



MAsterFlow: Cadeia de sistemas multiagente inteligentes para a criação de pipelines de aprendizagem automática e aprendizagem federada

RAFAEL ALEXANDRE SILVA BARBARROXA

julho de 2025



MAsterFlow: Chain of intelligent multi-agent systems to create pipelines for machine learning and federated learning

Rafael Alexandre Silva Barbarroxa

Student nº: 1181498

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science, Specialization Area of Artificial Intelligence

Supervisor: Luis Gomes

Co-Supervisor: Zita Vale

Evaluation Committee:

President:

Doctor Carlos Fernando da Silva Ramos; Full Professor, Polytechnic of Porto - School of Engineering

Members:

Doctor Luiz Felipe Rocha de Faria; Associate Professor, Polytechnic of Porto - School of Engineering

Doctor Luis Filipe de Oliveira Gomes; Assistant Professor, Polytechnic of Porto - School of Engineering

Porto, July 2025

DECLARAÇÃO DE INTEGRIDADE

DECLARAÇÃO DE INTEGRIDADE

Declaro ter conduzido este trabalho académico com integridade. Não plagiei ou apliquei qualquer forma de uso indevido de informações ou falsificação de resultados ao longo do processo que levou à sua elaboração.

Declaro que o trabalho apresentado neste documento é original e de minha autoria, não tendo sido utilizado anteriormente para nenhum outro fim.

Declaro ainda que tenho pleno conhecimento do Código de Conduta Ética do P.PORTO.

ISEP, Porto, 14 de junho de 2025

Rafael Barbarosa

Abstract

The growing demand for secure, privacy-preserving AI solutions is particularly noticeable in domains such as renewable energy or healthcare, where sensitive data is involved. As society continues to transition to AI-driven systems, the need for decentralized machine learning systems has become increasingly evident. Traditional machine learning methods rely heavily on centralized datasets, often compromising privacy and security.

Although federated learning addresses these concerns by enabling decentralized model training while maintaining data privacy, several challenges remain. These include the complexity of creating, configuring, and managing federated learning models, particularly when dealing with a large number of clients and different configurations. As federated learning grows in popularity, there is also a need for more automated solutions that can simplify this process for users with varying levels of expertise.

This dissertation presents MASTerFlow, a novel system that combines multi-agent systems with large language models to intelligently create machine learning models and federated learning federations. By integrating LLMs and Retrieval-Augmented Generation, MASTerFlow provides an efficient way to configure, execute, and analyze FL training simulations. The system streamlines the process by allowing users to interact with intelligent agents that manage different tasks, such as configuring machine learning models, setting up federated learning simulations, and analyzing training logs.

MASTerFlow is designed with a user-friendly web-based interface that allows users to engage with the system's agents and configure simulations according to their needs. Through extensive case studies, the dissertation benchmarks various multi-agent frameworks and demonstrates the effectiveness of combining multi-agent systems and large language models to automate the creation of machine learning and federated learning pipelines. The results indicate that MASTerFlow provides a more accessible, secure, and adaptable alternative to traditional machine learning methods, offering improved efficiency and usability for AI development.

Keywords: Federated Learning, Large Language Models, Machine Learning Pipelines, Multi-Agent Systems, Retrieval-Augmented Generation

Resumo

A crescente e contínua procura por soluções de inteligência artificial seguras e que preservem a privacidade é particularmente notável em domínios, que envolvem dados sensíveis, tais como a energia e a saúde,. À medida que a sociedade continua a transitar para sistemas impulsionados por inteligência artificial, a necessidade de sistemas de aprendizagem automática descentralizados torna-se cada vez mais importante. Métodos tradicionais de aprendizagem automática dependem fortemente de conjuntos de dados centralizados, muitas vezes comprometendo a privacidade e a segurança.

Embora a aprendizagem federada¹ resolva algumas dessas preocupações, permitindo o treino descentralizado de modelos enquanto mantém a privacidade dos dados, vários desafios permanecem, incluindo a complexidade de criar, configurar e gerir modelos de aprendizagem federada, particularmente aquando da existência de um elevado número de clientes e diferentes configurações. À medida que a aprendizagem federada cresce em popularidade, há também uma necessidade de soluções mais automatizadas que possam simplificar este processo para utilizadores com diferentes níveis de proficiência.

Esta dissertação propõe o MASTerFlow, um sistema inovador que combina sistemas multiagente de *large language models* para criar, de forma inteligente, modelos de aprendizagem automática e federações de aprendizagem. Adicionalmente, ao integrar *large language models* e geração aumentada via recuperação, o MASTerFlow oferece uma maneira eficiente de configurar, executar e analisar simulações de treino de aprendizagem federada. O sistema simplifica o processo permitindo que os utilizadores interajam com agentes inteligentes responsáveis por tarefas diferentes, como configurar modelos de aprendizagem automática, configurar simulações de aprendizagem federada e analisar *logs* de treino.

O MASTerFlow foi projetado com uma interface *web* que permite aos utilizadores interagir com os agentes do sistema e configurar simulações de acordo com suas necessidades. Através de estudos de caso extensivos, a dissertação avalia diferentes *frameworks* multiagente e demonstra a eficácia de combinar sistemas multiagente e *large language models* para automatizar a criação de *pipelines* de aprendizagem automática e aprendizagem federada. Os resultados indicam que o MASTerFlow oferece uma alternativa mais acessível, segura e adaptável comparada aos métodos tradicionais de aprendizagem automática, proporcionando uma maior eficiência e usabilidade para o desenvolvimento de inteligência artificial.

Palavras-chave: Aprendizagem Federada, *Large Language Models*, Pipelines de Aprendizagem Automática, *Retrieval Augmented Generation*, Sistemas Multiagentes

¹ Federated learning

Acknowledgments

I would like to thank GECAD for providing me with the opportunity and resources to develop and write this thesis.

I would like to thank my supervisors, Zita Vale and Luis Gomes, for their support and guidance during the development of the project and the writing of the dissertation.

And, finally, I would like to thank all my family and friends that helped me and supported me during this period.

Contents

1	Introduction	1
1.1	Contextualization	1
1.2	Problem definition	2
1.3	Objectives	2
1.4	Scientific publication	3
1.5	Thesis organization	4
2	State of the art	5
2.1	Large Language Models	5
2.2	Multi-agent systems	10
2.3	Multi-Agent Systems with Large Language Models	13
2.3.1	AutoGen	14
2.3.2	CrewAI	15
2.3.3	TaskWeaver	16
2.3.4	Other works	17
2.4	Retrieval Augmented Generation	18
2.5	Machine learning algorithms for forecasting	22
2.6	Federated Learning	26
3	Methods and tools	29
3.1	Models and frameworks	29
3.1.1	Large language models	29
3.1.2	Multi-agent systems	30
3.1.3	Machine learning	31
3.1.4	Federated learning	35
3.1.5	RAG framework	35
3.1.6	Python frameworks and libraries	36
3.2	Datasets	37
3.2.1	Energy consumption dataset	37
3.2.2	Community dataset	37
3.3	Data protection, security analysis, and ethical aspects	38
3.3.1	Data protection	38
3.3.2	Ethical aspects	39
4	MAStErFlow	41
4.1	Architecture	41
4.2	System model	42
4.2.1	ML model creation component	43
4.2.2	Integration of ML model in Flower Client component	49

4.2.3	FL federation creation component	52
4.2.4	FL training logs analyzer	58
4.2.5	User interface	62
5	Case Studies	67
5.1	Case Study 1 - Multi-agent system frameworks testing	67
5.2	Case Study 2 - ML model creation	71
5.2.1	Version 1 - Configuration file and zero-shot prompting	73
5.2.2	Version 2 - Template and multi-agent system	75
5.2.3	Version 3 - LLM dialog and multi-agent system	76
5.3	Case Study 3 - FL federation creation.....	78
5.3.1	User expertise determination	78
5.3.2	LLM dialog and multi-agent system.....	79
5.4	Case Study 4 - FL training logs analyzer	82
6	Conclusions and Future Work.....	89
	References	93
	Annex	119

List of Figures

Figure 1. Transformer model architecture (Vaswani et al., 2023)	7
Figure 2. Document Embedding in RAG	18
Figure 3. LLM Inference with RAG	19
Figure 4. Forward propagation and backward propagation (Gezici & Sengul, 2023)	23
Figure 5. FL process representation	26
Figure 6. MASTerFlow components	42
Figure 7. Dialog LLM system message – part one	43
Figure 8. Dialog LLM system message – part two	44
Figure 9. Confirmation LLM system prompt	44
Figure 10. Function to replace the string placeholders in the prompt template.....	45
Figure 11. Dataset percentage information in template to prompt function.....	45
Figure 12. Full prompt placeholder in the template to prompt function	46
Figure 13. Example prompt of ML model making	46
Figure 14. ML model creation component.....	49
Figure 15. Integration component diagram	50
Figure 16. Framework analyzing LLM system message	51
Figure 17. Automated FL federation component.....	53
Figure 18 User expertise determination LLM system prompt	54
Figure 19. Confirm FL configuration LLM system prompt.....	55
Figure 20. FL configuration file prompt base parameters.....	55
Figure 21. FL model-making tool usage instructions.....	55
Figure 22. FL model-making function call construction	56
Figure 23. FL training logs analyzer component.....	58
Figure 24. Example of Flower Framework logs	59
Figure 25. Embedchain app configuration - LLM parameters.....	60
Figure 26. Embedchain app configuration – embedder parameters	61
Figure 27. Embedchain app configuration – chunk parameters	61
Figure 28. Embedchain app configuration – vector database configuration.....	61
Figure 29. Log analyzer introduction prompt.....	62
Figure 30. MASTerFlow UI layout.....	63
Figure 31. UI folder contents visualization.....	63
Figure 32. Current multi-agent system information in UI.....	63
Figure 33. UI component stepper	64
Figure 34. UI chat box.....	64
Figure 35. UI current code component	65
Figure 36. MASTerFlow Terms and Conditions dialog	66
Figure 37. Communication between system components	66
Figure 38. Case study 1 versions	72
Figure 39. User expertise determination example.....	79

Figure 40. Example of FL making dialog with the user.....	80
Figure 41. Results from 200 rounds of FL training	82
Figure 42. FL logs analyzer dialog – part 1	83
Figure 43. FL logs analyzer dialog – part 2	84
Figure 44. FL training logs - initialization and client registration information.....	84
Figure 45. FL training logs – second round of training.....	85
Figure 46 FL training logs – termination information	85
Figure 47. FL training logs – summary section.....	85
Figure 48. FL logs analyzer dialog – part 3	86
Figure 49. FL logs analyzer dialog – part 4	86
Figure 50. FL logs analyzer dialog – part 5	87
Figure 51. An example of generated FL training logs.....	121
Figure 52. Web Meta-ERM public case studies display	121

List of Tables

Table 1. LLMs tested during the development.....	29
Table 2. Python libraries and frameworks	36
Table 3. Energy consumption dataset column description.....	37
Table 4. Community dataset column description	37
Table 5. AutoGen model-making group chat system prompts	47
Table 6. AutoGen integration group chat.....	51
Table 7. FL model-making group chat system prompts	57
Table 8. Relationship between objectives and case studies	67
Table 9. Case Study 1 results	70
Table 10. Case study 2 version 1 run percentages.....	73
Table 11. Case study 2 version 1 ML models created without error	73
Table 12. Case study 2 version 1 total runs.....	74
Table 13. Case Study model making group chats.....	75
Table 14. Case study 1 version 2 run percentages.....	75
Table 15. Metrics obtained by each group chat.....	76
Table 16. Configuration file confirmation LLM test results.....	77
Table 17. FeDis case study results.....	78
Table 18. FL training results	81

Acronyms

List of Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
AR	Augmented Reality
AV	Autonomous Vehicle
BDI	Belief-Desire-Intention
CDP	Central Differential Privacy
CEC	Constant Error Carousel
CoT	Chain-of-Thought Prompting
CNN	Convolutional Neural Network
CPT	Continual Pre-training
CRUD	Create, Read, Update, Delete
DP	Differential Privacy
DL	Deep Learning
DT	Decision Trees
ET	Extra Tree
EVS	Explain Variance Score
FJSP	Flexible Job Shop Scheduling Problem
FL	Federated Learning
GBM	Gradient Boosting Regression
GDPR	General Data Protection Regulation
GHI	Global Horizontal Irradiance
Non-IID	Non-Independent and Identically Distributed Data
LDP	Local Differential Privacy
LR	Linear Regression
LoRA	Low-Rank Adaptation
MAPE	Mean Absolute Percentage Error
MAE	Mean Absolute Error
MDR	Multi-Agent Data Retrieval
MedAE	Median Absolute Error
MER	Multi-Agent Evaluation and Reflection
MCP	Multi-Agent Collaborative Planning
MLE	Maximum Likelihood Estimation
ML	Machine Learning
MLP	Multilayer Perceptron
NN	Neural Network
NLP	Natural Language Processing
NPC	Non-Player Character
PCA	Principal Component Analysis
PEAK	Python-based Framework for Heterogeneous Agent Communities
PLMs	Pre-trained Language Models
PR	Polynomial Regression
PSP	Privacy-Aware Service Placement
RAG	Retrieval Augmented Generation
RF	Random Forest
RFR	Random Forest Regression
RL	Reinforcement Learning
RMSE	Root Mean Squared Error
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SCM	Supply Chain Management
SHAP	Shapley Additive Explanations
SVM	Support Vector Machine

TPU	Tensor Processing Unit
VLMs	Visual Language Models
k-NN	k-Nearest Neighbor
XAI	Explainable Artificial Intelligence

1 Introduction

This chapter introduces the context of the research, highlighting the significance of the problem at hand and the motivations driving this study. The core problem is identified, and the specific objectives of the research are outlined, providing a roadmap for the development of the proposed system. In addition, the scientific contributions of the dissertation are described, namely the writing of the scientific articles. Finally, this chapter presents the structure of the thesis, detailing the organization of the subsequent chapters and guiding the reader through the document.

1.1 Contextualization

Federated Learning (FL) has emerged as a solution for distributed machine learning (ML), particularly in environments where data privacy and security are of high priority. Unlike traditional centralized ML approaches, FL allows data to remain local on devices while enabling the training of ML models through the aggregation of model updates, ensuring privacy by never exposing raw data (C. Zhang et al., 2021).

However, FL is a complex process that requires the coordination of multiple devices or clients, efficient management of training rounds, and careful configuration of the FL system. Multi-Agent Systems (MAS) provide an effective solution to automate and optimize the management of FL processes. To this end, this dissertation's project utilizes MAS to enable the automation of FL simulations, facilitating tasks such as model configuration, log analysis, and user interaction. By delegating different responsibilities to specialized agents, the system becomes scalable, flexible, and capable of handling complex FL tasks without manual intervention.

While FL helps solve privacy concerns, it introduces new challenges in terms of training efficiency and the complexity of managing a large number of clients and model updates. Recent advances in hardware, particularly Graphical Processing Units (GPUs) (F. H. Khan et al., 2021), have enabled researchers to create more powerful machine learning models at larger scales. Additionally, the introduction of the Transformer architecture in 2017 revolutionized the capacity of Large Language Models (LLMs), allowing these models to process and generate

natural language more effectively. This evolution has opened new avenues for tackling challenges in FL, as LLMs, particularly those based on architectures like GPT-4, offer a promising solution. LLMs have proven themselves as capable aids for learning (Ghimire & Edwards, 2024). The better-performing models can accurately answer questions of a general nature (Dam et al., 2024), and if trained for it, can answer questions specific to an area, such as coding (X. Zhang et al., 2024). To further increase the knowledge of a subject the model has, methodologies such as Retrieval Augmented Generation (RAG) can be applied to provide the LLM with document data that is relevant to the questions asked by the users (Ke et al., 2024). LLMs can facilitate more efficient natural language interaction with users, enabling them to configure FL simulations, understand training logs, and obtain real-time answers to their queries related to the training process.

This has resulted in an increased usage and integration of artificial intelligence (AI) across various fields of study (W. X. Zhao et al., 2024). However, a lot of potential users interested in AI applications still have little to no experience in the field (Draxler et al., 2023).

The motivation behind this work is to bridge this gap by integrating LLMs and MAS to assist users in creating ML models, configuring FL simulations, and analyzing the results. Through a combination of automation, multi-agent collaboration, and natural language understanding, the system simplifies the process of federated learning, making it more accessible while maintaining the privacy and security of the data involved.

1.2 Problem definition

Although the usage of ML and FL is becoming more widespread, the development of such models typically requires programming knowledge, limiting their use and understanding by people from non-technical fields, preventing them from benefiting from the advantages that ML and FL models bring in certain scenarios.

To address this, this dissertation attempts to determine how capable multi-agent systems (MAS), powered by LLMs, are in creating ML models and FL federations while guiding users in their understanding and following the specified parameters. This also raises a secondary problem of understanding which of the various MAS frameworks would be the most adequate to create the LLM-using agents to handle the tasks of creating the ML and FL pipelines. Additionally, the capacity of the LLMs, utilizing RAG, in analyzing training logs and answering user questions about those accurately is tested in the dissertation.

1.3 Objectives

To address the issues raised in the context and problem definition, the platform named MASTerFlow was devised. This system will leverage various MAS, in a chain configuration, whose agents are powered by LLMs to empower their reasoning abilities. These MAS will be

responsible for working with the user to create ML and FL models. Additionally, an LLM will be equipped with RAG to be able to answer user questions regarding the logs generated by the FL training process.

Finally, a User Interface (UI) will be implemented to allow users to interact with MASTerFlow's agents via a chat box in a web interface.

Considering the context and problem definition, the following objectives outline the goals intended to be achieved during the development of the system discussed in this dissertation:

- **OB1:** Determining the state-of-the-art regarding MAS frameworks, and the most adequate to be used for the implementation of the chain of MAS.
- **OB2:** Implementation of the MAS powered by LLM agents to assist users in creating ML models.
- **OB3:** Implementation of the MAS powered by LLM agents to assist users in creating a FL Federation in the Flower framework.
- **OB4:** Development of an interactive system that employs an LLM and RAG to allow users to perform FL training log analysis.
- **OB5:** Test and validate the solution using realistic datasets and scenarios.

1.4 Scientific publication

During the writing of the dissertation, the following scientific articles were published:

- The article (Barbarroxa et al., 2024): "Barbarroxa, R., Ribeiro, B., Gomes, L., & Vale, Z. (2024). Benchmarking AutoGen with different large language models. *2024 IEEE Conference on Artificial Intelligence (CAI)*, 263–264. <https://doi.org/10.1109/CAI59869.2024.00058>";
- The paper (Barbarroxa et al., 2025): "Barbarroxa, R., Gomes, L., & Vale, Z. (2025). Benchmarking Large Language Models for Multi-agent Systems: A Comparative Analysis of AutoGen, CrewAI, and TaskWeaver. In P. Mathieu & F. De la Prieta (Eds.), *Advances in Practical Applications of Agents, Multi-Agent Systems, and Digital Twins: The PAAMS Collection* (pp. 39–48). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-70415-4_4";
- The article (Barbarroxa et al., 2023): "Barbarroxa, R., Silva, J., Gomes, L., Lezama, F., Ribeiro, B., & Vale, Z. (2023). *FeDis: Federated Learning Framework Supported by Distributed Ledger* (pp. 32–41). https://doi.org/10.1007/978-3-031-45155-3_4";
- And the article (Almeida et al., 2023): "Almeida, J., Barbarroxa, R., Lezama, F., Soares, J., Gomes, L., Oliveira, F., & Vale, Z. (2023). *Meta-ERM: Metaheuristic Optimization*

1.5 Thesis organization

This thesis document is structured into the following sections: Section 0 - Introduction provides contextualization, defines the problem being addressed, and outlines the objectives and scientific contributions of the research. It also includes an overview of the document structure.

Section 2 – State of the art, reviews the current state of knowledge in the relevant fields. It covers the concepts of LLMs, MAS, the integration of MAS with LLMs, RAG, ML algorithms for forecasting, and FL.

Section 3 – Method and tools details the models and frameworks considered and employed in the research. It covers the LLMs, MAS, ML models, and the FL framework used. The chapter also explores the Python frameworks and libraries applied in the development of the system. Additionally, the two datasets used for the case study are presented, along with a discussion of data protection, security considerations, and ethical aspects related to the system.

Section 4 – MASterFlow introduces the MasterFlow system, presenting its architecture and the components that make up the system. These include the ML model creation component, the integration of the ML model within the Flower client, the FL federation creation component, the FL training logs analyzer, and the user interface. The design and interaction of these components are discussed in detail.

Section 5 – Case Studies presents the case studies used to evaluate the system and discusses the results obtained. The case studies are linked directly to the objectives defined in Section 1.3, providing insights into how well the system meets the proposed goals.

Finally, Section 6 – Conclusions summarizes the main findings of the research and reflects on the contributions of the dissertation. It also discusses the potential for future work, considering improvements to the system and directions for further research in the field.

2 State of the art

This chapter will go over the recent developments in the areas that will be addressed in this dissertation, namely LLMs, MAS, RAG, ML algorithms and FL. Each subsection will contain an explanation of the subject matter, a presentation of some of the advantages and disadvantages of the concept, the areas of application it can be found in, and concrete examples where the technology or concept was applied.

2.1 Large Language Models

Large Language Models (LLMs) are a specialized version of AI models that are designed to process and generate human-like text (Minaee et al., 2024). These models are trained on vast amounts of data, such as textual datasets representing the Internet (Y. Liu et al., 2024), which requires a lot of computational power, and are capable of performing a large array of Natural Language Processing (NLP) tasks. These models are considered large due to the number of parameters, usually billions, that are part of the models. Parameters are the adjustable elements of the LLMs that are configured during training to optimize performance. By analyzing patterns in the trained data, LLMs can generate text that is coherent and contextually relevant to the query and what was previously said in the conversation. These models are also capable of translating languages (Z. He et al., 2024), text summarization (Supriyono et al., 2024), and chatting with users (Shanahan et al., 2023).

Before the creation of the Transformer architecture, NLP tasks were mostly realized by statistical methods and neural networks (NN) such as recurrent neural networks (RNNs) (Lavanya & Sasikala, 2021) and long short-term memory (LSTM) networks (Anusha et al., 2022). A statistical approach used was n-grams (Banerjee & Pedersen, 2003), which predicted the probability of a word given the sequence of words that came before. For example, bigram models (BEKKERMAN & Allan, 2003) determined the probability of a word based on the immediate predecessor. These types of models were limited by the size of the context they

could capture. Models using Maximum Likelihood Estimation (MLE) (Emezue, 2024) techniques were used to estimate the likelihood of a sequence of words and required a large *corpus* of text to realize the predictions. The neural-based models, such as the RNNs and the LSTMs, were better than the statistical models in handling sequential data but struggled with longer contexts due to the vanishing gradient issue (Noh, 2021)

In 2017, the architecture of LLMs was revolutionized by the introduction of the Transformer architecture in the paper “Attention is all you need” by researchers at Google (Vaswani et al., 2023). Unlike previous models, such as RNNs and LSTMs, transformers can process entire sequences simultaneously, improving training efficiency and the handling of dependencies between words in long sequences. As shown in Figure 1, the Transformer architecture is composed of the following components:

- Self-Attention Mechanism, which allows the model to weigh the importance of the different words in a sequence relative to each other. The computation of attention scores focuses the model on the relevant parts of the sequence, irrespective of position;
- Multi-Head Attention, this element of architecture allows the model to capture various aspects of the relationship between words by performing multiple attention operations in parallel;
- Positional Encoding, which is added to the embeddings of the input to provide information about word position in the sequence, since Transformers process sequences in parallel and not in order;
- Feed-Forward NN that receives the output of the attention layer. Typically composed of two linear transformations with a non-linear activation function in between, such as the rectified linear unit (ReLU) function;
- Residual Connections are wrapped around the attention and the feed-forward NN layers to improve the training of the model by adding the input of the layer back to the transformed output to preserve information;
- Layer Normalization is applied after residual connections, standardizing layer activations which stabilizes the training process.

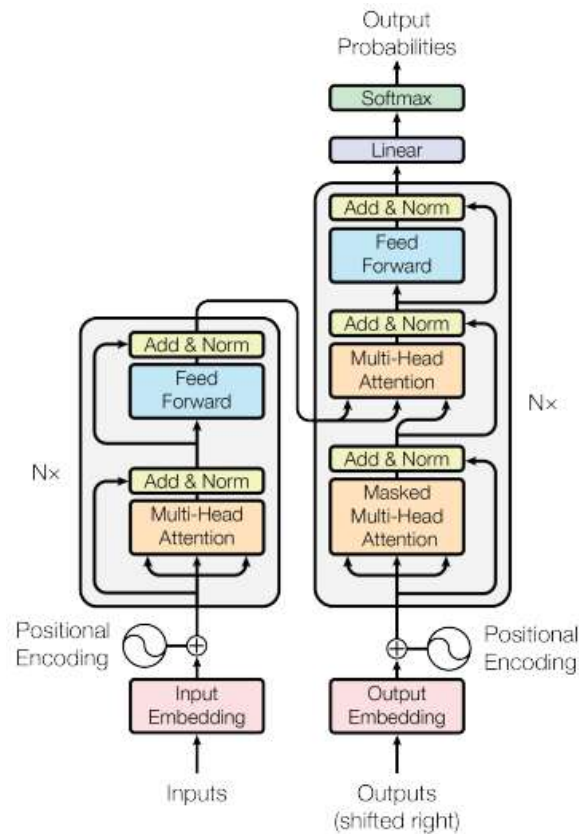


Figure 1. Transformer model architecture (Vaswani et al., 2023)

Additionally, the Transformer model consists of an encoder-decoder structure. The encoder is responsible for processing the input sequence and generating a set of representations. It is composed of multiple layers that contain the multi-head attention component followed by a feed-forward NN element. On the other hand, the decoder generates the output sequence based on the encoder's representations and previous outputs. It consists of multiple layers that contain a multi-head self-attention mechanism, multi-head attention mechanism, and a feed-forward NN.

