 22, 2025

Supervisor(s): Dr. Naser Motlagh, Dr. Martha A. Zaidan

Examiner(s): Prof. Sasu Tarkoma

UNIVERSITY OF HELSINKI
FACULTY OF SCIENCE
P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki

Tiedekunta — Fakultet — Faculty		Koulutusohjelma — Utbildningsprogram — Degree programme	
Faculty of Science		Master's Programme in Data Science	
Tekijä — Författare — Author			
Sergei Panarin			
Työn nimi — Arbetets titel — Title			
Comparative Assessment of Large Language Model-Driven Recommendation Systems in Smart Spaces			
Työn laji — Arbetets art — Level		Aika — Datum — Month and year	Sivumäärä — Sidantal — Number of pages
Master's thesis		May 22, 2025	97
Tiivistelmä — Referat — Abstract			
<p>Large Language Models (LLMs) are becoming increasingly prominent in Data Science. This thesis investigates how LLM-driven systems provide intelligent recommendations in smart spaces. Initially, fine-tuning and retrieval-augmented generation are studied in terms of their performance with IoT sensor data from smart spaces. The thesis then evaluates two LLM-based recommendation solutions refined on collected sensor data: (1) FAISS-based retrieval of vectorized sensor data and (2) query generation prompt with an execution agent. The tests are done on several LLMs chosen from the most prominent model families. Performance is assessed using precision, recall, F1 score, query completion time, and token generation latency. The thesis identifies an optimal configuration for further evaluation. In addition, it examines the system's performance when introducing a user preference adjusting mechanism that makes future queries take user preferences into account. The user experience and the quality of the interaction are also evaluated through a survey.</p> <p>The findings of this thesis revealed that the RAG based Query Generation with GPT-4 outperformed all other methods, achieving superior performance. Furthermore, a structured user perception survey showed moderate user satisfaction, slightly lower than the empirically achieved values. These results indicate practical utility and effectiveness of the developed recommendation system, but also signal the need for improvements to the overall user experience. The thesis concludes by emphasizing scalability and user perception studies as critical factors for future developments, along with recommending further extensive evaluations of new models and methods.</p> <p>ACM Computing Classification System (CCS): Information Systems → Recommender Systems Computing methodologies → Artificial Intelligence → Natural Language Processing</p>			
Avainsanat — Nyckelord — Keywords			
NLP, LLM, Information Retrieval, Streamlit, AI, RAG, Recommendation System, Smart Spaces.			
Säilytyspaikka — Förvaringsställe — Where deposited			
Helsinki University Library			
Muita tietoja — Övriga uppgifter — Additional information			

Acknowledgements

This master's thesis was completed at the Department of Computer Science, Helsinki University. I gratefully acknowledge the financial support provided by Research Council of Finland with project "evoS3: Evolution of Smart Space Systems", and by other projects within the 6G Research Group.

I would like to express my sincere gratitude to my supervisors, Dr. Naser Hossein Motlagh and Dr. Martha Arbayani Zaidan, for their continuous guidance, valuable insight, and unwavering support throughout the course of this thesis. Their expertise and patience made this research possible. I would like to extend my gratitude to Prof. Sasu Tarkoma for serving as the external reviewer of this thesis. I also would like to thank the members of the IRON Research Group for their helpful feedback, good discussions, and encouragement during various stages of my work. This appreciation extends to University of Helsinki's Departments of Computer Science and Data Science staff for providing the necessary resources and a supportive environment.

A special note of gratitude goes to my family and friends, particularly my parents, and to my dear university friends, Xinyang Chen, Haruka Buss and Max Väistö, whose moral support made this journey enjoyable and fulfilling. Outside of the university community, I am grateful to Sandra Sobolewska, whose constant encouragement and belief in me provided strength when it was most needed.

Contents

1	Introduction	1
1.1	Motivation	2
2	Theoretical background: Large Language Models	5
2.1	Fine-tuning and embeddings	6
2.2	Retrieval-Augmented Generation	9
2.3	Facebook AI Similarity Search and Vector search	11
3	Overview of Large Language Model Families	13
3.1	GPT Family of Models	13
3.1.1	Architecture and Training	14
3.1.2	Open Challenges	15
3.2	BERT Family of Models	16
3.2.1	Architecture	16
3.2.2	Variations and further improvements	17
3.3	Llama Family of Models	18
3.3.1	Architecture	18
3.3.2	Variations and further improvements	19
3.4	Gemini Family of Models	21
3.4.1	Architecture	21
3.4.2	Performance and limitations	21
3.5	Claude Family of Models	23

3.5.1	Architecture	23
3.5.2	Performance and limitations	23
3.6	LLM selection	24
4	Recommendation Systems	29
4.1	Traditional approaches	29
4.2	LLM integration	30
5	Smart Spaces Data	33
6	Fine-tuning or RAG?	37
6.1	Remote setup	38
6.1.1	Fine-tuning	38
6.1.2	RAG	39
6.1.3	Fine-tuning and RAG	39
6.2	Locally based models	40
6.2.1	Fine-tuning	40
6.2.2	RAG	41
6.2.3	Fine-tuning and RAG	42
6.3	Testing results	42
6.3.1	Performance Metrics Overview	43
6.3.2	API / Remote LLM Results	43
6.3.3	Local LLM Results	45
6.3.4	Comparative Analysis and Conclusion	47
7	Vector Search or Query Generation?	49
7.1	FAISS-based retrieval	50
7.1.1	LLM prompts	52
7.2	Query-Based Execution Approach	53
7.2.1	LLM prompts	56

7.3	Preference adjusting	59
7.3.1	Preference Integration Pipeline	63
7.4	Performance metrics	65
7.4.1	Evaluation Benchmarks for Large Language Models	67
8	Results and Evaluation	69
8.1	User Perception Study on the Smart Space Recommendation System	69
8.2	Heat Mapping Process	71
8.3	Streamlit Application Deployment	73
8.4	Results	75
9	Discussion and Future Works	79
10	Conclusions	83
	Bibliography	85
	Appendix A User-feedback questionnaire	97

1. Introduction

Large Language Models (LLMs) are revolutionizing the field of data analysis and the management of Big Data. These models, powered by deep learning and advanced neural network architectures, are able to process large amounts of text data to uncover patterns, relationships, and insights that were previously unattainable [71]. LLMs have caused a change in the way large datasets are handled and analyzed, allowing more intuitive and efficient ways to interpret complex data [75]. By understanding and generating human-like text, LLMs facilitate more natural interactions with data, allowing users to pose queries in everyday language and receive comprehensive, contextually relevant responses. This capability transforms data analysis from a purely technical task into a more accessible and user-friendly process. In the context of Big Data, LLMs enhance the ability to look through massive datasets, identify critical information, and generate actionable insights, thus significantly improving the efficiency and effectiveness of data handling and analysis.

This thesis aims to use and test these LLM capabilities to develop a recommendation system, which will integrate IoT data from various sensors-such as Passive Infrared (PIR) sensors, temperature sensors, humidity sensors, light sensors, and CO₂ sensors-and use these data to generate responses to user queries regarding the most up-to-date data available in data storage. Initially, sensor data are collected related to room temperature, humidity, light intensity, CO₂ levels, and PIR values. The data comes from a study on smart spaces by Motlagh et al. [34]. The data is pre-processed so that the LLM can use them effectively. This can be solved by using LLM techniques, namely fine-tuning and Retrieval-Augmented Generation (RAG). This will adapt a pre-trained

LLM to specialize in interpreting and responding to queries about the data and provide context to specific user queries. Fine-tuning allows for customization of the LLM for a specific domain, while RAG can enable the model to dynamically retrieve and utilize relevant data to generate accurate responses.

The first goal is to develop a query-response system that can operate in real-time and compare different combinations of LLM techniques; whether through API integration or local deployment, RAG, or a combination of RAG and fine-tuning; to decide which approach yields better results. The second stage is to choose and compare the way the recommendations are chosen. The system can be designed to compare the data stored in vectorized form or to allow the LLM to interpret the query and generate the code necessary to run in the database. In addition, a preference mechanism is implemented that allows users to change their preferences on recommendations. The final stage is to compare all the solutions in a range of performance metrics and develop an application in the best-performing solution. The application is then distributed to a target group of users, who will provide feedback on the recommendation system. The application code is available on GitHub*.

1.1 Motivation

The motivation for this thesis revolves around enhancing the interactivity and functionality of smart spaces, specifically within academic campus environments. Smart spaces equipped with IoT sensors generate significant amounts of data related to environmental conditions such as temperature, humidity, occupancy, and air quality. Traditionally, such detailed information has been accessible primarily through studies such as the work by Motlagh et al. [34, 33, 32, 35], on which much of this thesis is based. However, this has limitations when it comes to practical utility for general users.

*github.com/Panarin97/Workspace-Recommender

Large Language Models present a promising solution by transforming raw sensor data into easily understandable and human-readable formats. This transformation can significantly improve human-computer interaction, providing quick and intuitive access to complex data previously reserved for specialized users [75][14][12]. The integration of LLM-driven systems can not only improve the usability and value of smart campus infrastructures, but also ensure their competitive and technological viability in the rapidly evolving landscape of artificial intelligence and data science [31]. Current methodologies utilizing LLMs, such as Retrieval-Augmented Generation, have proven effective in dynamically retrieving and utilizing data to generate accurate, context-specific responses [12]. Techniques such as Active Retrieval-Augmented Generation and Retrieval-Augmented Fine-Tuning (RAFT) further enhance model adaptability to real-time data variations [19][69], offering frameworks for various domains, including data-intensive and rapidly changing environments. RAGLAB was developed as a modular framework for benchmarking and developing RAG algorithms [70]. For sparse or highly variable data, few-shot learning can also be performed with RAG [17]. Knowledge graphs can be integrated into RAG, as discussed by Kang et al. [23]. Research by Karpukhin et al. shows that Dense Passage Retrieval is possible, offering techniques that are highly applicable to querying real-time databases [24]. Although existing studies such as Data Interpreter by Hong et al. [14] or the Traffic Analytics Framework by Wang [57] offer insight into the practical implementation of LLMs, they do not fully address the distinct contextual analysis required by specific smart campus sensor data scenarios. This thesis differentiates itself by evaluating the deployment strategies of LLMs: comparing local hosting versus API access, and comparing search done with vector embeddings with letting the LLM generate code by itself. In addition, user preference handling and adaptability are emphasized, employing feedback-driven mechanisms to continuously refine recommendations.

Through comparative analysis and practical experimentation, this thesis aims to provide actionable insights and recommendations for the effective deployment of LLM-based recommendation systems in smart spaces, ensuring optimal performance and user satisfaction. To achieve this, the research questions were defined as follows:

- **RQ1:** How do FAISS-based retrieval and query generation approaches compare in terms of precision, recall, and F1 score for answering user queries about IoT sensor data?
- **RQ2:** What are the differences in query completion time and token generation latency between the two approaches when using LLMs of varying sizes (GPT-4, Claude 3.5 Sonnet, Gemini) and SLMs (CodeBERT, Codellama 7B)?
- **RQ3:** How accurately can LLMs incorporate user feedback to improve future recommendations or queries?
- **RQ4:** How do users perceive the effectiveness and usability of the smart space recommendation system?

2. Theoretical background: Large Language Models

Large Language Models are a class of artificial intelligence (AI) systems designed to understand, generate, and manipulate human language. These models use a large amount of text data to learn patterns, structures, and the semantic intricacies of language. The underlying architecture typically employs deep learning techniques, particularly transformer networks, which have advanced the field of natural language processing (NLP) [31]. LLMs are built using deep neural networks, specifically the transformer architecture introduced by Vaswani et al. in 2017 [56]. Transformers use self-attention mechanisms to process and generate text, allowing the model to weigh the importance of different words in a sentence when making predictions. LLMs are trained on extensive datasets, often encompassing billions of words from various sources such as books, articles, and websites, such as the newest GPT models from OpenAI, Llama models from Meta AI, or BERT by Google. They are trained to understand context by considering the relationships between words over long distances in a text, which makes them capable of generating coherent and contextually relevant responses. LLMs can generalize from their training data to perform various language tasks, such as translation, summarization, question-answering, and text completion. However, if a model cannot provide the desired output, it can be further enhanced to better process user queries. In the context of human language tasks, models can be either retrained from scratch or fine-tuned with new datasets in order to adjust their weights. If other types of data are required to be processed by the model, Retrieval-Augmented Generation

combined with data preprocessing can be used to transform the data into the format an LLM can understand.

2.1 Fine-tuning and embeddings

Fine-tuning is a process of adapting a pre-trained LLM to a specific task or domain. Although pre-trained LLMs possess broad language understanding capabilities, fine-tuning allows them to specialize by training on a narrower dataset related to the desired application [15]. Firstly, a pre-trained model that serves as the base is needed in order to perform fine-tuning. Then, a dataset relevant to the specific task or domain is gathered. It can be either generated by a data engineer, or can be found from open-source projects such as Kaggle. In the scope of this thesis, this involves collecting sensor data and related documentation and converting those data into a dataset of instructions for the model. The last step is needed, because LLMs operate with natural human language and are generally not very good at interpreting numerical data. In order for the model to be able to process that data, it needs to be turned into a set of instructions that would provide the model with ideal answers based on potential user queries. An example of an entry in such a dataset can be seen in Figures 2.1 2.2 2.3. Furthermore, when manual fine-tuning is performed, manual tokenization has to be performed on the text data. It is the process of breaking down a string of text into smaller pieces called tokens. These tokens can be as small as individual characters or as large as entire words or sub-words, depending on the tokenization strategy used. The purpose of tokenization is to convert the raw text into a format that can be easily processed by the model. For example, "Fine-tuning is easy!" can be tokenized as ["Fine-tuning", "is", "easy", "!"], or ["Fine", "-", "Tuning", "is", "easy", "!"], or ["F", "i", "n", "e", "-", "t", "u", "n", "i", "n", "g", " ", "i", "s", " ", "e", "a", "s", "y", "!"]. These tokens are then converted into numerical representations (embeddings) that the model can process. The model uses these embeddings to generate responses, understand the context, and perform various language tasks [15].


prompt
string · lengths

35+276 76.1%
Arianna has 12 chocolates more than Danny. Danny has 6 chocolates more than Robbie. Arianna has twice as many chocolates as Robbie has. How many chocolates does Danny have?
Write a plot summary for a comedic novel involving Elon Musk and sea travel.
Create a 3 turn conversation between a customer and a grocery store clerk - that is, 3 per person. Then tel...
Write a poem about the sun and moon.
Does Searle believe that AI can think? Explain your

Figure 2.1: The instructional dataset sample from HuggingFace. The "prompt" column.


completion
string · lengths

1+312 69.4%
Denote the number of chocolates each person has by the letter of their first name. We know that $A = D + 12$ $D = R + 6$ $A = 2 * R$ Thus, $A = (R + 6) + 12 = R + 18$ Since also $A = 2 * R$, this means $2 * R = R + 18$ Hence $R = 18$ Hence $D = 18 + 6 = 24$
Elon Musk hires a team of experts to build the ultimate yacht, but when the yacht is completed, he...
Clerk: How are you doing today? Customer: Great! I'm buying groceries to cook my wife a homemade steak and...
The sun and the moon, the guards from the sky One works the day, other watches the night As the rays of...
No, Searle does not believe that AI can think. Here is

Figure 2.2: The instructional dataset sample from HuggingFace. The "completion" or "response" column

meta
dict
<pre>{ "id": 0, "motivation_app": null, "prompt": "Arianna has 12 chocolates more than Danny. Danny has 6 chocolates more than Robbie. Arianna has twice as many chocolates as Robbie has. How many chocolates does Danny have?", "input": null, "completion": "Denote the number of chocolates each person has by the letter of their first name. We know that\nA = D + 12\nD = R + 6\nA = 2 * R\n\nThus, A = (R + 6) + 12 = R + 18\nSince also A = 2 * R, this means 2 * R = R + 18\nHence R = 18\nHence D = 18 + 6 = 24", "source": "surge", "category": "Question Answering", "subcategory": "Math" }</pre>
<pre>{ "id": 1, "motivation_app": null, "prompt": "Write a plot summary for a comedic novel involving Elon Musk..." }</pre>
<pre>{ "id": 2, "motivation_app": null, "prompt": "Create a 3 turn conversation between a customer and a grocery..." }</pre>
<pre>{ "id": 3, "motivation_app": null, "prompt": "Write a poem about the sun and moon.", "input": null, "completion": "The sun and moon are both so bright, They shine upon the earth at night. The sun is hot and full of life, The moon is cool and full of strife. They both are part of our world, And both are part of our soul. The sun is the heart of our day, The moon is the heart of our night. They both are part of our world, And both are part of our life." }</pre>
<pre>{ "id": 4, "motivation_app": null, "prompt": "Does" }</pre>

Figure 2.3: The instructional dataset sample from HuggingFace. The metadata column.

This dataset was obtained from the HuggingFace platform dataset archive. The HuggingFace is a platform for all AI and Deep Learning enthusiasts, it hosts various training datasets, LLMs trained or fine-tuned by other users, and more. In this particular dataset, the prompt column indicates potential user queries, while the column completion provides a desired response to the corresponding query. This particular dataset has only 327 rows and is used for demonstration purposes. Once the dataset has been generated, it can be split for testing and training purposes. The model is trained on the train dataset and tested against the test dataset. The output is compared with the test data set and the performance of the model is evaluated. During this phase, the model adjusts its parameters to better fit the new data, improving its performance on domain-specific tasks. The general purpose behind fine-tuning is

to enable the model to perform specialized tasks more accurately by leveraging the knowledge acquired during pre-training and adapting it to the nuances of the new train data. This approach is useful for formatting the way an LLM responds to user queries; however, it has limits to understanding numerical data. Once trained and (if) fine-tuned, the model remains static without further improvement unless it is subjected to fine-tuning again. This presents challenges in applying this approach to data analysis. The fine-tuning approach can be used to transform the way an LLM responds to the user query, but the provision of accurate numerical data is challenging, because the process needs to be repeated for every time there is new data available. The process of fine-tuning is illustrated in Figure 2.4.

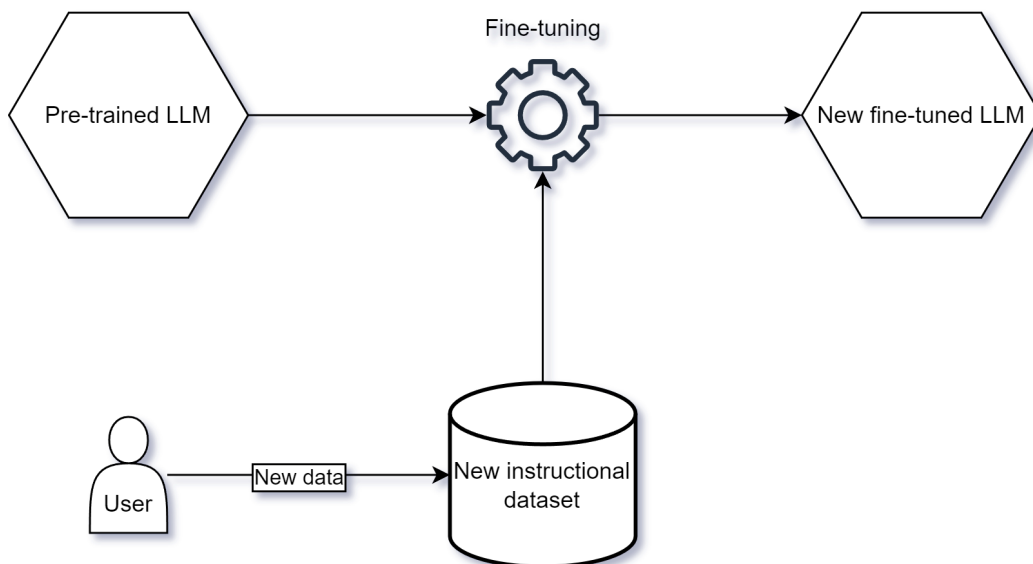


Figure 2.4: The fine-tuning process of an LLM.