Training LLMs is a rather complex endeavor that requires a high degree of computational resources, vast datasets, and specialized technical expertise (Hadi et al., 2023). Due to the models containing billions of parameters, high-performance hardware accelerators such as Graphics Processing Units (GPUs) or Tensor Processing Units (TPUs) are essential to handle all the processing needs. The vast amount of data is necessary to ensure that the model can perform a variety of language tasks. Finally, technical expertise is necessary to design model architectures, implement efficient training algorithms, manage large-scale data pipelines, and optimize the time and resources spent on the training process.

To refine the knowledge of a pre-trained LLM to a specific task or domain, fine-tuning can be utilized via training the model on a smaller and more specialized dataset. Methods such as Low-Rank Adaptation (LoRA) (Z. Han et al., 2024) allow for fine-tuning with reduced computational resource usage by adjusting only a subset of the model's parameters. However, it poses

challenges such as data availability, overwriting general knowledge (catastrophic forgetting), and managing computational costs.

An alternative to fine-tuning is the use of prompt engineering (B. Chen et al., 2024). It consists of crafting input prompts to guide the LLMs in generating the desired output. Instead of utilizing zero-shot learning, where the model receives a prompt without prior examples, the following techniques can be applied:

- Few-shot learning, where the LLM receives a few examples within the prompt to guide the response;
- Chain-of-Thought (CoT) Prompting, where the user encourages the model to generate intermediate reasoning steps for arriving at a conclusion, which enhances the LLM's problem-solving capabilities

Thus, prompt engineering can be considered an important technique to optimize LLM performance and accuracy without the need to spend time and computational resources on expensive training. Another technique, Retrieval Augmented Generation, is presented in Section 2.4.

For works that utilize LLMs, in (Long et al., 2023), the GPT-3 model is leveraged to assist STEM (Science, Technology, Engineering, and Mathematics) experts in crafting engaging hooks for educational tweets (“tweetorials”) by supporting the writing process. Similarly, article (Ibrahim & Al, 2021) introduces FunSearch, where a model from the PaLM2 family, named Codey, is paired with a systematic evaluator to minimize hallucinations while addressing the cap set problem, a challenge in the field of extremal combinatorics. In (S. Xu & Zhang, 2023), the authors propose using LLMs, specifically GPT-3.5-Turbo, to simulate real students from diverse backgrounds. This approach aims to enhance education quality by enabling researchers to better understand learning processes and design more effective teaching methods. Additionally, the Fabric project (Miessler, 2024/2024) extends the content-generation capabilities of LLMs by providing users with detailed prompts, called Patterns, designed to address various use cases. Fabric also allows users to create custom prompts, host them on a server (Mill) for access by client-side applications (Looms), and combine Patterns into more complex workflows (Stitches) for advanced applications.

The work presented in (Värtinen et al., 2024) proposes a fine-tuned GPT-2 model, named Quest-GPT-2, for automated quest generation in role-playing games. The article (Y. Xu et al., 2024) proposes a tuning-free framework for engaging LLMs in communication games, incomplete information games that depend heavily on natural language interactions. Also in the gaming area, the paper (Volum et al., 2022), utilizes an LLM named OpenAI Codex for dynamic, free-form interactions between players and Non-Player Characters (NPC). In addition to conversation, an agent utilizing Codex is also able to dynamically modify the game environment via code generation. FinGPT is proposed in (H. Yang, Liu, et al., 2023), an open-source LLM for the finance sector, aiming to democratize Internet-scale financial data. The model has an automatic data curation pipeline, which enables continuous updates, and a LoRA for efficient fine-tuning without excessive computation costs. Alternatively, article (S. Wu et al., 2023)

presents BloombergGPT, a closed-source LLM comprised of 50 billion parameters, specifically designed for financial applications. It was trained on both Bloomberg's financial data and general-purpose datasets.

In (Guha et al., 2023), LegalBench is presented, a collaboratively constructed legal reasoning benchmark consisting of 162 tasks across six types of legal reasoning to assess LLMs' effectiveness in legal reasoning.

SteloCoder is presented in (Pan et al., 2023), a decoder-only StarCoder-based LLM designed for multi-programming language to Python code translation. The model supports translation from C++, C#, JavaScript, Java, and PHP to Python translation without requiring explicit specification of the input language.

In (Jain et al., 2023), an LLM-assisted code cleaning pipeline to improve natural language to code generation is proposed, enhancing code quality, structure, and readability. Unlike prior work focused on dataset size and correctness, the proposed approach renames variables, modularizes complex functions, and inserts natural-language plans via LLM transformations.

The article (Mandal et al., 2023) proposes SpecSyn, a framework that utilizes LLMs to automatically synthesize software specifications from natural language sources. The model attempts to address the lack of a centralized and structured repository for software documentation which might happen as software systems grow in complexity and specifications are often dispersed across manuals, code comments, and online discussions.

SelfEvolve is presented in (S. Jiang, Wang, et al., 2023), a novel two-step code evolution framework that enhances LLM-based code generation by treating LLMs as both knowledge providers and self-reflective programmers. The framework extracts relevant knowledge directly from input prompts and generates intermediate code. Afterwards, it uses LLMs to act as expert debuggers, refining the generated code based on interpreter error messages rather than predefined test cases.

Paper (D. Guo et al., 2024) introduces DeepSeek-Coder, a revolutionary series of open-source code models ranging from 1.3B to 33B parameters, designed for software development. These models were trained from scratch on 2 trillion tokens, utilizing a high-quality project-level code corpus and incorporating a fill-in-the-blank task within 16k context windows to enhance code generation and infilling capabilities. Extensive evaluations demonstrate that DeepSeek-Coder achieves state-of-the-art performance among open-source models and even outperforms closed-source alternatives like Codex and GPT-3.5 across multiple coding benchmarks. The models are under a permissive license allowing for both research and unrestricted commercial usage.

2.2 Multi-agent systems

Multi-Agent Systems (MAS) are characterized by multiple autonomous agents interacting inside an environment. Key features of the MAS include the autonomy of each agent, the ability to communicate and share knowledge, and the social organization, forming an agent society (Du et al., 2024).

MAS can be organized under various paradigms, ranging from centralized hierarchies to completely decentralized networks (Yan et al., 2025). In a hierarchical organization, agents are arranged in a tree-like command structure with higher-level agents managing or controlling the lower-level ones. This can facilitate clear decision authority and simplify coordination but may suffer from a single point of failure at higher levels. On the other hand, a decentralized structure has no fixed leader – all agents operate more or less as peers, coordinating through distributed protocols. This offers robustness and high adaptability, at the cost of more complex consensus-building when agent numbers grow large. In practice, many MAS opt to adopt hybrid structures that blend hierarchical and decentralized structures, achieving a balance between control and stability.

Effective communication is extremely important in any MAS. Agents must exchange information, requests and responses to cooperate or negotiate. To enable this, standard Agent Communication Languages (ACLs) have been developed. A notable example is the FIPA-ACL (FIPA, 2004), a communication language standardized by the Foundation for Intelligent Physical Agents (FIPA) for agent interoperability. FIPA-ACL provides a framework for agents to exchange messages through standardized formats that support various communication acts such as requests, responses, and declarations, ensuring meaningful interactions between agents from different platforms .

Similarly, Knowledge Query and Manipulation Language (KQML) was a widely used ACL that treats messages as speech acts (for example, inform, query, request) with defined semantics (Kone et al., 2000). Both FIPA-ACL and KQML provide a set of performatives (message types) and an ontology/syntax so that heterogeneous agents can understand each other's intent. These ACLs are grounded in speech-act theory from linguistics and ensure that an agent sending a message like inform (with certain content) conveys a well-defined meaning to the receiving agent.

In addition to the languages, MAS also rely on interaction protocols, which are patterns of message exchange that achieve some collaborative outcome. For instance, the Contract Net Protocol (CNP) is a coordination protocol where the manager and the other agents bid for it (Smith, 1980). The task gets awarded to the best bid. This enables decentralized task allocation through negotiation. Other examples include auction protocols, English and Dutch actions for example, for resource allocation and voting or consensus protocols for group decision-making. Agents often follow these protocols to negotiate agreements or coordinate in a structured way.

Communication in MAS can be either explicit or implicit. Explicit communication means agents send direct messages to one another. Implicit communication relies on observing changes in

the environment or shared context as signals, which is common in swarm systems, where agents coordinate by sensing each other's behaviors. This is visible in multi-robot systems that might only use local sensor information about the location of each other to coordinate movements, without relying on explicit messaging (Ismail & Sariff, 2018).

Each agent in a MAS is an AI entity with its own internal architecture, namely the design of its sensing, reasoning, and acting components. MAS architectures can be from the following categories (Palanca et al., 2023):

- Reactive agents act on stimulus-response rules without an internal symbolic model of the world, excelling at real-time responses and robustness. However, due to their purely reactive nature these agents show some limitations, without an internal world model or planning, they can exhibit only a short-term view which makes it hard to engineer complex, goal-directed behaviors.
- Deliberative agents, on the other hand, maintain an internal state and symbolic representations (beliefs, goals) and use reasoning or planning algorithms to decide actions. These are similar to classical AI planners or knowledge-based systems. While deliberative architectures can handle complex tasks, they tend to be slower and may not react swiftly to dynamic changes.
- Hybrid architectures are a combination of both approaches, and many practical agent systems use a hybrid design. Typically, a hybrid agent has layered control: a reactive layer for quick routine responses, and a higher deliberative layer for long-term planning. The reactive layer might handle low-level behaviors (for example, obstacle avoidance in a robot), while the deliberative layer handles strategic goals. Researchers have argued that neither purely reactive nor purely deliberative designs are sufficient for building intelligent agents, which led to many hybrid designs in the literature (S. Yang et al., 2017). One of the most well known architectures is the Belief-Desire-Intention (BDI) model.

The BDI model was proposed by Bratman (Bratman, 1987) and formalized by Rao and Georgeff (Rao & Georgeff, 1995). In BDI, an agent's decision-making is structured around mental attitudes: beliefs (information the agent has about the world), desires (objectives or motivations), and intentions (current plans or commitments to action). The BDI architecture has been widely adopted in agent programming languages and frameworks, because it provides an intuitive way to model rational decision-making. A BDI agent constantly updates its beliefs based on perceptions, deliberates to choose desires or goals, and from those adopts intentions which it then executes. This approach integrates reasoning and reactivity, the agent can react to changes, updating its beliefs, but also has goal-directed persistence via intentions.

Regardless of specific paradigm, typical components of an agent include:

- Perception modules to sense the environment;
- A knowledge base or memory to store state;

- A reasoning/decision module to select actions,
- Actuators to perform actions in the environment.

A framework utilizing the BDI architecture can be seen in the article (Shang et al., 2024) where a MAS is used for dynamic resource scheduling by integrating BDI with fuzzy logic and time windows. The framework is designed to handle uncertainties in emergency resource allocation, such as data transmission delays and sudden environmental changes.

Agent interactions typically fall into two categories: cooperation and competition (J. Wang et al., 2022). In a cooperative MAS, agents work together to achieve shared objectives or solve a problem that is beyond the capability of any single agent. Achieving cooperation between agents requires organizing agents, by roles or teams, and establishing protocols for information sharing and joint decision-making. Common techniques to divide work include distributed planning, in which agents develop a plan together, consensus algorithms, in which agents iteratively agree on certain values or strategies, and task allocation strategies.

In competitive MAS, each agent pursues its own interests, which can conflict with others. This scenario is common in economics (for example, trading agents) and any system that models autonomous agents with different stakeholders or goals, where negotiation is vital. Via negotiation, self-interested agents communicate proposals and counterproposals to reach mutually acceptable agreements. It can involve two or more agents. Techniques for automated negotiation in MAS include auction-based methods, bargaining models, and more sophisticated argumentation based negotiation where agents exchange not only offers but also justifications or arguments (Carrera & Iglesias, 2015). Another approach in competitive scenarios is to design market mechanisms within the MAS: for instance, allocate resources via a virtual market where agents “pay” or bid, ensuring an efficient outcome under certain game-theoretic conditions.

Multi-agent systems have been applied to many different areas, such as intelligent transport systems (Bretas et al., 2023), multi-robot systems (J. Chen et al., 2025), and smart grids (D. Zhang et al., 2021). Article (Pal et al., 2023) presents a decentralized MAS for solving the Flexible Job Shop Scheduling Problem (FJSP) considering setup and transportation times, offering a more efficient alternative than traditional centralized metaheuristic approaches. (Lim et al., 2013) proposes the usage of a MAS for integrating process planning and production scheduling across multi-site manufacturing facilities. This is achieved through a currency-based agent iterative bidding mechanism to allow agents to coordinate and make optimization decisions. Additionally, a genetic algorithm (GA) is used to optimize the currency values for agent bidding, improving decision-making. Article (Weerasinghe et al., 2016) presents a distributed MAS designed to reduce resource wastage in a Local Area Network (LAN). Using the MaSMT framework, the system includes several agents: the managing agent, file send agent, file receive agent, download agent, and load balancing and dynamic scheduling agent, that communicate with each other to optimize file downloading. In the sector of agriculture, article (Goonatilleke et al., 2019) presents Rice Express, a MAS designed to improve communication within Sri Lanka’s rice production industry and reduce production costs. Utilizing the MaSMT framework,

the system includes agents representing farmers, buyers, and transporters. By facilitating effective communication between these agents, Rice Express helps streamline operations and significantly reduce transport costs.

2.3 Multi-Agent Systems with Large Language Models

Traditionally, MAS research emphasized explicit communication protocols and hand-crafted decision policies. The advent of LLMs has introduced a new paradigm, integrating agents with powerful general reasoning and natural language communication skills. These kinds of systems utilize LLMs to enable more flexible communication, collaborative decision-making, on-the-fly reasoning, and autonomous coordination (S. Han et al., 2024)

In essence, LLMs serve as powerful brains and act as communication facilitators for agents, enabling more human-like and flexible behaviors within a MAS. Instead of rigid protocols, agents are able to exchange information, ask questions, and negotiate with each other in natural language. This improves coordination, as agents can clarify misunderstandings and align on goals dynamically (T. Guo et al., 2024). MAS can employ various communication paradigms, from fully cooperative teams to adversarial or debate-style interactions, which use LLMs to generate and interpret messages. Different communication structures can be adopted, such as centralized, decentralized peer-to-peer, hierarchical layers or publish/subscribe message pools.

LLMs have demonstrated impressive reasoning abilities such as, for example, when using chain-of-thought prompting. These abilities can be harnessed by MAS for better decision-making. Each agent can reason about complex tasks, decompose problems, and critique or verify other agent's conclusions, which leads to more informed and reliable group decisions. For example, one agent might propose a plan which another agent evaluates for flaws, and a third one might suggest improvements. In comparison to a single agent, a team of LLM-based agents brings collective intelligence (Zou et al., 2023), where multiple autonomous agents engage in planning, discussions and decision-making, which mirrors the cooperative nature of human group work.

Another major benefit of integrating LLMs into MAS is the adaptability it confers. LLM agents can be rapidly adapted to new instructions or changes in the environment simply by updating their prompts or context, rather than requiring code changes. This means that a MAS with LLM-powered agents can be repurposed for different tasks and scenarios with minimal reconfiguration. Additionally, LLM agents can employ in-context learning to adjust their behavior based on recent interactions. Via Learning through Communication (LTC), where the dialogue between agents is logged and used to fine-tune the LLMs or inform future decisions (K. Wang et al., 2024). In addition, LLM-based agents often have vast prior knowledge (from pre-training), which allows them to adapt to unfamiliar situations by drawing analogies or using their general knowledge, something fixed-rule agents lack. An LLM-powered agent, when faced in a complex task, might suggest bringing in another helper agent or using an external tool, which reflects their adaptability and flexibility when faced with new problems. This flexibility expands the range of environments and problems MAS can handle.

In MAS, agents often benefit from specialization, each agent focuses on what it does best. With the usage of LLMs, this is enhanced by the enabling of specialized agent roles that can still understand and communicate with others. For example, one agent might take on the role of a researcher (skilled in information gathering), another acts as a planner (skilled in organizing steps), and another as a critic (skilled in evaluation). Because each agent utilizes an LLM, these roles aren't rigidly defined by code, but by prompts that indicate the agent a certain persona or expertise. This ability of LLMs to embody diverse professional or personality-based roles has been leveraged in systems like MetaGPT (Hong et al., 2024) and AutoGPT (H. Yang, Yue, et al., 2023), where an assembly line of agents with human-like domain expertise collaborate on a complex project, such as software development. This collaborative specialization leads to more efficient workflows in which tasks are divided naturally according to each agent's strengths, yet the group works toward a common objective, making them more effective than a monolithic agent (S. Han et al., 2024).

By combining the above capabilities, rich communication, strong reasoning, adaptability, and specialized collaboration, MAS integrated with LLMs achieve a high degree of autonomy. They can operate with minimal human intervention, handling complex goals through multi-step dialogues and actions among themselves. Researchers have demonstrated emergent behaviors in such systems that resemble social or cognitive dynamics. For example, in a simulated town populated by generative agents (each controlled by an LLM), the agents formed new plans (throwing a party) and coordinated attendance purely based on their own interactions, without explicit programming of that scenario (Park et al., 2023). In addition, LLM agents can leverage tools (for example, calling APIs, executing code) to act on the world, further increasing their autonomy.

2.3.1 AutoGen

AutoGen (Q. Wu et al., 2023) is an open-source framework developed at Microsoft for the construction of agentic applications by utilizing multiple LLM-powered agents that converse and cooperate with each other. The framework provides a high-level API to create agents integrated with LLMs and to define the conversational workflows between them. The design philosophy of the framework is to make multi-agent interactions feel like orchestrating a chat among experts, which can include AI agents, tool-using agents, and even humans in the loop. AutoGen acts as a controller that routes messages among agents according to a specific pattern, or dynamically via a manager agent.

AutoGen's architecture is highly modular and event driven. It utilizes asynchronous messaging to support both request-response style chats and event-driven triggers. This decoupled design improves scalability, allowing agents to operate concurrently and even distributed across machines. The framework has a layered structure with a Core messaging layer, an AgentChat layer for managing dialogues, and an Extensions layer for additional capabilities. Users are able to easily plug in to the framework custom agents, tools, and models. For example, the user can

create a new agent class that uses a specific LLM (via OpenAI API, Azure, or locally hosted) and then integrate it into an AutoGen conversation.

The framework also enables complex conversation patterns between agents. Out-of-the-box, it supports patterns like one-on-one chats and multi-agent group chat. Developers can define roles and initial prompts for each agent (for example, a “Planner” agent who breaks a task into steps, and a “Solver” agent who executes them) and then use AutoGen to let them dialogue until a solution is reached. The AutoGen agents can also call tools or APIs as part of the conversation. The framework will pass tool invocation requests to the actual tool and return the results into the chat. This integration of external tools allows the agents to perform web searches or run code and share the results with each other in natural language. Another feature is human-in-the-loop capability, where a human can be inserted as one of the agents in the conversation, to guide or correct the AI agents during the conversation. AutoGen also provides observability and debugging aids, with which developers can trace the message history, enable logging, and even step through conversations for analysis.

2.3.2 CrewAI

CrewAI (Moura, 2023), is another in development open-source framework that provides a structured platform for building MAS. It is designed to create multi-agent workflows with a degree of structure and control.

The architecture of the framework is modular and hierarchical and composed of four components:

- Agent, an LLM-powered entity with a specific role, persona, and a set of tools. Agents also have a memory mechanism to be able to remember conversation context.
- Task, a unit of work to be accomplished, for example, “Gather information on the topic of MAS” or “Draft a blog post”. Tasks are assigned to agents.
- A Crew is the top-level manager that holds a group of agents and a collection of tasks to be completed.
- The Process is the workflow that defines how tasks are carried out by the crew and how the agents interact with each other. CrewAI currently supports three strategies:
 - Sequential process, in which tasks are executed in a fixed sequence by one agent after another and outputs also flow from one agent to the next.
 - In the Hierarchical process, there is a manager agent that delegates tasks to other agents, provides feedback and coordinates results. The manager agent is able to dynamically decide the order of execution of tasks.
 - The Consensual process is a planned strategy where agents openly discuss and work jointly on tasks, taking decisions democratically or through debate. It is similar to AutoGen’s group chat.

The modular design of the framework means that different LLMs from different providers can be easily plugged in to serve the Agents, which is achieved by the abstraction layer provided by

LiteLLM that allows the swapping between providers such as OpenAI, Anthropic, Azure or Ollama. Additionally, while CrewAI supports custom tool creation, it also comes with pre-built tools ready to be used out of the box, to enable actions such as directory, file and website reading, code interpretation and image generation.

CrewAI also supports Flows, a counterpart to Crews that are used to build more deterministic automation sequences, incorporating agent actions at certain steps. This allows for the combination of static workflows with agent intelligence, enabling finer precision at the cost of agentic autonomy. Finally, the framework also provides a dashboard for monitoring and tracking agent performance on tasks.

2.3.3 TaskWeaver

Also created by Microsoft, TaskWeaver (Qiao et al., 2024) is a MAS framework with a code-first focus, aimed at tasks involving complex data analytics or workflow automation tasks. TaskWeaver addresses scenarios where an AI agent needs to generate and execute code to fulfill a user's request. The framework combines LLM reasoning capabilities with traditional programming. The models plan and write code, which is then run to obtain results. This approach is slightly different from AutoGen and CrewAI that placed more emphasis on the conversation between agents.

Architecture-wise, TaskWeaver has three main components:

- The Planner, an LLM-powered agent that interacts with the user and breaks down the task into a series of sub-tasks, creating a plan for solving the problem.
- Part of the Code Interpreter, the Code Generator component takes each planned sub-task and generates executable code for it. The Code Generator can use plugins, like tools, to have access to more actions, and examples, which guide the agent in creating the code.
- The second part of the Code Interpreter is the Code Executor. This component does not employ any LLMs, instead it is used to run the generated Python code and return the results to the Code Generator.

These components work in a loop, the Planner decides on a step, the Code Generator produces code for it and the Code Executor executes it. Both the Code Generator and the Planner are capable of self-reflection, analyzing the results and determining the next step. The process continues until the user's request is fulfilled, then the Planner returns the final result in a user-friendly answer, like a graph or natural language. The framework's plugins are also an important component of TaskWeaver, as they allow the agents new capabilities such as database data retrieval or DataFrame anomaly detection.

2.3.4 Other works

Integrating the reasoning capabilities of LLMs with MAS expands the range of tasks that the agents can accomplish. LLMs enhance explainability and transparency by answering user questions, making them valuable tools for improving user understanding. By combining the task-solving capabilities of MAS with the reasoning abilities of the LLMs, it is possible to develop platforms that more efficiently address user tasks and requests. For instance, in (Kumar et al., 2023) ChatGPT is employed to orchestrate scientific ML processes based on user prompts. Similarly, (Chuor et al., 2024) describes the classification of user stories using ML methods in conjunction with LLMs. (Z. Liu et al., 2024) proposes an ensemble model comprising an LLM, a small ML model, and a task allocator to manage their collaboration effectively while reducing costs. In (Talebirad & Nadiri, 2023), the authors propose a framework that enhances the capabilities of LLMs by fostering collaboration within a MAS. Each agent, referred to as an Intelligent Generative Agent (IGA), is assigned unique attributes and roles to contribute effectively to the system. In (C. Qian et al., 2023), the ChatDev framework is introduced, an LLM-powered, agent-based system for software development. Agents, including roles such as chief officers, programmers, test engineers, and designers, work collaboratively to deliver software solutions tailored to client requirements. Similarly, GPT Pilot (*Pythagora-io/Gpt-Pilot*, 2023/2024) enables users to design software through the collaboration of AI agents, such as the Specification Writer, Architect, Developer, and Reviewer. These agents work together to generate tasks and code while ensuring user requirements are met. Additionally, the framework supports debugging with the user to refine the final output.

In (F. Jiang et al., 2024), CommLLM is introduced. A MAS designed to enhance the capabilities of LLMs for 6G communications by integrating retrieval, planning, memory, evaluation, and reflection. The system consists of three components: multi-agent data retrieval (MDR), whose agents refine and summarize domain-specific knowledge, multi-agent collaborative planning (MCP), where planning agents generate solutions based on the retrieved knowledge, and multi-agent evaluation and reflection (MER), where an evaluation agent assesses the created solutions, while reflection and refinement agents suggest improvements. Article (H. Li et al., 2024) investigates the usage of MAS and LLMs for knowledge tagging, addressing complex cases that involve intricate knowledge definitions and strict numerical constraints. (Quan & Liu, 2024) proposes InvAgent, which utilizes LLMs to manage multi-agent inventory systems for supply chain management (SCM). It adopts the usage of zero-shot learning for LLMs and offers explainability and transparency via Chain-of-Thought (CoT) reasoning. The work demonstrates dynamic adaptability to various scenarios, cost minimization, and stockout avoidance. Paper (H. Chen et al., 2025) introduces the usage of MAS driven by LLMs in consensus-seeking tasks. The agents had to negotiate to reach a shared consensus value, but each agent started with a different numerical value. The paper explores how LLM-driven agents typically settle on an average value strategy to reach consensus by default. Additionally, the paper also presents the usage of LLMs for consensus-seeking in a multi-robot aggregation task.

PEAK (Python-based Framework for Heterogeneous Agent Communities) (Ribeiro, Dias, et al., 2025) is a MAS development tool that enables agents from different environments to interact,

exchange knowledge, and work towards common goals. Built on the SPADE framework, PEAK supports advanced features like simulation, agent organization, debugging, and integration with external services and AI models. It is designed to create scalable, autonomous agents that can operate in real-world scenarios, such as energy systems and FL.

Finally, (Brienza et al., 2024), a multi-agent architecture for task planning is proposed which utilizes a single image of the environment and leverages common sense knowledge to handle free-form domains. The MAS, consisting of LLMs and Visual Language Models (VLMs), receives the image and the task and generates a plan to complete the assigned task.

2.4 Retrieval Augmented Generation

This section explains the inner workings of RAG. While multiple implementations and variations of the RAG methodology exist, this section explains in more detail the Naïve variant (Gao et al., 2024), since this version was implemented in the system and utilized during the case study. Naïve RAG operates in two main phases: the first phase involves inserting documents into a vector database, as shown in Figure 2, and the second phase involves processing user queries, as shown in Figure 3. The two phases work together to allow the RAG system to provide enhanced responses based on external knowledge.

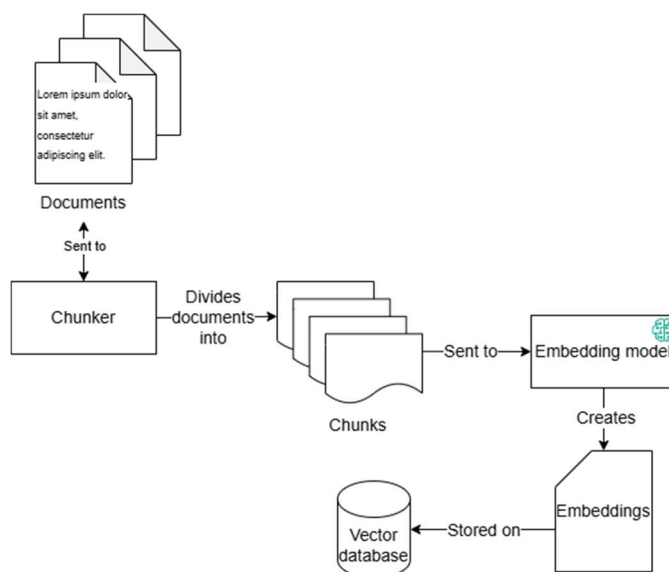


Figure 2. Document Embedding in RAG

When a document is uploaded into the RAG system, it undergoes an initial chunking process. the document is divided into smaller, semantically meaningful chunks. Each chunk is then passed through an embedding model, which transforms the text into vector representations. These vectors, or embeddings, are numerical values that encode the semantic meaning and relationships within the text (S. Wu et al., 2024), allowing for efficient similarity comparison later in the process. The embeddings are stored in a vector database for future use, where they can be retrieved based on similarity to user queries.

Upon receiving a user query, the RAG system processes it similarly to how it handles documents. The retriever component first transforms the user's query into a vector using the same embedding model that was applied to the documents. A similarity search is then performed to compare the query vector with the vectors of the text chunks stored in the vector database. Common similarity metrics include cosine similarity and Euclidean distance. The most relevant chunks are retrieved and appended to the original user query, enriching it with contextual information. This enriched input, containing both the original query and the retrieved relevant text chunks, becomes the input to the LLM. The model, now equipped with this additional context, generates a response that incorporates the knowledge stored in the database.

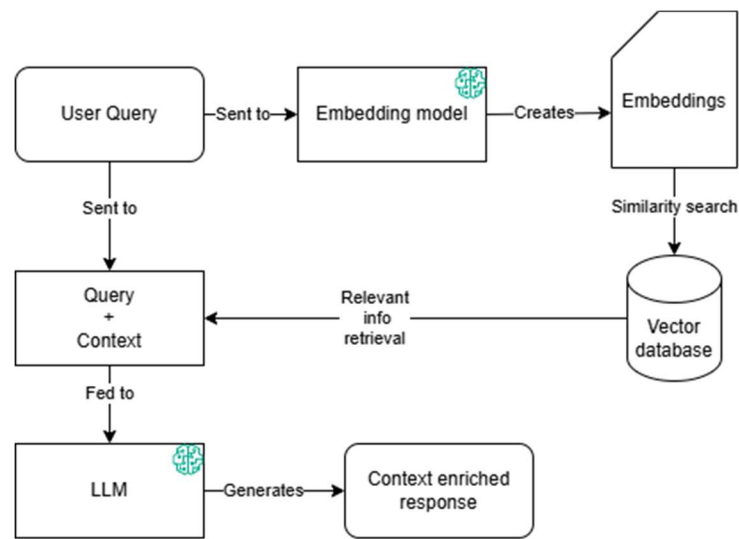


Figure 3. LLM Inference with RAG

Other variants of the RAG methodology include advanced and modular RAG (Zhou et al., 2024), and Graph-based RAG.

Advanced RAG enhances the traditional RAG framework by introducing specialized components to address challenges in both the retrieval and generation stages of the pipeline, categorized into pre-retrieval and post-retrieval stages. In the former, the main issue lies in the vagueness or shortness of the queries, often leading to irrelevant retrieval results. To address this, a rewriter is incorporated to clarify or expand the query. Rewriters can be a direct prompt to an LLM or training rewriter models using feedback from the generator. These methods improve the specificity and relevance of the query. In the post-retrieval stage, challenges such as noisy or excessively long retrieved content can negatively impact generation quality. A reranker can be employed to reorder retrieval results based on their relevance to the query. The rerankers often use cross-encoder architectures and can improve relevance by more accurately measuring the similarity between the query and the retrieved documents. A refiner can also be employed to summarize or compress the retrieved documents, utilizing techniques such as LLM-based summarization or training dedicated summarizers through supervised or reinforcement learning (RL). However, despite its flexibility, the sequential structure of Advanced RAG limits its adaptability in complex step-by-step reasoning scenarios.

Modular RAG is considered the next evolutionary step of RAG research. It is focused on flexibility and adaptability through the use of modular components. In Modular RAG, the various components of the RAG process are considered interchangeable modules, which allow researchers to design and customize pipelines that are optimized for different case studies. This modularity offers more versatility compared to Advanced and Naïve RAG, as it supports dynamic configurations and optimizations. The pipelines can be classified into four main types:

- Sequential Pipelines:
 - The query is processed linearly;
 - Includes distinct pre-retrieval and post-retrieval stages, similar to Advanced RAG.
- Conditional Pipelines:
 - Queries are routed based on their characteristics;
 - Simple queries might bypass retrieval, while complex ones undergo multi-round retrieval.
- Branching Pipelines:
 - Executes multiple paths simultaneously for a given query;
 - Combines results to produce the final output;
 - Addresses instability in single-path reasoning by aggregating probabilities or selecting the best answer.
- Looping Pipelines:
 - Involves iterative interactions between the retriever and generator;
 - Enables multiple rounds of reasoning and retrieval;
 - It is considered the most complex type of pipeline.

Graph-based RAG extends traditional RAG by combining graph-based structures into the retrieval and generation processes, enabling models to understand not just textual content but also the relationships between the entities (Peng et al., 2024). In Graph-based RAG, knowledge graphs have nodes that represent entities and edges that represent relationships, allowing for more context-specific retrieval. To improve performance, the retrieval process can only extract relevant subgraphs, that represent specific portions of the larger graph that are contextually significant. Graph-based RAG is useful for domains that require deep understanding and contain complex datasets, such as cybersecurity, healthcare, and politics (Abu-Salih, 2021). In (Hu et al., 2024), a framework named Graph Retrieval-Augmented Generation (GRAG) is introduced, employing a divide-and-conquer strategy, retrieving the most relevant ego-graphs and refining them via a soft graph pruning mechanism, providing an approximate solution without searching all subgraphs. Article (X. He et al., 2024) proposes G-Retriever, a framework that enables users to interact with textual graphs through a conversation interface. The user asks natural language questions and receives both textual responses and visual highlights of relevant graph components.

Some works that leveraged RAG include the article (Ma et al., 2024) that proposes a multi-expert collaboration framework, where LLMs provide expert knowledge to pre-trained language models (PLMs), enhancing their ability to analyze logs and perform more complex

tasks. In (Guan et al., 2024) the usage of both LLMs and PLMs for anomaly detection in logs is investigated, demonstrating the potential of these models for identifying abnormal patterns in data.

In the same vein, (Ji et al., 2024) applies continual pre-training (CPT) to equip LLMs with domain-specific knowledge, which enables LLMs to perform better in log analysis, where domain expertise is important.

Paper (Pan et al., 2024) proposes a method where a database of normal logs is compared with a log entry posed as a question, determining whether the query log corresponds to normal or abnormal behavior.

Article (W. Zhang et al., 2024) presents a two-phase approach to log anomaly detection. In the first phase, log tokens are analyzed using a deep one-class classifier to identify anomalies. In the second phase, LLMs using RAG are employed to validate the anomalies detected in the first phase, improving the robustness and accuracy of the anomaly detection process.

The article (Y. Lyu et al., 2025) introduces a comprehensive benchmark for evaluating RAG systems, addressing limitations in existing evaluations that focus mostly on question-answering applications. The authors categorize RAG applications into four CRUD actions: Create (generating original content), Read (answering complex questions), Update (revising pre-existing texts), and Delete (summarizing extensive texts). For each category, different datasets were developed to assess the performance of the RAG systems and to analyze the impact of the different RAG components, such as the retrievers, context length, knowledge base construction, and LLMs.

Paper (Ding et al., 2024) proposes RealGen, a retrieval-based in-context learning framework for generating realistic and controllable traffic scenarios to support autonomous vehicle (AV) development, which generates new scenarios through the combination of behaviors from multiple retrieved examples, in a gradient-free way. Additionally, the scenarios can be edited, and critical ones can be produced.

The article (Yu et al., 2024) presents a novel instruction fine-tuning framework that unifies context ranking and answer generation within RAG systems. Unlike traditional RAG systems, whose retrievers rank the top-k contexts, RankRAG trains a single LLM to perform both tasks by incorporating a small fraction of ranking data into its instruction fine-tuning process.

Paper (Ribeiro, Gomes, et al., 2025) proposes a new human-machine interface for smart homes, integrating MAS and natural language models with RAG. The system aims to enhance user interaction and facilitate adaptation to smart home technologies. The assistant allows users to control their home and provides information, even when issues arise within the system.

Finally, RAGAs (Retrieval Augment Generation Assessment) is introduced in (Es et al., 2023), and it is a framework for reference-free evaluation of RAG systems. Evaluating RAG systems is complex, requiring an assessment of multiple variables, such as the accuracy of the retrieval, the faithfulness of the LLM's usage of the retrieved contexts, and the quality of the generated

response. These challenges are addressed by RAGAs by introducing a suite of evaluation metrics that do not rely on ground truth human annotations, which contributes to the faster evaluation of RAG architectures and their development.

2.5 Machine learning algorithms for forecasting

This subsection will go over forecasting algorithms based on ML. In addition, it will also go over the usage of these algorithms in the energy domain, namely for the forecast of energy generation and energy consumption.

ML is a subset of AI focused on enabling computers and machines to imitate the human learning process (Paleyes et al., 2022), allowing algorithms to learn from data to improve their performance on specific tasks without being specifically programmed. ML uses statistical algorithms to analyze and make predictions or decisions based on patterns in data. These algorithms can generalize beyond the training data they were given, allowing them to handle new situations they haven't encountered before. There are three main types of ML: supervised, unsupervised, and reinforcement learning. ML algorithms have wide-ranging applications across various areas, such as energy (Kiasari et al., 2024), computer vision (CV) (Safyari et al., 2024), fraud detection (Huang et al., 2024), predictive maintenance (Oliveira et al., 2023), decision-making (Mohr & van Rijn, 2024), and recommendation systems (Hasan & Ferdous, 2024).

Supervised learning is a type of ML where the algorithm is trained on labeled data that pairs inputs, the features, with the correct outputs, the targets (Valkenburg et al., 2023). The objective is for the model to learn patterns that allow it to perform predictions on new, unseen data. This approach can be applied to classification tasks, where the model assigns inputs to predefined categories, or to regression tasks, where continuous values are predicted. Examples of classification models include Decision Trees (DT) (Costa & Pedreira, 2023), Random Forests (RF) (Sharma et al., 2021), and Support Vector Machines (SVM) (Bansal et al., 2022). Regression models include Linear Regression (LR) (Gyawali et al., 2024) and Polynomial Regression (PR) (Tsai et al., 2022). This type of learning can be used for applications such as cybersecurity (Alduailij et al., 2022), sentiment analysis (Tabany & Gueffal, 2024), and house price prediction (Adetunji et al., 2022).

Unsupervised learning is the type of ML where algorithms learn from data with no labels. The models try to identify hidden structures, relationships, or groupings in the data (Naeem et al., 2023). Common unsupervised learning techniques include clustering, where algorithms group similar data points, and dimensionality reduction, where features are reduced while keeping important information. An example of the former is the K-means clustering algorithm (Ikotun et al., 2023), and of the latter the Principal Component Analysis (PCA) algorithm (Marukatat, 2023). Unsupervised learning algorithms can be applied to tasks such as customer segmentation (Saxena et al., 2024), fraud detection (S. Jiang, Dong, et al., 2023), and sports performance (Casimiro-Artés et al., 2024).

The third type of ML is Reinforcement Learning (RL), where the AI agent learns by interacting with the environment and gets rewards or penalties depending on its actions (Towers et al., 2024). The goal of the AI agent is to maximize cumulative rewards over time. RL agents follow a policy, that dictates the actions they must take based on the current state. The policy gets refined using feedback from the environment.

Additionally, RL can be divided into two types. In model-free RL, the agents learn a policy directly from interactions with the environment without possessing a direct model of it (Mhammedi et al., 2023). On the other hand, in model-based RL, the agent builds a model of the environment’s dynamics and uses it to plan future actions (Luo et al., 2024). Q-learning (J. Lyu et al., 2022) is an example of a model-free RL algorithm, while Dyna-Q (Y. Liu et al., 2022) represents a model-based RL algorithm. RL is commonly used in areas such as robotics (B. Singh et al., 2022), games such as chess, Go, and video games (Patankar et al., 2024), and AVs (Xiao et al., 2024).

Deep learning (DL) is a subsection of ML that utilizes neural networks (NN) with multiple layers to simulate complex decision-making abilities inspired by the structure of the human brain (Dastres & Soori, 2021). NN consists of interconnected neurons, or nodes, divided into layers. Firstly, inputs are fed to the network and passed through the multiple layers, forward propagation, and afterward the error between the prediction and the real value is calculated, and weights are adjusted to improve accuracy, in a process known as back propagation, as shown in Figure 4. While DL models provide higher accuracy and performance when compared to traditional ML models, they require more computational resources, and their training process is highly time-consuming (Mei et al., 2024).

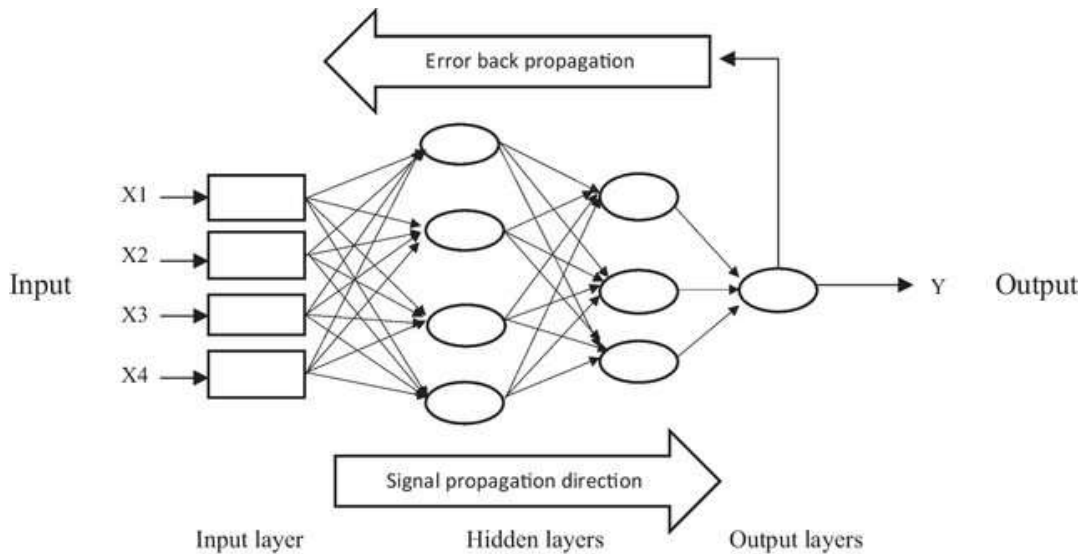


Figure 4. Forward propagation and backward propagation (Gezici & Sengul, 2023)

Some common DL architectures include Convolutional Neural Networks (CNNs) for image and video processing, Recurrent Neural Networks (RNNs) for speech and natural language, and LSTM models. Taking into account the nature of the data presented in the energy datasets used for the case studies that test the components of the system, the model type that was chosen

to be generated by the component defined in Section 4.2.1 was the LSTM model (Hochreiter & Schmidhuber, 1997). It is a type of recurrent neural network (RNN) designed to effectively capture and utilize long-range dependencies in sequential data. Usually, traditional RNNs struggle with the vanishing gradient problem, where the influence of earlier inputs diminishes exponentially as the sequence length increases, hindering the learning of long-term patterns. LSTMs address this limitation through the usage of Constant Error Carousel (CEC) memory units (Schmidhuber, 2015). These models have been successfully applied in many fields, such as Natural Language Processing (NLP) (J. Liu et al., 2024), speech recognition (Al-Fraihat et al., 2024), and time series prediction (M. Wang et al., 2024). However, they also present some disadvantages (Menghani, 2023), such as computational complexity, which leads to higher resource usage and time spent, they are prone to overfitting without regularization, and limited parallelism due to sequential data processing. One of the other issues present in DL is models being treated as “black boxes”, it stems from the fact that the internal decision-making processes that lead to an output from an input are highly complex and opaque, making the predictions not easily interpretable by humans (Hassija et al., 2024). This lack of transparency translates into a lack of trustworthiness in the decisions of the model, which is crucial in areas like healthcare (Fernandez-Quilez, 2023) and finance (Bücker et al., 2022).

DL models see usage in areas such as transportation (Khalil et al., 2024), finance (C. Zhang et al., 2024), drug discovery (Fluetsch et al., 2024), content recommendation (N. Tan, 2024), and crop monitoring (Slimani et al., 2024).

Concerning ML forecasting-related works, the paper (Theocharides et al., 2020) presents a method that forecasts the day-ahead photovoltaic (PV) power generation. An Artificial Neural Network (ANN) was used to predict the power output of the PV. Afterward, the forecast was enhanced via the utilization of k-means clustering applied to the global horizontal irradiance (GHI) dataset to determine the best parameter for the post-processing analysis. The model obtained a mean absolute percentage error (MAPE) of 4.7% in a hot location and a MAPE of 6.3% in a cold semi-arid location.

In (Mahmud et al., 2021) machine learning models are applied for PV generation forecasting in the area of Alice Springs, Australia. The performance analysis was performed in the following models: LR, PR, SVR, DTR, Random Forest Regression (RFR), Multilayer Perceptron Regression (MLP), and LSTM. The algorithms were tested on normalized and un-normalized data, and the performance metrics utilized were MAE, MSE, R^2 , Median Absolute Error (MedAE), and Explain Variance Score (EVS). For normalized data, RFR is the model that possesses the best performance metrics, with MLP being the one with the least performance. Generally, unnormalized data meant that the models achieved worse performance, yet RFR remained the best performing out of the six.

A comparison between LR, RF, SVR, and MLP is presented in (Jebli et al., 2021). The forecasting models' case study is the PV energy generation in the province of Errachidia, Morocco, which possesses a semi-desert climate. In addition, Pearson correlation was used during the data pre-processing to extract the features more relevant to the predictions made by the models. All models overfitted in a scenario without Pearson correlation. In the case of real-time prediction,

the Random Forest model is the best performing, while the MLP ANN is the best model in the scenario of daily prediction. The trained models were applied to another case study, the tropical climate region of Pirapora in Brazil, and once again MLP ANN and the RF were the best performers.

A study of different machine learning methods applied to wind power production is presented in (U. Singh et al., 2021). Before the forecasting, a Supervisory Control And Data Acquisition (SCADA) system is used to gather the data from the wind farm which is then pre-processed. Five forecasting models were trained on the dataset, RFR, k-Nearest Neighbor (k-NN) Regression, Gradient Boosting Trees, DT, and Extra Tree (ET). The accuracy and error obtained during the testing of the models were compared, and the gradient boosting regression (GBM)-based ensemble algorithm was determined to be the most accurate with a Mean Absolute Error (MAE) of 0.0277, MAPE value of 0.3310, Root Mean Square Error (RMSE) value of 0.0672, Mean Square Error (MSE) value of 0.0045 and coefficient of determination (R^2) value of 0.9651. The authors determined that the DT model was the worst performing, with an MAE of 0.0336, a MAPE of 0.3309, an RMSE value of 0.0884, an MSE value of 0.0078, and an R^2 value of 0.9497.

In (Olu-Ajayi et al., 2022) the authors propose utilizing ML models to forecast the energy consumption of a building. The dataset utilized contained the metadata of 300 real residential buildings around the UK, available online on the Ministry of Housing Communities and Local Government (MHCLG) Repository. Three models were considered: an Artificial Neural Network Multilayer Perceptron, a Support Vector Machine, and a Decision Tree model. The MLP was the best performer in terms of R^2 , MAE, RMSE, and MSE parameters. The DT model was the worst performing, yet it had the shortest training time.

In (P. W. Khan et al., 2020) the authors propose a hybrid ensemble model that combines SVR, MLP, and CatBoost, an algorithm based on decision trees. The paper uses the energy consumption data of Jeju, in South Korea, to train the hybrid model for energy consumption forecasting. The ensemble model was evaluated on the MAE, MSE, RMSE, and Root Mean Squared Logarithmic Error (RMSLE) metric against the Lasso, Ridge, GradientBoost, MLPRegressor, SVR, and XGBoost models. The proposed model was the best performer, with a 15.73 MAE error and a 21.75 RMSE error value, while the second-best model was XGBoost with 23.51 MAE and 27.88 RMSE error values. The worst performer was the SVR model, with an MAE error value of 75.81 and an RMSE error value of 98.76.

Article (Teixeira et al., 2024) proposes using K-Means clustering to enhance Explainable Artificial Intelligence (XAI) in complex ML models. By identifying relevant data points and creating similarity clusters, the explainable AI is focused on key information. The proposed method was tested in a PV generation forecasting case study.

Paper (Ramos et al., 2024) discusses the use of XAI techniques, particularly Shapley Additive Explanations (SHAP), to analyze and improve energy consumption forecasting in buildings. The energy sector faces challenges due to the uncertainty of energy demand and user behavior, making forecasting crucial for optimizing energy efficiency. Various ML algorithms, such as ANN, KNN, XGB, are commonly employed for forecasting.

2.6 Federated Learning

Federated Learning (FL) (Zhan et al., 2022) is a decentralized machine learning methodology where different actors, usually named clients, collaborate to train a global model, in a way that provides data privacy to the participants.

Machine learning algorithms usually depend on a large quantity of data, normally stored in a single location, to train models with high prediction accuracy. However, the decentralized nature of FL allows for data to be split between multiple clients. These clients will be tasked with training a local model on their data. Afterward, the gradients of the trained local model of each client will be sent to a server that will be responsible for aggregating all the model gradients into a global model. Finally, the global model will be sent to the clients once the aggregation process is finished. Figure 5 provides a visual representation of this process.