2.2 Retrieval-Augmented Generation

Retrieval-Augmented Generation, or simply RAG, is an approach that combines the strengths of retrieval-based and generation-based methods in NLP. It improves the ability of LLMs to generate accurate and contextually relevant responses by incorporating external knowledge sources. It involves retrieving relevant documents or pieces of infor-

mation from a large corpus or database. This is achieved using a retriever model that identifies and ranks documents based on their relevance to the input query. Most of the time it is accompanied by an agent that acts on the data and retrieves the necessary information [28]. For example, a user asks a query about some SQL-formatted data but uses natural language to describe the desired output. The retriever is then tasked with deciphering that information into an SQL query that is then executed by the agent. The agent returns the results of the query to the retriever model. Once relevant documents are recovered, they are fed into a generation model. The model uses this retrieved information to generate a response that is informed by external knowledge. Possibly the biggest benefit of RAG is that RAG allows models to integrate up-to-date information dynamically, making them more versatile and applicable in real-time scenarios. It creates a system where the LLM can retrieve sensor data from a database and generate answers based on these data without the need to retrain the model every time the data are updated. The RAG process is illustrated in Figure 2.5.

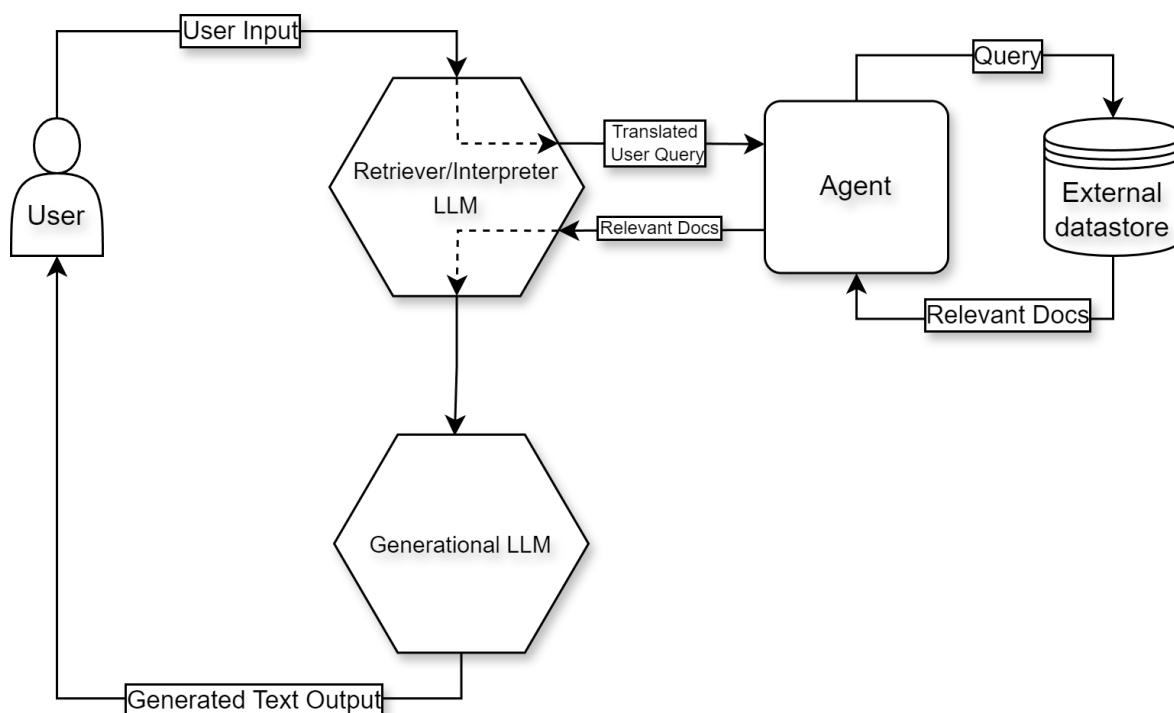


Figure 2.5: Retrieval-Augmented Generation process.

Sometimes an agent is not necessary due to the nature of the user query. In

that case, the retriever LLM can retrieve the necessary information on its own. For example, if the user uses a simplified generational LLM acting as a chat-bot, while the retriever actually parses the necessary information to the chatbot. This can be useful if the retriever model is too large to be on the device where the chatbot is located.

2.3 Facebook AI Similarity Search and Vector search

RAG in particular is useful when combined with vector databases [12]. Searching through indexed documents in vector form can use powerful algorithms to quickly sort through large databases.

Facebook AI Similarity Search (FAISS) is a library developed to perform efficient similarity searches and clustering of dense vectors, which are typical outputs of deep learning models in various NLP and machine learning applications [10][2]. Originally developed by Facebook AI Research, FAISS addresses performance bottlenecks associated with high-dimensional vector search [10], making it particularly useful in real-time data retrieval scenarios where rapid response times are needed. FAISS achieves high efficiency through two main mechanisms: indexing and vector quantization. The indexing methods implemented in FAISS significantly reduce computational complexity by limiting the search space. Among these, the Inverted File Index (IVF [5]) and the Hierarchical Navigable Small World (HNSW) graphs [30] stand out. The IVF method clusters database vectors into centroids, thus significantly reducing search time during queries. HNSW, in contrast, relies on a graph-based structure that connects data points through hierarchical layers, allowing rapid traversals to approximate nearest neighbors efficiently and accurately even with billions of vectors [20]. Quantization in FAISS primarily involves Product Quantization (PQ) and Residual Quantization (RQ). PQ divides high-dimensional vectors into multiple sub-vectors, each quantized separately, reducing memory usage while preserving accuracy. Formally, PQ approximates

a vector x as:

$$q(x) = q_1(x_1) + q_2(x_2) + \dots + q_m(x_m) \quad (2.1)$$

where each $q_i(x_j)$ represents a quantized sub-vector of the original vector. Residual Quantization, an advanced variant, sequentially encodes vector residuals, progressively improving accuracy at each quantization step [10].

$$c_m = \operatorname{argmin}_{j=1,\dots,K} \left\| \sum_{i=1}^{m-1} T_i[c_i] + T_m[j] - x \right\|^2 \quad (2.2)$$

where c_m is the quantization index chosen at step m . $T_m[j]$ represents the reconstruction from the j -th entry of the m -th codebook (set of vectors to be optimized). K is the number of entries per codebook.

The core functionality of FAISS involves Approximate Nearest Neighbor (ANN) searches defined mathematically by the Asymmetric Distance Computation (ADC) method as follows:

$$L_{ADC} = \mathbf{k}\text{-argmin}_{i=0:l} \|x - q(y_i)\|^2 \quad (2.3)$$

where $q(y_i)$ represents the quantized approximation of the database vector y_i , and x is the query vector [10][20]. This operation efficiently retrieves vectors closest to the query without requiring exhaustive computational resources, thus ideal for environments demanding high throughput and low latency.

FAISS also capitalizes on GPU acceleration to significantly boost performance. Johnson et al. demonstrate an implementation capable of achieving up to 8.5 times the performance of previous GPU-based state-of-the-art methods, enabling the rapid construction of k -nearest neighbor graphs over billion-scale databases in hours, thus highlighting its suitability for large-scale real-time applications [20].

3. Overview of Large Language Model Families

This chapter provides an overview of the model families used in this thesis, explores their characteristics, and provides information on the reasons behind the selection process.

3.1 GPT Family of Models

The Generative Pre-trained Transformer (GPT as they are widely known) models, developed by OpenAI, have been a big part of natural language processing evolution and enabling of advanced language understanding and generation capabilities. Based on transformer architecture, GPT models employ a decoder-only design with a self-attention mechanism that allows them to process and generate human-like text. GPT models have evolved from the initial GPT-1 through to the advanced GPT-4, each iteration introducing improvements in scale and performance. GPT-1 established the framework with 12 layers and demonstrated the efficiency of unsupervised pre-training for downstream NLP tasks [63]. GPT-2 expanded significantly on this with a deeper model and increased parameter count, showcasing the capabilities to generate coherent long-form text. GPT-3 also scaled up to 175 billion parameters, doing better in contextual understanding and zero-shot learning [63]. This scaling highlighted the model's ability to generalize across various NLP tasks without task-specific fine-tuning [66].

3.1.1 Architecture and Training

GPT models are built on the transformer architecture, originally introduced by Vaswani et al. [56] and introduced in Chapter 2. It uses a mechanism known as self-attention to weigh the importance of each word in a sequence relative to others. The self-attention mechanism allows the model to understand context by assessing relationships between all words in a given input, regardless of distance. This is achieved by computing scaled dot product attention, where input tokens are mapped through three learned matrices to produce query, key, and value vectors. Attention scores are calculated using these vectors, enabling the model to focus on different parts of the input when generating the output.

A typical transformer block, the core of GPT’s architecture, consists of multihead self-attention layers followed by position-wise feedforward networks. These blocks are stacked sequentially and each layer normalizes its output through residual connections and layer normalization, ensuring that deep networks remain stable during training. Unlike full transformers used in other models such as BERT, GPT uses only the decoder part of the transformer, making it efficient for autoregressive tasks such as text generation [63]. This decoder-only setup is designed to predict the next word in a sequence based solely on preceding words, which is important for tasks requiring continuation and language generation.

The training methodology of GPT models involves pretraining on vast, diverse corpora to predict the next word in a sequence, facilitating a robust understanding of language context. GPT-3 and GPT-4 introduced fine-tuning with reinforcement learning from human feedback (RLHF), which was used to better align the model’s outputs with user expectations and real-world applications [37], enhancing alignment with human expectations and improving performance on specific tasks. These advancements make GPT models particularly good at handling ambiguous queries and generating contextually appropriate responses. This allows researchers to make educated guesses regarding the data users want but are unable to describe their exact meaning precisely.

One of the significant advances in GPT-4 is the support for larger context windows, extending up to 32,768 tokens [63]. This capability allows for better handling of complex, multi-step data processing, which is particularly beneficial for analyzing complex and detailed data, such as raw sensor data streamed from smart areas. Additionally, GPT-4's integration of multi-modal inputs, though primarily experimental, supports the trend toward frameworks capable of versatile data interpretation. This multimodality can theoretically enhance scenarios in which text and other data types must be synthesized to generate comprehensive insights [37]. Although in the scope of this thesis, multimodality is only one of the considerations for the future and is not being implemented currently.

The choice of GPT models for developing a system for user interaction is mainly supported by their extremely strong ability to interpret and generate complex natural language. This adaptability ensures that users can query the model in various formats and receive contextually relevant human-like responses. In the case of a recommendation system for a smart campus area, where user queries and stored data could include textual descriptions, user behavior patterns, and multisource context, GPT models can offer enhanced data representations and facilitate interaction through more advanced prompt engineering than most other solutions [21]. These capabilities can effectively bridge the gap between raw data interpretation and actionable intelligence, which is one of the goals of this thesis.

3.1.2 Open Challenges

Despite these advances, there are inherent challenges associated with GPT models. One critical issue is the generation of hallucinated content, where the results may appear plausible but lack factual precision [37]. This requires either implementing robust verification processes when deploying these models in applications that require precise information or developing a pipeline with the help of agents that would retrieve the data for the model to return as part of the response. For the purposes of this the-

sis, the solution must be chosen based on sensor data analysis in smart environments. Furthermore, although the size of the context window in GPT-4 has been significantly expanded, it still presents limitations when processing data that exceed this threshold [63]. This limitation can affect the reliability of responses in tasks that involve extensive data sequences. Another consideration is the computational resources required to deploy and fine-tune such large-scale models, which can be substantial. High resource demands could impact the practicality of their deployment in constrained environments or when processing sensor input [58]. Moreover, while hallucinations in natural language responses might not be as critical in general use cases, they pose a notable challenge when integrating GPT models into data pipelines where input accuracy and consistency must be maintained at all stages.

3.2 BERT Family of Models

3.2.1 Architecture

The architecture of the Bidirectional Encoder Representations from Transformers (BERT) model also builds on the transformer framework, which uses self-attention mechanisms to process text by jointly attending to the left and right context [9], but also introduces deep bidirectional learning, setting it apart from earlier unidirectional language models. More specifically, BERT is designed with a multilayer transformer encoder that learns representations by processing input sequences bidirectionally. This architecture allows BERT to better understand the relationships between words and context, unlike unidirectional models, like GPT models, which process sequences from left to right only, thus only relying on past context.

The model consists of encoder blocks with layers of self-attention and feedforward networks, interspersed with layer normalization and residual connections [9] [29]. Each word token passes through this stack of encoder layers, enhancing its contextual representation by considering both preceding and succeeding tokens. BERT also uses

Masked Language Modeling as a pre-training objective, which randomly masks a portion of input tokens and tasks the model with predicting them based on the unmasked context [9]. Additionally, BERT uses a Next Sentence Prediction task to improve sentence-pair understanding, a component used for tasks like question answering and natural language inference [9] [41].

3.2.2 Variations and further improvements

The model’s pre-training requires substantial computational resources, which is usually addressed with large-scale hardware. Additionally, BERT’s use of the [MASK] token during pre-training introduces a discrepancy, as masked tokens do not appear in real-world downstream tasks, which can affect fine-tuning [9]. Further research demonstrated [29] that BERT did not fully utilize its potential, forcing researchers to adopt more extensive data and refine training strategies.

Improvements in pre-training have led to variations such as RoBERTa, which were used to show that BERT’s original training was underoptimized. RoBERTa removed the NSP task and used longer training with larger batches and more data, resulting in state-of-the-art performances as shown by Yinhan Liu et al.[29]. Additionally, ALBERT introduced parameter reduction techniques, such as factorized embedding parameterization and cross-layer parameter sharing, making BERT-like models more memory-efficient without significant performance trade-offs [27].

Other models were also built on top of BERT. For example, BERT’s architecture has influenced subsequent models like XLNet, which addresses the limitations of BERT’s independence assumption by introducing permutation-based language modeling, enhancing the ability to capture bidirectional context while avoiding discrepancies between pre-training and fine-tuning phases [62]. This extension also utilizes Transformer-XL’s segment-level recurrence and relative position encoding, boosting performance for longer sequences [62].

3.3 Llama Family of Models

The Llama model family is developed by Meta AI and includes open-access foundation models ranging from 1 billion to 405 billion parameters. According to Meta, these models are designed for performance with NLP tasks with a priority focused on efficient scaling and accessibility [52]. Llama models have shown competitive results in NLP benchmarks while using fewer parameters than other leading models like GPT-3. For example, the Llama 2 13B model has been found to outperform GPT-3 on many tasks, despite having around 90% fewer parameters [54].

3.3.1 Architecture

The Llama models use an optimized transformer architecture, featuring RMSNorm instead of traditional layer normalization, which improves training stability. They also employ the SwiGLU activation function, chosen over ReLU for its efficiency and effectiveness in large-scale models, inspired by Google’s PaLM model [54]. Thus, the architecture consists of stacked layers that integrate multi-head attention mechanisms and feedforward networks, each supported by the efficient RMSNorm and SwiGLU activations. For positional encoding, Llama models incorporate Rotary Positional Embeddings (RoPE), which enable better handling of longer input contexts, particularly in tasks that require sequential dependencies [52] [47]. RoPE applies rotation to token embeddings, allowing for more dynamic position encoding that improves model adaptability in processing sequential information across layers. During pre-training, Llama models were trained on extensive data sources to capture a broad understanding of language patterns and structures. Llama 2 models, for example, were trained on a dataset of approximately 1.4 trillion tokens, sourced from publicly available resources such as Common Crawl and C4 [54].

3.3.2 Variations and further improvements

Since the Llama models are available for open-source development, there are many models that could be used in the scope of this thesis. The first choice to look at would be Llama 2, the latest widely available version before Llama 3. It introduced expanded context lengths and Grouped-Query Attention, an efficiency improvement that optimizes memory use during multi-query attention processes, particularly effective in larger parameter models like the 70B variant [54]. With Llama 3, Meta introduced models of the 8B, 70B, and 405B parameters. The 405B variant uses a dense transformer model, capable of processing context windows up to 128K tokens, which is good for long-context applications in multilingual and cross-disciplinary tasks. Llama 3 models have multilinguality, tool usage, and complex reasoning, positioning them close to the top proprietary models, such as GPT-4 in terms of versatility and depth in tasks. The Llama 3 architecture includes grouped query attention with 8 key-value heads and adopts RoPE with an extended base frequency of 500,000 to enhance long-sequence handling up to 128K tokens [53].

Another type of model to be considered is smaller, borderline SLM, Llama models. These are available freely and can be easily integrated with different LLM management tools such as LM Studio. This thesis aims to explore smaller models as well, and Meta, as well as the HuggingFace community [11], provide smaller versions of Llama 2 and 3 with 4, 7 or 10B parameters. For example, a specialized variant of Llama 7B intended for code-related tasks, trained with additional data and architectural modifications to support extended context windows of up to 100,000 tokens, is a good candidate for testing. This variant is divided into three submodels: the general-purpose Code Llama, a Python-optimized version, and Code Llama - Instruct, each fine-tuned to support different coding languages and completion tasks. Code Llama uses extended rotary embeddings to improve token level understanding in code generation, allowing it to handle complex programming languages and perform well in benchmarks for code-related tasks [43]. It could be potentially a powerful solution for both fine-tuning and

RAG, providing queries in Python or SQL that need to be executed by CSV or SQL agents.

Another consideration included Llama-Adapter, an efficient fine-tuning solution with zero initialized attention within Llama’s transformer layers, which significantly reduces the number of learnable parameters required for downstream applications. Llama-Adapter’s parameter reduction methods bring the total number of learnable parameters to approximately 1.2 million, which is useful for multimodal tasks, such as vision language processing [67].