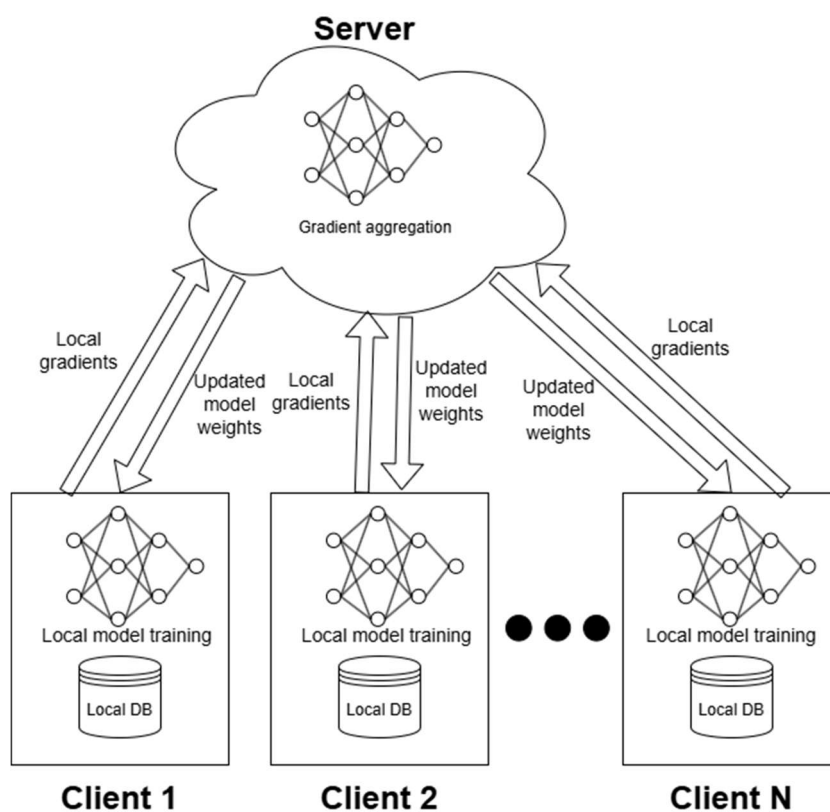


Figure 5. FL process representation

This process means that the data used for training does not have to be transferred to any centralized data silo. To safeguard the transfer of model gradients from the clients to the server, methods such as Differential Privacy (DP) or k-anonymity are applied to the FL process (Q. Li et al., 2023).

Differential privacy (Wei et al., 2020) is a way to prevent private information from being leaked via the analysis of the model parameters that are sent to the server. It works by adding artificial

Gaussian noise to the model gradients at a local level before they're sent to the server to be aggregated.

K-anonymity is a process that applies generalization and suppression processes to data (Liang & Samavi, 2020). The former means replacing an attribute in the data with a broader category. The latter, suppression, means the record can be deleted to achieve k -anonymity. A balance must be made between guaranteeing the anonymity of data while still maintaining the utility of the data.

One of the challenges Federated Learning has to face is the fact that edge devices most likely possess non-independent and identically distributed data (IID). The usage of non-IID data generally means that the accuracy of the federated training model is reduced (Zhao et al., 2018). However, techniques such as sharing a small subset of data between all the edge devices or FedProx (T. Li et al., 2018) can be used to increase the accuracy of the trained model. FedProx is an FL aggregation method based on Federated Averaging (FedAvg) but introducing partial work tolerances to aid with systems heterogeneity as well as proximal terms to limit the impact of local model updates to reduce the divergence of the methods due to the heterogeneous nature of the data.

Federated Learning can be applied to areas such as edge computing, and recommender systems (Q. Li et al., 2023). In (Niknam et al., 2020) the authors present the usage of a FL-trained model to predict which information is popular enough to be pre-fetched and stored locally in edge terminals to reduce latency, which could be applied to an Augmented Reality (AR) context. A privacy-aware service placement (PSP) scheme is proposed by (Y. Qian et al., 2019) where federated learning is used to predict user's preferences about services via the data residing on their phone while protecting the user's privacy. This is used in combination with a greedy algorithm to realize the service placement scheme. Article (Y. Zhao et al., 2019) proposes a system that combines Federated Learning with Blockchain technologies that enables home appliance manufacturers to request customers to participate in the federated training of a model for home appliance improvement, in a crowdsourcing effort. The distributed ledger is used to store the gradients of the local models trained by customers and where the global aggregated model is then stored by one of the miners. Customers will be rewarded based on their contribution with maintenance and upgrade services. The data on a large number of edge devices, like mobile phones, can be used to train a global model without compromising the privacy of the phone owner's data, such as Google's international scale model used to improve virtual keyboard search suggestions (T. Yang et al., 2018).

The federated recommender system (FedRecSys) is presented in (B. Tan et al., 2020). The system utilizes federated learning in combination with recommendation algorithms to provide users with content recommendations, product recommendations, and online advertising while keeping the user's data private. According to the authors, the system has increased the average reading time by 22%. In (Muhammad et al., 2020) FedFast is proposed, the authors of the paper affirm that users in the federated learning process incur the majority of the costs, such as the increase of the communication payload during the training process. In addition, the authors also state that the users have to wait until the late stages of the training process for quality

recommendations from the trained model. The proposed framework speeds up the convergence rate of the recommendation models while also providing more accurate recommendations without users having to wait until a later training stage. FedFast is divided into an algorithm that chooses which users to participate in the training process and another algorithm that speeds up training by smartly combining the local models.

Paper (Jie et al., 2023) proposes a personalized federated recommendation system designed to tackle privacy concerns due to strict regulations and the challenges posed by the non-IID nature of data in FL. Traditional recommendation systems often rely on centralized user data, raising privacy issues. In response, the system integrates historical learning parameters from clients with global parameters using a time decay factor, ensuring better adaptation to local data. Additionally, parameters are clustered on the server to account for data diversity, improving the model's ability to handle heterogeneous datasets while maintaining privacy.

Article (S. Wang et al., 2024) proposes FeB4RAG, a novel dataset designed to enhance federated search withing RAG frameworks. Federated search systems aggregate results from multiple search engines to improve result quality and align with user intent. FeB4RAG consists of 790 conversational queries tailored for chatbot applications, along with the top results from each resource.

3 Methods and tools

This section will present the ML, FL, and LLM models considered and used in the development of the MASTerFlow system. The Python frameworks and libraries used for implementation are also explained. Afterward, the datasets that were used for testing the case studies are presented and described. Finally, an overview regarding the data protection and security analysis requirements of the system is provided, as well as the ethical aspects that need to be taken into consideration.

3.1 Models and frameworks

This section outlines the models, frameworks, and libraries that were studied and utilized during the development of the system. For each, an overview of their functionalities and how they were used in the system is given.

3.1.1 Large language models

During the implementation of the system, several LLMs, shown in Table 1, were considered to be used for either dialoguing with the user or by an agent inside one of the group chats of MASTerFlow. As the case studies were executed, the results obtained were used to filter the chosen LLMs that were used in the final version of the system.

Table 1. LLMs tested during the development

Model name	Model type
GPT-4	Closed weights
GPT-4o	Closed weights
GPT-4o-mini	Closed weights
GPT-3.5-turbo	Closed weights

Llama 2 13b Chat	Open weights
Llama 3 8b instruct	Open weights
TinyLlama 1.1B	Open weights
Mistral 7b instruct v0.3	Open weights
Zephyr 7b beta	Open weights
Codestral v0.1 22b	Open weights

Created by OpenAI, the GPT family of models leads the field due to their advanced reasoning and conversational capabilities, making them highly capable general models that can be used in a broad range of applications. Sorted by release order, the models considered were GPT-3.5-turbo, GPT-4, GPT-4o, and GPT-4o-mini. The latter provides a faster inference time when compared to GPT-4o, sacrificing only a minimal amount of quality.

Developed by Meta AI, the Llama models are free open weights models that can be hosted locally, providing a level of privacy not found in the closed GPT models. The cost of hosting local models is the electricity expenditure and the potency of hardware required. Thus, models with fewer parameters, such as 8b or 13b, are less powerful versions of LLMs that run on more types of hardware. Llama 2 13b Chat focuses on conversational AI due to being fine-tuned for that objective. On the other hand, Llama 3 8b instruct is a smaller model to be used for instruction-following tasks, making it better at responding to code requests.

TinyLlama 1.1B is a small, compact LLM sharing the same architecture as Llama 2. It was used to test how small a model can be parameter-wise and still be usable.

Mistral 7b instruct v0.3, developed by MistralAI, is tailored for instruction-following use cases, able to outperform the larger Llama 2 13b (A. Q. Jiang et al., 2023). Zephyr 7b beta is a fine-tuned version of the Mistral model, trained to act as a helpful assistant. While it performs better than its predecessor, the lack of model filtering of response means the model can produce problematic outputs (Tunstall et al., 2023). The largest, parameter-wise, open weights model considered is Codestral v0.1 22b, a programming-focused LLM also created by MistralAI.

3.1.2 Multi-agent systems

For the development of the chain of MAS system, three MAS were considered:

- AutoGen;
- CrewAI;
- TaskWeaver.

The case study in Section 5.1 served to benchmark the frameworks and to determine which one would be used for implementation in MASTerFlow. At the time of development, the Swarm framework was unreleased, so it was excluded from consideration.

Developed by OpenAI, Swarm (OpenAI, 2024) is an education-focused lightweight MAS framework designed to streamline agent coordination and execution. The two main components of Swarm are Agents and handoffs. The former includes instructions and tools and

is capable of handing off a conversation to another Agent at any time. These foundational elements are versatile and enable the modeling of complex interactions between tools and agent networks, avoiding a steep learning curve when creating scalable, real-world solutions.

AutoGen (Q. Wu et al., 2023), developed by Microsoft, is a multi-agent system (MAS) framework that allows users to customize the roles and number of agents needed to solve various tasks. These tasks can include code generation, problem-solving, planning, and answering math questions. The framework also supports human involvement, enabling users to interact with agents to provide corrections or additional details, fostering collaboration in completing tasks.

AutoGen is available as a Python library, enabling local execution by calling the necessary functions in code. While the framework runs locally, it interacts with large language models (LLMs) hosted remotely through OpenAI API-compatible endpoints. This compatibility allows AutoGen to work with OpenAI models such as GPT-3.5 and GPT-4, as well as open models hosted on platforms like LM Studio or Ollama.

Alternatively, AutoGen can be accessed via a user-friendly interface on the Microsoft AutoGen Group Chat Playground hosted on HuggingFace. Similar to the Python library, the UI lets users define the number and types of agents, configure their system messages, and communicate with them via text. A locally hosted version of this interface, AutoGen Studio, is also available.

TaskWeaver (Qiao et al., 2024), another framework developed by Microsoft, focuses on delivering code-first solutions to user requests, particularly in data analytics. The framework devises a plan to address the user's request, generates the required code, and executes it to provide a solution. Additionally, it allows for the integration of plugins to aid in task completion.

Finally, CrewAI (Moura, 2023) is an open-source framework that allows users to customize agents with specific roles, goals, and code-based tools. Users can define tasks and assign them to individual agents, who can also delegate tasks to one another autonomously. The framework supports both sequential and hierarchical agent workflows, where a designated agent can act as the manager of the crew, planning and delegating tasks. These configurations are defined within the framework's process.

CrewAI can be run locally by installing the required packages in a Python environment. Similar to AutoGen and TaskWeaver, it enables the use of large language models (LLMs) hosted remotely. The framework supports GPT models from OpenAI and open models hosted on platforms like Ollama, HuggingFace, or other tools with OpenAI API-compatible endpoints, such as LM Studio, text-gen-web-ui, or Mistral API.

3.1.3 Machine learning

In MASTerFlow, the MAS agents powered by LLMs are tasked with generating code for an ML model while taking into consideration the parameters in the configuration file filed by a user that the system had engaged in conversation with. However, while the final version of the system has the type of ML model to be created in the configuration file, earlier versions of

MAsterFlow allowed the agents themselves to choose the type of the ML model to be created to address the user's uploaded dataset. The following ML models were the ones chosen by the agents:

- Autoregressive integrated moving average (ARIMA);
- LR;
- LSTM;
- RFR;
- XGBoost Classifier (XGBC).

Autoregressive integrated moving average

The ARIMA model (X. Wang et al., 2023) is a classical statistical model used for time series forecasting. It operates by expressing future values of a time series as a linear function of its past values and past errors. These models are denoted "ARIMA(p,d,q)" meaning:

- Autoregressive (AR): Order p indicates the number of past observations included. The model incorporates relationships where the current value depends on a weighted sum of the previous p values.
- Integrated (I): Order d is the number of times the series is differenced to remove trends and achieve stationarity. Differencing is used to eliminate non-stationary behavior.
- Moving Average (MA): Order q specifies the number of past forecast errors included. The model uses past error terms (noise) in a moving average manner to adjust future forecasts.

The model fitting involves identifying the appropriate orders (p,d,q) and estimating the coefficients such that the residual errors are as small as possible (often done via maximum likelihood or least squares). ARIMA is commonly applied to univariate time series data, such as economic indicators, financial time series, sensor readings or load forecasting, where the goal is to make short- to mid-term forecasts based solely on the series' past behavior.

Linear Regression

LR (Schneider et al., 2010) is one of the simplest and most widely used models for understanding and predicting a quantitative outcome. It models the relationship between one dependent variable and one or more independent variables by fitting a linear equation to observed data. In a LR, the prediction \hat{y} for an outcome y is given by a linear combination of input features, as shown by Equation 1, where x_1, x_2, \dots, x_n are features (predictor variables) and $\beta_0, \beta_1, \dots, \beta_n$ are the parameters (coefficients) of the model. These coefficients are learned from training data by minimizing the difference between predicted values and actual values. The result is a best-fit linear hyperplane through the data. The key components of a linear regression model are the coefficients (weights) assigned to each feature and the intercept term β_0 , which is the predicted value of the outcome when all the features are zero.

Equation 1. LR equation

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$$

LR is used across various fields, such as economics, engineering and social sciences. It can be applied not only for prediction but also for inference, to understand which factors significantly affect the outcome and by how much.

Long short-term memory

LSTM networks (Staudemeyer & Morris, 2019) are a type of RNN architecture designed to model sequence data and capture long-range dependencies, since traditional RNNs suffer from the vanishing gradient problem, making it difficult for them to learn relationships in long sequences. The memory cell introduced by LSTM addresses this problem, it can maintain information over long time steps, along with gated mechanisms that regulate information flow. As data sequences (time series) are processed step by step, the LSTM can decide to retain information from earlier steps or forget it, and to incorporate new input information as needed, allowing the model to learn complex temporal patterns, which makes them powerful for tasks where context and order matter.

Each LSTM cell has three main gates that modulate the cell's internal state:

- The forget gate decides which information in the cell state to erase. At each time step, the gate examines the previous hidden state and the current input, and the output generated is a number between 0 and 1 for each piece of information in the cell state. The number 0 means “completely forget this” and the number 1 means “keep this entirely”;
- The input gate controls what new information to add to the cell state. It takes the current input and the previous hidden state to determine which values, that have passed through a nonlinear transformation, should be written to the cell state
- The output gate determines what information from the cell state to output and pass on to the next time step as the hidden state. It filters the cell state via a sigmoid layer and typically applies a *tanh* function to scale the output.

The cell state of the LSTM acts as a conveyor of information, and the hidden state is the output of the cell that also feeds into the next time step. During training, LSTM networks adjust their weights via gradient-based optimization (backpropagation through time) to learn the appropriate sequence patterns.

LSTMs are usually used in scenarios involving sequential or time-dependent data. These models have been used in NLP tasks, such as language modeling, machine translation and text generation. In time series analysis, LSTMs are applied to problems like forecasting complex patterns, like finance or weather data.

Random Forest Regression

RFR (Liaw & Wiener, 2001) is an ensemble learning method that builds multiple decision trees and combines their outputs to produce a more robust prediction. In a random forest, each individual decision tree is trained on a random subset of the training data and, typically, at each split in the tree, a random subset of features is considered. By averaging the predictions of the de-correlated trees, the model achieves higher accuracy and stability than any single decision tree.

The random forest algorithm trains an ensemble of randomized decision trees to ensure diversity among models. Each tree is trained on a bootstrapped sample (random sample with replacement) of the dataset. This means each tree sees a slightly different set of observations, promoting variety. At each node where a tree splits, it does not consider all available features, but rather a random subset of features. This enforces even more diversity among trees by preventing the ensemble from relying too heavily on a single feature. For regression trees, once all trees are grown, the prediction for a new instance is obtained by averaging the predictions of all individual trees. In a classification context, a majority vote or probability average is used instead. The averaging process reduces variance, while individual trees might overfit to noise or peculiarities in their bootstrap sample, their errors tend to cancel out when averaged.

RFR is used in a variety of regression tasks where a nonlinear, flexible model is needed and interpretability is not the primary concern. Some of the areas that have employed RFR models include house price prediction, product demand estimation, or environmental data modelling, such as rain or temperature.

XGBoost Classifier

XGBoost (Extreme Gradient Boosting) is an implementation of gradient boosted decision trees, widely used for classification and regression tasks (T. Chen & Guestrin, 2016). As a classifier, XGBC builds an ensemble of shallow decision trees in sequence, where each successive tree is trained to correct the errors of the combined ensemble of previous trees. The “gradient” in gradient boosting refers to the use of gradients of a specified loss function to guide the improvement of the model; XGBoost extends this approach with efficient algorithms and additional features that make it extremely powerful in practice.

XGBoost’s training process is based on the gradient boosting framework, albeit with several enhancements. The model starts with an initial prediction (often a constant value). Trees are then added one by one. Each new tree is fit on the residuals or gradients of the loss, effectively learning how to improve upon the mistakes of the current model ensemble. For example, if the current ensemble is under-predicting the probability of a certain class for certain instances, the next tree will be trained to increase those predictions.

XGBoost uses both first-order (gradient) and second-order (Hessian) information of the loss function to make the training more efficient and accurate. Incorporating second-order information allows for better choices of tree split points and leaf weights.

Unlike basic boosting, XGBoost includes regularization terms in its objective to penalize model complexity. It controls the depth of trees, the minimum child weight (minimum data points in a leaf), and other aspects to prevent overfitting. After growing trees, XGBoost can prune branches that do not improve the objective, further simplifying the model.

Since XGBoost is engineered for speed, it leverages parallel processing for tree construction and can distribute computations across clusters for very large datasets. Its algorithm is optimized to gracefully handle sparse data and missing values, making it faster and more memory-efficient than many other implementations of boosting.

During training, hyperparameters such as the learning rate (shrinkage factor for each tree's contribution), number of trees, maximum tree depth, and regularization penalties are tuned to balance model fit and generalization. The final model is an ensemble where the prediction is the sum of contributions from all the constructed trees.

As a classifier, XGBC has been applied in areas like fraud detection, customer churn prediction, click-through rate prediction in advertising, disease diagnosis from medical data, and any problem with tabular data requiring high accuracy.

3.1.4 Federated learning

For the development of the FL system component, the chosen framework was Flower (Beutel et al., 2022). It is an open-source FL framework initially developed at the University of Oxford and now maintained by Flower Labs. The primary goal is to simplify and streamline FL across various machines. The framework offers a high-level API that reduces technical complexities, and it supports a variety of federated optimization algorithms, such as Federated Averaging, while also providing developers with the capability to implement custom algorithms. Built on a client-server architecture, it is compatible with popular ML frameworks such as PyTorch, TensorFlow, and Scikit-learn.

On the side of the server, the framework performs global operations (client selection, parameter configuration, gradient aggregation). On the side of the client, the framework performs model training and evaluation. It has components such as ClientProxy, for client-server communication, and the Virtual Client Engine, to execute FL simulations on the same physical device.

The framework supports heterogeneous workloads and devices, such as computers, edge devices such as Raspberry Pi and NVIDIA-Jetson, and Android and iOS mobile devices. Flower states it is capable of supporting scaling workloads to millions of clients.

3.1.5 RAG framework

For the implementation of the FL training logs analyzer, the framework used was EmbedChain (Taranjeet Singh, 2023). This open-source framework streamlines the creation of AI applications

that interact with document data. It supports data from various sources, such as PDFs, JSON files, text, video, and audio files. By default, EmbedChain can divide the data into chunks, perform vectorization, and store it on a database, which is then used for quick and accurate matching of user queries with relevant information. The developer can customize these parameters thoroughly, from the prompt to be used, the size of the chunks, the embeddings, and the vector database to be used by the system. Both these options showcase the flexibility and ease of implementation of the framework.

3.1.6 Python frameworks and libraries

Python was chosen as the programming language for development due to the vast ecosystem of libraries and frameworks in the ML area. Table 2 shows the Python libraries and frameworks used during the implementation.

Table 2. Python libraries and frameworks

Library/framework	Description
TensorFlow	ML and DL framework
PyTorch	ML and DL framework
Scikit-learn	ML framework
Matplotlib	Data analysis and visualization
Pandas	Data manipulation
NumPy	Numerical computation

Developed by Google, TensorFlow (Martín Abadi et al., 2015) is an open-source, versatile framework for building and deploying ML models, such as deep learning and reinforcement learning models. PyTorch (Paszke et al., 2019) is a popular deep-learning framework developed by MetaAI, due to its ease of use and flexibility it can be used for both academic research and real-world applications. Scikit-learn (Pedregosa et al., 2011) is a lightweight and user-friendly ML library, that provides implementations of various algorithms for tasks such as classification, regression, clustering, and dimensionality reduction. It also integrates well with other Python libraries, such as NumPy and matplotlib.

Regarding data analysis libraries, matplotlib (Hunter, 2007) is a powerful library that allows for the creation of static, interactive, and animated data visualizations in Python, offering a high degree of customization on the generation of graphs, charts, and plots. Pandas (McKinney, 2010) is a data manipulation and analysis library, mostly known for the DataFrame structure, allowing for data cleaning, aggregation, and transformation in a simple and easy-to-use manner. NumPy (Harris et al., 2020) is a numerical computation Python library, that supports large, multi-dimensional arrays and matrices as well as mathematical operations to operate on them.

3.2 Datasets

This section of the dissertation describes the datasets that were used for the training and testing of the ML models that were created during the case studies. While there's no specific area that the forecasting models must belong to, the datasets considered belong to the energy area. Due to one of GECAD's focuses being energy, the availability of data in this sector was one of the reasons that contributed to its choice.

3.2.1 Energy consumption dataset

To test some of the case studies, an energy consumption dataset was used, consisting of data between the 1st of January 2019 and the 30th of December 2022, registered in 15-minute intervals, totaling 84178 rows. The characteristics of each column are shown in Table 3.

Table 3. Energy consumption dataset column description

Column name	Data type	Description
Datetime	Datetime	The datetime of each row
Total	Float	The energy consumption, in Watts
Temp_x10	Float	The mean temperature, in Celsius

3.2.2 Community dataset

The community dataset (Goncalves et al., 2022) is an energy-related dataset consisting of generation and consumption data of 20 residences and a community. The dataset spans from 00:00 on January 1, 2020, to 23:45 on December 31, 2020, and each column represents a 15-minute interval. The data of each house is divided into a list of 10 appliances in addition to the flexible and rigid load values, as shown in Table 4.

Table 4. Community dataset column description

Column name	Data type	Unit	Description
Time	Datetime	Datetime	The timestamp indicates when the data was recorded.
Generation	Float	kWh	The value of energy generated.
Consumption	Float	kWh	The value of energy consumed.
AC1	Float	kWh	Energy consumed by Air conditioner (AC) 1.
AC2	Float	kWh	Energy consumed by AC 2.
AC3	Float	kWh	Energy consumed by AC 3.
AC4	Float	kWh	Energy consumed by AC 4.
Water heater	Float	kWh	Energy consumed by the water heater.
TV	Float	kWh	Energy is consumed by the television.
Microwave	Float	kWh	Energy consumed by the microwave.
Kettle	Float	kWh	Energy is consumed by the kettle.
Lighting	Float	kWh	Energy is consumed by lighting.
Refrigerator	Float	kWh	Energy consumed by the refrigerator.
Flexible Load	Float	kWh	Quantity of energy load that can be shifted over time.
Rigid Load	Float	kWh	Quantity of energy load that is time critical.

The values in the Consumption column are the sum of the consumption of the appliances present in the house. Additionally, the dataset is divided into 21 sheets. The H_01 to H_20 sheets represent each house that is a part of the community, and the Community sheet is the sum of the row values of all houses. Considering this, when used in case studies model making, the community sheet was the only one used. In addition, this sheet was altered and only the first three columns were picked: Time, Generation, and Consumption.

3.3 Data protection, security analysis, and ethical aspects

This section will introduce the aspects the project will have to consider regarding data protection, security, and ethics.

3.3.1 Data protection

Considering that one of the main components of the system is the usage of ML models, and the training of those requires a lot of data, this introduces challenges regarding the safeguarding of user information. Since the datasets used in the case studies are from the energy area, and the one described in section 3.2.2 contains appliance-specific information, this data could be used to create a user's energy consumption profile.

FL inherently promotes data privacy by decentralizing the training process. With FL, user data remains on local devices, and only model gradients (not raw data) are sent to the server for aggregation into a global model. This decentralized approach significantly reduces the risk of data breaches and unauthorized access.

Although differential privacy is not currently implemented in the system, the Flower framework supports both Central Differential Privacy (CDP) and Local Differential Privacy (LDP). These privacy-enhancing techniques can be easily integrated to further safeguard the data during the training process.