One of the primary concerns is the computational cost associated with training, which remains substantial, especially in large-scale versions like the 70B models. Training the Llama 2 series, for example, required around 3.3 million GPU hours, resulting in an estimated 539 tons of CO₂ emissions, although Meta AI offsets these emissions as part of its sustainability programs [54]. Training Llama 3’s 405B model required an estimated 16,000 H100 GPUs and 3.8×10^{25} FLOPs [53], which would likely be several times longer than that of Llama 2’s biggest models. The setup used in this thesis only includes 16BG of GPU memory, so for SLM testing, a less powerful model will be needed. In addition, the open-access nature of Llama introduces potential risks around misuse, as the model’s flexibility allows it to be adapted for a wide range of purposes. To address these risks, Meta AI took a red-team approach and incorporated safety-focused data annotations to identify and mitigate potential biases and misuse cases in the Llama 2 release [54] and, in the case of Llama 3, created Llama Guard, a safety feature to manage input-output quality which has undergone improvements to reduce potential biases and harmful content generation. Post-training safety measures are strengthened through supervised fine-tuning, rejection sampling, and Direct Preference Optimization, which optimizes the model alignment with human preferences without resorting to complex reinforcement learning algorithms, thus maintaining stability in fine-tuning [53].

3.4 Gemini Family of Models

Gemini models, developed by Google DeepMind made a significant advancement in multimodal large language models, showcasing capabilities in text, vision, audio, and video processing. These models have demonstrated competitive performance in domains such as knowledge-based question answering, recommendation systems such as medical assistants, and reasoning in complex contexts in general.

3.4.1 Architecture

Gemini models utilize an advanced transformer-based architecture specifically optimized for multimodal integration. This architecture incorporates multiple modality-specific embedding layers, attention mechanisms, and specialized routing functions within a sparse Mixture-of-Experts framework [49]. The MoE architecture significantly enhances model efficiency by conditionally activating a subset of parameters based on input modality and context, allowing Gemini to handle extremely long-context inputs efficiently. Gemini 1.5 Pro, for example, can process inputs up to 10 million tokens across multimodal data, substantially exceeding other contemporary models like GPT-4 Turbo and Claude 3 [50]. Gemini 1.5 models have demonstrated near-perfect information retrieval and reasoning in extensive multimodal inputs, including text, audio, and video, positioning them uniquely in tasks involving complex multimodal interactions [50].

3.4.2 Performance and limitations

Studies evaluating Gemini against other leading models like GPT-4 and GPT-3.5 Turbo in language tasks have found that Gemini Pro offers competitive performance [3]. Gemini excels particularly in language translation tasks, significantly outperforming GPT-3.5 Turbo in multilingual translations [3]. Gemini generally achieves slightly lower accuracy compared to GPT-3.5 Turbo in knowledge-based tasks such as MMLU. For

example, Gemini’s proficiency in medical tasks, as evaluated through benchmarks like MultiMedQA, Med-HALT, and medical visual question answering (VQA), indicates clear limitations. Although Gemini shows robust capability in synthesizing medical literature, it trails specialized medical models such as MedPaLM 2 and GPT-4 in diagnostic accuracy and exhibits notable vulnerability to hallucinations and overconfidence. For example, Gemini achieved a VQA accuracy of 61.45%, significantly below the accuracy of GPT-4 88% [38]. Detailed analysis of Gemini’s performance in common-sense reasoning tasks reveals noteworthy competencies, particularly in general and contextual reasoning domains. The performance of Gemini Pro was marginally better than that of GPT-3.5 Turbo on language-only tasks, but significantly behind that of GPT-4 Turbo, especially on tasks of temporal and social reasoning [61]. Furthermore, multimodal evaluation showed that Gemini Pro Vision struggled particularly in interpreting emotional and psychological signals from images, suggesting challenges in more nuanced emotional contexts[61].

Gemini can still achieve good performance in knowledge-based retrieval and coding tasks. Gemini 1.5 Flash, specifically designed for efficient processing with minimal quality regression, demonstrates impressive capabilities in long-context retrieval tasks and general multimodal tasks, maintaining high performance levels while significantly reducing compute requirements. This efficiency enables the Gemini 1.5 Flash variant to deliver a performance comparable to Gemini 1.0 Ultra on many benchmarks, while substantially improving the feasibility of deployment in computationally constrained environments [50]). Additionally, Gemini 1.5 Pro demonstrated strong coding capabilities in API function call completion tasks, achieving high accuracy (cosine similarity of 0.813), indicating its suitability for API-intensive coding scenarios. However, the model showed notable weaknesses in algorithmic block and control-flow completion tasks, with accuracy dropping significantly (cosine similarity of approximately 0.603 and 0.632 respectively). Gemini 1.5 Flash, optimized for latency-sensitive contexts, showed moderate but consistent performance in all task types, effectively balancing

accuracy and efficiency [65].

3.5 Claude Family of Models

3.5.1 Architecture

Claude is a large language model developed by Anthropic, designed around a novel alignment paradigm called Constitutional AI. At its core, Claude shares the same foundational elements as other transformer-based large language models: layers of multi-head self-attention, feed-forward networks, and residual connections that enable it to generate contextually relevant sequences of text. However, unlike many LLMs that rely extensively on human-annotated "harmlessness" data, Claude's alignment strategy primarily employs a set of guiding principles-its "constitution". The model is initially pre-trained on broad text corpora to learn general linguistic patterns and then refined through a stage called Reinforcement Learning from AI Feedback (RLAIF). In this process, Claude generates multiple candidate outputs for potentially harmful or questionable prompts, compares them against its constitutional principles, and updates itself to favor the best-aligned responses. This two-part pipeline-pre-training followed by RLAIF-is detailed in the work by Bai et al. [6] on "Constitutional AI" (2022).

3.5.2 Performance and limitations

Code generation and debugging are key performance highlights for Claude, especially in multi-step reasoning tasks. Evaluations on recently proposed benchmarks such as LiveCodeBench indicate that Claude consistently competes with top-tier LLMs, although it may sometimes lag behind GPT-4 in specialized code scenarios [18]. Its relatively strong performance arises from Anthropic's deliberate inclusion of programming data during the training phase, along with emphasis on chain-of-thought rationales. This combination helps Claude tackle subtle coding tasks-for instance, producing a function stub, detecting errors in a snippet, or guiding the user with short explanatory notes.

Despite its robust alignment framework, Claude faces several notable limitations. Like other large transformers, it can hallucinate, producing well-formed but factually incorrect statements when confronted with niche or incomplete data. Additionally, although Anthropic has extended Claude’s context window, extremely lengthy or elaborate multi-document queries can overwhelm its attention mechanism, causing incomplete or tangential responses [71]. There is also a natural tension between helpfulness and refusal policies; occasionally, Claude can over-refuse borderline requests if its constitutional guidelines yield excessive caution. Finally, because Anthropic has not disclosed full technical details such as precise parameter counts or hyperparameters, aspects of Claude’s exact training design and performance remain somewhat opaque outside of reported test evaluations.

3.6 LLM selection

Each model family offers distinct strengths across several dimensions, such as performance, resource requirements, and feasibility of deployment. However, there are many different versions of these models, including thousands of open-source models.

The models are chosen based on their overall performance, specialized training and/or architecture, and overall popularity. Popularity is a different metric to measure, but for this thesis, a public hub for language models was used: HuggingFace (HF). Hugging Face is possibly the largest machine learning and data science platform that helps users build, deploy, and train machine learning models, including language models. Specifically for language models, HF has a leaderboard [11]. For this thesis, 2 models were chosen that had good ranking in most categories, were specifically designed for the task of code generation and comprehension, and were often used/downloaded by the users. In addition to those 2, 3 of the biggest names among LLMs were also chosen: GPT, Claude, Gemini.

GPT models have demonstrated unparalleled capabilities in complex tasks involving multi-step reasoning, linguistic understanding, and cross-domain generalization.

GPT-4, with an estimated 1 trillion parameters and extensive 32,768-token context length, achieves high accuracy on benchmarks like MMLU (87.4%) and Mathematical Reasoning (92%) [74]. Its proficiency in instruction following and versatility make it suitable for API-based deployments where high computational capacity can be used to handle diverse user queries, large contexts, and perform rigorous reasoning tasks [13]. For this thesis, GPT-4 was chosen as the latest available model.

Llama models offer an open-access alternative that is efficient and adaptable for both local and remote deployment scenarios. The Llama 2 series, available in sizes ranging from 7 billion to 70 billion parameters with optimizations RMSNorm, SwiGLU activations, and RoPE, improve training stability and handling of longer context lengths. Llama 2 7B with 7 billion parameters is particularly suited to applications with limited computational resources, supporting up to 4,096 tokens and demonstrating reasonably competitive performance on MMLU (64.6%) and multi-step reasoning tasks [48] [60]. The Llama 2 13B and 70B variants have enhanced capacity and are better suited for scenarios that demand higher performance. Llama 2 70B achieves notable results with 80.3% on MMLU and 52.8% on math benchmarks, which is a good result, even considering some closed-source models. These models have extended context handling, with support for up to 8,192 tokens, making them more versatile in complex applications such as multi-turn dialogue and long document processing [46] [42]. However, in the scope of this thesis, the LLM is mostly needed to communicate with the agent and, briefly, the user; it does not need to read long documents. The ability to have longer dialogues is still appreciated due to possible adjustments needed by the user. The latest Llama 3 series has even better scalability and has up to 405 billion parameters. Additionally, it is equipped with Grouped-Query Attention and enhanced RoPE embeddings, and achieves an MMLU accuracy of 87.1% and Math benchmark accuracy of 82.3%, placing it on par with proprietary models like GPT-4 [13] [60]. However, it would likely require several GPU stations to simply run this model. Despite their impressive capabilities, the larger Llama 3 models have high computational demands

and are designed for environments with ample hardware resources, given their extensive training FLOPs and memory usage [46]. Llama 3 70B supports up to 128,000 tokens, ideal for applications requiring long-context support, such as cross-document summarization or processing large databases. However, it is still quite resource intensive. Smaller Llama models, for instance Code Llama 7B, are more specialized and are used for support for code-related tasks and instruction following. They are able to work with Pandas dataframes and SQL queries and are optimized with additional data for programming languages. Code Llama 7B performs well in code-specific tasks with a context length of up to 100,000 tokens. The 7B Instruct variant, in particular, combines efficient parameter usage with instructional capabilities, making it a good choice for secure, local deployments where data privacy and lower power consumption are critical [48] [74]. It is chosen alongside 1 other model for the SLM representation in this thesis.

For BERT models, many of the tested models are slightly older and not as thoroughly tested in data-intensive environments. They include variations such as RoBERTa and ALBERT, which are reliable options for NLP tasks that require interpretability and shorter context handling. These models are great in areas like extractive question answering and sentiment analysis due to their bidirectional encoding and NSP-free training, some like RoBERTa achieving F1 scores over 90% on SQuAD [29]. However, their 512-token context limit and inability to handle multi-turn conversations constrain their applicability for dynamic, instruction-following tasks necessary for advanced conversation-based RAG systems. Still, it is possible to use them for inference in certain agentic structures when multi-turn conversation is not needed. For the purposes of this thesis, CodeBERT (125M params) was chosen because it is specifically designed to handle coding tasks. It is a smaller model that would be compared with the chosen Code Llama 7B model in the SLM field.

For Claude models, Claude 3.5 Sonnet is particularly impressive for its performance on a variety of benchmarks, surpassing its predecessor Claude 3 Opus and

showing results on par with GPT-4. Specifically, Claude 3.5 Sonnet achieves 90.4% accuracy on general reasoning tasks (MMLU), 92.0% on coding tasks (HumanEval), and demonstrates strong multimodal capabilities with 95.2% on document understanding (DocVQA) and 94.7% on interpreting scientific diagrams (AI2D) [4]. These attributes make Claude 3.5 Sonnet an excellent choice for tasks that require detailed reasoning, coding proficiency, and multimodal interpretation, especially in smart space environments where accurate structured information retrieval is important.

Finally, Gemini models are popular due to their multimodal processing and efficient handling of large-context tasks, as the other big model families [49]. Gemini 1.5 Flash, a more computationally efficient variant within the Gemini family, demonstrates great performance in long-context retrieval and multimodal understanding scenarios. It provides near-perfect recall for contexts that extend into millions of tokens, making it suitable for applications involving extensive data analysis and complex query handling. Its optimized speed and strong benchmark performance, including improved reasoning in tasks such as MathVista and visual question answering, make Gemini 1.5 Flash a good candidate for deployments in smart campus environments [50]. Thus, Gemini 1.5 Flash is used to test the recommendation system.

4. Recommendation Systems

Recommendation systems have become an indispensable component of modern digital services, focusing on personalization in domains such as e-Commerce, streaming platforms, and social networks. Traditional recommendation techniques, such as content-based filtering and collaborative filtering, have long been used to generate personalized suggestions. However, these methods frequently face challenges related to data sparsity, scalability, and adaptability to evolving user preferences. Rapid advancement of deep learning, particularly the rise of LLMs, has introduced new capabilities that improve recommendation systems by offering advanced contextual understanding, interactivity, and personalization. LLMs, such as GPTs and LLaMAs, can analyze and interpret large amounts of unstructured text data, such as user reviews, product descriptions, and conversational inputs, which helps to understand the intent of the user when designing a recommendation system [72].

4.1 Traditional approaches

Traditional recommendation systems are based primarily on content-based filtering (CBF) and collaborative filtering (CF). Content-based filtering suggests items similar to those previously preferred by the user by analyzing item attributes such as descriptions and features. Techniques like Term Frequency-Inverse Document Frequency and cosine similarity have been widely used in these systems. However, CBF faces limitations, particularly with new users and items, as it is heavily dependent on descriptive metadata, which may not always be available [26]. Collaborative filtering, in contrast,

predicts user's preferences based on similarities with other users or items derived from past interactions. Techniques such as matrix factorization and singular value decomposition are effective in discovering latent patterns within user-item matrices. Despite its success, collaborative filtering is prone to data-sparsity issues, where insufficient interactions hinder accurate predictions, and the cold start problem, where new users or items lack historical data for reliable recommendations [26]. Hybrid recommendation systems have been proposed to mitigate the weaknesses of both approaches by combining CBF and CF techniques [72]. These hybrid systems offer improved performance by using user-item interaction data along with content-based features, although they often require significant computational resources and complex optimization strategies.

4.2 LLM integration

The integration of LLMs into recommendation systems has opened up several new routes and possibilities. LLMs can understand natural language queries, analyze unstructured textual data, and dynamically generate personalized recommendations based on user preferences. An important advantage of LLMs is their ability to generate rich embeddings that encapsulate complex relationships between users and items in a rather simple form of a vector. Studies have shown that LLM-generated embeddings can improve recommendation accuracy by up to 15% compared to traditional methods [73]. It is possible to divide the integration process into 3 parts: pre-training, fine-tuning, and prompting, each offering different advantages to utilize the powerful language understanding and generation capabilities of LLMs [72]. Pre-training focuses on learning universal user-item representations from large-scale textual data, improving model generalization across diverse recommendation scenarios. Fine-tuning allows LLMs to adapt to specific recommendation tasks, such as rating prediction or personalized content suggestions, by refining model parameters with domain-specific data. Prompting, on the other hand, facilitates zero-shot and few-shot learning, enabling LLMs to provide recommendations without extensive retraining by making use

of task-specific queries. Furthermore, instruction tuning further refines LLM performance by customizing model responses to align with specific recommendation tasks. It involves training the model on curated datasets with explicit instructions to guide its behavior, resulting in improved controllability and reduced generation biases.

Using RAG improves LLM-based recommendation systems by incorporating external knowledge sources as well. RAG allows LLMs to access up-to-date and domain-specific information, mitigating issues related to model hallucination and knowledge limitations [59]. It also allows for the usage of new retrieval methods, such as graph-based retrieval, such as the K-RagRec framework, which has further improved the efficiency of RAG by using structured knowledge graphs to provide more accurate recommendations. This method has been shown to significantly improve the performance of the recommendation, reducing irrelevant suggestions by 18% compared to traditional RAG approaches [59].

Conversational LLM-powered recommendation systems are another significant development in the field. These systems facilitate interactive dialogues with users to refine their preferences iteratively. Experiments have shown that conversational recommender systems that utilize LLM outperform traditional systems by achieving a 23.3 % improvement in Recall@20 benchmark on the ReDial dataset [26]. However, challenges such as response latency and maintaining coherence across multi-turn dialogues remain active areas of research [8].

Despite the promising capabilities of LLM-powered recommendation systems, there are several challenges still exist. One of the primary concerns is computational efficiency. LLMs require substantial computational resources, making real-time inference costly and difficult to scale for high-traffic applications. Studies have shown that LLM-based recommendation systems can experience inference times that are up to five times slower compared to traditional models, significantly impacting the user experience [73]. Bias and fairness present additional concerns. LLMs, trained on large datasets, often inherit biases present in the training data, which can lead to fairness

issues in recommendations. For example, it has been observed that recommendations disproportionately favor certain demographic groups, leading to suboptimal user experiences and regulatory concerns [25]. Efforts to mitigate these biases include fairness-aware classification algorithms and improved data set curation techniques, although these approaches require careful tuning and validation before they are ready to be put into production [25].

Generally, the application of LLMs in recommendation systems has multiple directions, providing significant improvements in personalization and adaptability. According to Yu et al. [64], “These models work best by extracting insights from unstructured data on user preferences that were not easy to address by a traditional recommendation system due to cold start problems and data sparsity. This versatility has also been demonstrated in a number of domains, including e-commerce, social networks, and streaming platforms, increasing user engagement and satisfaction with a service.” The same applies to conversational recommendation systems. These systems allow users to refine their preferences through iterative interactions, improving the relevance of recommendations over time. The integration of LLMs into conversational recommenders has resulted in better user satisfaction, with response times optimized using RAG techniques and dynamic user intent prediction models [8].

In this thesis, the focus is mostly on prompting and prompt engineering. Chosen LLMs are quite powerful and do not require additional fine-tuning for the data that are being used. In addition, RAG approaches are sufficient with the search for k nearest neighbors for the embeddings of user queries.

5. Smart Spaces Data

Smart spaces are technologically enhanced indoor environments that seamlessly incorporate sensors, communication networks, and computing systems to provide a wide array of services tailored to user needs [55]. By embedding artificial intelligence and large language models in these environments, smart spaces evolve into dynamic and interactive ecosystems capable of understanding, predicting, and responding to human behavior [45]. This integration can optimize resource efficiency and automation, but can also significantly improve user comfort, personalization, and overall satisfaction [44].

The data used for this thesis come from two previous studies on sensor data in the premises of the TellUs smart space at the University of Oulu [35, 32]. The first work focuses on the feasibility of repurposing existing occupancy monitoring and environmental sensors to monitor social distancing and estimate disease transmission risks [32]. For this thesis, data from this study are used. The study uses data from CO₂ and PIR (passive infrared) motion detectors collected over 410 days from 2017 to 2018 in a smart space at the University of Oulu. The smart space is equipped with 352 LoRaWAN sensor nodes (Elsys ERS sensors), which measure temperature, humidity, CO₂, motion and light. Each sensor node is geo-positioned and powered by two 3.6 V AA lithium batteries, placed approximately two meters apart. Sensor nodes send data packets every 15 minutes to a remote server using LoRaWAN radio access network technology. The data is collected by a LoRA gateway connected to an external antenna, then transferred using the MQTT protocol to the ThingWorx commercial cloud platform and stored on local servers using Python scripts. The dataset contains

9,917,848 lines of readings, including CO₂, humidity, light, temperature, and motion data. Each entry has a sensor ID and timestamp. The data covers the period from June 26, 2017, to November 20, 2018. The output CSV files are used for data preprocessing.

The second paper deals with the creation of digital twins in smart spaces [35], highlighting data management, network management, and the lifecycle of sensor deployments. The study uses data from a sensor network deployed in the TellUs smart space at the University of Oulu, collected from July 2020 to May 2021. Currently, there are only 68 LoRaWAN sensor nodes that measure temperature, humidity, CO₂, motion, and light.

The original dataset comes in the form of an XLSX Microsoft Excel file and a CSV file. The Excel file contains information on the rooms, their exact coordinates, and name conventions, as well as short explanations. The CSV file contains the original data and has 6 columns: id, timestamp, CO₂, humidity, light, temperature, and PIR. Each entry corresponds to a sensor measurement taken at a certain time. All rows for each sensor are combined into one averaged out row to simulate the real-time data flow. Aggregation is done by sensor ID. The rest of the data are also presented as summary statistics for each of the original categories. There are columns for mean, std, min and max values, as well as rank, that were chosen based on the generally accepted values for those categories. This last category was introduced in order to give LLMs a head start when it comes to understanding what is good for humans when it comes to their environment, knowledge an LLM might not necessarily possess. To determine good values, the work on Occupancy Detection technologies by Priyadarshini and Mehra [39] was used. The first 5 entries in the final data set can be seen in Figure 5.1.

This dataset is used for all future trainings and embeddings unless otherwise stated. From this dataset was also built a QA dataset, that contains sets of instructions for further fine-tuning of the LLMs. For each of the entries, a list of questions was generated and the answers were supplied with the desired data from the original data set (see Figure 5.2). Another metadata column is also added. It makes it easier for

	Location	temperature_mean	temperature_std	temperature_min	temperature_max	humidity_mean	humidity_std	humidity_min	humidity_max
0	AM01-0101	21.334663	1.528104	17.7	27.0	31.650062	12.610286	5	72
1	AM01-0102	21.590877	1.613845	17.6	28.4	31.033046	12.613084	5	73
2	AM01-0201	21.320821	1.528981	16.8	27.5	32.277841	12.926867	5	75
3	AM01-0202	21.509642	1.505879	17.0	27.2	31.380057	13.136004	4	75
4	AM01-0301	21.184210	1.559464	17.1	26.9	31.927196	12.924792	5	76

	co2_mean	...	pir_max	light_mean	light_std	light_min	light_max	CO2_Rank	Humidity_Rank	Light_Rank	Temperature_Rank	PIR_Rank
526.859299	...	29	36.750146	59.588009	0	2325	Good	Good	Dark	Comfortable	Occupied	
524.464478	...	29	48.294141	85.043368	0	2316	Good	Good	Dark	Comfortable	Free	
549.126885	...	29	28.503551	49.761334	0	2336	Good	Good	Dark	Comfortable	Occupied	
510.603747	...	29	57.301445	97.285650	1	2292	Good	Good	Dark	Comfortable	Occupied	
353.001019	...	29	32.391336	56.098000	1	2326	Good	Good	Dark	Comfortable	Occupied	

Figure 5.1: The new dataset sample.

the LLM to retrieve the context relevant to that specific query. The dataset size is 3311 rows. To approximate the pattern of arrival of live sensor traffic, the data set is slightly altered every hour. The interval represents the shortest time that remains compatible with the current fine-tuning pipeline. Gaussian noise ($\mu = 0$, $\sigma = 5\%$ of the feature range) is injected into every numeric reading, and the timestamps are adjusted accordingly. This procedure generates a pseudo-stream while preserving the original correlation structure. A faster replay cadence was avoided because of the chosen serving models and their restrictions imposed by OpenAI’s policies and local setup’s GPU power availability. One final thing to keep track of is the data backups for all datasets. By the nature of natural language query generation, there is always a small possibility that the LLM interprets the query as a command to delete or modify some of the data from the database. To prevent that, secure copies of all datasets need to be kept and updated regularly to prevent possible data loss in a real deployment.

	questions	answers
0	What is the average temperature in room AM01-0...	The average temperature in room AM01-0101 is 2...
1	Is the temperature in room AM01-0101 comfortable?	The temperature in room AM01-0101 is comfortable.
2	What is the CO2 level in room AM01-0101?	The CO2 level in room AM01-0101 is 526.86 ppm.
3	Is the CO2 level in room AM01-0101 hazardous?	The CO2 level in room AM01-0101 is good.
4	What is the humidity level in room AM01-0101?	The humidity level in room AM01-0101 is 31.65%.

Figure 5.2: The new QA dataset sample.

6. Fine-tuning or RAG?

First, a system is developed as an initial step of this thesis, capable of analyzing sensor data using LLMs, with two primary approaches tested: connecting to an LLM through an API and downloading and fine-tuning a local model. Each approach employs three distinct methods: Fine-Tuning, Retrieval-Augmented Generation, or a combination of both. The approach schema is illustrated in Figure 6.1. A local LLM CodeLlama 7B Instruct is selected for the local approach due to its size and capabilities, and GPT4o-mini is chosen for the remote approach, where model size is less important, but performance is the most important factor. This section outlines the methodologies tested, discussing their utility, and the processes involved. The recommender system developed for this thesis is built on the results of this initial stage.

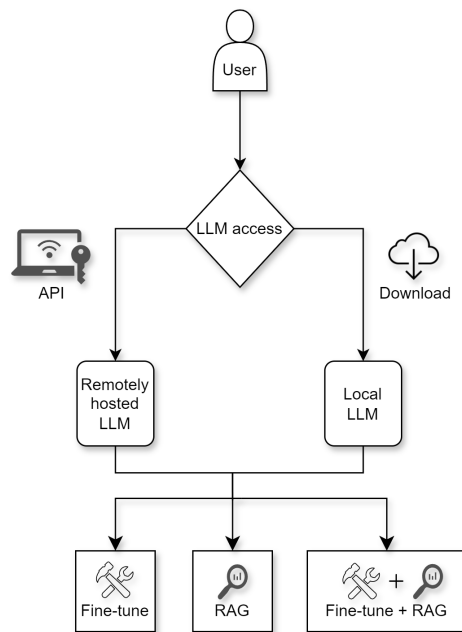


Figure 6.1: The two approaches of handling the LLMs.

6.1 Remote setup

6.1.1 Fine-tuning

Initially, the GPT-4o mini LLM is fine-tuned through an API to the specific domain using sensor data converted into strings of textual examples. Sensor data is collected and preprocessed, including PIR values, room temperature, humidity, light, and CO₂ levels. The API is then utilized to upload the data and initiate fine-tuning. OpenAI provides the tools necessary to fine-tune models through their fine-tuning service using the OpenAI Python library. The process is illustrated in Figure 6.2. Fine-tuning by itself will not be enough due to the nature of the task, requiring specific information to be retrieved over a large dataset. This will be showcased in the comparison of the results where fine-tuned models are queried regarding the data they were fine-tuned on. However, it is a viable approach for smaller datasets because there are fewer specific data to memorize for models.

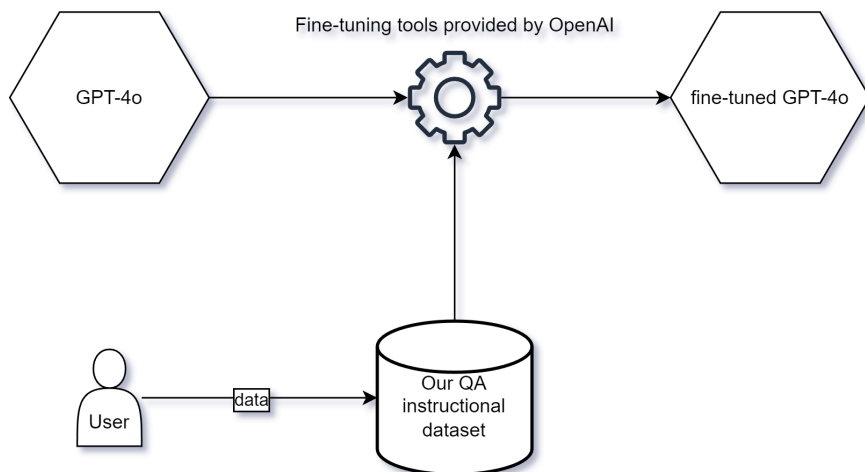


Figure 6.2: Fine-tuning with remote connection.

The next stage involves pre-processing the QA dataset into JSON format suitable for the OpenAI library. Each row in the CSV is transformed into a JSON object containing a prompt (instruction) and a completion (output), which is then written into a JSONL file. Afterward, the dataset is split into training and validation sets

using Python’s sklearn library to ensure proper model validation and avoid overfitting. The datasets are uploaded to OpenAI’s servers via their API, and each file receives a file ID. These IDs initiate the fine-tuning job, specifying parameters such as epochs and batch size. Once completed, the fine-tuned model generates answers to queries.

6.1.2 RAG

To test the RAG setup, the Langchain library and its experimental features are used. Langchain is a Python library designed to streamline the development of applications using LLMs. It simplifies prompt management, chaining LLMs, and integrating models into complex systems [51]. The process is illustrated in Figure 6.3.

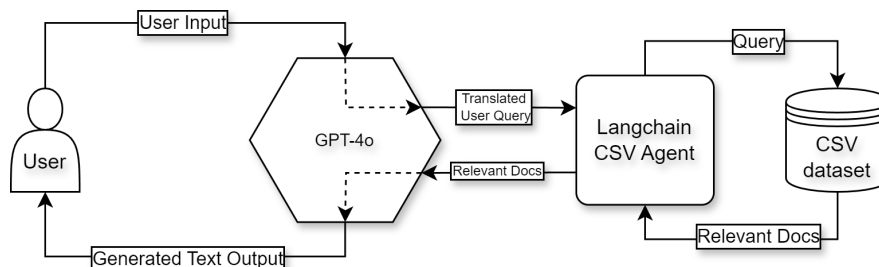


Figure 6.3: RAG with remote connection.

An agent interacts with the CSV dataset through OpenAI’s model. The OpenAI instance is configured with a temperature of 0 for deterministic output and requires an API key. The agent uses the ZERO SHOT REACT DESCRIPTION type, enabling query handling without prior examples by interpreting task descriptions. The agent runs queries against the CSV dataset and generates contextually relevant responses.

6.1.3 Fine-tuning and RAG

Combining RAG with fine-tuning uses structured data and enhanced model capabilities. RAG generates context-specific responses based on the retrieved data. Fine-tuning improves the accuracy of the base model by aligning it with the nuances of the data set. This combination could give increased accuracy, reduced hallucinations, and improved scalability over the two methods separately. However, it increases compu-

tational and architectural complexity, requiring better infrastructure. Data quality is important because it impacts both methods. Regular updates may be needed for relevance as domain knowledge evolves. The implementation sequentially combines RAG and fine-tuning, initially fine-tuning the model, and subsequently performing RAG.

6.2 Locally based models

For local deployment, the CodeLlama 7b Instructional model by Meta is selected from the HuggingFace hub due to its effectiveness with code and general performance [11]. Its compact size (7 billion parameters) enables execution on consumer-grade hardware, facilitating local deployment without external dependencies. The "Instructional" variant is good for structured query-response systems.

The machine running the model has 16 GB video memory and 64 GB RAM, sufficient for smaller model deployment. The model primarily utilizes GPU resources to improve speed through parallel computing capabilities [22].

Model monitoring is performed using LM Studio, providing local server hosting and detailed logs of output generation and interpretation.

Because only one hardware configuration was available for testing, no attempt was made to model the variance introduced by alternative CPU/GPU classes or longer network paths. The latency values reported in this thesis should therefore be interpreted as lower-bound estimates; future work would require controlled A/B benchmarking across a matrix of hardware tiers and network conditions to enable precise generalization.

6.2.1 Fine-tuning

The same preprocessed and summarized data from previous methods is used. The local environment serves the LLM through an API endpoint (localhost:1234/v1). The data set is submitted along with training parameters such as learning rate, epochs, and batch size. The primary difference from remote setup involves manually specifying the tokenizer, typically sourced from the Transformers Python library. The process is

illustrated in Figure 6.4.

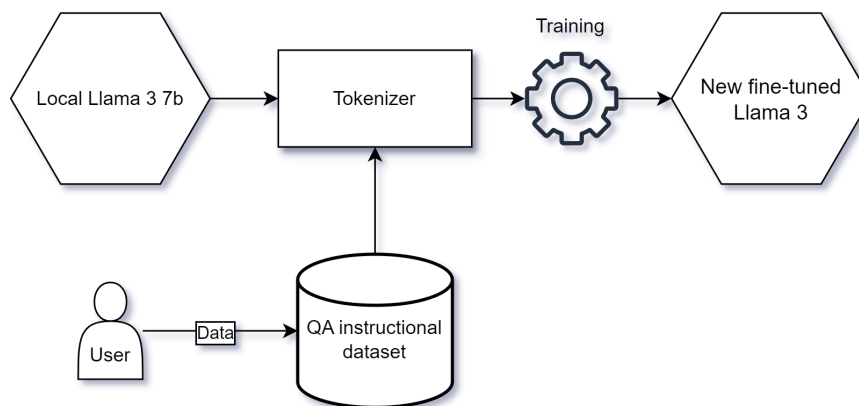


Figure 6.4: Fine-tuning with local connection.

6.2.2 RAG

The RAG system integrates the local Llama 3 model with a CSV-based sensor data analysis tool and is embedded within a Streamlit user interface. The file downloader functionality facilitates rapid additions and testing of the dataset directly through the user interface. CSV sensor data are converted to embeddings using HuggingFaceEmbeddings and stored in a FAISS database for efficient vector similarity searches. Queries from the Streamlit UI are similarly embedded and used to retrieve relevant sensor data. Retrieved contexts combined with the original query and a predefined prompt template guide the model's response generation. The process is illustrated in Figure 6.5.

Generated responses undergo post-processing to extract relevant information, removing extraneous code artifacts and irrelevant text, prioritizing location-specific and numeric data. A special cleaner function first removed common code-related artifacts such as code blocks, return statements, and print statements. It then splits the text into sentences, cleans each sentence by removing extraneous whitespace and quotes, and filters out sentences that are too short or lack proper punctuation. The function prioritizes sentences that contain both location information (e.g., "Location TY-2007") and numerical data, as these are likely to be the most relevant answers to sensor data questions. If no such sentence is found, it defaults to the first substantial sentence.

The resulting concise response is displayed through the user interface. This method forms the foundation for the thesis’s FAISS retrieval solution, requiring minimal changes in prompts to align with recommendation system goals.

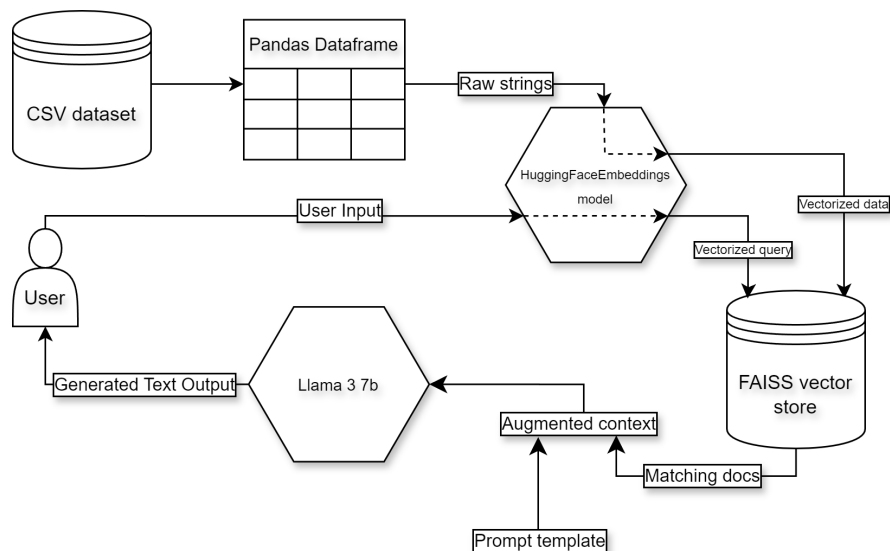


Figure 6.5: RAG with local connection.

6.2.3 Fine-tuning and RAG

Local environment fine-tuning combined with RAG mirrors remote implementation. Fine-tuning helps with accuracy, while RAG provides dynamically retrieved data-driven responses.

6.3 Testing results

This section presents performance outcomes from testing the API-connected remote LLMs and locally deployed LLMs. Both setups utilize Fine-Tuning, RAG, and their combination. Performance is evaluated based on accuracy, comprehension, and performance across data samples.

6.3.1 Performance Metrics Overview

This section presents the results of the performance testing of the remote LLM connected to the API and the locally deployed LLM. Both set-ups utilize Fine-Tuning, RAG, and their combination.

Primary evaluation metrics include:

- **Accuracy:** Proportion of correct responses to user queries. Meaning that the returned location was suitable for the user
- **Comprehension:** Clarity and relevance of responses as natural language. Meaning that the result could be understood by a human reader and had no code or other artifacts
- **Specific Dataset Sample Accuracy:** Accuracy of fine-tuned models updated for specific datasets.

Metrics range from 0 (poor) to 1 (excellent). Results for API/Remote LLM and Local LLM deployments are detailed below.

The primary aim at this stage is to identify the most promising approach among several candidates to further proceed with the implementation of the recommendation system. Accuracy and comprehension are not as precise as F1 scores, for instance, but allow for a quick and practical comparison, ensuring that resources are allocated efficiently for more in-depth analysis in subsequent phases. Additionally, both accuracy and comprehension play a role in several benchmarks like BertSCORE [68] and are generally used in evaluating LLM performance [7].

6.3.2 API / Remote LLM Results

The results for the Remote LLM deployment approach are presented in Table 6.1 and Figure 6.6.

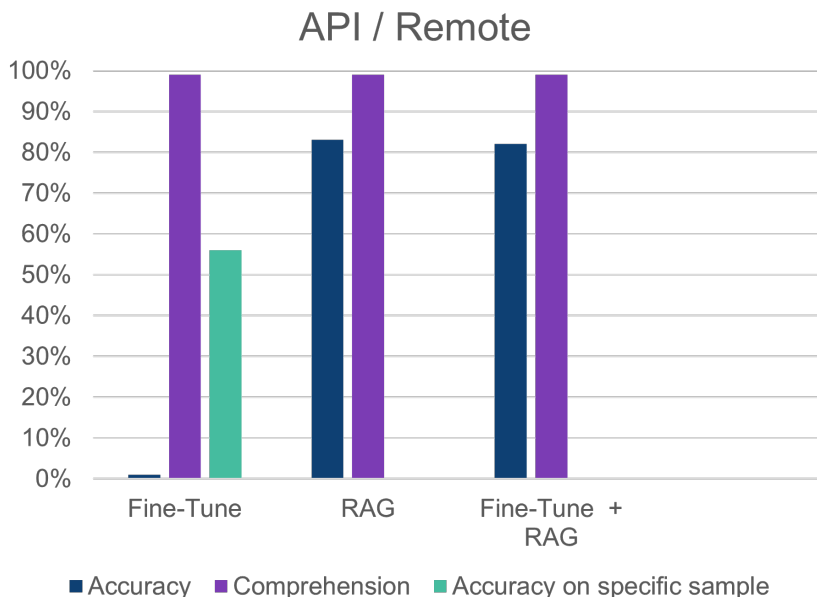


Figure 6.6: RAG with remote connection.