- CDP: In this approach, DP is applied at the server-side after the model updates from clients are aggregated. The server adds noise to the aggregated model updates to prevent sensitive information about each client's data from being inferred. While CDP requires trust in the central server, it ensures that the globally aggregated model does not leak information about individual clients' data. The process involves clipping the model updates and adding Gaussian noise to the aggregated model, ensuring that no single client's data can be easily extracted.
- LDP: LDP adds noise to each local model update before they are sent to the server, which prevents any sensitive information from being leaked during the communication process. This increases privacy and ensures that individual data points remain unidentifiable. By enabling LDP, the system ensures that even if a malicious actor intercepts the communication, they would not be able to extract sensitive information

from the gradients. This improves the privacy and security of the system by minimizing the risk of re-identification.

Regarding General Data Protection Regulation (GDPR) compliance, users are required to accept terms and conditions before using the application. While there is no direct mechanism for users to delete their data, developers can remove the user's data folder from the system, which contains all information they've uploaded to the system, effectively deleting the user's data.

3.3.2 Ethical aspects

Transparency is a key component of the system. Users have full access to the code generated by the MAS. All code is downloadable, and users can see the real-time conversation between the agents, providing transparency into the decision-making process of the system. The user can also view the current step of the system's process and the definition of the active MAS agents. This ensures that users are always aware of which part of the process the system is executing and the role of each agent in the pipeline.

Additionally, during phases where the LLM agent interacts with the user, the user has the opportunity to ask questions and gain a deeper understanding of concepts such as FL or ML. This interaction promotes learning and empowers users to engage with the system more effectively. Moreover, the LLM-based FL training logs analyzer component enables users to interact with the LLM to better understand the training logs generated during the FL training process. This helps users interpret the logs, enhancing their overall experience and comprehension of the system.

4 MASTerFlow

MASterFlow is a system that integrates MAS, LLMs and RAG to streamline the process of creating ML models and FL federations. The system is designed to be accessible to users with no coding expertise, enabling them to benefit from the power of ML and FL in their domains, such as renewable energy forecasting, healthcare, or financial analysis, while ensuring data privacy and security. Users are guided through the process of creating pipelines for ML models and FL federations via an intuitive UI that facilitates communication with LLM-powered agents. These agents assist users by answering questions and performing complex tasks autonomously, such as filling out configuration files with the user's parameters, integrating ML models into the FL framework, and analyzing training logs. MASTerFlow is divided into five main components:

- ML model creation component;
- Integration of ML in Flower Client component;
- FL federation creation component;
- FL training logs analyzer;
- User interface.

This section will start by providing an overview of the architecture and of the elements that MASTerFlow is composed of. Afterward, each component will be examined in greater detail, highlighting its role within the overall system.

4.1 Architecture

The architecture of the MASTerFlow system, as shown by Figure 6, is organized into four main components that interact with one another, each consisting of several sub-elements.

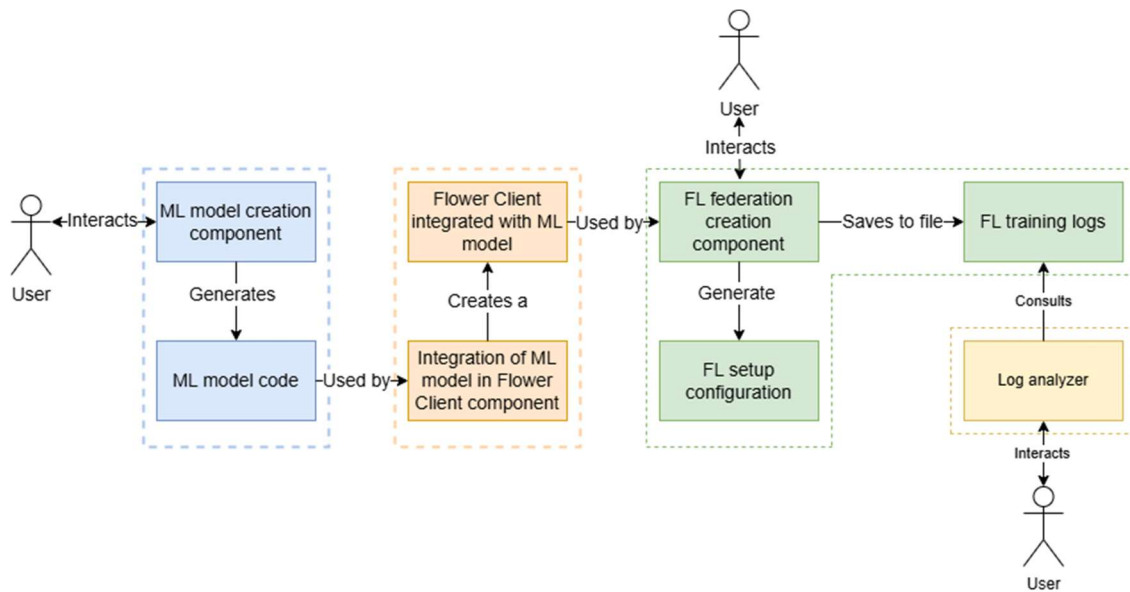


Figure 6. MASTERFlow components

Colored blue, the first component of the system, “ML model creation component”, consists of automated machine learning model creation. It is responsible for interacting with the user and filling up the configuration file for the machine learning model, and the user can opt to work exclusively with this component if desired. Alternatively, the generated ML model code can be integrated into FL simulation in subsequent stages.

For the integration, the system will integrate the ML model that was created into a Flower Client via the second component, colored orange and named: “Integration of ML model in Flower Client component”. This is a necessary step to ensure that the ML model code is compatible with the Flower framework and can be used during the FL training simulation.

Labeled “FL federation creation component” and colored green, this component autonomously generates the FL federation setup based on the configuration file and the Flower Client created in the previous step. It also produces the FL training logs, which can be analyzed later.

Finally, the system will employ the fourth component, the “log analyzer”, colored yellow. This component employs an LLM and RAG to analyze the training logs produced by the FL federation and respond to user queries, offering insights into the training process.

4.2 System model

In this section, the implementation will be described by expanding on each component described in the architecture at a time. The system allows the users to interact with it in three main modes. The first mode provides the user with the creation of an ML model only. The second mode allows the user to choose a previously created ML model that will be converted to a Flower Client and used for FL training. The third mode is the full process, where the user starts by creating a new ML model and the system converts and uses it in the FL training process.

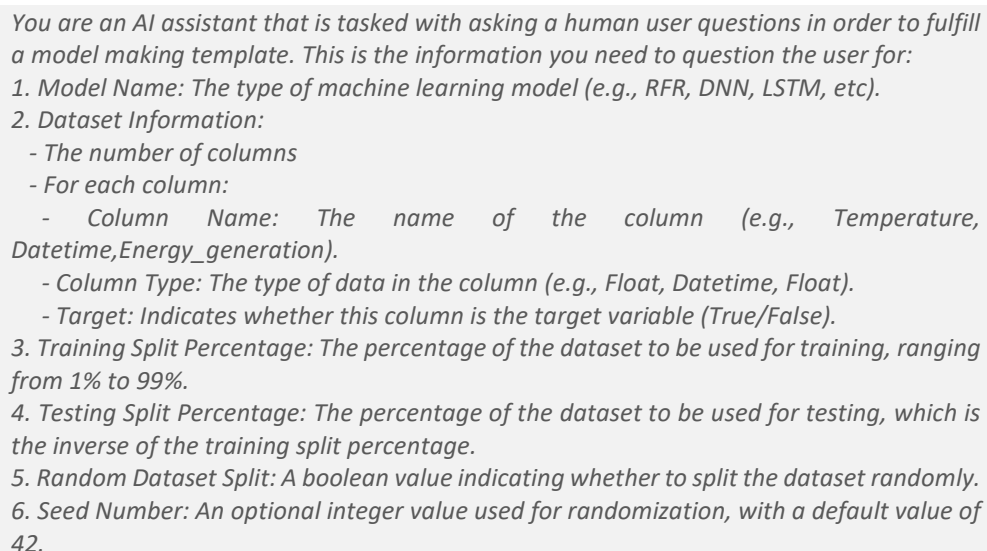
4.2.1 ML model creation component

The purpose of this first component of the system is to create an ML model while following the configurations the user has determined. By automating the ML model creation process after the user has finished completing the configuration file, the creation and usage of models becomes more accessible to users who might not have the technical expertise to create an ML model via programming.

To assist the users in filling out the configuration file, a Dialog LLM is engineered to dialog the user and obtain the configuration parameters that way. The following parameters are the ones present in the file:

- The type of machine learning model (e.g., RFR, DNN, LSTM).
- Information about the columns of the dataset:
 - The name of the column.
 - The type of data in the column.
 - A Boolean value indicates whether the column is a target variable.
- The percentage of the dataset to be used for training ranged from 1% to 99%.
- The percentage of the dataset to be used for testing is the inverse of the training split percentage.
- A Boolean value indicates whether to split the dataset randomly.
- An optional integer value is used for train-test split randomization, defaulting to 42.

The Dialog LLM was configured with a system message, shown in Figure 7, that provided instructions about its task and the necessary information that it must obtain from the users.



You are an AI assistant that is tasked with asking a human user questions in order to fulfill a model making template. This is the information you need to question the user for:

- 1. Model Name: The type of machine learning model (e.g., RFR, DNN, LSTM, etc).*
- 2. Dataset Information:*
 - The number of columns*
 - For each column:*
 - Column Name: The name of the column (e.g., Temperature, Datetime, Energy_generation).*
 - Column Type: The type of data in the column (e.g., Float, Datetime, Float).*
 - Target: Indicates whether this column is the target variable (True/False).*
- 3. Training Split Percentage: The percentage of the dataset to be used for training, ranging from 1% to 99%.*
- 4. Testing Split Percentage: The percentage of the dataset to be used for testing, which is the inverse of the training split percentage.*
- 5. Random Dataset Split: A boolean value indicating whether to split the dataset randomly.*
- 6. Seed Number: An optional integer value used for randomization, with a default value of 42.*

Figure 7. Dialog LLM system message – part one

The second part of the system message instructs the Dialog LLM on the correct formatting it should use for the configuration file, so the rest of the component can extract it and send the information to the MAS. The details of this formatting can be seen in Figure 8.

```
When you have all the information, return the information in the template:  
ACABADO  
MODEL_NAME: <model_name>  
NUMBER_COLUMNS: <number_of_columns>  
COLUMNS:  
- NAME: <column_name_1>  
TYPE: <column_type_1>  
TARGET: <boolean>  
- NAME: <column_name_2>  
TYPE: <column_type_2>  
TARGET: <boolean>  
- NAME: <column_name_3>  
TYPE: <column_type_3>  
TARGET: <boolean>  
TRAIN_SPLIT_PERCENTAGE: <train_split_percentage>  
TEST_SPLIT_PERCENTAGE: <test_split_percentage>  
RANDOM_DATASET_SPLIT: <boolean>  
SEED_NUMBER: <number>  
ACABADO_TEMPLATE
```

Figure 8. Dialog LLM system message – part two

The reason for the usage of the Portuguese word “ACABADO” is to contrast with the rest of the English text generated by the LLM. This way, it is guaranteed that the configuration file will be between a “reserved” keyword which will facilitate the extraction via regular expression.

Having an LLM assist with the filling out of the ML model configuration file allows for a more dynamic interaction with the user when compared with a manual collection of information. The LLM is able to adapt to diverse inputs, reducing the possibility of human error. The dynamic nature of the interaction also allows the component to scale to different datasets and use cases.

Due to the autonomous way the Dialog LLM operates, the filled-up candidate configuration file is extracted from the LLM’s messages via a Python script that uses a regular expression. Afterwards, a second LLM, labeled as Confirm configuration LLM, receives the configuration file and is responsible for evaluating if all the parameters have been correctly filled out. If something is missing, the Dialog LLM will be informed and will continue dialoguing with the user to obtain the lacking information. The system prompt in Figure 9, was used to instruct the Confirm configuration LLM in this task. The rest of the system prompt for the confirmation LLM is the same configuration file template from Figure 8.

```
You are an AI assistant tasked with confirming if the user's machine learning template has  
been filled with all the info. Reply with True if valid. Otherwise, reply with False:<reason>  
  
This is the mandatory template:  
[...]
```

Figure 9. Confirmation LLM system prompt

The usage of an LLM to first confirm the validity of the configuration file before it is used by the rest of the component serves as an additional layer of validation, catching missing or incorrect

parameters that could lead to errors in the generated ML model code and supporting the Dialog LLM by cross-checking the configuration file. The interaction between the dialog and the confirmation LLM serves as a self-correcting mechanism, reducing the possibility of potential user input errors resulting in an unusable configuration file.

When compared to a verification of the configuration file via code, the utilization of the LLM ensures that the system can adapt and scale to support more types of ML models and configuration files without requiring changes to the underlying confirmation code.

If the configuration file is confirmed as correct, a Python script will transform the configuration file into a natural language prompt. The script starts by transforming the column information into sentences, as shown in Figure 10, by iterating over the information and interpolating it into the placeholders in the pre-written strings.

```
FOR each column IN columns_info WITH index idx STARTING FROM 1 DO
  IF column.TARGET IS TRUE THEN
    SET target_status TO "is Target"
  ELSE
    SET target_status TO "not Target"
  END IF
  APPEND the following string TO columns_str:
  " {idx}. Column name: {column.NAME}, data type: {column.TYPE}, {target_status}"
END FOR
```

Figure 10. Function to replace the string placeholders in the prompt template

Afterward, the generated column strings, saved to a variable, are inserted into a placeholder that also includes information about the percentages and seeds used to split the dataset, shown in Figure 11.

```
SET dataset_template TO:
"""
The dataset has the following columns:
{columns_str.strip()}

Dataset split:
- Training: {parsed_template.TRAIN_SPLIT_PERCENTAGE}%
- Testing: {parsed_template.TEST_SPLIT_PERCENTAGE}%
- Random split seed: {parsed_template.SEED_NUMBER}
"""
```

Figure 11. Dataset percentage information in template to prompt function

The next step of prompt creation for the script, shown in Figure 12, is to call the `generate_few_rows_string`, which returns five rows of the dataset, already formatted to be inserted into the natural language prompt. This step also adds the file path of the dataset the user uploaded to the system into the prompt.

All the obtained information is combined into a single string that starts with the instruction to the LLM to write the ML model code following the specification. It is also advised to turn off

verbose model training which causes unnecessary token spending by printing a loading bar in text.

```
CALL generate_few_rows_string WITH:
  csv_file = uploaded_dataset_filepath
  number_of_rows = n_rows
STORE the results IN first_rows AND encoding

SET message TO:
"""
Write a Python script for the machine learning model following these specifications, turn
off verbose:
### Model Details
- Model: {parsed_template.MODEL_NAME}

### Dataset Information
- Filename: {uploaded_dataset_filepath}
- Encoding: {encoding}

### Column Details
{dataset_template}

### First {n_rows} Rows of the Dataset
{first_rows}
"""
```

Figure 12. Full prompt placeholder in the template to prompt function

Once all placeholder variables are filled, the prompt is complete, as shown in Figure 13.

```
The model to be coded is: LSTM
The dataset filename is:
E:\\model_making\\autogen_model_making\\minus_one_week_energy_consumption.csv
The dataset has the following columns:
Column number 1 with the name datetime, data type datetime and isn't Target
Column number 2 with the name total, data type int and is Target
Column number 3 with the name temp_x10, data type int and isn't Target
The percentage of the dataset used for training is: 80
The percentage of the dataset used for testing is: 20
The dataset should be randomly split
The seed for the random number generator is 33
First few rows of the dataset:
      datetime total temp_x10
2019-01-01 00:00:00  472     101
2019-01-01 00:15:00  478     100
2019-01-01 00:30:00  476     100
2019-01-01 00:45:00  473     100
2019-01-01 01:00:00  475     100
```

Figure 13. Example prompt of ML model making

The prompt is then fed to a Society of Mind (SoM) agent. This is an AutoGen framework-specific agent that leverages a multi-agent system group chat internally but appears as a single agent to the exterior. It was used to facilitate the retrieval of the Python model code from the group

chat’s messages. Since it was encapsulated inside a single agent, it was guaranteed that the SoM’s response would contain the completed model code generated by the group chat.

By using an AutoGen MAS, the process of creating the code for the ML model can be divided into specialized agents, each responsible for a specific task. This approach increases the modularity and the flexibility of the component while allowing each agent to have a single responsibility and focus on a single task.

The multi-agent system responsible for autonomously creating the machine learning model consists of the following five agents:

- The group chat manager, universal to all AutoGen group chats, is responsible for organizing the messages between all the participants as well as choosing the agent that will speak next;
- A user proxy agent, not supported by an LLM, exists to provide the group chat with code execution capability. The User proxy can be configured to execute code in either the local command line or in a Docker container;
- The dataset analyzer agent is responsible for providing a summary of the dataset, identifying patterns, or anomalies, suggesting features for the machine learning model and the preprocessing steps, and creating visualization mechanisms for the dataset, such as histograms, charts, and plots;
- The preprocessing agent is the one that generates the Python code necessary for handling missing values, encoding categorical values, and normalizing numerical features;
- The machine learning expert agent interprets the user’s natural language description of the model and creates the Python code necessary to initialize and configure the model with the correct parameters and specifications, also instructed to integrate their code with the Preprocessing agent’s code;
- A consolidator agent is used to guarantee that the code generated by the other agents gets integrated into a unified script.

All LLM-using agents belonging to the group chat were created using AutoGen’s ConversableAgent as a base. This class is the most customizable and serves as the base class for most types of agents created by AutoGen. Table 5 shows the system prompts used for each agent in the group chat.

Table 5. AutoGen model-making group chat system prompts

Agent name	System prompt
Dataset analyzer	<p>You are an expert in analyzing datasets. Your tasks include:</p> <ol style="list-style-type: none"> 1. Providing a summary of the dataset, including basic statistics for each column. 2. Identifying any patterns, trends, or anomalies present in the dataset. 3. Suggesting potential features for a machine learning model based on your analysis. 4. Visualizing important aspects of the dataset using appropriate plots (e.g., histograms, scatter plots). 5. Highlighting any preprocessing steps that may be necessary before feeding the data into a machine learning model.

	<p>Do not use <code>plt.show()</code>, use <code>plt.savefig("plot.png")</code> Do not use <code>seaborn</code>.</p> <p>Ensure your Python code is well-commented and easy to understand. If the code should be saved to a file before execution, include <code># filename: <filename></code> at the start of the code block. Provide clear instructions or comments within your code to indicate how other agents can integrate and use your analysis code.</p>
Preprocessing	<p>You are responsible for preprocessing a dataset. Given the first few rows, along with the column names and data types, your tasks include:</p> <ol style="list-style-type: none"> 1. Handle missing values appropriately (e.g., imputation, removal). 2. If they exist, encode categorical variables (e.g., one-hot encoding, label encoding). 3. If they exist, normalize numerical features (e.g., <code>MinMaxScaler</code>, <code>StandardScaler</code>). 4. Return the preprocessed dataset directly without creating new CSV files. <p>Ensure the Python code is well-commented and easy to understand. If the code should be saved to a file before execution, include <code># filename: <filename></code> at the start of the code block.</p> <p>Provide clear instructions or comments within your code to indicate how other agents can integrate and use your preprocessing code. Do not use <code>plt.show()</code>, use <code>plt.savefig("plot.png")</code></p> <p><code># filename: preprocessing_code.py</code></p>
Machine learning expert	<p>You are an expert Python developer specializing in machine learning. Your task is to generate Python code for a machine learning model based on the provided description. Follow these steps:</p> <ol style="list-style-type: none"> 1. Interpret the user's natural language description of the model. 2. Extract and define the necessary parameters and specifications. 3. Generate clear and concise Python code to initialize and configure the model. 4. Use the preprocessing code generated by the Preprocess Agent (if applicable) to prepare the data before training the model. Integrate your code with the preprocessing code seamlessly. 5. Include comments to explain key sections and choices. <p>The code must be formatted within triple backticks <code>```</code>. If the code should be saved to a file before execution, include <code># filename: <filename></code> at the start of the code block. Do not use <code>plt.show()</code>, use <code>plt.savefig("plot.png")</code></p> <p><code># filename: ml_model_code.py</code></p>
Consolidator	<p>Your role is to consolidate and integrate Python code and analysis results generated by other agents into a unified and cohesive script. Ensure the final code includes:</p> <ol style="list-style-type: none"> 1. Detailed dataset analysis and insights from the Analyzer Agent (<code>dataset_analysis.py</code>). 2. Preprocessing steps from the Preprocess Agent (<code>preprocessing_code.py</code>). 3. Machine learning model setup from the Machine Learning Expert Agent (<code>ml_model_code.py</code>). 4. Clear and concise comments explaining each section and its purpose. <p>Ensure the Python code is well-commented, follows best practices, and is ready for execution. Do not use <code>plt.show()</code>, use <code>plt.savefig("plot.png")</code></p> <p><code># filename: final_script.py</code></p>

Once the MAS has finished running, the output of the component is a code file with the implementation of the ML model code according to the user specifications defined in the configuration file. The diagram presented in Figure 14 demonstrates how the numerous elements of the ML model creation component are connected, and how they interact.

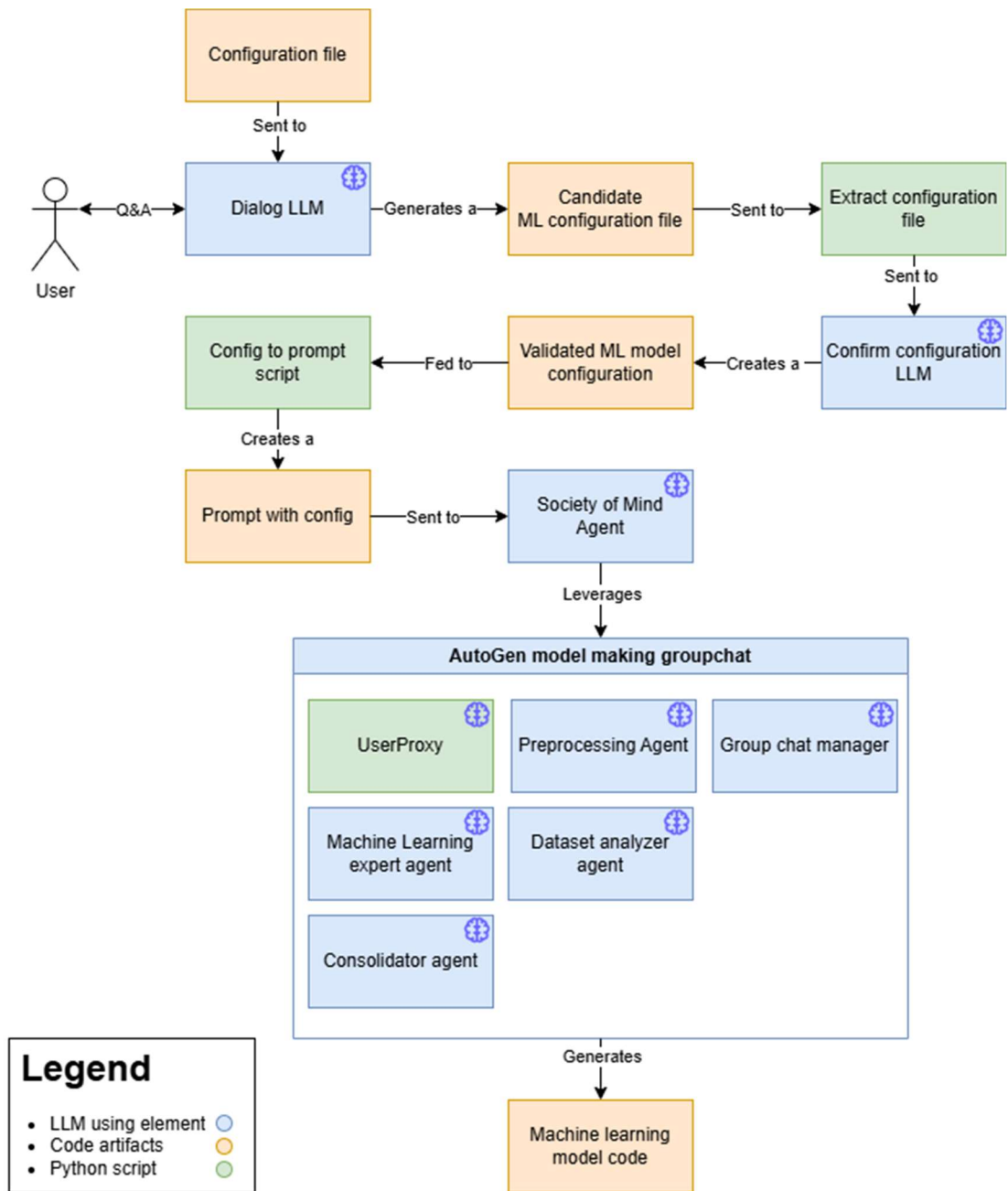


Figure 14. ML model creation component

4.2.2 Integration of ML model in Flower Client component

The purpose of this component is to bridge the ML model creation and the FL federation creation components, displayed in Figure 6. While a normal ML model can be trained individually, FL models require a specific structure to ensure that the models trained collaboratively across different clients and aggregated into a global model are compatible. This component ensures this integration, allowing the generated ML model to be used in a federated setting.

The reason this component and the previous one, ML model creation component, are separated is to allow for modularity and flexibility, enabling users to continue into the integration step after creating a ML model or choose previously built models and adapt them for FL tasks. This decoupling approach simplifies the system and makes it more adaptable to different needs and scalable to changes. As shown in Figure 15, this component takes the Python code of the generated ML model as input and generates a Flower Client integrated with the ML model code as output.