Method	Accuracy	Comprehension	Specific Sample Accuracy
Fine-Tuning	0.00	1.00	0.56
RAG	0.83	1.00	NA
Fine-Tuning + RAG	0.82	1.00	NA

Table 6.1: Performance Results for API/Remote LLM Approach.

Key Observations:

- RAG by itself showed almost exactly the same performance in accuracy as the combination of Fine-Tuning and RAG.
- Fine-Tuning on its own is not at all effective in increasing accuracy but maintains high comprehension due to a well trained model. Sometimes, the model cannot be fine-tuned in time for the change in the dataset, which makes accurate recommendations impossible. It also showcases, that fine-tuning by itself is better suited for a different kind of task, not requiring speed.
- Both fine-tuning + RAG and RAG alone provided good results in accuracy and comprehension.

- Fine-tuning for every dataset is, as expected, not practical and does not deliver good results.

6.3.3 Local LLM Results

The results for the Local LLM deployment approach are presented in Table 6.2 and Figure 6.7.

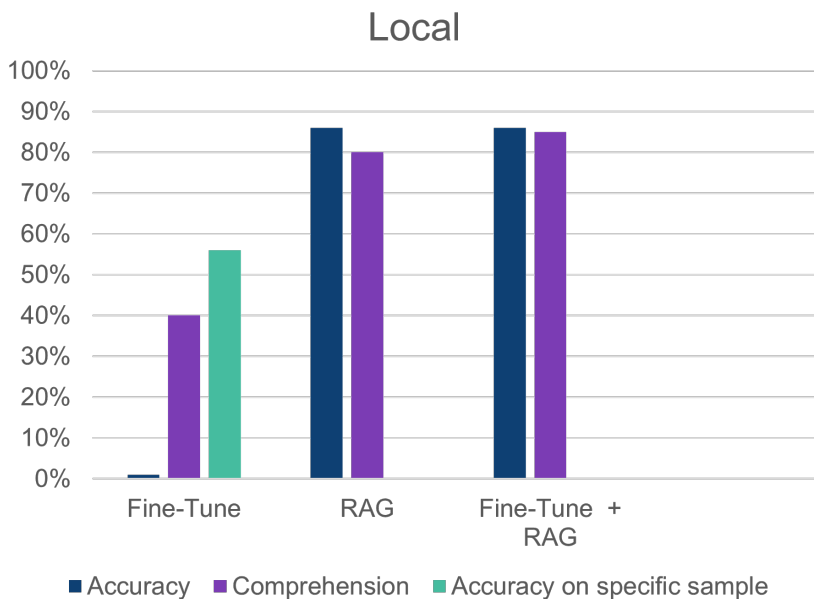


Figure 6.7: RAG with remote connection.

Method	Accuracy	Comprehension	Specific Sample Accuracy
Fine-Tuning	0.00	0.40	0.55
RAG	0.86	0.80	NA
Fine-Tuning + RAG	0.86	0.85	NA

Table 6.2: Performance Results for Local LLM Approach.

Key Observations:

- The combination of Fine-Tuning and RAG yielded similar results to RAG alone in terms of accuracy and comprehension.

- Fine-Tuning alone performed as poorly as with the remote case.
- The Local LLM setup showed similar accuracy for specific samples when using RAG compared to the API approach.
- The overall accuracy is slightly lower than that in the API case.

Like mentioned in Chapter 7, output filtering was performed. The results of that approach are presented in Table 6.3 and Figure 6.8.

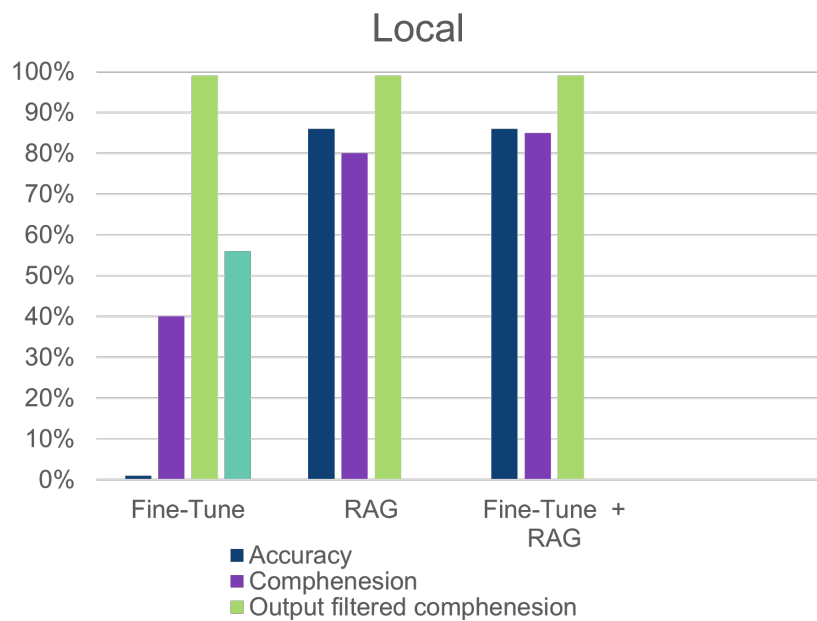


Figure 6.8: RAG with remote connection.

Table 6.3: Performance Results for Filtered Local LLM Approach

Method	Accuracy	Comprehension	Filtered comprehension
Fine-Tuning	0.00	0.40	1.00
RAG	0.86	0.80	1.00
Fine-Tuning + RAG	0.86	0.85	1.00

Key Observations:

- Filtering does not improve accuracy in any way.
- Filtering completely eliminates the comprehension issue.

6.3.4 Comparative Analysis and Conclusion

- **Overall Performance:** The Local LLM setup combined with RAG and fine-tuning achieved the same accuracy and comprehension scores as RAG. However, with well-trained models, especially with bigger LLMs, fine-tuning can be a long process and require even more resources if done locally. RAG by itself is the most effective method for the analysis of sensor data based on these results.
- **Resource Utilization:** The Local setup, while offering more control and slightly better performance, required a powerful machine unlike the API/Remote approach.
- **Practicality:** The API/Remote approach was more practical for deployment in environments where resources are limited or where maintaining a local LLM is not feasible.

To conclude, a remotely hosted model like the GPT-4o mini with RAG is chosen for further implementing and testing the recommendation system.

7. Vector Search or Query Generation?

Based on the results of Chapter 6, RAG alone was selected as the optimal solution for the sensor data analysis task. In order to evaluate the effectiveness and practicality of LLM-driven recommendation systems in smart spaces, two more in-depth approaches are investigated: FAISS-based retrieval and LLM-based code and query generation. The first approach utilizes vector embeddings to represent sensor data, using the FAISS library for rapid retrieval of contextually relevant information. The role of the LLM in this scenario is primarily response generation, relying on FAISS to conduct efficient and accurate retrieval based on user queries. In contrast to the first approach, the query generation method uses the generative capabilities of LLMs more comprehensively. The LLM is tasked with interpreting natural-language queries, dynamically deciding the appropriate code operations to execute against the sensor dataset, determining if and how to adjust user preferences based on query feedback, and ultimately formulating the natural language response.

Both methods are tested with the selected LLMs: GPT-4, Claude, Gemini-2, CodeBERT, and a locally deployed Llama-based SML CodeLlama 7B Instruct-to evaluate differences in accuracy, response latency, and adaptability. Following a comparative analysis, the most effective configuration is selected for further evaluation through a practical deployment. This deployment involves creating a user-interactive recommender system implemented via a Streamlit application. Through this interactive platform, the thesis further examines user perceptions of system effectiveness,

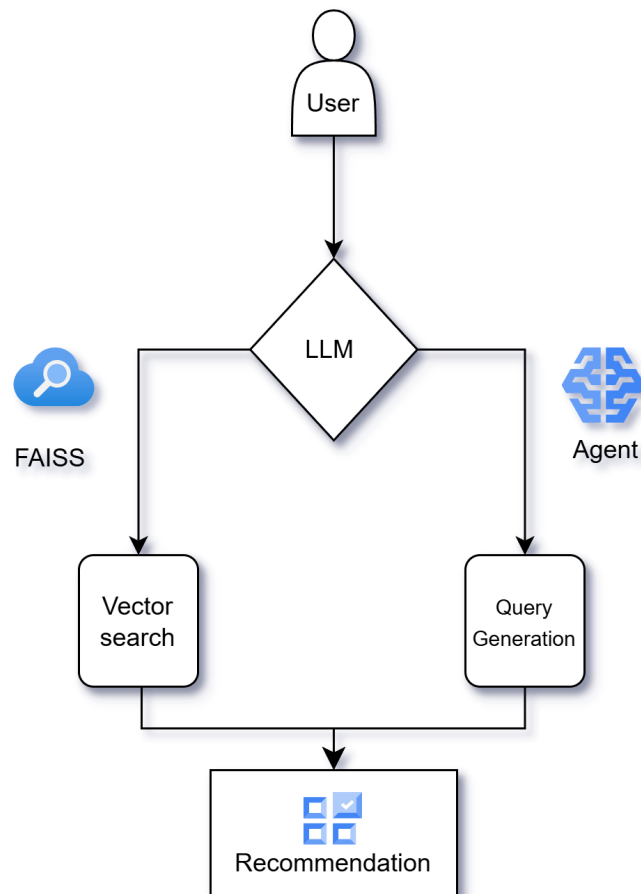


Figure 7.1: The two approaches of handling the LLMs.

responsiveness, and satisfaction, complemented by feedback analysis. The following sections provide comprehensive details about each experimental methodology and the implementation of the chosen solution.

7.1 FAISS-based retrieval

The FAISS method uses vector embeddings to represent sensor data, compare it with the user query (also in vector form), and retrieve relevant records based on those queries. The primary role of the Large Language Model in this method is to generate meaningful and contextually appropriate responses based on the retrieved sensor data. The process begins by converting each sensor data record into a descriptive string containing its identifying attributes: room location, temperature, humidity, light, CO₂

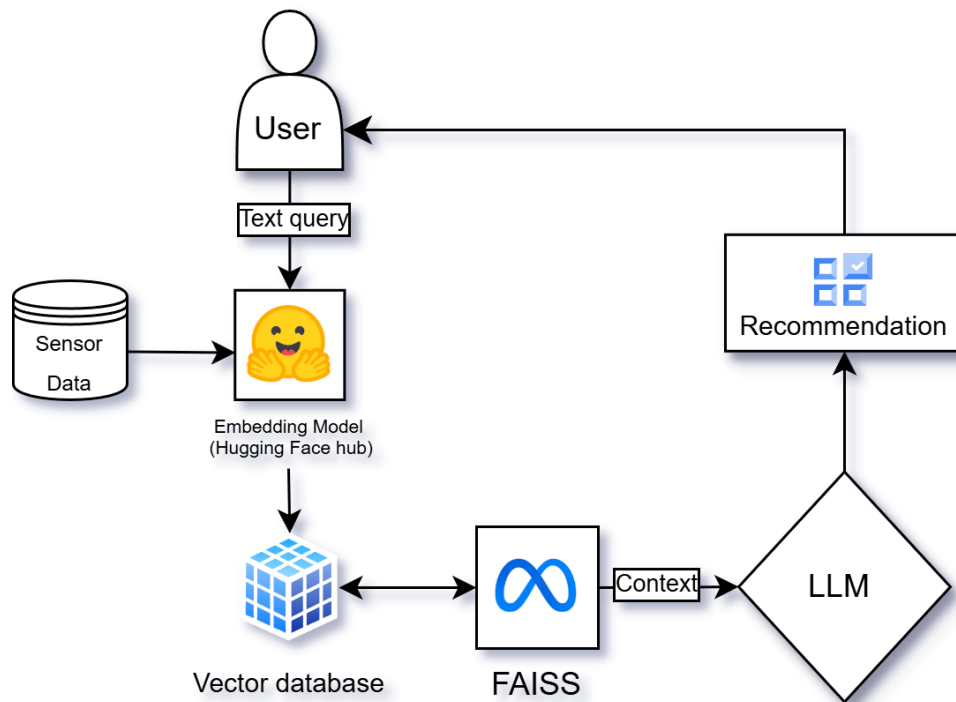


Figure 7.2: FAISS-based retrieval.

and PIR measurements from smart space sensors. Each record is then transformed into an embedding vector by a chosen transformer-based embedding model, in this case HuggingFace sentence-transformers imported from langchain library with HuggingFaceEmbeddings and using default sentence-transformers/all-MiniLM-L6-v2 embeddings. The FAISS library indexes these embedding vectors so that, upon receiving a query, a matching embedding can be produced for the query text, and the most similar sensor data records are identified. In the retrieval phase, the query is embedded by the same model used for the sensor data. FAISS looks up the top-matching embeddings using a similarity metric (in this case, default cosine similarity) and returns the data judged most relevant to the question. Once retrieved, these records are combined with the user query into a carefully structured prompt for the LLM. The language model is instructed to take the retrieved sensor information and craft a concise response that addresses the user's request.

Algorithm 1 FAISS Retrieval Logic for TerminalRAG (Pseudocode)

```

1: procedure INITIALIZEFAISSINDEX(sensorData, embeddingModel)
2:   Convert sensorData into descriptive strings
3:   Generate embeddings using embeddingModel
4:   Build FAISS index with resulting embeddings
5:   Store or cache the FAISS index for later use
6: procedure ASKQUESTION(userQuery, FAISSIndex, embeddingModel, LLM)
7:   queryEmbedding ← embeddingModel(userQuery)    ▷ Vectorize the query
8:   retrievedDocs ← FAISSIndex.similarity_search(queryEmbedding, k = 5) ▷
   Fetch top-k docs
9:   combinedContext ← Combine retrievedDocs with userQuery to form prompt
10:  response ← LLM.invoke(prompt = combinedContext)    ▷ Generate answer
11:  return response
12: procedure TERMINALRAGFAISS(sensorData, embeddingModel, LLM)
13:  INITIALIZEFAISSINDEX(sensorData, embeddingModel)
14:  Wait for query from user or system
15:  ASKQUESTION(userQuery, FAISSIndex, embeddingModel, LLM)
16:  Return generated answer

```

When a question involves recommendations about rooms, the system follows a specific guideline mentioned in the docstring of the code. In that instruction, the assistant is instructed to present the recommendation by describing one primary suggestion, an alternative option with relevant trade-offs, and any additional considerations that might help contextualize the choice further.

7.1.1 LLM prompts

Below is the prompt used by the LLM when delivering its answers. This prompt describes the system context, indicates the presence of retrieved FAISS data, and specifies how the model should articulate any room recommendations.

FAISS RAG Prompt

You are an AI assistant specialized in analyzing sensor data retrieved from FAISS and providing personalized recommendations. Provide answers in a single, concise sentence whenever possible. Please provide your response in natural, conversational language while maintaining accuracy and specificity.

You have the information about the location in the context {context}.

IMPORTANT: You must ONLY provide information and recommendations for the CURRENT USER specified at the start of the context {context} and only to the question asked {question}.

If asked about other users, respond that you can only provide information for the currently logged-in user {current_user}.

If user asks for multiple locations, provide as many as you can from the context.

If multiple locations meet the criteria, rank them and select the top one.

Use specific numerical data to support your recommendations.context: {context}

The user's question: {question}

Current user ID: {current_user}

Remember to incorporate the retrieved FAISS data (context) in your explanation.

7.2 Query-Based Execution Approach

This section focuses on the query-based methodology in which an LLM generates executable instructions that directly query sensor data. Unlike the FAISS approach, which is built around embedding vectors for retrieval, this method delegates more of the operational logic to the language model itself. The LLM receives user input in natural language and responds with a suitable piece of code designed to filter, aggregate, or otherwise manipulate sensor data. In the simplest implementation, the system prompts the LLM to produce a one-line pandas snippet that operates on a

DataFrame of smart space sensor data. By translating user intent into a compact command, the model automates the act of building complex queries. For instance, if a user asks "Which rooms are cooler than average?", the language model responds with a single command that filters rows in the DataFrame by a temperature value below the aggregate mean. Once the user's query has been transformed into workable code, a Python interpreter executes that snippet. The result is returned as text or used to inform a final recommendation. To accommodate variations in user intent, the LLM prompt includes a brief summary of available columns and typical metrics. The model is instructed to avoid returning extraneous output or multiple lines of code. By constraining the acceptable answer format, the system remains predictable and easier to handle programmatically. If queries grow more complicated (for example, including group-by operations or multi-column conditions), the same approach still applies: the user expresses a query in natural language, and the LLM fabricates a single-line code snippet that addresses the request precisely.

In addition, the modular design ensures that changing the underlying language model requires few modifications outside the prompt's instructions. Because the system always expects one concise snippet, it can integrate different versions of GPT or other LLMs. This maintains the uniformity of the output across potential updates or expansions to the architecture. In this implementation, the system also uses an analysis prompt that interprets and contextualizes the code's resulting data for the user. A separate preferences prompt, which specifically handles user comfort parameters, feedback, and adaptability, will be introduced later in this chapter.

Algorithm 2 Query-Based Execution Logic (Pseudocode)

```

1: procedure INITIALIZEQUERYCHAIN(llmModel)
2:   Load the chosen LLM model
3:   Create a prompt template describing the single-line query requirements
4:   return a chain object capable of producing code snippets
5: procedure GENERATESINGLELINECODE(userQuery, queryChain)
6:   Provide userQuery as input to queryChain
7:   codeSnippet ← queryChain.invoke(prompt = userQuery)    ▷ LLM output
8:   return codeSnippet                                     ▷ E.g. a single-line pandas statement
9: procedure EXECUTEDATAFRAMEOPERATION(codeSnippet, sensorData)
10:  Evaluate codeSnippet against sensorData in a safe environment
11:  filteredData ← results of the operation
12:  return filteredData
13: procedure GENERATEANALYSIS(filteredData, analysisChain)
14:  Provide filteredData as input to analysisChain
15:  analysisResponse ← analysisChain.invoke(prompt = filteredData)    ▷
    LLM generates explanation
16:  return analysisResponse
17: procedure QUERYBASEDAPPROACH(userQuery, sensorData, llmModel)
18:  queryChain ← INITIALIZEQUERYCHAIN(llmModel)
19:  snippet ← GENERATESINGLELINECODE(userQuery, queryChain)
20:  resultDF ← EXECUTEDATAFRAMEOPERATION(snippet, sensorData)
21:  analysisChain ← INITIALIZEQUERYCHAIN(llmModel)    ▷ Reusing LLM for
    analysis
22:  response ← GENERATEANALYSIS(resultDF, analysisChain)
23:  return response                                     ▷ Final human-readable output

```

In the above pseudocode, the system begins by loading or creating a chain object capable of generating code snippets to query the sensor dataset. This chain is initialized

with the chosen LLM model, which is responsible for interpreting user requests and constructing the corresponding data operations. Once the user provides a query, it is passed to the chain, where the language model produces a concise snippet representing a valid pandas command. By restricting the LLM's output to a single line of code, the system enforces a degree of structure on how user queries are converted into data manipulations. This makes the final code easy to parse, so it can be executed cleanly on the sensor data DataFrame in Python. After obtaining the code snippet, the interpreter safely executes it, capturing any subset or aggregation of data that is requested. The results, in the form of JSON data, are then processed through the analysis procedure. The retrieved data are provided as input to an LLM-driven analysis chain, which produces a human-readable response contextualizing the results. Unlike a retrieval-based system, this method directly translates user instructions into code, outsourcing the search and filtering logic to Python rather than an embedding or vector-based index.

7.2.1 LLM prompts

The query generation prompt is designed so that the LLM produces a structured single-line pandas query that effectively retrieves and filters sensor data based on user input. The restrictions in the prompt prevent the model from generating multi-line responses or including unnecessary code constructs. By using a strict format, consistency in execution is more certain, and the risk of unintended code injection is reduced.

Prompt for Query Generation (Part 1)

You are a data query assistant that converts natural language questions into pandas DataFrame operations. ONLY RETURN CODE, NOTHING ELSE!

YOUR OUTPUT MUST BE IN THIS EXACT FORMAT:

```
.assign(score = abs(temperature_mean - TEMP)/2 + abs(humidity_mean - HUM)/5 + abs(co2_mean - CO2)/100 + abs(light_mean - LIGHT)/10 + abs(pir_mean - PIR)).sort_values('score').head(N)
```

Where:

- TEMP, HUM, CO2, LIGHT, PIR = user's preference values
- N = 1 if single room requested, up to 5 if multiple rooms requested
- Determine N from the question context (default to 5 if unclear)

DO NOT:

- Define variables
- Write multiple lines
- Use Python syntax outside of pandas operations
- Return anything except the single-line query

AVAILABLE DATA:

DataFrame 'df' contains room measurements:

- Location (str): Room identifier
- temperature_mean (float): Average temperature in C
- humidity_mean (float): Average humidity percentage
- co2_mean (float): Average CO2 levels in ppm
- light_mean (float): Average light levels in lux
- pir_mean (float): Average occupancy (PIR sensor readings)
- description (str): Room description including conditions ranked as Good/Average/Hazardous

Prompt for Query Generation (Part 2)

top_rooms_json: {top_rooms_json}

The user's preference info: {context}

The user's question: {question}

Current user ID: {current_user}

Ensure the generated query aligns with user constraints and retrieves data efficiently.

Return only a single valid pandas command, formatted correctly for execution.

Prompt for Analysis Generation

You are a system assistant responsible for analyzing sensor data and explaining the meaning of filtered results. Provide a concise but informative response summarizing the retrieved data.

SYSTEM CONTEXT:

- You are analyzing filtered sensor data retrieved from a structured database query.*
- Provide a clear, numerical summary of the relevant parameters (temperature, humidity, CO₂, etc.).*
- Highlight how the retrieved data aligns with or deviates from the average conditions.*
- If the retrieved data set is empty, provide an alternative approach or suggest a broader query scope.*

Filtered Data Summary: {filtered_data}

User Query: {question}

Explain the retrieved data in simple terms, focusing on key numerical insights.

If no data is available, suggest a possible reason and a modification to the query.

The analysis generation prompt exists so that the system does not return raw

numerical output but instead provides a structured interpretation of the query results. The assistant explains whether the retrieved data meet the user's conditions and suggests improvements if necessary.

7.3 Preference adjusting

This section focuses on RQ3 and on how the system updates user comfort preferences via a dedicated "preference parser" prompt within the query-based approach. The preference adjustment for the FAISS method was considered as well; however, it would have to be hard coded and LLM would not participate in any part of the preference adjustment process. As such, it was decided to focus on implementing it with the query method instead. Unlike the primary query prompt, which produces a single-line pandas command, this preference parser strictly interprets statements such as "I would like the room to be slightly cooler" and converts them into JSON specifying updated temperature, humidity, or other parameter values. It links natural language feedback and the numeric fields stored in the user's preference record. The idea is that if a user expresses a desire for an adjustment to their preference, the LLM calculates a feasible offset based on the standard deviation for that metric, constrained by min and max values derived from the dataset's distribution. If the user pushes a preference beyond the dataset's observed limits, the LLM sets the "action" field to "none", preserving data integrity. The LLM is instructed to return only a JSON object. In this way, the code layer can consume the JSON, update the relevant fields, and store or display them (if asked). Any domain-specific logic, such as reading how far "slightly" or "much" should shift the user's preference, is handled within the prompt instructions, leaving the final decision to the LLM. Below is the pseudocode depicting the preference adjusting mechanism:

Algorithm 3 Preference Adjustment Logic (Query-Based Approach)

```

1: procedure PARSEPREFERENCESTATEMENT(userId, userText, paramStats,
   llm)
2:   Build JSON string from paramStats showing min, max, std for each metric
3:   Fetch existing preferences for userId (in JSON form)
4:   parserPrompt ← ConstructPrompt(paramStats, currentPrefs, userText)
5:   parserOutput ← llm.invoke(prompt = parserPrompt)           ▷ Valid JSON
6:   action, changes ← EXTRACTUPDATES(parserOutput)
7:   if action = 'update' then
8:     APPLYPREFERENCECHANGES(userId, changes)
9:   else if action = 'show' then
10:    Display or return current preferences
11:  else if action = 'none' then
12:    pass

```

The following is the prompt that provides (`{param_stats_json}`) and the user's existing preference fields to the LLM. The LLM's reply is always expected to be strictly valid JSON containing one or more updated preference values, a special "show" directive, or an indication that no changes are necessary.

Prompt for Preference Parser (Part 1)

You are a "Preference Parser" for a sensor-based recommendation system.

****IMPORTANT****: Return valid JSON *only*, with *no* triple backticks, code fences, or extra text.

Only a single JSON object as your entire output.

We have the dataset stats (min, max, stdev) for each param in JSON:

`{param_stats_json}`

The user's current preferences in JSON: `{current_prefs_json}`

The user's latest message is: `{user_text}`

Instructions:

1. If the user says "slightly warmer," interpret that as $+0.5 * \text{stdev}$ for temperature,
if "much warmer," consider $+2$ or $+3$ stdev if feasible, etc.
2. If the offset or absolute value goes beyond min-max, either do partial offset
or
set "action": "none" if still not feasible.
3. If user only wants to show preferences, set "action": "show," everything else null.

Prompt for Preference Parser (Part 2)

4. If user says "colder than last recommended reading by X," do the math.
 e.g. last reading=21.3 => new temperature_preference=21.3 - X if in range.

5. If user does not mention any preference change, set "action": "none."

6. Return EXACT JSON with the shape:

```
{
  "action": "<'update','show','none'>",
  "temperature_preference": <float or null>,
  "temperature_sensitivity": <string or null>,
  "temperature_min": <float or null>,
  "temperature_max": <float or null>,
  "humidity_preference": <float or null>,
  "humidity_sensitivity": <string or null>,
  "humidity_min": <float or null>,
  "humidity_max": <float or null>,
  "co2_preference": <float or null>,
  "co2_sensitivity": <string or null>,
  "co2_min": <float or null>,
  "co2_max": <float or null>,
  "light_preference": <float or null>,
  "light_sensitivity": <string or null>,
  "light_min": <float or null>,
  "light_max": <float or null>,
  "occupancy_preference": <float or null>,
  "occupancy_sensitivity": <string or null>,
  "occupancy_min": <float or null>,
  "occupancy_max": <float or null>,
  "notes": "<brief explanation of how you computed the final numeric>"
}
```

No extra lines, no triple backticks. Only valid JSON.

If the user's request references the last recommended reading, the LLM can compute direct offsets as long as they remain feasible. Similarly, if the user merely wants to see their stored preferences, "action": "show" can be returned, prompting the system to reveal those values without making any modifications.

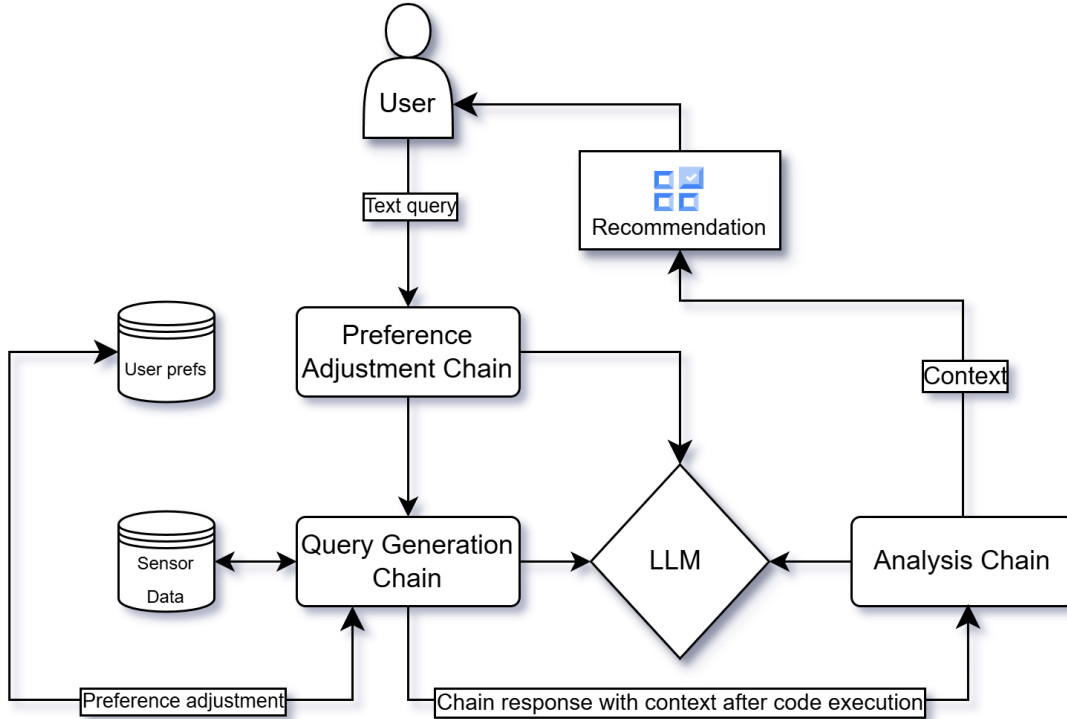


Figure 7.3: Query generating agent's architecture.

7.3.1 Preference Integration Pipeline

Natural language feedback is first converted into a strict JSON record using the *Preference-Parser* LLM described in Algorithm 3 (Preference Adjustment Logic). The current preference vector and empirical statistics (\min, \max, σ) for every environmental parameter are injected into the prompt, ensuring that the LLM is aware of feasible numeric ranges. The prompt contract allows only for a single JSON object with an `action` field and any modified values; no other text is accepted, which makes downstream parsing deterministic. The backend verifies each proposed value before it is persisted. A preference is accepted only if $\min \leq v \leq \max$; requests outside this

envelope are logged and ignored, and "action" is downgraded to "none". After validation, the record is written to `user_preferences.csv`. Subsequent query-generation prompts read the active preference row directly from this file, so the latest settings are always respected. During query generation, the preference vector acts as *soft constraint*. If the strict filter produced by the LLM yields an empty result set, the prompt instructs the model to return to a Euclidean distance ranking, thus guaranteeing at least one recommendation.

Example pipeline.

1. User input: "CO₂ should be much lower than before."
2. LLM reply: {"action": "update", "co2_preference": 450.0} (interpreted as -2σ from the previous target).
3. Backend range-checks the value and stores 450 ppm.
4. At the next query the scoring term $\text{abs}(\text{co2_mean}-450)/100$ is injected into the single-line pandas command generated for retrieval.

For every user, a single-row CSV file is maintained with the column representing `user_id`, as well as 4 columns for each of the parameters in the form: `parameter_preference`, `parameter_sensitivity`, `parameter_min`, `parameter_max`. When *Preference-Parser* returns a JSON object, the back-end loads the corresponding row and overwrites only the keys present in the patch. No version history is retained; therefore, a conflicting sequence such as 'slightly warmer' followed by 'much colder' resolves deterministically to the latter value. Each column `*_min*` and `*_max*` stores the empirical bounds seen in the training corpus, so any update that would exceed those limits is clipped before the row is written. The `*sensitivity*` columns hold user-supplied integers (1 = low, 2 = medium, 3 = high) and serve as weights in the retrieval score. The current preference vector \mathbf{p} and weight vector \mathbf{w} are

appended to the LLM prompt, producing a pandas command of the form:

```
df.assign(score=-sum_k w_k |x_k - p_k|).sort_values("score").
```

Higher sensitivity therefore increases the impact of the corresponding attribute on ranking, while the unmentioned attributes inherit the default mid-range values stored in the row.

7.4 Performance metrics

Given the two primary tasks-query generation and FAISS retrieval-augmented output generation-the following metrics were chosen for their ability to provide a comprehensive assessment of model performance: precision, accuracy, recall, f1, query completion time, token generation latency. These metrics are typical for LLM tests [16][36][1] and can be used for several notable benchmarks such as BERTScore [68].

Precision measures the ratio of true positive predictions to the total predicted positives. Quantifies the accuracy of the relevant output of the model. High precision means that the generated queries align with the intended data manipulation, which minimizes incorrect output.

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}} \quad (7.1)$$

Accuracy reflects the overall correctness of the model predictions by evaluating the ratio of correctly predicted outputs to the total number of cases.

$$\text{Accuracy} = \frac{\text{TP} + \text{True Negatives (TN)}}{\text{Total Predictions}} \quad (7.2)$$

Recall measures the ability of the model to correctly identify all relevant outputs. In the context of this work, recall indicates how effectively the model retrieves accurate sensor data or constructs correct queries. High recall is useful to ensure that critical

data points are not missed.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{False Negatives (FN)}} \quad (7.3)$$

The F1 score is the harmonic mean of precision and recall. It balances both metrics into a single value, making it suitable for tasks where both false positives and false negatives must be minimized. This metric is also useful in evaluating LLMs specifically, as a model that prioritizes precision over recall (or vice versa) may still fail to provide reliable outputs in real-world applications.

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (7.4)$$

Query completion time measures the total time taken by the model to generate a response or execute a query. It is important to note that this thesis uses this metric to measure the time it takes for the LLM to generate a response once it has received it, not the total amount of time it takes to run the program. This depends on the Streamlit server and/or local hardware.

Token generation latency, or sometimes referred to as Inter-Token Latency (ITL), represents the average time (in milliseconds) required for the model to generate a single token. This metric helps analyze the computational efficiency of LLMs. It is useful when comparing larger models like GPT-4 and Claude 3.5 Sonnet with smaller, locally deployable models like Codellama 7B or CodeBERT. Lower token latency translates into faster response times, which can be used in resource-constrained environments. This helps estimate both the time the model needs to complete queries and the power consumption likely required. It is harder to be more specific in these metrics, as the hardware and latency of the Internet connection can influence the inference speed of the model and the delivery of the generated output.

7.4.1 Evaluation Benchmarks for Large Language Models

BERTScore evaluates the similarity between generated text and reference text by using contextual embeddings from pre-trained transformer models like BERT. Unlike traditional string-matching methods such as BLEU, *BERTScore* computes cosine similarity between token embeddings, capturing semantic similarity. It is defined as follows:

$$\text{BERTScore} = \frac{\sum_{i=1}^n \text{cosine}(E_{\text{gen},i}, E_{\text{ref},i})}{n}, \quad (7.5)$$

where $E_{\text{gen},i}$ and $E_{\text{ref},i}$ are embeddings of tokens from the generated and reference texts, respectively, and n is the total number of tokens. *BERTScore* is particularly effective in tasks involving paraphrasing or semantic understanding, as it accounts for contextual meanings rather than exact word matches.

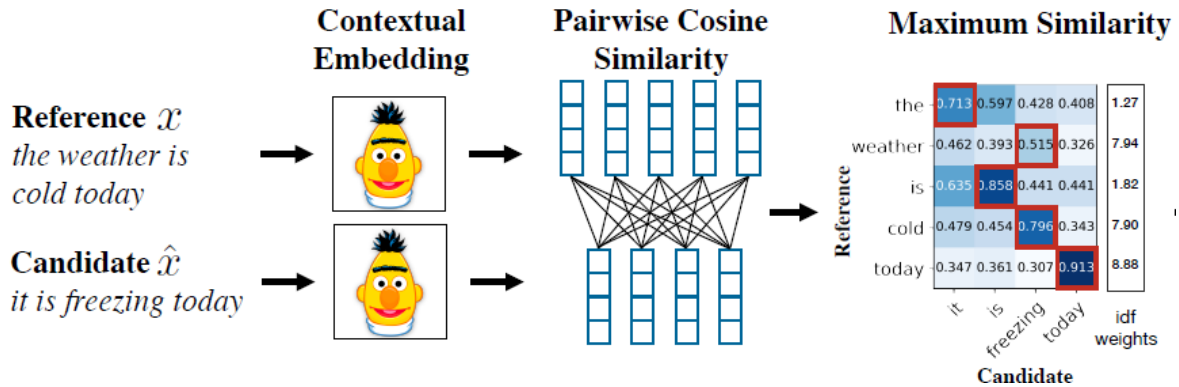


Figure 7.4: BERTScore structure.

The *BLEU Score* (Bilingual Evaluation Understudy) measures the n-gram overlap between the generated and reference texts. It penalizes outputs lacking precise word matches and is commonly used for machine translation and language generation tasks. BLEU calculates precision for n-grams and applies a brevity penalty for shorter outputs:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^N w_n \log p_n\right), \quad (7.6)$$

where BP is the brevity penalty, p_n is the precision for n-grams, and w_n are weights for different n-grams.

The *ROUGE* (Recall-Oriented Understudy for Gisting Evaluation) suite evaluates the recall-oriented overlap between generated and reference text. It is widely used for summarization tasks. ROUGE variants include:

- ROUGE-N: Overlap of n-grams.
- ROUGE-L: Longest Common Subsequence (LCS) similarity.
- ROUGE-S: Skip-bigram similarity allowing for non-contiguous matches.

ROUGE is effective in assessing recall, ensuring that critical content from the reference is kept in the generated text.

Perplexity measures how well a language model predicts a sample of text. Lower perplexity indicates that the model assigns a higher probability to correct sequences. Perplexity is calculated as:

$$\text{Perplexity} = \exp \left(-\frac{1}{N} \sum_{i=1}^N \log P(w_i | w_1, \dots, w_{i-1}) \right), \quad (7.7)$$

where N is the total number of tokens and $P(w_i)$ is the probability assigned to token w_i . Perplexity is a fundamental metric in the pre-training and evaluation of language models.

As this thesis aims to test several model families, including BERT, as well as focusing specifically on measuring the ability of the system to recommend data based on ground truth defined independently, BERTScore was chosen as the appropriate testing benchmark.

8. Results and Evaluation

This chapter covers the user survey structure, the Streamlit app, heat mapping, the results obtained from the comparison between vector similarity search and query generation approaches between 5 different models and briefly discusses the results.