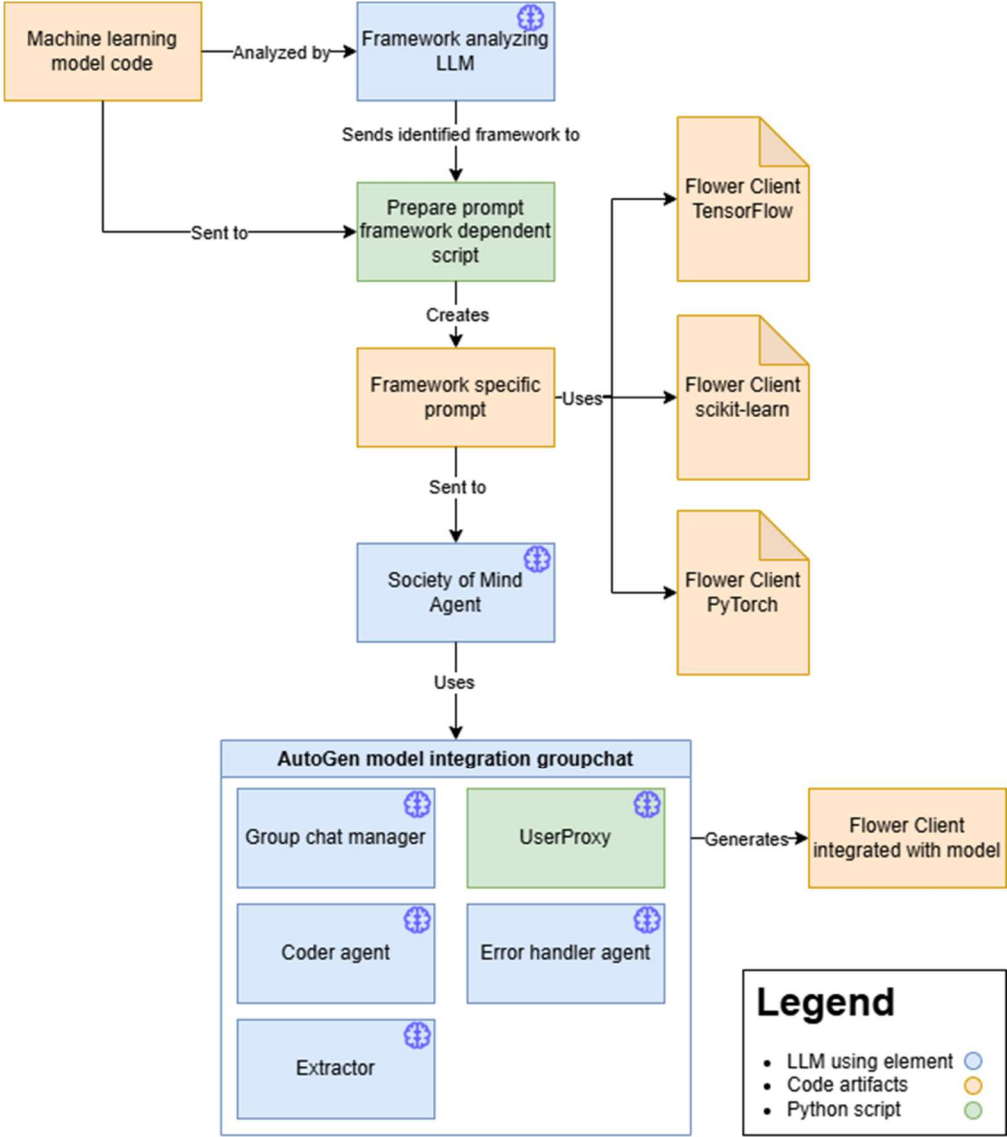


Figure 15. Integration component diagram

The ML model code is analyzed by an LLM to determine the machine learning framework that was used to generate the code. This way, the integration of a previously created ML model into an FL environment becomes seamless, avoiding a manual adjusting of code. This is a crucial step

because each framework has its requirements for integrating with the Flower Client, while avoiding a less dynamic code-based detection system. At the time of writing, the system supports three ML frameworks, TensorFlow, scikit-learn, and PyTorch. Figure 16 shows the system message used to instruct the framework analyzing LLM.

```
Analyze the user's machine learning code and determine the library and framework used to
build the model.
Reply with one of the following options:
TensorFlow
scikit-learn
PyTorch
none: <the reason why you can't find the framework>
```

Figure 16. Framework analyzing LLM system message

Both the Python code of the ML model and the framework identifier are sent to a Python script that builds the prompt for the AutoGen agents. The script accesses information files containing a baseline framework-dependent Flower Client, as well as information about the inputs and outputs of the various functions that are used by the Flower Client class, building the prompt with that information and the ML model code. By injecting the framework specific Flower Client information into the prompt, the success rate of the integration done by the MAS improves, as it avoids hallucinations due to the LLMs not having information about the Flower framework in the training data, as well as possible confusions that could appear because while the code of each framework’s Flower Client is similar, it is not interchangeable.

This framework-specific prompt is then fed to AutoGen’s Society of Mind agent, which will utilize the AutoGen model integration group chat to create the Flower Client code integrated with the ML model. The AutoGen integration group chat is defined by the following agents:

- The AutoGen universal group chat manager, to handle agent communication and next speaker choice;
- The UserProxy agent, which is not powered by an LLM, provides the group chat with the capacity for executing code;
- Coder agent, responsible for generating the code that integrates the ML model code with the Flower Client class;
- Error handler agent, that monitors the code outputs for potential issues and suggests fixes;
- The extractor agent saves a version of the Flower Client code to a file.

Table 6 displays the system prompts to instruct each of the agents in their tasks. As with the previous group chat, these agents were also built on top of the ConversableAgent class.

Table 6. AutoGen integration group chat

Agent name	System prompt
Coder	You are an expert in Python and the Flower federated learning framework. Your role is to:

	<ol style="list-style-type: none"> 1. Review the provided Flower federated learning code or configuration for issues and improvements. 2. Ensure the implementation adheres to federated learning principles and best practices. 3. Provide feedback and suggest optimizations to enhance compatibility, performance, and maintainability. 4. Explain any necessary modifications and include comments to clarify your changes.
Error handler	<p>Your role is to handle errors during the development process. You should:</p> <ol style="list-style-type: none"> 1. Monitor the outputs for potential issues. 2. Provide clear feedback and suggestions for fixes. 3. Ensure that any generated code meets expected standards and runs without errors. <p>Your feedback should be constructive and detailed.</p>
Extractor	<p>Extract the completed FlowerClient class from the text and save to file.</p> <p>If you want the user to save the validated code in a file, put # filename: <filename> inside the code block as the first line.</p>

4.2.3 FL federation creation component

The FL federation creation component is responsible for automating the setup of a FL federation. As shown in Figure 17, the component receives inputs that include the user’s level of expertise, the filled-up FL configuration file, and a Flower Client integrated with the ML model, the output from the previous component.

FL is a complex process, involving decentralized training across multiple clients while ensuring that data privacy and model integrity are maintained. By automating the creation of FL federations, the process becomes simpler for users, regardless of their experience with FL, while the system ensures that all required parameters for the FL simulation are accurately configured.

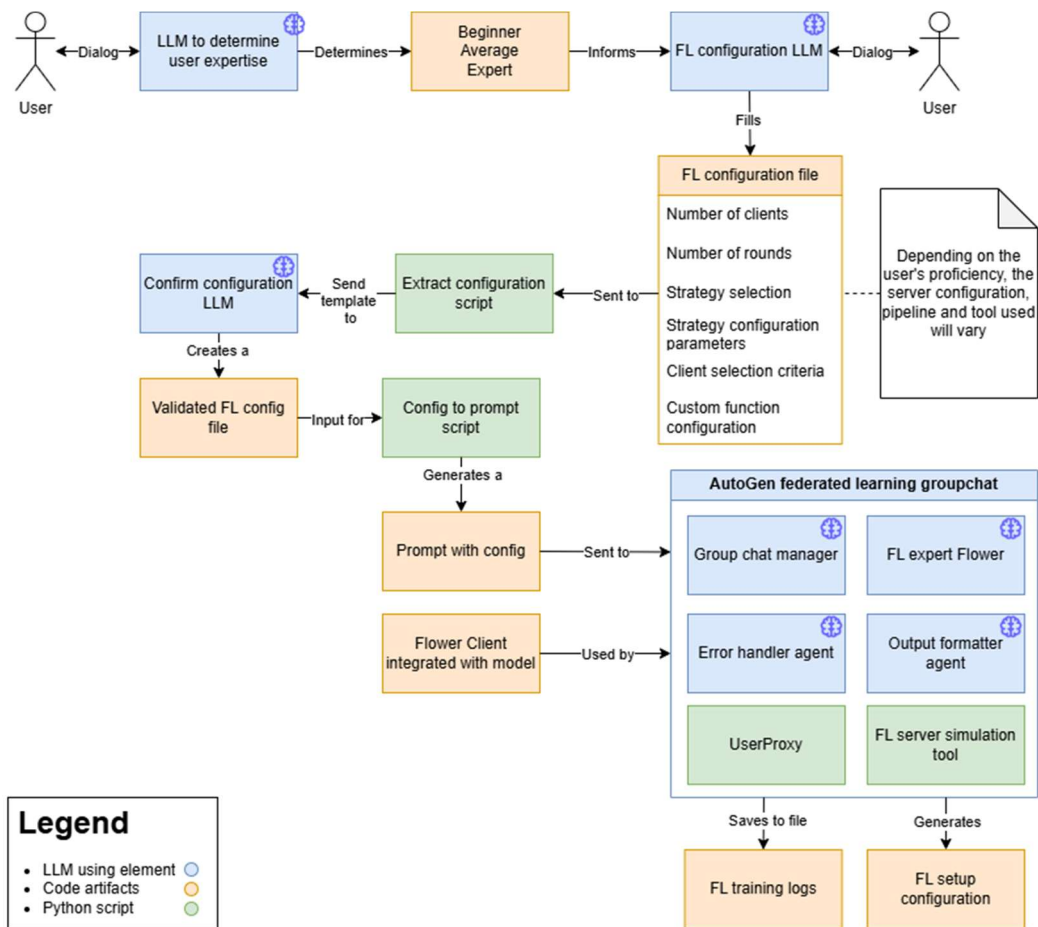


Figure 17. Automated FL federation component

Given the intricacies of FL, the system is designed to assess the user’s expertise regarding FL, specifically the Flower FL framework chosen for implementation. This allows the system to provide configuration options tailored to the user’s proficiency, ensuring a more personalized experience. To determine a user’s expertise, an LLM is tasked with dialoguing with the user to gather information and assess their knowledge. Based on their responses, the system assigns one of three expertise levels:

- **Beginner:** Little to no knowledge of FL;
- **Average:** Prior experience using FL;
- **Expert:** Regular use of FL or a deep understanding of its principles.

The system message that guides the LLM in determining the user’s proficiency is shown in Figure 18. After the LLM assesses the user’s expertise, it assigns them a proficiency level, which influences the available FL strategies and configuration options.

*You are an AI assistant part tasked with figuring out the user's proficiency with federated learning using the Flower framework.
Assign the user one of the following: beginner, average, expert
The user can be a beginner, with small to no knowledge of federated learning.
An average user, that has used federated learning before
An expert, that regularly uses or is well-aware of how federated learning works.*

```
When you have gathered all the necessary information, return it in the following template:  
USER_PROFICIENCY_TEMPLATE  
<user_proficiency_here>  
USER_PROFICIENCY_TEMPLATE_END
```

Figure 18 User expertise determination LLM system prompt

Once the user's proficiency is determined, a second LLM is responsible for guiding them through the process of filling out the configuration file for the FL simulation. Regardless of the user's expertise, the following core parameters must be configured:

- **Num_clients**: The number of clients in the simulation.
- **Fraction_fit**: The fraction of clients used during training.
- **Fraction_evaluate**: The fraction of clients used during validation.
- **Min_fit_clients**: Minimum number of clients required during training.
- **In_evaluate_clients**: Minimum number of clients required during validation.
- **Min_available_clients**: Minimum number of available clients in the system.
- **Accept_failures**: Whether to accept rounds with failures.
- **Num_rounds**: The number of training rounds in the simulation.

The user's expertise level influences the strategies available to them. Beginner users are restricted to using the Federated Averaging strategy, which aggregates the local model gradient updates via averaging. Average expertise users gain access to three strategies: FedAvg, FedAvgM and FedMedian, with the additional option to customize the client selection criteria in the `configure_fit` function. Meanwhile, expert users have access to all available strategies, including FedTrimmedAvg and Krum, and can also view and customize the `configure_fit` function directly.

The system ensures that the LLM does not show code to users with lower expertise levels, instead translating their natural language inputs into function code. For expert users, the system allows them to directly engage and customize the code of the `configure_fit` function.

Once the configuration file is filled up, it undergoes confirmation by another LLM to ensure that all required information is present and valid, similar to the ML model creation component, in Section 4.2.1. If the configuration is incomplete or erroneous, the LLM will indicate what needs to be fixed, ensuring that only valid configurations proceed to the next step. The confirmation LLM operates using a system prompt, as shown in Figure 19, to verify the configuration file's integrity.

```
You are an AI assistant tasked with confirming if the user's federated learning  
template has been filled with all the info. Reply with True if  
valid. Otherwise, reply with False:<reason>
```

```
This is the mandatory template:  
strategy = <string>  
num_clients = <int>  
fraction_fit = <float>  
fraction_evaluate = <float>
```

```
min_fit_clients = <int>
min_evaluate_clients = <int>
min_available_clients = <int>
accept_failures = <bool>
num_rounds = <int>
HAS_CUSTOM_FUNCTION= <boolean>
```

Figure 19. Confirm FL configuration LLM system prompt

The last parameter, the Boolean HAS_CUSTOM_FUNCTION, does not exist when the LLM confirms the configuration file of a beginner user. After the configuration is confirmed, the parameters are fed into a Python script that constructs a natural language prompt for the MAS group chat. This prompt includes all the key parameters common to all expertise levels, including strategy string, client number, and number of rounds. These parameters are shown in Figure 20.

```
User strategy_string:
{strategy_string}
User num_clients:
{num_clients}
User num_rounds:
{num_rounds}
User client_class_path:
{client_class_path}
User SESSION_ID:
{session_id}
User log_filepath:
{log_filepath}
```

Figure 20. FL configuration file prompt base parameters

Additionally, all prompts, independent of the user's expertise level, contain instructions on how to use the tool provided to the group chat for FL simulation. The prompt provides the group chat agent's LLMs with the name of the tool, as shown in Figure 21, and the conversion information between the configuration file parameters and the tool parameters.

```
Use v2_server_tool to run the federated learning server and correct any errors that appear.
1. strategy_string should be passed as the tool parameter strategy_string
2. num_clients should be passed as the tool parameter num_clients
3. num_rounds should be passed as the tool parameter num_rounds
4. client_class_path should be passed as the tool parameter client_class_path
5. SESSION_ID should be passed as the tool parameter session_id
6. log_filepath should be passed as the tool parameter log_filepath
```

Figure 21. FL model-making tool usage instructions

In the case of average or expert users, if they have customized the `configure_fit` function, the system will add this custom function to the prompt, along with an example of how to call the tool with the customized parameters, as shown by the function in Figure 22.

```

IF user_proficiency IS "average" OR "expert" AND has_custom_function IS TRUE THEN
  ADD the following to the prompt:
  """
  User has customized configure_fit. Here's the function:
  {custom_function}
  7. The customized configure_fit function should be passed as the tool parameter
  configure_fit_str
  """
END IF

SET example_call TO:
  "v2_server_tool(strategy_string=strategy_string, num_clients=num_clients,
  num_rounds=num_rounds, client_class_path=client_class_path, session_id=session_id,
  log_filepath=log_filepath)"

IF user_proficiency IS "average" OR "expert" AND has_custom_function IS TRUE THEN
  APPEND ", configure_fit_str=configure_fit" TO example_call
END IF

APPEND ")" TO example_call

ADD the following to the prompt:
  "Example:
  {example_call}"

RETURN prompt

```

Figure 22. FL model-making function call construction

The creation of the FL federation according to the configuration defined by the user is handled by a MAS, which is responsible for executing the configuration, running the simulation, and managing errors. The MAS consists of the following agents:

- **Group Chat Manager Agent:** This agent handles message organization and decides which agent speaks next in the group chat.
- **User Proxy Agent:** A non-LLM agent that provides the group chat with the capability to execute code.
- **Federated Learning Expert Agent:** This agent creates the federated learning code, ensuring that the configuration aligns with federated learning principles.
- **Error Handler Agent:** It monitors the execution of the simulation, identifies issues, and offers feedback to ensure that the simulation runs correctly.
- **Output Formatter Agent:** Responsible for formatting the final configuration and output data, ensuring that everything is structured for extraction.

Table 7 shows the system messages that were used to define the role of each agent in the FL-making group chat. Like the previously described group chats, these agents were constructed by using the AutoGen ConversableAgent class.

Table 7. FL model-making group chat system prompts

Agent name	System prompt
FL expert	<p>You are an expert Python developer specializing in federated learning with the Flower framework.</p> <p>The user will provide a federated learning configuration or setup, and your task is to:</p> <ol style="list-style-type: none"> 1. Analyze the provided configuration or code for errors, inefficiencies, or missing components. 2. Use the `v2_server_tool` to validate the user's configuration, ensuring it aligns with federated learning principles and the Flower framework. 3. Provide feedback on any issues found by the `v2_server_tool` and offer suggestions for improvement based on the tool's output. 4. Explain the results of the validation, including any changes the user needs to make to ensure proper client-server interaction in federated learning. 5. Include comments to clarify the validation process and highlight important adjustments. <p>The corrected configuration must be formatted within triple backticks ```. If you want the user to save the validated configuration in a file, put # filename: <filename> inside the code block as the first line. # filename: federated_learning_flower_config.py</p>
Error handler	<p>Your role is to handle errors during the development process. You should:</p> <ol style="list-style-type: none"> 1. Monitor the outputs for potential issues. 2. Provide clear feedback and suggestions for fixes. 3. Ensure that any generated code meets expected standards and runs without errors. <p>Your feedback should be constructive and detailed.</p>
Output formatter	<p>You are responsible for formatting the user defined configurations into a file: Create code to save strategy_string, num_rounds and configure_fit to a .txt file. If you want the user to save any validated configuration in a file before executing it, put # filename: <filename> inside the code block as the first line.</p>

These agents collaborate to generate the necessary configuration for the FL federation and run the v2_server_tool to simulate federated learning training with multiple clients. This collaborative effort ensures that the federation setup is fully automated, consistent, and adheres to the principles of federated learning.

The v2_server_tool receives the configuration parameters and runs the FL simulation, utilizing the Flower Client generated in the previous component. The tool takes as input the FL strategy, the number of clients, rounds, the custom configure_fit function (if applicable), the user session id and the log file paths, executing the simulation and generating logs that capture the training process. These logs can be analyzed by other components to evaluate the simulation's performance.

By using LLMs for expertise profiling, configuration gathering and validation, the system ensures an intuitive and efficient setup process, while also allowing advanced users to engage with and customize the FL configuration. The use of a MAS further optimizes the configuration and simulation execution, ensuring that the entire process is automated and scalable.

4.2.4 FL training logs analyzer

The FL training logs analyzer component in the system utilized RAG to provide an LLM with context-specific information, enabling it to analyze FL training logs and deliver accurate, context-aware answers to users regarding the FL training process. The diagram in Figure 23 provides a brief overview of the system component.

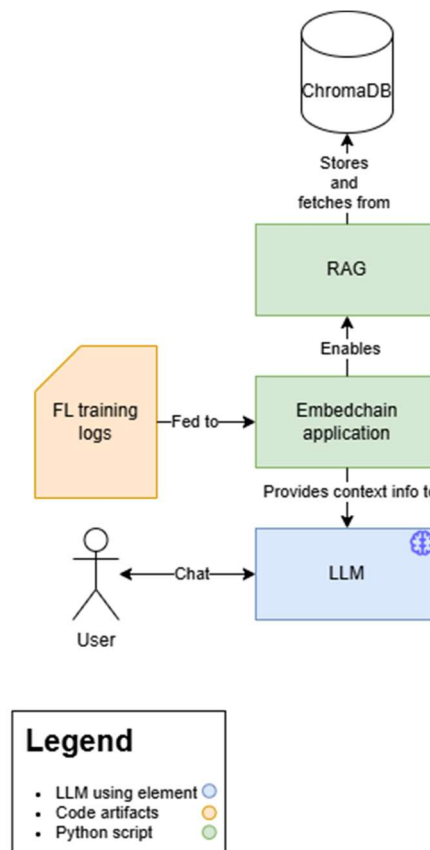


Figure 23. FL training logs analyzer component

Once a FL simulation is executed via the Flower framework, logs are generated that track the training process, including the simulation’s characteristics and the per-round performance details. The purpose of this component is to allow the system to analyze the logs and enable users to interact with them in a natural, conversational manner. This is useful for troubleshooting, performance tracking, or understanding the evolution of the model during training. The default format of the Flower framework logs was used, which contains the following parameters:

- **Identifier:** A unique ID for the session and log entries.
- **Log message level:** Indicates whether the log is an informational message (INFO), a debug message (DEBUG), or a warning (WARNING).
- **Timestamp:** The date and time the log entry was created.
- **Filename and line number:** Where the log message originated in the code.
- **Log message:** The content or action described by the log entry.

The logs follow a format string like: "%(levelname)s %(name)s %(asctime)s | %(filename)s:%(lineno)d | %(message)s". An example of a generated log is shown in Figure 24, which displays the logs for an FL session across multiple rounds, including metrics such as loss per round, client sampling, and evaluation results.

```
47e5b628-2024_12_04_12_13_36-3 | INFO flwr 2024-12-04 12:37:45,132 | server.py:113 | [ROUND 3]
47e5b628-2024_12_04_12_13_36-3 | INFO flwr 2024-12-04 12:37:45,132 | server.py:226 |
configure_fit: strategy sampled 5 clients (out of 5)
47e5b628-2024_12_04_12_13_36-3 | INFO flwr 2024-12-04 12:40:39,957 | server.py:240 |
aggregate_fit: received 5 results and 0 failures
47e5b628-2024_12_04_12_13_36-3 | INFO flwr 2024-12-04 12:40:39,961 | server.py:178 |
configure_evaluate: strategy sampled 5 clients (out of 5)
47e5b628-2024_12_04_12_13_36-3 | INFO flwr 2024-12-04 12:43:29,873 | server.py:192 |
aggregate_evaluate: received 5 results and 0 failures
47e5b628-2024_12_04_12_13_36-3 | INFO flwr 2024-12-04 12:43:29,873 | server.py:496 |
47e5b628-2024_12_04_12_13_36-3 | INFO flwr 2024-12-04 12:43:29,873 | server.py:497 |
[SUMMARY]
47e5b628-2024_12_04_12_13_36-3 | INFO flwr 2024-12-04 12:43:29,874 | server.py:498 | Run finished
3 round(s) in 816.61s
47e5b628-2024_12_04_12_13_36-3 | INFO flwr 2024-12-04 12:43:29,874 | server.py:500 |
History (loss, distributed):
47e5b628-2024_12_04_12_13_36-3 | INFO flwr 2024-12-04 12:43:29,878 | server.py:500 |
round 1: 0.02667810209095478
47e5b628-2024_12_04_12_13_36-3 | INFO flwr 2024-12-04 12:43:29,878 | server.py:500 |
round 2: 0.02679949440062046
47e5b628-2024_12_04_12_13_36-3 | INFO flwr 2024-12-04 12:43:29,879 | server.py:500 |
round 3: 0.0268742386251688
47e5b628-2024_12_04_12_13_36-3 | INFO flwr 2024-12-04 12:43:29,879 | server.py:501 |
47e5b628-2024_12_04_12_13_36-3 | DEBUG flwr 2024-12-04 12:43:29,882 | run_serverapp.py:89 |
ServerApp finished running.
47e5b628-2024_12_04_12_13_36-3 | DEBUG flwr 2024-12-04 12:43:29,882 | run_simulation.py:349 |
ServerApp finished running.
47e5b628-2024_12_04_12_13_36-3 | DEBUG flwr 2024-12-04 12:43:29,882 | run_simulation.py:353 |
Triggered stop event for Simulation Engine.
47e5b628-2024_12_04_12_13_36-3 | DEBUG flwr 2024-12-04 12:43:30,888 | ray_actor.py:450 |
Terminated 6 actors
47e5b628-2024_12_04_12_13_36-3 | DEBUG flwr 2024-12-04 12:43:31,820 | raybackend.py:193 |
Terminated RayBackend
47e5b628-2024_12_04_12_13_36-3 | DEBUG flwr 2024-12-04 12:43:31,821 | run_simulation.py:455 |
Stopping Simulation Engine now.
```

Figure 24. Example of Flower Framework logs

The logs provide detailed records of the training process, including information like:

- **Round-wise loss:** Information about the global model's loss per round
- **Client Activity:** Details about the clients sampled for each training round.
- **Failures:** Information about failures in aggregation or evaluation.