8.1 User Perception Study on the Smart Space Recommendation System

To evaluate user perception of the recommendation system, a structured survey was conducted using an online Google Form, specifically tailored to reflect the dimensions of user interaction and feedback in smart spaces. This survey was based on the ResQue evaluation framework [40], which provides a comprehensive method to evaluate the qualities of the system, the attitudes, beliefs, and behavioral intentions of users through targeted survey questions. Participants were selected from a diverse pool of users who were familiar with the University of Oulu's Smart Campus area. The survey was administered online, allowing anonymous responses to encourage honest feedback. For a better user experience, an app was designed, which will be explained in the next section. A total of 10 complete responses were collected using the questionnaire. This section outlines the types of questions, but all items in the questionnaire are reproduced verbatim in the Appendix A. The survey consisted of 13 targeted questions designed to capture detailed feedback on the recommendation system. It included three main sections:

- **Preliminary questions**, gathering general demographic and sensitivity information:
 - Age of users.
 - General sensitivities such as allergies and migraines.
 - Specific environmental sensitivities during work or study, including noise, crowding, smells, temperature changes, and light variations.
 - Types of queries previously posed to the system, including aspects such as light, noise, CO₂ levels, air quality, temperature, and occupancy.

- **Performance questions**, directly related to the perceived effectiveness and usability of the system:
 - Users rated whether the recommended rooms met their expectations on a scale from 1 (none met expectations) to 10 (all rooms met expectations). This will be compared with the accuracy results from other tests.
 - Users assessed if rooms they expected were missed by the recommendations. This will help measure recall compared to other evaluations.
 - Users evaluated whether inappropriate rooms were recommended by mistake. This metric will be used to assess precision relative to other system evaluations.
 - The users gave an overall performance rating for the system.
 - The ease of use and clarity of the recommendations provided by the system were also rated.
 - The likelihood of recommending the system to others was evaluated.
 - The frequency of app usage was recorded to understand the level of engagement.

- **Additional questions**, collecting qualitative insights:
 - An open-ended feedback question where users could freely express comments or suggestions for improvement.

Questions Q5-Q7 provide the information required to approximate standard classification metrics. A score of ≥ 6 on the 10-point scale was treated as a positive response. According to this convention *precision* was derived from Q7, where a high score denotes few unsuitable rooms; responses below the threshold were counted as false positives. *Recall* was obtained from Q6, which asked whether any relevant rooms had been omitted; sub-threshold answers indicated false negatives. *True positives* were inferred when Q5 ("rooms met expectations") and Q7 were both positive, while *true negatives* occurred when Q5 was negative and Q6 was positive. Accuracy and the harmonic mean F_1 were calculated from these four counts and reported alongside the descriptive means in the final results table. Using this approach, the results of the survey provide clear indicators of system performance from the user's point of view. The precision, accuracy, and recall metrics derived from the user ratings will be used in comparison with the theoretical values to measure the effectiveness of the recommendations, while qualitative comments help with the understanding of user satisfaction and highlight potential areas for improvement. This evaluation verifies the practical utility of the recommendation system, but also aligns with previous assessments carried out within smart campus environments [34, 32], ensuring methodological continuity and relevance.

8.2 Heat Mapping Process

In order to visualize how different rooms in the environment compare on the available sensor metrics (temperature, humidity, CO₂, lighting, occupancy), a systematic approach was created to create color-coded heat maps. These maps help users quickly see which rooms meet certain conditions (for instance, 'cooler', 'brighter', or 'less crowded')

and also allow for easy comparisons across multiple parameters.

All aggregated data are collected and then linked to certain locations provided in the original dataset from the paper by Motlagh et al. [35] on digital twins. To translate these numerical readings into an intuitive graphic, a coordinate-based layout of the building with sensor locations was created. Each room is placed on a scatter plot at specific x-y positions that match its real-world location. The testing version uses Matplotlib, while the application version uses Plotly. Each point is colored by the sensor reading (the 'coolwarm' parameter is used for heat mapping with matplotlib plots, with warmer rooms colored red and cooler rooms colored blue). This makes it straightforward to identify outlier rooms and spot clusters with similar conditions for users. The heatmaps are overlaid on the real map in order to make it easier for users to understand it. Since multiple parameters need to be visualized at once for clarity, several maps (each has its own tab) are arranged at once-one for each sensor metric-so a reader can switch between multiple color-coded charts. Each map uses the same layout or coordinate system, ensuring that location positions never change. Only the color scale changes to reflect the chosen metric (for example, the first map shows temperature differences, the second displays humidity variations, etc.). This consistency can also help reveal how different factors overlap and whether a comfortable temperature could coincide with high or low humidity, for example.

The specific recommended room is highlighted on every map so that users can see if the recommendation matches their request. A simple marker is drawn in the form of a red circle on top of that room's center coordinates. By doing so, the user can easily locate the recommended option. Unfortunately, the original dataset did not provide full dimensions for the rooms, only the sensor locations. As such, a fully continuous 'surface' is more appropriate because the entire rectangle of the map is folded that way. An interpolation step is used: this estimates the values between sensor locations and fills all empty spaces with the 'closest' or 'averaged' reading. Although this yields a smoother and more intuitive view, it does rely on assumptions about the similarity

between neighboring rooms. However, it does present a much nicer view for the users as well as model a real environment in an acceptable manner. A user then asks "Can you please give me a location slightly warmer than average?", which the LLM then interprets as a warm room, and the recommendation system changes the recommended room to one that is calculated to be above average. Figure 8.1 shows all maps side by side, where the recommended locations are highlighted on the last heatmap.

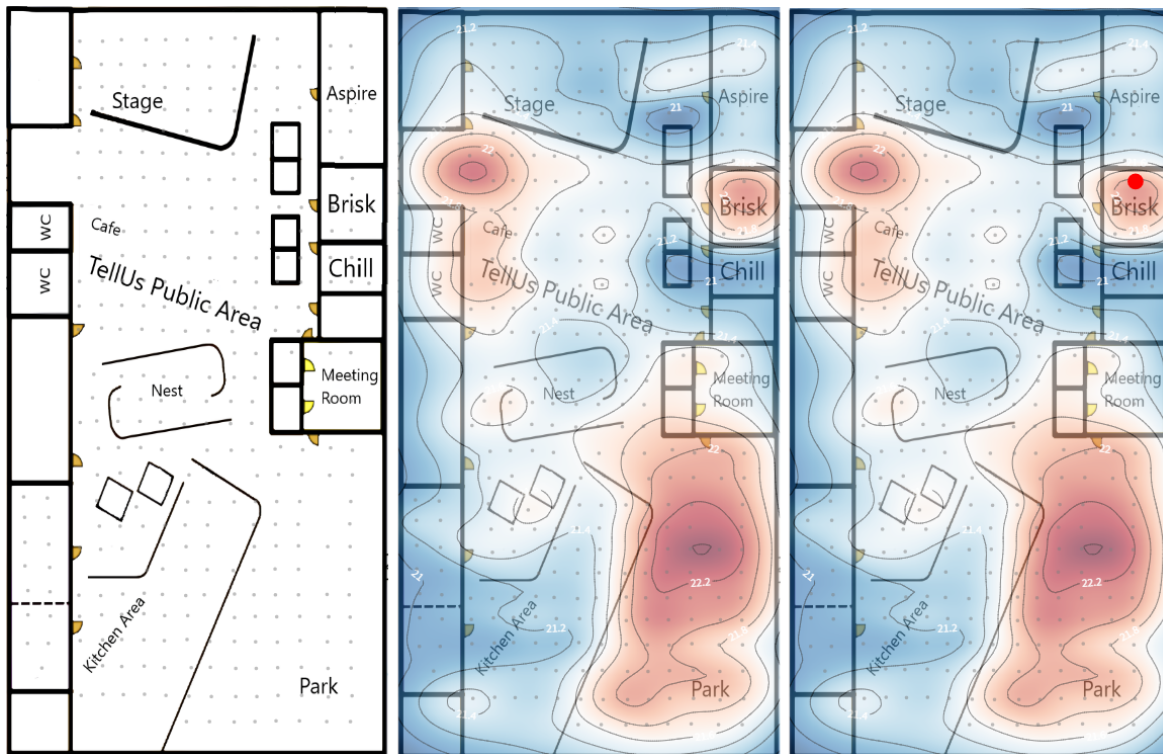


Figure 8.1: All maps side by side.

8.3 Streamlit Application Deployment

To help users better interact with the recommender system, a Streamlit Web application was designed. This application provides a browser-accessible interface through which users can query the sensor-based dataset, adjust their environmental preferences, and visualize real-time conditions for various spaces with heat maps. The system is hosted on Streamlit's cloud platform and uses the same smart space sensor data described in earlier sections, which originate from the Oulu-based experiments referenced

in Motlagh et al. [35]. When opening the link and launching the app, users are greeted by a main screen, prompting them to either choose from predefined examples or formulate their own queries in natural language. Queries invoke the Query-Generation based LLM retrieval with a "Find Workspace" button, which then triggers the backend logic, returning recommendations about suitable rooms or preferences. Collapsible sections, such as "How to adjust your preferences", provide short instructions on how to phrase requests. The application uses the same interface for adjusting the preferences. Users simply type the feedback or a subsequent query into the same input field. Any preference changes are stored and can be reused in subsequent queries. To provide visual clarity, the app also displays heatmaps corresponding to each environmental metric. Each visualization is updated dynamically with fresh data or new queries so that the user can observe how environmental conditions vary between rooms. The selected rooms are highlighted in red circles and every room can be seen on the map due to Plotly integration with Streamlit. Because the application is web-based, the distribution to participants for user feedback studies is done through a publicly accessible Streamlit link, allowing users to interact with the system from any browser.

8.4 Results

All 5 models performed well on the Query Generation, with GPT-4 in the lead, which managed to achieve the F1 score of 0.95 over 100 samples. Claude 3.5 and Gemini 1.5 were closely tied in second place. CodeBert performed the worst, which is somewhat expected due to its size. Codellama 7b is second to last with a relatively good 0.81 F1 score. It appears that big models with dozens of billions of parameters do slightly better on these tasks. Smaller models like CodeBERT and CodeLlama 7b perform worse but offer a viable alternative for solutions that cannot rely on Internet access or API subscriptions.

Table 8.1: Performance metrics for different LLMs in Query Generation

Metric	GPT-4	Claude 3.5	Gemini 1.5	CodeBERT	Codellama 7B
Precision	0.95	0.90	0.88	0.76	0.80
Accuracy	0.93	0.89	0.87	0.74	0.78
Recall	0.96	0.91	0.89	0.67	0.82
F1	0.95	0.90	0.88	0.74	0.81
QTC (s)	0.8	1.2	1.5	19.2	15.9
TL (ms/token)	20	25	30	15	15

Table 8.2: Performance metrics for different LLMs in Query Generation (with preference adjusting)

Metric	GPT-4	Claude 3.5	Gemini 1.5	CodeBERT	Codellama 7B
Precision	0.94	0.91	0.85	0.72	0.78
Accuracy	0.92	0.87	0.89	0.70	0.75
Recall	0.95	0.91	0.86	0.68	0.79
F1	0.94	0.91	0.85	0.69	0.78
QTC (s)	1.00	1.25	1.40	18.5	16.3
TL (ms/token)	23	27	33	17	17

In general, the integration of preference did not affect the performance of the LLMs too much. In some cases, the preferences were not adjusted correctly, but those

were rare occurrences. Most of the queries were unaffected due to the nature of the questions they posed. Only a subset of the questions had actual values when it came to determining the ground truth relative to preference adjustments. The models performed roughly the same with Gemini 1.5 being slightly worse in this case than GPT-4 and Claude 3.5. Overall, the preference integration does not improve the models' performance, but it does add a tool for the users to interact with.

Table 8.3: Performance metrics for different LLMs with FAISS Retrieval

Metric	GPT-4	Claude 3.5	Gemini 1.5	CodeBERT	Codellama 7B
Precision	0.92	0.88	0.89	0.65	0.77
Accuracy	0.91	0.87	0.87	0.63	0.75
Recall	0.94	0.89	0.87	0.63	0.78
F1	0.93	0.88	0.88	0.61	0.77
QTC (s)	0.7	1.1	1.2	17.2	10.8
TL (ms/token)	18	22	20	12	13

Vector search retrieval with FAISS has demonstrated somewhat lower F1-scores and accuracies than the query generation approach, however not by much. Gemini 1.5 even performed slightly better than with the integration of preference adjustment and query generation. It shows that vector search is still a very powerful technique that can be utilized for recommendation systems and offers competitive performance compared to query generation. Another upside of this approach is that the context that the model receives does not depend on the model's characteristics. Only the response is shaped by the model. As such, it shows that vector search by itself can compete with the best models that create queries for an agent to retrieve the necessary data.

As for the user feedback, this thesis takes the best performing solution:GPT-4 with query generation approach, and analyzes the user feedback from using the Streamlit application with this setup. A Google form is sent along with the application with the questions from the User Study section of Chapter 7. The analysis of the survey responses yielded the following average scores: precision of 0.65 (indicating moder-

ate accuracy in avoiding unsuitable room suggestions), accuracy of 0.76 (indicating generally positive perceptions of overall system performance), recall of 0.73 (suggesting the system typically included relevant room recommendations) and an F1 score of approximately 0.69. These metrics indicate moderate performance, with room for improvement particularly in precision to enhance user satisfaction. Given sample size 10, the results are presented as descriptive proportions; formal significance testing was not attempted because the sample is too small for reliable inference. This evaluation does verify the practical utility of the recommendation system, but also shows certain weaknesses and areas where further improvements are needed. In particular, users noted that TellUs Smart Space might not be the best match for this system due to the necessity of booking rooms before they can be accessed. Additionally, certain users expressed concerns that the system might recommend locations with low PIR values even if there is already a person inside who previously used this application to find a quiet place. There were also jokes, particularly about the availability of therapy cats on campus, which is a natural part of any feedback collected by users. Those were not taken into account during the evaluation. Overall, users agreed that the app is useful, but some further improvements are needed and the targeted areas need to be revisited.

Table 8.4: Feedback Evaluation Metrics for GPT-4 Query Generation Retrieval

Model	Precision	Accuracy	Recall	F1
GPT-4	0.65	0.76	0.73	0.69

9. Discussion and Future Works

The results of this thesis highlight the potential of Large Language Models in the context of sensor data analysis. Specifically, the combination of Retrieval-Augmented Generation and Query Generating AI agents. This approach allows for more accurate and contextually relevant responses by using the strengths of both RAG and LLM's ability to generate code, providing the flexibility to dynamically incorporate new data without the need for constant retraining. In fact, any model can be used with these 2 solutions without preliminary training. The data is retrieved in real time with RAG and then passed as a context to the LLM. Fine-tuning has proven to be ineffective in enhancing the output of the model. The constant need for retraining and the nature of the data make it very challenging to effectively use fine-tuning in the context of real-time sensor data analysis.

However, there are notable limitations to this solution. The current system appears to be not perfectly suited for deployment on booked spaces due to the inability of the users to freely choose their spots and the fact that the system can recommend locations that are already booked. This limitation necessitates incorporation into the University of Oulu's booking system. In addition, security is an issue to consider in the future. Ideally, all user preference data will be hosted on secure remote servers that are regularly backed up and have a good authentication system. Moreover, the LLM field is constantly evolving and, as such, it is important to regularly check for newer models, possible AI agents, and vector database solutions for faster and more accurate results. All performance measurements were also taken on a single workstation; the effect of different hardware accelerators or high-latency links remains unquantified and

is left for future investigation.

In the future, the focus will also be on developing a recommendation system that uses these approaches for real-time sensor data analysis. The system will receive data in real time through an API, and the data will be regularly updated. This is the end goal for such systems anyway due to the need for users to have up-to-date information. Another step would be to test the developed recommendation system within the Ubikampus smart space at the University of Helsinki and TellUs Smart Space at the University of Oulu directly. Ubikampus is an open working space of approximately 400 square meters, visited by university staff and students daily [34]. The space is equipped with some motion detectors installed on the ceiling and low-cost air quality sensors strategically placed around the area. These sensors measure various parameters, including levels of CO₂ and PM_{2.5}, providing a rich data set to test the recommendation system in a real world environment. The similarity is true for the TellUs Smart Space. The goals of future work include the following.

- Work more on the data: Move all data to secure storage, separate different types of data, keep track of metadata for each response. This means implementing a solution that allows the system to securely store and utilize the data and collect user feedback for further analysis.
- Expanding the User Interface : Further development of the UI to make it more user-friendly, intuitive, and capable of handling the complex interactions required for real-time data analysis.
- Making the response times faster: Keeping track of response times and trying to minimize them as much as possible.
- Model Updates: Continuously updating the LLM models as new advancements are made in the field, ensuring that the recommendation system remains at the cutting edge of technology.
- Testing on actual live data from Thunderboard Sensors: Conduct extensive tests

using live data from sensors to validate the system's effectiveness in a real-world scenario and refine the model based on relevant real-time sensor data.

- Deployment in Ubikampus: Testing the system within the Ubikampus smart space, using the existing deployed motion detectors and air quality sensors to assess the system's performance in monitoring and improving indoor air quality.
- Incorporate room booking: Make sure that the locations are not recommended if already booked.

10. Conclusions

This thesis has demonstrated the significant potential of integrating Large Language Models into real-time sensor data analysis. Using the combination of Retrieval-Augmented Generation and Agentic AI, a system has been developed, capable of providing mostly accurate and contextually relevant responses to user queries based on sensor data from smart spaces. Two solutions were tested and both have shown performance, with F1 scores exceeding 0.8 for most models. The GPT-4 query generator setup proved to be the most effective and received high marks from the surveyed users.

For **RQ1**, the Query Generation method slightly outperformed FAISS-based retrieval, achieving higher precision (0.95 vs 0.92), recall (0.96 vs 0.94), and F1 score (0.95 vs 0.93) using GPT-4, making it the recommended approach for precise query handling. However, the difference is quite small. In addition, Gemini 1.5 actually has a higher F1 score (0.85 vs 0.88) in the FAISS approach than in the query generation with preference adjusting. This shows that both approaches are viable for a recommendation system using smart spaces, and the best performing system may change depending on the sample size and retrieved data. Other metrics, of which F1 is composed, follow the same trend. Overall, both methods show a high amount of correctly retrieved locations as well as a low amount of incorrect responses.

For **RQ2**, FAISS retrieval provided faster query completion times (0.7 s, 1.1 s, 1.2 s, 17.2 s, 10.8 s) compared to query generation methods (0.8 s, 1.2 s, 1.5 s, 19.2 s, 15.9 s), thus preferable for speed-sensitive applications. This can be explained by the fact that the query generator has 3 agentic chains while the FAISS approach only has 1. This means that with the FAISS approach, LLM only generates the response text with

the provided context, thus only spending time generating tokens during inference for the final response. In the query generator, the LLM has to generate the query and the response. Also, with the preference adjustment mechanism, the query generator has another chain, the preference chain. The other chains have to wait until the preference chain completes its own analysis of the user query. It is reflected in the higher QTCs across all models except CodeBERT (1 s, 1.3 s, 1.4 s, 18.5 s, 16.3 s). The token latency (in ms/token), ideally, should be the same for each model. However, the token latency can be thought of as the model's speed of generating tokens. It will depend, for instance, on the input prompts. The prompts for both approaches are quite different, including a separate prompt for preference adjustment. This corresponds to the values measured during the experiments. The methods with the biggest prompts took the longest to generate tokens. Finally, server latency for remotely accessed models is always a factor, even though there was a stable internet connection of 500+ Mbit/s during all the experiments.