Figure 24 also shows log entries such as:

- [ROUND 3] indicating the start of the third round.

- `aggregate_fit`: received 5 results and 0 failures describing the outcome of an aggregation step.
- History (loss, distributed) listing the loss values for each training round.

These logs also contain debugging messages that indicate the simulation's status, such as "ServerApp finished running" or "Terminated RayBackend", which inform users about the progress or termination of the simulation.

Once the FL training process has terminated, the logs are saved to a text file. User interaction with the logs is optional, and the user is first prompted to decide if they wish to analyze the logs. If the user chooses to proceed, the logs are processed using EmbedChain and RAG techniques to make them searchable and interpretable by the system. The RAG technique combines retrieval from a database with the generative power of an LLM, allowing the system to provide relevant answers based on the logs.

The logs are divided into chunks using the EmbedChain application, which converts the chunks into vector representations using an embedding model and stores these vectors in a ChromaDB vector database for efficient retrieval during queries.

The yaml file that configures the EmbedChain application can be divided into four parts: the LLM, embedder, chunker, and vectordb. Figure 25 shows the LLM configuration parameters fed to the application.

```

llm:
  provider: openai
  config:
    model: 'gpt-4o'
    prompt: |
      History of your talk with the user:
      $history

      Use the following pieces of context to answer the query at the end.
      If you don't know the answer, just say that you don't know, don't try to make up an answer.

      $context

      Query: $query

      Helpful Answer:
      system_prompt: |
        You are a helpful and knowledgeable AI assistant, that answers questions about federated learning
        logs.
        Keep your answers relevant and professional, but friendly, adapting to the user's tone and needs.
        If additional context or suggestions could enhance the user's understanding or help them reach their
        goals, feel free to provide it.
        Focus on accuracy and clarity, and keep explanations straightforward, guiding the user step-by-step
        if needed.

```

Figure 25. Embedchain app configuration - LLM parameters

The provider is defined as “openai” since the model used is GPT-4o. The LLM uses a custom system prompt that guides it to act as an expert assistant knowledgeable in FL training logs. The prompt also contains the following fields:

- The conversation history with the user to ensure continuity in the chat;
- An instruction for the LLM to not make up answers and reply that it does not know the answer instead;
- The context, which is retrieved from the vector database via RAG;
- The user’s query.

Additionally, the embedder is also defined, as shown in Figure 26. The embedder converts the chunks of log data into vector form using the “text-embedding-3-large” model from OpenAI. This model is selected because of its ability to handle large embeddings (up to 3072 dimensions), which is necessary for accurately encoding complex log data.

```
embedder:  
provider: openai  
config:  
model: 'text-embedding-3-large'
```

Figure 26. Embedchain app configuration – embedder parameters

As shown in Figure 27, the details of each chunk’s constitution are also provided by the configuration file. The chunker is responsible for breaking the log file into manageable text chunks. Parameters such as “chunk_size”, “chunk_overlap” and “min_chunk_size” determine how the text is split to ensure efficient processing. Additionally, the “length_function” indicates which function is to be used to calculate chunk length.

```
chunker:  
chunk_size: 4000  
chunk_overlap: 500  
length_function: 'len'  
min_chunk_size: 1000
```

Figure 27. Embedchain app configuration – chunk parameters

As displayed in Figure 28, a Chroma vector database is used to store the vectorized log chunks. The logs are saved under a collection named “log-collection”, which can be queried to retrieve relevant chunks based on the user’s query.

```
vectordb:  
provider: chroma  
config:  
collection_name: 'log-collection'
```

Figure 28. Embedchain app configuration – vector database configuration

Once the log data is embedded and stored, the system is ready to engage with the user. The LLM that has RAG capabilities is tasked, as shown by the prompt in Figure 29, with introducing itself to the user by outputting an overview of the federated learning training process.

"Provide me with an overview of the federated learning training process."

Figure 29. Log analyzer introduction prompt

Both the introductory message and further answers from the LLM are improved via RAG. The system uses it to retrieve information relevant to the user's prompts from the logs and supply the LLM with the context needed to generate a helpful answer. The LLM then processes the user query alongside the retrieved context, offering answers that are both accurate and relevant to the specific training session. This way, RAG ensures that the LLM provides answers based on the logs' actual contents, improving the response's accuracy and relevance. Overall, this component significantly enhances the usability and accessibility of FL simulations, allowing users to gain deeper insights into their training process with ease.

4.2.5 User interface

The user interface (UI) developed for MASTerFlow was created based on the interfaces previously developed for two different projects: FeDis (Barbarroxa et al., 2023) and Web Meta-ERM (Almeida et al., 2023). The former was an interface for a framework of the same name (FeDis), which integrated FL with a blockchain distributed ledger as the communication method between clients and the server. It allowed the users to manage and visualize the status of clients and server nodes from the dashboard.

In Web Meta-ERM, a dashboard was developed so that users could interact with a MATLAB platform without needing to install the program on their machines. The UI allowed users to authenticate and authorize themselves in the system, upload case studies and metaheuristic algorithms related to the energy resource management area and execute simulations with them. Additionally, administrators of the platform possessed tools to manage the other users. Figure 52, present in the Annex, shows a screenshot of Web Meta-ERM's UI. MASTerFlow's UI is composed of five main components, with the layout shown in Figure 30: the session folder information, the current MAS information, the MASTerFlow component stepper, the LLM chatbox, and the current code window.

The first component of the UI provides the user with information about the content of the session folder where the ML and FL code files, the dataset, and the FL training logs are stored. As shown in Figure 31, a button for folder selection is displayed. Due to security measures of the File System Access API (*File System API - Web APIs | MDN, 2024*), the user of the UI has to be the one choosing a folder and authorizing the system to access the file system.

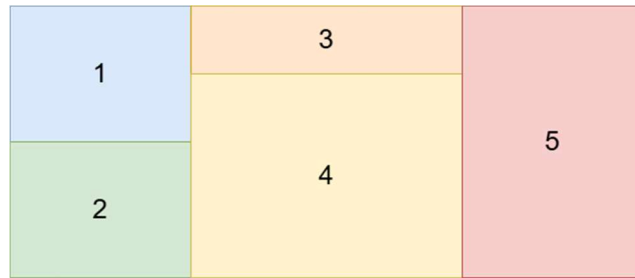


Figure 30. MAStErFlow UI layout

The second component contains information about the current MAS being executed in MAStErFlow. The user can mouse over any agent of the group chat and a pop-up with a description of the agent will appear, as shown in Figure 32.

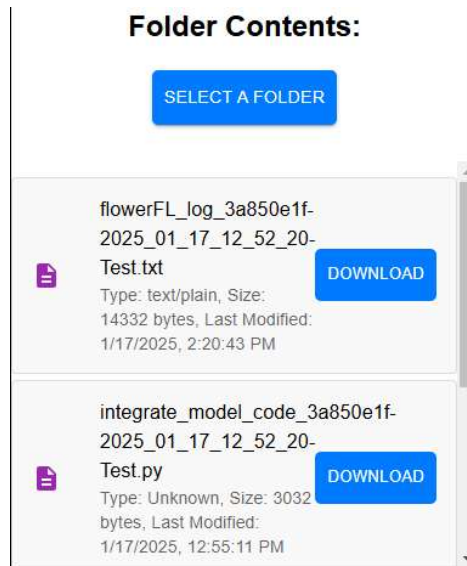


Figure 31. UI folder contents visualization

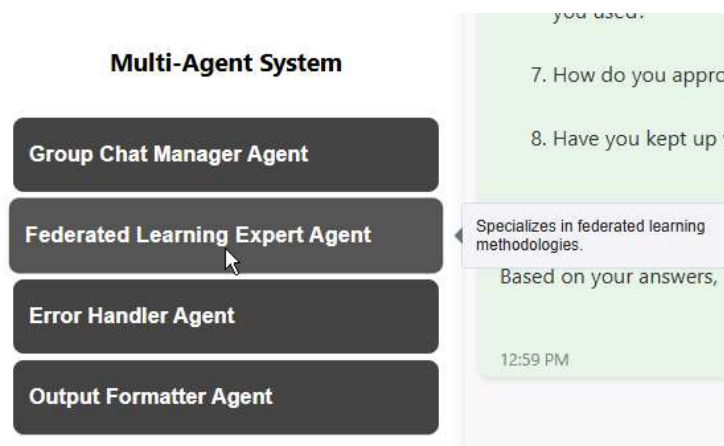


Figure 32. Current multi-agent system information in UI

As shown in Figure 33, the third component displays information about the current MASTERFlow session, the user ID, session ID, and directory are displayed in the header. Additionally, a stepper element shows which step of the MASTERFlow chain of MAS the system is currently at, which steps have been completed, and the future ones.

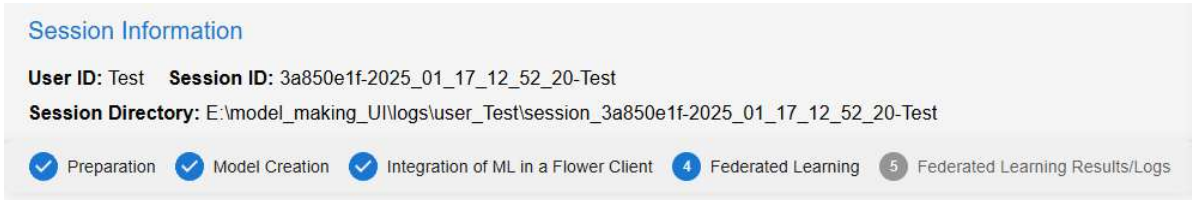


Figure 33. UI component stepper

The fourth and main component of the UI is the chat box, shown in Figure 34, where the user interacts with the various LLMs that ask questions about the configuration of ML and FL models, and answer user questions about the domains and the FL training logs.

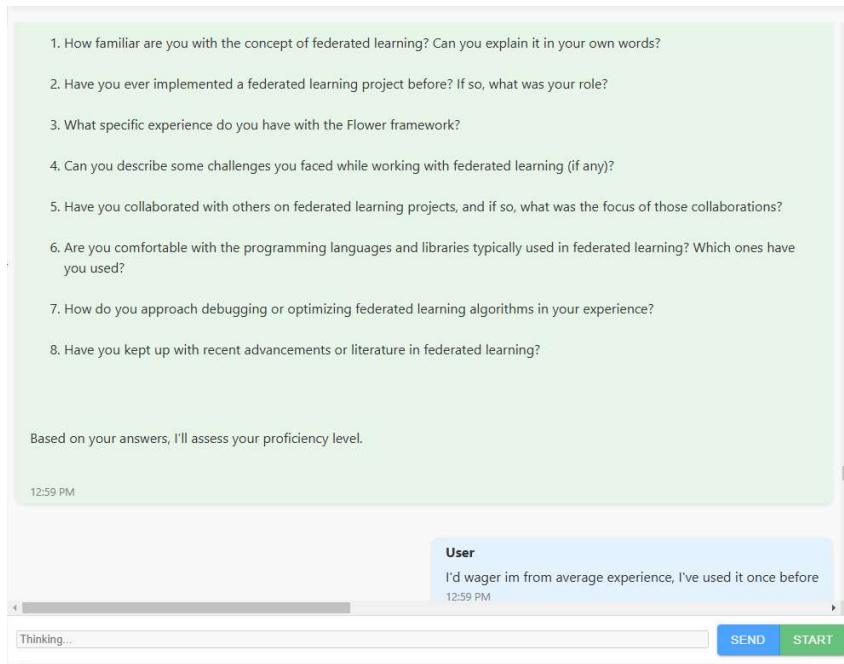


Figure 34. UI chat box

The fifth and final component, Figure 35, displays the most recent code generated by the system in a box with syntax highlighting, with a button that allows the user to copy the code to their clipboard. Additionally, it also shows the number of code versions the agents have created. When new code is generated, the code version number is incremented, and the current code box is updated.

Current Code:

Code Version: 1

 COPY TO CLIPBOARD

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras.callbacks import EarlyStopping
from flwr.client import NumPyClient

# Load the dataset (absolute path)
data_path = r'E:\model_making_UI\logs\user_Test\session_1
data = pd.read_csv(data_path, encoding='ascii', parse_da