For **RQ3**, GPT-4 demonstrated an effective incorporation of user feedback, maintaining high accuracy (0.92) and precision (0.94) even after dynamic preference adjustments. However, the difference is not large enough to note a significant improvement either. The choice of whether or not to include preference adjustments should be made on the basis of the user preference.

For **RQ4**, users reported generally positive perceptions of the recommendation system, with an average accuracy rating of 7.56/10, indicating satisfactory overall system performance. Precision, which means suitability of recommended rooms, received an average rating of 6.5/10, indicating some room for improvement to avoid inappropriate recommendations. Recall, which measures the effectiveness of the system in including relevant rooms, was rated at 7.3/10, indicating a moderate level of effectiveness. The calculated F1 score of approximately 6.88 shows that the system has achieved a reasonable balance between precision and recall, but indicates a need for further optimization.

Bibliography

- [1] A. Agrawal, A. Agarwal, N. Kedia, J. Mohan, S. Kundu, N. Kwatra, R. Ramjee, and A. Tumanov. Etalon: Holistic performance evaluation framework for llm inference systems. *arXiv preprint arXiv:2407.07000*, 2024. <https://arxiv.org/abs/2407.07000> (visited on 05/01/2025).

- [2] M. AI. Faiss official documentation. <https://faiss.ai>, 2024. (visited on 12/12/2024).

- [3] S. N. Akter et al. An in-depth look at gemini’s language abilities. *arXiv preprint arXiv:2312.11444*, 2023. <https://arxiv.org/abs/2312.11444> (visited on 22/01/2025).

- [4] Anthropic. Claude 3.5 sonnet model card addendum, 2024. https://www-cdn.anthropic.com/fed9cc193a14b84131812372d8d5857f8f304c52/Model_Card_Claude_3_Addendum.pdf, (visited on 26/10/2024).

- [5] K. Aoyama, K. Saito, and T. Ikeda. Inverted-file k-means clustering: Performance analysis. *arXiv preprint arXiv:2002.09094*, 2020. <https://arxiv.org/abs/2002.09094> (visited on 13/12/2024).

- [6] Y. Bai et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022. <https://arxiv.org/abs/2212.08073> (visited on 22/11/2024).

-
- [7] N. B. Brown. Enhancing trust in llms: Algorithms for comparing and interpreting llms. *arXiv preprint arXiv:2406.01943*, 2024. <https://arxiv.org/abs/2406.01943> (visited on 10/12/2024).
- [8] N. Chen, Q. Dai, and X. Dong. Evaluating conversational recommender systems with large language models: A user-centric evaluation framework. *arXiv preprint*, 2025. <https://arxiv.org/abs/2501.09493> (visited on 02/02/2025).
- [9] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2019. <https://arxiv.org/abs/1810.04805> (visited on 04/01/2025).
- [10] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazare, M. Lomeli, L. Hosseini, and H. JÃ©gou. The faiss library. *arXiv preprint arXiv:2401.08281*, 2024. <https://arxiv.org/abs/2401.08281> (visited on 12/12/2024).
- [11] C. Fourrier, N. Habib, A. Lozovskaya, K. Szafer, and T. Wolf. Open llm leaderboard v2. https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard, 2024. (visited on 10/10/2024).
- [12] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, Q. Guo, M. Wang, and H. Wang. Retrieval-augmented generation for large language models: A survey. *ArXiv*, abs/2312.10997, 2023. <https://api.semanticscholar.org/CorpusID:266359151> (visited on 22/11/2024).
- [13] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021. <https://arxiv.org/abs/2103.03874> (visited on 05/11/2024).
- [14] S. Hong, Y. Lin, B. Liu, B. Wu, D. Li, J. Chen, J. Zhang, J. Wang, L. Zhang, M. Zhuge, T. Guo, T. Zhou, W. Tao, W. Wang, X. Tang, X. Lu, X. Liang,

- Y. Fei, Y. Cheng, Z. Xu, C. Wu, L. Zhang, M. Yang, and X. Zheng. Data interpreter: An llm agent for data science. *ArXiv*, abs/2402.18679, 2024. <https://api.semanticscholar.org/CorpusID:268063292> (visited on 28/10/2024).
- [15] J. Howard and S. Ruder. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 328–339, 2018.
- [16] T. Hu and X.-H. Zhou. Unveiling llm evaluation focused on metrics: Challenges and solutions. *arXiv preprint arXiv:2404.09135*, 2024. <https://arxiv.org/abs/2404.09135> (visited on 09/01/2025).
- [17] G. Izacard, P. Lewis, M. Lomeli, L. Hosseini, F. Petroni, T. Schick, J. Dwivedi-Yu, A. Joulin, S. Riedel, and E. Grave. Atlas: few-shot learning with retrieval augmented language models. *J. Mach. Learn. Res.*, 24(1), 2024.
- [18] N. Jain, K. Han, A. Gu, W.-D. Li, F. Yan, T. Zhang, S. Wang, A. Solar-Lezama, K. Sen, and I. Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024. <https://arxiv.org/abs/2403.07974> (visited on 02/01/2025).
- [19] Z. Jiang, F. Xu, L. Gao, Z. Sun, Q. Liu, J. Dwivedi-Yu, Y. Yang, J. Callan, and G. Neubig. Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7969–7992, Singapore, 2023. Association for Computational Linguistics.
- [20] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017. <https://arxiv.org/abs/1702.08734> (visited on 02/11/2024).
- [21] Z. R. JÄänki and V. Bilicki. Rule-based architectural design pattern recognition with gpt models. *Electronics*, 12(15):3364, 2023. DOI: 10.3390/electronics12153364.

- [22] C. Kachris. A survey on hardware accelerators for large language models. *Applied Sciences*, 15(2):586, Jan. 2025. DOI: 10.3390/app15020586.
- [23] M. Kang, J. M. Kwak, J. Baek, and S. J. Hwang. Knowledge graph-augmented language models for knowledge-grounded dialogue generation. *ArXiv*, abs/2305.18846, 2023. <https://api.semanticscholar.org/CorpusID:258967643> (visited on 08/12/2024).
- [24] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online, Nov. 2020. Association for Computational Linguistics.
- [25] K. Kaur, M. Chadha, and V. Gupta. Efficient and responsible adaptation of large language models for robust and equitable top-k recommendations. *arXiv preprint*, 2025. <https://arxiv.org/abs/2501.04762> (visited on 11/04/2025).
- [26] W. Kim, S. Lim, G. Kim, and S. Choi. Extracting implicit user preferences in conversational recommender systems using large language models. *Mathematics*, 13(2):221, 2025. <https://www.mdpi.com/2227-7390/13/2/221> (visited on 03/02/2025).
- [27] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020. <https://openreview.net/forum?id=H1eA7AEtvS> (visited on 01/01/2025).
- [28] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of the 34th Inter-*

- national Conference on Neural Information Processing Systems, NIPS '20*, Red Hook, NY, USA, 2020. Curran Associates Inc.
- [29] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019. <https://arxiv.org/abs/1907.11692> (visited on 04/01/2025).
- [30] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *arXiv preprint arXiv:1603.09320*, 2018. <https://arxiv.org/abs/1603.09320> (visited on 13/12/2024).
- [31] S. Minaee, T. Mikolov, N. Nikzad, M. A. Chenaghlu, R. Socher, X. Amatriain, and J. Gao. Large language models: A survey. *ArXiv*, abs/2402.06196, 2024. <https://api.semanticscholar.org/CorpusID:267617032> (visited on 12/12/2024).
- [32] N. H. Motlagh, P. Toivonen, M. A. Zaidan, E. Lagerspetz, E. Peltonen, E. Gilman, P. Nurmi, and S. Tarkoma. Monitoring social distancing in smart spaces using infrastructure-based sensors. In *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*, pages 124–129. IEEE, 2021.
- [33] N. H. Motlagh, M. Zaidan, P. Fung, S. Varjonen, A. Rebeiro-Hargrave, P. Nurmi, and S. Tarkoma. Poster: IoT-Based Indoor Air Quality Monitoring for Health Risk Assessment and Well-Being. In *Proceedings of the 23rd Annual International Conference on Mobile Systems, Applications and Services (ACM MobiSys 2025)*, June 2025.
- [34] N. H. Motlagh, M. A. Zaidan, E. Lagerspetz, S. Varjonen, J. Toivonen, J. Mineraud, A. Rebeiro-Hargrave, M. Siekkinen, T. Hussein, P. Nurmi, et al. Indoor air quality monitoring using infrastructure-based motion detectors. In *2019 IEEE*

- 17th International Conference on Industrial Informatics (INDIN)*, volume 1, pages 902–907. IEEE, 2019.
- [35] N. H. Motlagh, M. A. Zaidan, L. Lovén, P. L. Fung, T. Hänninen, R. Morabito, P. Nurmi, and S. Tarkoma. Digital twins for smart spaces-beyond iot analytics. *IEEE internet of things journal*, 11(1):573–583, 2023.
- [36] NVIDIA Corporation. *Metrics - NVIDIA NIM LLMs Benchmarking*, 2025. <https://docs.nvidia.com/nim/benchmarking/llm/latest/metrics.html> (visited on 23/03/2025).
- [37] OpenAI. Gpt-4 technical report. Technical report, OpenAI, 2024. <https://cdn.openai.com/papers/gpt-4.pdf> (visited on 14/12/2024).
- [38] A. Pal and M. Sankarasubbu. Gemini goes to med school: Exploring the capabilities of multimodal large language models on medical challenge problems hallucinations. *arXiv preprint arXiv:2402.07023*, 2024. <https://arxiv.org/abs/2402.07023> (visited on 22/01/2025).
- [39] R. Priyadarshini and R. Mehra. Quantitative review of occupancy detection technologies. *International Journal of Radio Frequency Design*, 1:1–19, 01 2015.
- [40] P. Pu and L. Chen. A user-centric evaluation framework for recommender systems. In *Proceedings of the Fifth ACM Conference on Recommender Systems (RecSys’10)*, pages 157–164. ACM, 2011.
- [41] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.
- [42] D. Rein, B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, and S. R. Bowman. Gpqa: A graduate-level google-proof qa benchmark. *arXiv preprint arXiv:2311.12022*, 2023. <https://arxiv.org/abs/2311.12022> (visited on 05/11/2024).

- [43] B. RoziÅšre, J. Gehring, F. Gloeckle, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2024. <https://arxiv.org/abs/2308.12950> (visited on 19/10/2024).
- [44] A. Saleh, P. K. Donta, R. Morabito, N. H. Motlagh, and L. Lovén. Follow-Me AI: Energy-efficient user interaction with smart environments. *IEEE Pervasive Computing*, 2025. <https://arXiv.org/abs/2404.12486> (visited on 24/04/2025).
- [45] A. Saleh, S. Tarkoma, P. K. Donta, N. H. Motlagh, S. Dustdar, S. Pirttikangas, and L. Lovén. Usercentrix: An agentic memory-augmented ai framework for smart spaces. *arXiv preprint arXiv:2505.00472*, 2025. <https://arxiv.org/abs/2505.00472> (visited on 04/05/2025).
- [46] Z. Sprague, X. Ye, K. Bostrom, S. Chaudhuri, and G. Durrett. Musr: Testing the limits of chain-of-thought with multistep soft reasoning. *arXiv preprint arXiv:2310.16049*, 2024. <https://arxiv.org/abs/2310.16049> (visited on 10/11/2024).
- [47] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864v5*, 2023. <https://arxiv.org/abs/2104.09864v5> (visited on 28/11/2024).
- [48] M. Suzgun, N. Scales, N. SchÅœrli, S. Gehrmann, Y. Tay, H. W. Chung, A. Chowdhery, Q. V. Le, E. H. Chi, D. Zhou, and J. Wei. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022. <https://arxiv.org/abs/2210.09261> (visited on 28/10/2024).
- [49] G. Team, R. Anil, et al. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2024. <https://arxiv.org/abs/2312.11805> (visited on 22/01/2025).

- [50] G. Team, P. Georgiev, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024. <https://arxiv.org/abs/2403.05530> (visited on 22/01/2025).
- [51] O. Topsakal and T. C. Akinci. Creating large language model applications utilizing langchain: A primer on developing llm apps fast. In *Proceedings of the International Conference on Applied Engineering and Natural Sciences (ICAENS)*, volume 1, pages 1050–1056. International Conference on Applied Engineering and Natural Sciences, July 2023.
- [52] H. Touvron, T. Lavril, G. Izacard, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. <https://arxiv.org/abs/2302.13971> (visited on 19/10/2024).
- [53] H. Touvron, T. Lavril, G. Izacard, et al. Llama 3: Scaling open foundation language models to 405 billion parameters. *arXiv preprint arXiv:2401.02457*, 2024. <https://arxiv.org/abs/2401.02457> (visited on 20/10/2024).
- [54] H. Touvron, L. Martin, K. Stone, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023. <https://arxiv.org/abs/2307.09288> (visited on 18/10/2024).
- [55] A. Varol, N. H. Motlagh, M. Leino, S. Tarkoma, and J. Virkki. Creation of ai-driven smart spaces for enhanced indoor environments—a survey. *arXiv preprint arXiv:2412.14708*, 2024. <https://arxiv.org/abs/2412.14708> (visited on 02/02/2025).
- [56] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Neural Information Processing Systems*, 2017. <https://api.semanticscholar.org/CorpusID:13756489> (visited on 23/11/2024).

- [57] B. Wang. *Real-Time Data Informed Traffic Analytics Framework Powered by Large Language Model (LLM)*. PhD dissertation, University of Washington, 2024.
- [58] L. Wang et al. Generative ai for rf sensing in iot systems. *IEEE Transactions on Signal Processing*, 2024. <https://arxiv.org/abs/2407.07506> (visited on 15/12/2024).
- [59] S. Wang, W. Fan, Y. Feng, and X. Ma. Knowledge graph retrieval-augmented generation for llm-based recommendation. *arXiv preprint*, 2025. <https://arxiv.org/abs/2501.02226> (visited on 02/02/2025).
- [60] Y. Wang, X. Ma, G. Zhang, Y. Ni, A. Chandra, S. Guo, W. Ren, A. Arulraj, X. He, Z. Jiang, T. Li, M. Ku, K. Wang, A. Zhuang, R. Fan, X. Yue, and W. Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*, 2024. <https://arxiv.org/abs/2406.01574> (visited on 07/11/2024).
- [61] Y. Wang and Y. Zhao. Gemini in reasoning: Unveiling commonsense in multi-modal large language models. *arXiv preprint arXiv:2312.17661*, 2023. <https://arxiv.org/abs/2312.17661> (visited on 22/01/2025).
- [62] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in Neural Information Processing Systems*, 32:5753–5763, 2019.
- [63] G. Yenduri et al. Gpt (generative pre-trained transformer)â comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions. *IEEE Access*, 12:54608–54627, 2024.
- [64] P. Yu, Z. Xu, J. Wang, and X. Xu. The application of large language models in recommendation systems. *arXiv preprint*, 2025. <https://arxiv.org/abs/2501.02178> (visited on 02/02/2025).

- [65] H. Zhang, Y. Shen, L. Wang, C. Shi, S. Du, Y. Tao, and Y. Shen. Comparative analysis of large language models for context-aware code completion using safim framework. *arXiv preprint arXiv:2502.15243*, 2025. <https://arxiv.org/abs/2502.15243> (visited on 22/03/2025).
- [66] M. Zhang and J. Li. A commentary of gpt-3 in mit technology review 2021. *Fundamental Research*, 1(6):831–833, 2021.
- [67] R. Zhang, J. Han, C. Liu, et al. Llama-adapter: Efficient fine-tuning of large language models with zero-initialized attention. In *International Conference on Learning Representations (ICLR)*, 2024.
- [68] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*, 2020. <https://arxiv.org/abs/1904.09675> (visited on 04/01/2025).
- [69] T. Zhang, S. G. Patil, N. Jain, S. Shen, M. Zaharia, I. Stoica, and J. E. Gonzalez. RAFT: Adapting language model to domain specific RAG. In *First Conference on Language Modeling*, 2024. <https://openreview.net/forum?id=rzQGHXNReU> (visited on 30/11/2024).
- [70] X. Zhang, Y. Song, Y. Wang, S. Tang, X. Li, Z. Zeng, Z. Wu, W. Ye, W. Xu, Y. Zhang, X. Dai, S. Zhang, and Q. Wen. Raglab: A modular and research-oriented unified framework for retrieval-augmented generation. *arXiv, abs/2408.11381*, 2024. <https://arxiv.org/abs/2408.11381>.
- [71] W. X. Zhao, K. Zhou, J. Li, T. Tang, et al. A survey of large language models: Transforming ai research. *arXiv preprint arXiv:2303.18223*, 2023. <https://arxiv.org/abs/2303.18223> (visited on 03/11/2024).
- [72] Z. Zhao, W. Fan, J. Li, Y. Liu, X. Mei, Y. Wang, Z. Wen, F. Wang, X. Zhao, J. Tang, and Q. Li. Recommender systems in the era of large language models

- (llms). *IEEE Transactions on Knowledge and Data Engineering*, xx(xx):1–50, 2023.
- [73] H. Zhou, H. Gu, and X. Liu. The efficiency vs. accuracy trade-off: Optimizing rag-enhanced llm recommender systems using multi-head early exit. *arXiv preprint*, 2025. <https://arxiv.org/abs/2501.02173> (visited on 07/02/2025).
- [74] J. Zhou, T. Lu, S. Mishra, S. Brahma, S. Basu, Y. Luan, D. Zhou, and L. Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023. <https://arxiv.org/abs/2311.07911> (visited on 11/12/2024).
- [75] A. Zubiaga. Natural language processing in the era of large language models. *Frontiers in artificial intelligence*, 6, 2024. DOI: 10.3389/frai.2023.1350306.

Appendix A. User-feedback questionnaire

The online survey contained the following items (single choice unless stated).

1. Age bracket
2. Sensitivities (tick-all-that-apply)
3. Environmental triggers during work/study (tick-all-that-apply)
4. Topics of the questions asked to the system (tick-all-that-apply)
5. Q5: "Did the suggested rooms meet your expectations?" (1 = none, 10 = all)
6. Q6: "Were there any rooms you think should have been included?" (1 = many, 10 = none)
7. Q7: "Did it suggest unsuitable rooms?" (1 = many, 10 = none)
8. Q8: "Overall performance of the system" (1 = poor, 10 = excellent)
9. Q9: "Ease of use" (1 = unclear, 10 = easy)
10. Q10: "Do the recommended locations make sense to you?" (1 = unclear, 10 = easy)
11. Q11: "Would you recommend the system to others?" (1 = never, 10 = definitely)
12. Q12: "Number of times the app was tried" (1 = once – 5 = more)
13. Q13: Free-form comments