# Extract features and target
X = data[['temp_x10']].values # features
y = data['total'].values # target variable

# Normalize the features
scaler = MinMaxScaler(feature_range=(0, 1))
```

Figure 35. UI current code component

Upon opening the dashboard for the first time, the user is presented with a dialog displaying the terms and conditions of MASTerFlow, as shown in Figure 36. Currently, these terms are placeholders but can be easily updated if the application is made publicly accessible.

To integrate the MASTerFlow system with the user interface, some components had to be adapted. The way the messages were exchanged between the user and the system needed to take into consideration the UI. AutoGen's group chat agents log their messages to the terminal the system is running on, and the system redirected the print() function to a ZeroMQ socket.

By using ZeroMQ, the system initializes two sockets: a publisher and a subscriber socket. Messages generated by the system, such as the LLM's questions to the user, or the conversation between the agents in the various group chats, get transmitted to the UI via the publisher system. On the other hand, the subscriber socket receives incoming messages from the UI. Since the UI was built on the React framework, for integration reasons, a secondary application was created that bridges the communications between the UI and the system. Messages sent to and from the server are sent via the ZeroMQ sockets, while UI messages are sent via a React-compatible WebSocket. The connection between these elements is shown in Figure 37.

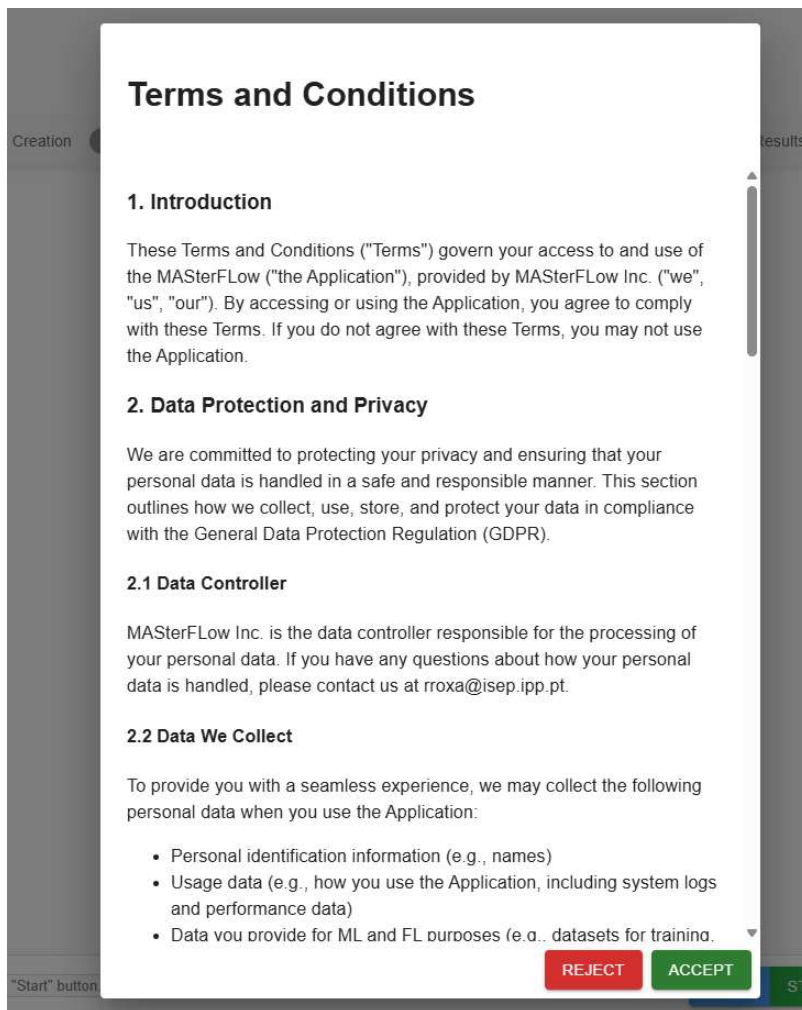


Figure 36. MAsterFlow Terms and Conditions dialog

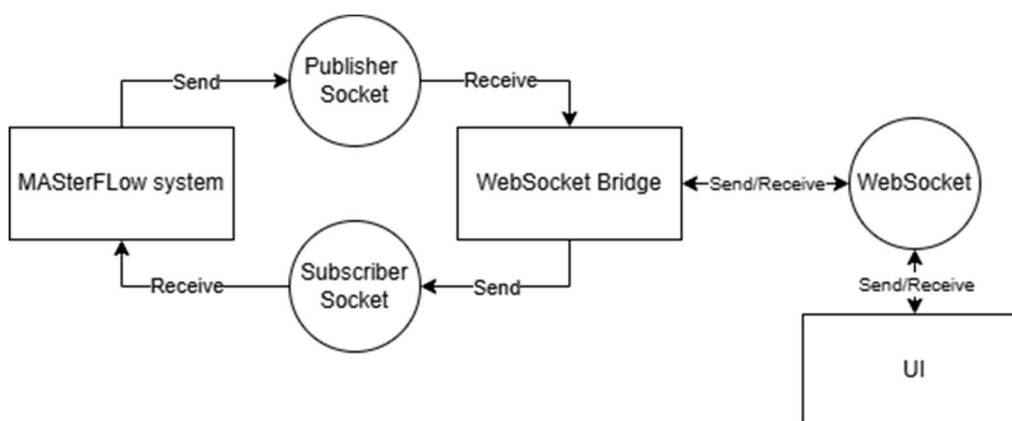


Figure 37. Communication between system components

5 Case Studies

This section presents the case studies that were designed to test the various components of MASTerFlow. The first case study presents a benchmark that compares the three MAS frameworks considered for development: AutoGen, TaskWeaver, and CrewAI. The second case study shows the results of the utilization of LLMs and MAS to create ML models following user configurations. Similarly, the third case study presents the results obtained by orchestrating LLMs and MAS to work with the user to adapt and create an FL federation for distributed model training. The final case study tests the integration of all the components of the system working together and evaluates the final results, namely the capacity of the LLM in analyzing the FL training logs generated by the system. Table 8 displays how the case studies answer the defined objectives.

Table 8. Relationship between objectives and case studies

Objective	Case Study
OB1	Case Study 1
OB2	Case Study 2
OB3	Case Study 3
OB4	Case Study 4
OB5	Case Studies 1-4

5.1 Case Study 1 - Multi-agent system frameworks testing

Before the implementation of the system began, multiple MAS frameworks were tested to figure out which one better fit with the rest of the system and had the necessary requirements. This benchmark case study was published on:

- “Barbarroxa, R., Gomes, L., & Vale, Z. (2025). Benchmarking Large Language Models for Multi-agent Systems: A Comparative Analysis of AutoGen, CrewAI, and TaskWeaver. In P. Mathieu & F. De la Prieta (Eds.), *Advances in Practical Applications of Agents, Multi-*

Agent Systems, and Digital Twins: The PAAMS Collection (pp. 39–48). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-70415-4_4;

In this case study, the agents belonging to the LLM-utilizing MAS frameworks were tasked with generating Python language code to train an ML model to forecast the energy consumption of a building. The dataset used is the one presented in Section 3.2.1. The prompt that was given to the MAS was: *“If I wanted to forecast the energy consumption of my home, what is the best machine learning algorithm to use and why? How could I implement it if I had a csv file named: energy_consumption.csv with three columns, the first is datetime, the second “total” is the total consumption and the third “temp_x10” is temperature multiplied by 10? What measure/metric/function should I choose and how should I use it to validate the model performance?”*. The models considered for this case study were GPT-4 and GPT-3.5 models from OpenAI; Llama 2 13B, Llama 2 13B chat and CodeLlama 13B from Meta AI; Zephyr 7B from Hugging Face, Mistral 7B and Mistral 7B instruct from MistralAI; and the OpenHermes 2.5 model, a Mistral 7B fine-tuned by Teknium.

The LM Studio was the platform used to host the local LLMs used by the AutoGen and TaskWeaver frameworks, via the OpenAI API compatible routes provided. On the other hand, Ollama was used to server the local models used by the CrewAI framework, as recommended by this framework’s documentation.

For the AutoGen framework, two agents were considered, with the following system messages:

- Product Manager agent, with the system message: *“Creative in software product ideas”*, was responsible for coming up with the solution to the user prompt;
- Code agent, responsible for creating the code that will define and train the ML model.

Regarding TaskWeaver, the default setup created by the framework was used, consisting of three agents:

- Planner agent: An LLM-powered agent responsible for creating the sub-tasks to be completed.
- Code Interpreter agent: An agent that utilizes the Code Generator to create and execute code.
- Code Generator agent: An LLM-powered agent designed to generate code for solving tasks.

CrewAI’s agents proved to be the most customizable. Three agents were created, with different goals and backstories:

- Machine Learning Engineer agent: with the goal of *“Develop and apply machine learning solutions to solve real-world problems”* and the backstory: *“You are a Machine Learning Engineer at a cutting-edge research lab. You combine your passion for mathematics, data analysis, and programming to build intelligent systems that learn and improve over time. You strive to use*

your expertise ethically and responsibly to make a positive impact on the world.”

- Machine Learning Quality Assurance Engineer agent: it was defined by the role *“Ensure the quality, fairness and reliability of machine learning models through various testing and validation methods.”* The agent’s backstory is defined as: *“You are a Machine Learning Quality Assurance Engineer, passionate about ensuring responsible and effective ML development. You combine your expertise in testing methodologies, data analysis, and ML concepts to evaluate models for accuracy, performance, fairness, and robustness. You collaborate with developers and data scientists to identify and mitigate biases, data quality issues, and potential harms, contributing to trustworthy and impactful ML applications.”*
- Chief Machine Learning Quality Assurance Engineer agent: its goal is: *“Lead and oversee the quality assurance process for machine learning models, ensuring their effectiveness, reliability, and responsible development.”* To aid in completing said goal, the agent was imbued with the backstory: *“You are a visionary leader in the field of Machine Learning Quality Assurance. You are passionate about building high-performing teams that ensure the quality, fairness, and responsible development of ML models. You combine your deep Benchmarking Large Language Models for Multi-Agent Systems 7 understanding of ML concepts, leadership skills, and strategic thinking to establish best practices, guide your team, and collaborate with stakeholders to deliver impactful and trustworthy ML applications.”*

Each CrewAI agent was assigned a specific task, with sequential tasks designed to address different parts of the original prompt. These sequential tasks, encapsulating the original prompt represented by {original_prompt}, provide additional context to assist the LLM in completing the task:

- Code task: it is the first task assigned to the Machine Learning Engineer agent, with the following description: *“You will create a software using python, these are the instructions: Instructions ----- {original_prompt} Your final answer must be the full python code, only the python code and nothing else.”;*
- Review task: it is the second performed task and it was assigned to the Machine Learning Quality Assurance Engineer agent, it was described as: *“You are helping create a software using python, these are the instructions: Instructions -----

{original_prompt} Using the code you got, check for errors. Check for logic errors, syntax errors, missing imports, variable declarations, mismatched brackets, and security vulnerabilities. Your final answer must be the full python code, only the python code and nothing else.”;*
- Evaluate task: the third task was assigned to the Chief Machine Learning Quality Assurance Engineer agent, described as: *“You are helping create a software using python, these are the instructions: Instructions -----{original_prompt} You will look*

over the code to ensure that it is complete and does the job that it is supposed to do. Your final answer must be the full python code, only the python code and nothing else.”

To compare the models generated by each framework, the last week of the dataset, containing 672 rows was used for testing predictions. These forecasts and the real values from the test dataset were used to calculate the Root Mean Squared Error (RMSE) of the models. The frameworks were able to generate the following model types:

- ARIMA;
- LR;
- LSTM;
- RFR;
- XGBoost Classifier.

As shown in Table 9, the ARIMA model was the best performant of the AutoGen framework, with an RMSE value of 64.86. Achieved by the GPT-4 as Coder and GPT-3.5 as Product Manager.

Regarding TaskWeaver, the GPT-3.5 powered agents were able to achieve the lowest RMSE metric, 25.04, with an LSTM model.

Concerning the CrewAI framework, most agent configurations opted to solve the problem by implementing a linear regression (LR) model. The best-performing setup featured CodeLlama 13B as the ML Engineer agent, with OpenHermes fulfilling both QA roles. This configuration achieved an RMSE of 256.089.

Table 9. Case Study 1 results

Framework	Coder/ML Engineer	Product Manager / ML QA	Chief ML QA	RMSE Value	Model
TaskWeaver	GPT-3.5	N/A	N/A	25.04	LSTM
TaskWeaver	Mistral Instruct	N/A	N/A	37.37	LR
AutoGen	GPT-4	GPT-3.5	N/A	64.86	ARIMA
CrewAI	CodeLlama13B	OpenHermes	OpenHermes	256.09	LR
AutoGen	Mistral	GPT-4	N/A	256.24	LR
AutoGen	GPT-3.5	GPT-3.5	N/A	256.26	LR
CrewAI	OpenHermes	OpenHermes	OpenHermes	256.26	LR
CrewAI	GPT-3.5	GPT-3.5	GPT-3.5	282.06	LR
CrewAI	OpenHermes	OpenHermes	OpenHermes	282.06	LR
AutoGen	GPT-4	Zephyr	N/A	312.5	ARIMA
AutoGen	GPT-3.5	GPT-4	N/A	312.5	ARIMA
AutoGen	GPT-3.5	Zephyr	N/A	312.5	ARIMA
AutoGen	Llama 13B	GPT-3.5	N/A	501.5	LSTM
CrewAI	GPT-4-preview	GPT-4-preview	GPT-4-preview	913.45	RFR
AutoGen	GPT-4	GPT-4	N/A	913.47	RFR
TaskWeaver	GPT-4	N/A	N/A	nan	LSTM
CrewAI	Llama13B Chat	Llama13B Chat	Llama13B Chat	nan	XGBoostClassifier

The TaskWeaver setup that uses GPT-4 was unable to generate the code to load the dataset, ignoring the “datetime” feature, utilizing the target variable “total” in the feature data, and using the feature “temp_x10” as the target variable.

The worst-performing setup used the Llama13B Chat model. In this case, the generated code mistakenly attempted to read a non-existent dataset column, “total_consumption,” instead of the actual column, “total.” Additionally, the incorrect target column was used to construct the features dataset. Furthermore, an inappropriate attempt to use a classifier model for a regression task led to error-filled, non-executable code.

Among the three tested frameworks, TaskWeaver demonstrated the best performance in terms of RMSE. Its agents, powered by either GPT-3.5 or the local Mistral Instruct model, outperformed the other frameworks. This success can be attributed to TaskWeaver’s specialization in data analytics tasks. The only framework approaching these results was AutoGen, with GPT-3.5 and GPT-4 agents that successfully generated an ARIMA model. In contrast, CrewAI’s performance revealed the limitation that, by default, the framework lacks a built-in capability to execute its generated code and analyze the results to identify and correct errors during execution. This missing functionality likely contributed to the results observed.

For this system, the chosen MAS framework was AutoGen since it provided good results in the task of generating ML models, can run code natively, unlike CrewAI, can execute functions via tools, and is easy to integrate with the rest of the system’s components. One of the issues that TaskWeaver presented was the rigidity of the format of outputs and inputs required from the LLM, which meant open-source LLMs had more difficulty in using TaskWeaver.

5.2 Case Study 2 – ML model creation

The purpose of this case study was to test the capacity of large language models in creating machine learning models per user specifications. This case study was developed in three versions. The first version had the user interact with a chat interface, programmed in Python, to fill out the configuration file, which was then sent to an LLM for the creation of the ML model. The second version of the case study replaced the single LLM with a MAS where each agent was powered by an LLM, and responsible for a single task in the pipeline of creating an ML model. The third version of the case study replaces the Python chat interface with an LLM that dialogues with the user, asking questions to fill out the configuration file.

While the system version in Section 4.2.1 is the third version of this case study, as shown in Figure 38, the case study was developed in a step-by-step manner, the various features were added incrementally to the system after the previous version was tested. A preliminary study of utilizing AutoGen was published on the paper:

- “Barbarroxa, R., Ribeiro, B., Gomes, L., & Vale, Z. (2024). Benchmarking AutoGen with different large language models. *2024 IEEE Conference on Artificial Intelligence (CAI)*, 263–264. <https://doi.org/10.1109/CAI59869.2024.00058>”;

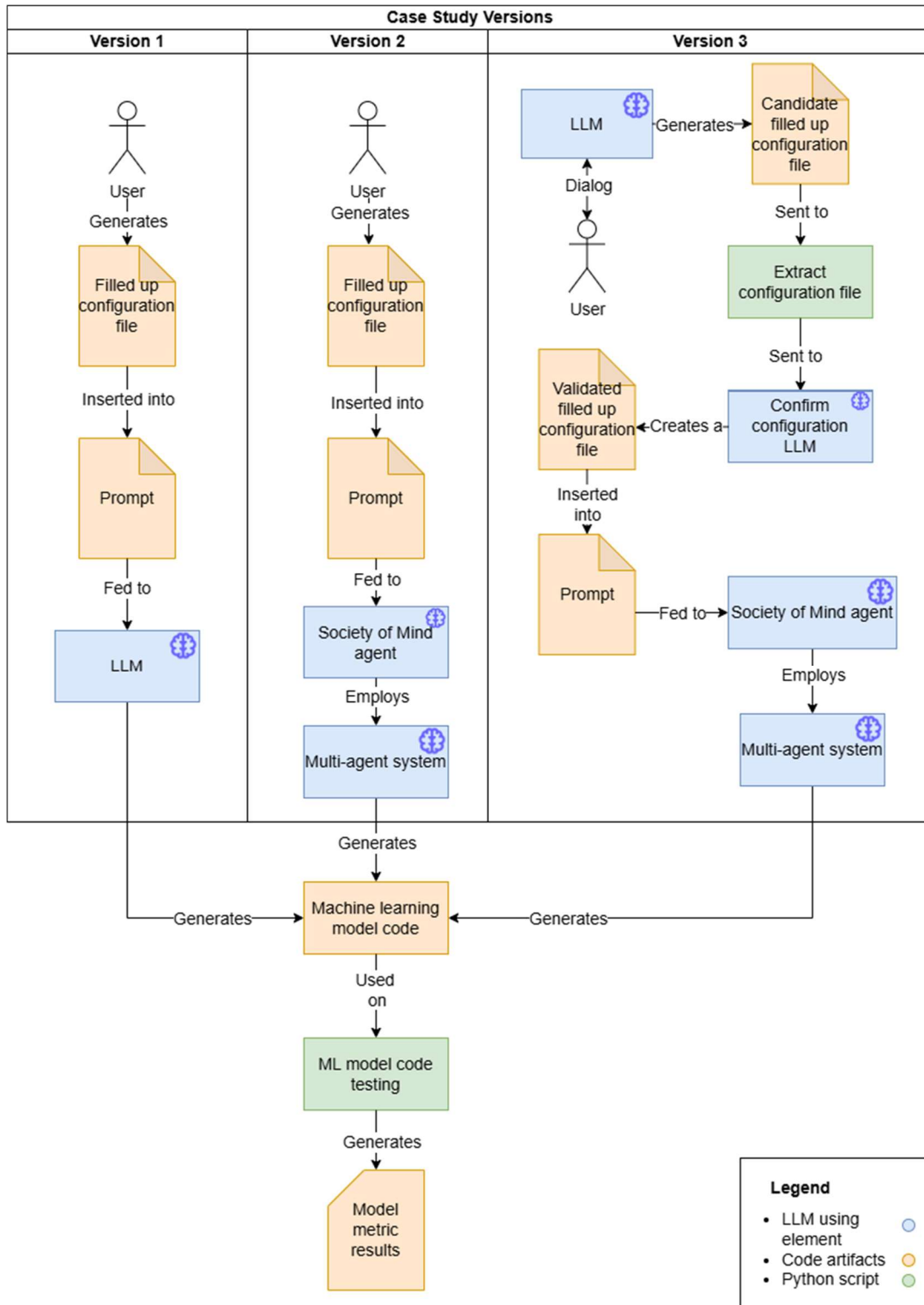


Figure 38. Case study 1 versions

5.2.1 Version 1 – Configuration file and zero-shot prompting

The purpose of this version of the case study was to test the quality of the creation of machine learning models through the usage of a template and an LLM. The configuration file was correctly filled by the user via a chat interface, programmed in a Python script. The first row of the diagram in Figure 38 showcases the implementation of this version of the case study. Afterward, the filled configuration file was inserted into a zero-shot prompt for the LLM tasked with generating Python code to load the dataset, create a ML model, train and test it following the metrics defined by the users in the configuration file.

To test this case study, an energy consumption dataset was used, as described in Section 3.2.1. Additionally, the dataset was split into two. The first half, with 83,505 rows, was given to the system to be used in the code that was autonomously generated by the LLMs. The second half, containing the last week of the dataset, was used to manually test the models generated by the system when used to predict unseen data.

The ML model the LLM was instructed to create was an LSTM. If the code was created with errors, it would be discarded. Otherwise, if there were no errors or the ones present were minimal, such as missing or wrongly imported packages, the model would be used, and the test set would be generated from the second half of the dataset, representing the last week.

In this case study, different LLMs were tested with the same prompt, and for comparison between the generated models the RMSE metric was used to determine the ranking of each LLM. Each of the LLMs was tested 3 times. Table 10 shows the success and failure rate of the case study.

Table 10. Case study 2 version 1 run percentages

Category	Count	Percentage
Total runs	27	100%
Successful runs	8	30%
Runs with errors, considered	2	7%
Runs with errors, discarded	17	63%

Throughout the case study, only approximately 30% of the runs were able to generate code without any errors, however, while the majority (approximately 63%) of runs presented some sort of error, not all of those results were discarded, 7% were considered as having minimal errors. Curiously, there was one single run where the LLM did not generate any code, belonging to the TinyLlama 1.1B Chat v1.0 model, which replied with information on how to create the ML model instead of generating the code by itself. Table 11 displays the rankings of the LLMs that were able to create ML models with no errors. The metrics, MSE and RMSE, were then averaged out by the number of successful runs and were used to rank the LLM models.

Table 11. Case study 2 version 1 ML models created without error

Rank	Model	Open weights	ML models created without error	Avg MSE	Avg RMSE
------	-------	--------------	---------------------------------	---------	----------

1	GPT-3.5-turbo	No	2	79082.30	205.33
2	Llama 3 8b instruct	Yes	1	44202	210.24
3	Codestral v0.1 22B	Yes	2	63543.60	250.06
4	Mistral 7b instruct v0.3	Yes	1	69967.10	264.51
5	GPT-4o	No	1	83590	289.12
6	Llama 2 13b chat	Yes	1	128110	357.92

On the other hand, Table 12 displays all the runs that are a part of the case study, grouped by LLM, indicating how many of the total 3 runs done by each LLM resulted in error.

Table 12. Case study 2 version 1 total runs

Rank	Model	Open weights	Total runs	Error	Error %	Avg MSE	Avg RMSE
1	Llama 3 8b instruct	Yes	3	2	66.67	44202.04	210.24
2	GPT-3.5-turbo	No	3	1	33.33	56445.68	234.08
3	Codestral v0.1 22B	Yes	3	1	33.33	63262.98	250.17
4	Mistral 7b instruct v0.3	Yes	3	2	66.67	69967.10	264.51
5	GPT-4o	No	3	2	66.67	83590.04	289.12
6	Llama 2 13b chat	Yes	3	2	66.67	128110.27	357.92
7	GPT-4	No	3	3	100	N/A	N/A
7	Zephyr 7b beta	Yes	3	3	100	N/A	N/A
7	TinyLlama 1.1B Chat v1.0	Yes	3	2	100	N/A	N/A

When comparing the two tables, the addition of the MSE and RMSE metrics obtained by models that were generated with errors by the LLMs had an impact on the rankings. GPT-3.5-turbo was dethroned by an open weights model, Llama 3 8b instruct. While the latter's LSTM model maintained the same average RMSE, 210,24, the models generated by GPT-3.5-turbo increased the average RMSE from 205,33 to 234,08.

Surprisingly, the third OpenAI model, GPT-4, was unable to create any code without errors or whose errors were minimal. This puts the LLM at the bottom of the table, with Zephyr 7b beta and TinyLlama 1.1B Chat v1.0. The error that appears in the GPT-4 attempts is the following: "ValueError: Failed to convert a NumPy array to a Tensor (Unsupported object type Timestamp)". This error appears because the code generated by the LLM erroneously converts the datetime column of the energy consumption dataset, defined in Section 3.2.1. Some of the minimal errors that appeared during the runs of the case study were the following: "NameError: name 'dt' is not defined. Did you mean: 'df'?" and "NameError: name 'np' is not defined". To correct the former, a proper datetime import can be used: "import datetime as dt". To correct the latter, the code was missing the import of the NumPy library with an alias: "import numpy as np".

Considering the results of this case study, the LLMs chosen for the next case study were GPT-4o, GPT-3.5-turbo, Codestral v0.1 22B, Mistral 7b instruct v0.3, and Llama 3 8b instruct.

5.2.2 Version 2 - Template and multi-agent system

In version 2, the single LLM was replaced by a multi-agent system, where each agent was powered by an LLM. The purpose of this case study was to study whether the usage of a multi-agent system where each agent had a single responsibility would surpass the capability of a single LLM with multiple responsibilities. Like the previous version, the multi-agent system also received the specifications of the machine learning model via the prompt that was created based on the user’s configuration file. However, the prompt for this version also contained the first five rows of the dataset chosen by the user, giving the multi-agent system some insight into the structure of the dataset. Figure 13 shows an example prompt.

The multi-agent system responsible for autonomously creating the machine learning model used is defined and described in Section 4.2.1. In the second row of Figure 38, the components of this case study are presented.

For this case study the LLMs with the best performance in Section 5.2.1, GPT-4o, GPT-3.5-turbo, Codestral v0.1 22B, Mistral 7b instruct v0.3 and Llama 3 8b instruct were used. Additionally, newly released LLMs were added to the list of LLMs used in the Case Study, namely GPT-4o-mini and Llama 3.1 8b instruct. Table 13 displays the composition of the different group chats that were tested.

Table 13. Case Study model making group chats

Group chat	Manager	Analyzer agent	Preprocessing agent	ML expert agent	Consolidator agent
1	GPT-4o	GPT-4o	GPT-4o	GPT-4o	GPT-4o
2	GPT-4o	GPT-3.5-turbo	GPT-3.5-turbo	GPT-3.5-turbo	GPT-3.5-turbo
3	GPT-4o	Codestral v0.1 22B	Codestral v0.1 22B	Codestral v0.1 22B	Codestral v0.1 22B
4	GPT-4o	Mistral 7b instruct v0.3	Mistral 7b instruct v0.3	Mistral 7b instruct v0.3	Mistral 7b instruct v0.3
5	GPT-4o	Llama 3 8b instruct	Llama 3 8b instruct	Llama 3 8b instruct	Llama 3 8b instruct
6	GPT-4o	GPT-4o-mini	GPT-4o-mini	GPT-4o-mini	GPT-4o-mini
7	GPT-4o	Llama 3.1 8b instruct	Llama 3.1 8b instruct	Llama 3.1 8b instruct	Llama 3.1 8b instruct

Equally to version 1, in Section 5.2.1, the same dataset described in Section 3.2.1 was used with the same data split to test the LLM’s capacity to create an LSTM model. These models were also compared via the RMSE metric. Table 14 showcases the run percentages of this version of the case study.

Table 14. Case study 1 version 2 run percentages

Category	Count	Percentage
----------	-------	------------

Total runs	21	100%
Successful runs	4	19%
Runs with errors, considered	2	10%
Runs with errors, discarded	15	71%

In this version, the percentage of discarded runs has increased when compared to the previous version testing runs. However, the GPT-4o-mini group chat was able to significantly improve on the ML model creation, being able to achieve an average RMSE of 152,6, as shown in Table 15, which displays the results obtained by each group chat combination with the three runs for each averaged out.

Table 15. Metrics obtained by each group chat

Rank	Group chat	Runs with error	Error %	Avg MSE	Avg RMSE
1	6	1	33	41036.80	152.60
2	1	1	33	81060.90	285.60
3	2	2	66	182141.50	426.80
4	3	2	66	249376.80	499.40
5	4	3	100	N/A	N/A
5	7	3	100	N/A	N/A
5	5	3	100	N/A	N/A

When compared to Version 1’s results, in this test the GPT family of model has risen to the top, with GPT-4o-mini in first place, followed by GPT-4o and GPT-3.5-turbo. Codestral v0.1 22B was the only one of the open-weight models that was capable of creating a ML model in the MAS configuration. The remaining models, Mistral 7b instruct v0.3, Llama 3.1 8b instruct, and Llama 3 8b instruct were incapable of generating code that worked. This failure in results could be attributed to the better capacities of the GPT models, as well as the fact that the AutoGen framework was built with GPT LLMs considered as default.

5.2.3 Version 3 - LLM dialog and multi-agent system

In this version, the necessity for the user to directly fill a configuration file of questions was replaced by a dialog interaction between the user and the LLM. By asking questions to the user, the LLM can autonomously fill in the configuration file and send it to the MAS to create the machine learning model code. For the LLM to understand the questions it had to ask the user, the prompt in Figure 7 was used.

Replacing a static configuration file completion with an LLM that chats with the user has various benefits. First, it simplifies the process of filling in the configuration file, which might overwhelm a less experienced user. Abstracting the configuration into a conversation allows for less technical, more natural responses. Additionally, the LLM can answer any question the user has about the meaning or purpose of one or more parameters present in the configuration file, leading to the bonus benefit of teaching the user. The components of this version of the case study are shown in the third row of Figure 38.

Because in this version of the case study, the configuration file is completed by an LLM, before it is passed on to the rest of the system, an LLM is instructed to check it and confirm it follows

the required format, has all values and the proper data type for each parameter of the configuration file. In case the file is missing something, the LLM will notify the dialog LLM with the reason. The latter will work with the user to solve the problem with the configuration file. Table 16 shows the various tests done to the confirm configuration file LLM, displaying the results of each test and the reasoning behind the choice the LLM made. A result of True indicates that the configuration file is complete and approved by the confirmation LLM. A result of False means the file was denied, with the Reasoning field providing the LLM's justification for the denial.

Table 16. Configuration file confirmation LLM test results

Test	Result	Expected Result	LLM Reasoning
Control, working template	True	True	N/A
Template with multiple missing parameters	False	False	The template is incomplete, missing NUMBER_COLUMNS, COLUMNS, TRAIN_SPLIT_PERCENTAGE, TEST_SPLIT_PERCENTAGE, and RANDOM_DATASET_SPLIT.
Column count and actual columns mismatch	False	False	Missing information for column 4 and column 5, which are required for NUMBER_COLUMNS: 5.
Missing model name in template	False	False	MODEL_NAME is missing.
Empty model name	False	False	MODEL_NAME is missing.
Missing type for column	False	False	<column_name_1 does not have a type specified>
Missing type and target for columns	False	False	Missing TYPE for column 'total' and 'temp_x10' does not have TARGET specified.

The GPT family of models, namely GPT-4o and GPT-4o-mini were the ones considered for dialog with the user due to their speed and capabilities (Agarwal & Nasim, 2024). The speed of response is important when talking with the user since it makes the system feel more responsive. A smaller local LLM model could be used, however, these are less capable to respect a template format and are more susceptible to hallucinations during the dialog. The Instruction-Following Evaluation (IFEval) is an important metric in this scenario, as it tests the model's ability to strictly adhere to formatting and generation instructions. The IFEval score indicates that larger models, such as GPT-4o, demonstrate higher accuracy in following explicit formatting instructions compared to smaller models. The Hugging Face open LLM leaderboard (Fourrier et al., 2024) can be consulted as proof, as models with a larger number of parameters can be seen dominating the leaderboard.

On the other hand, utilizing a local LLM model with a larger number of parameters, such as 70b or above, was not considered due to the lack of hardware with sufficient performance to guarantee fast enough inference for a responsive dialog.

5.3 Case Study 3 – FL federation creation

The purpose of this case study was to test the capacity of LLM in creating a FL federation autonomously, taking into account the parameters and configurations set by the user. Since federated learning can be considered a complex process, the system was also tasked to determine the level of expertise of the user, and the options it provides are dependent on that. The explanation of this case study will be divided into the different features that this part of the system provides. As described in Section 4.2, this case study is only possible if the integration of the ML model code into a Flower Client is successful. The results of an application of FL using the Flower framework can be seen in the following paper, where the communication between the clients and the server was done via distributed ledger technology:

- “Barbarroxa, R., Silva, J., Gomes, L., Lezama, F., Ribeiro, B., & Vale, Z. (2023). *FeDis: Federated Learning Framework Supported by Distributed Ledger* (pp. 32–41). https://doi.org/10.1007/978-3-031-45155-3_4”;

As shown by Table 17, the average accuracy value of the experiment while using FL and a distributed ledger is higher than the value of training a model without FL.

Table 17. FeDis case study results

Client	Experiment 1 (base case without FL)		Experiment 2 (using FeDis with DL and FL)		Training Time (minutes)
	Accuracy	Loss	Accuracy	Loss	
Client 1 (Set1_timefeature)	84.2%	5.6%	81.7%	5.5%	45
Client 2 (Set2_timefeature)	76.8%	5.8%	77.4%	5.6%	25
Client 3 (Set3_timefeature)	73.3%	5.4%	75.9%	5.5%	55
Average value	78.1%	5.6%	81.7%	5.5%	41

Additionally, the case study concludes that FL enhances data privacy within distributed ledgers by enabling knowledge sharing through model gradients instead of raw data. On the other hand, incorporating distributed ledgers into the FL process adds an extra layer of security and traceability, thanks to the ledger’s decentralized structure and its ability to immutably record all transactions, including potentially malicious ones.

5.3.1 User expertise determination

The initial step of the system, depicted in Figure 17, was to assess the user’s expertise in FL, specifically within the Flower framework used in this case study. This assessment was carried out by assigning an LLM to interact with the user and determine their expertise through a series of questions. Three levels of expertise were defined in section 4.2.3. Figure 39 shows an example of how the system asks questions and identifies the user as a beginner.

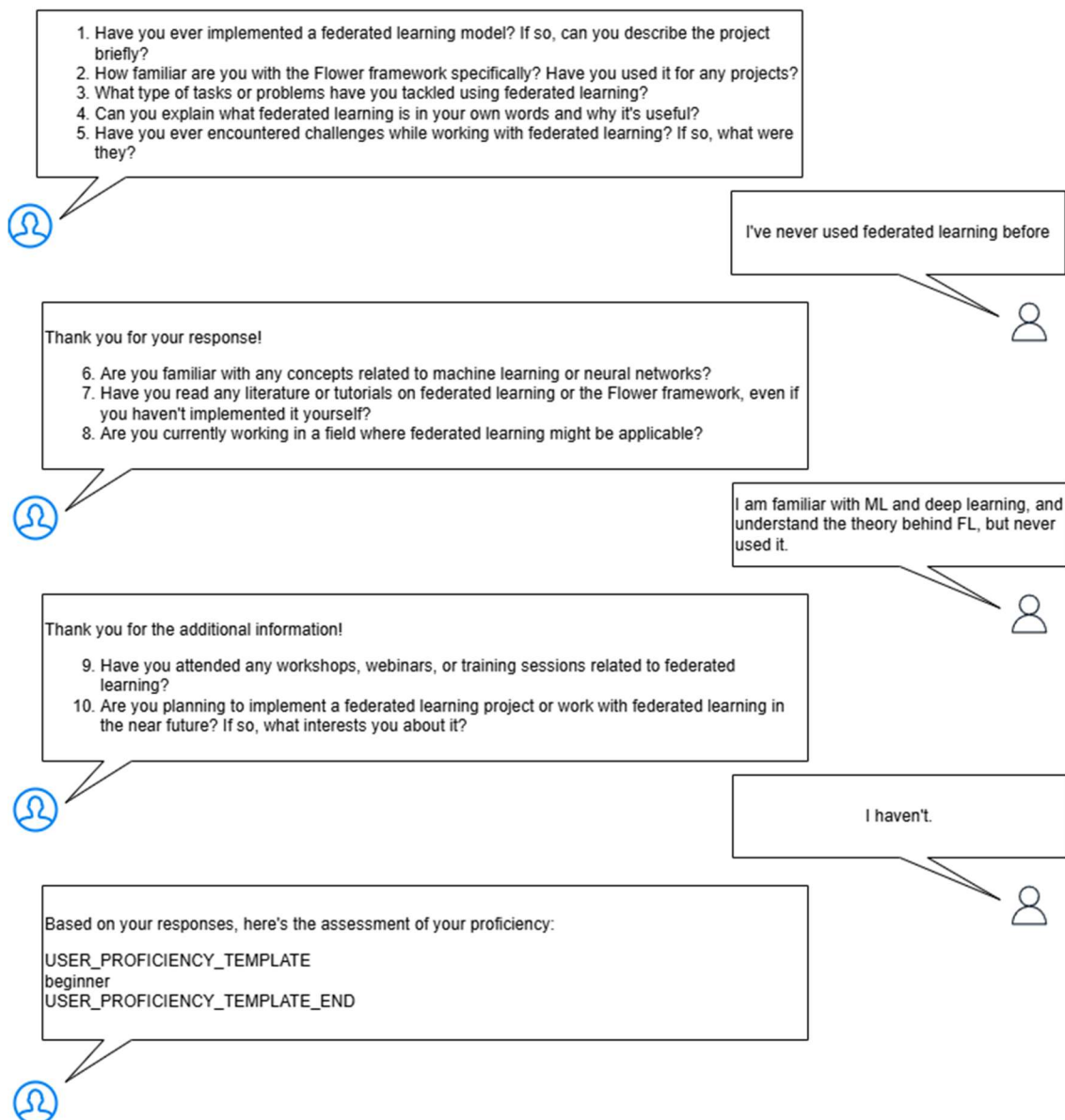


Figure 39. User expertise determination example

5.3.2 LLM dialog and multi-agent system

After the user's expertise has been determined and assigned to one of the three available levels, beginner, average, or expert, another LLM was tasked with dialoguing with them to fill up a template containing the configuration parameters for a Flower federated learning simulation. Like the previous case study, Section 5.2, the template was extracted and confirmed by another LLM before being used by the system. Figure 40 shows an example dialog with a beginner user.

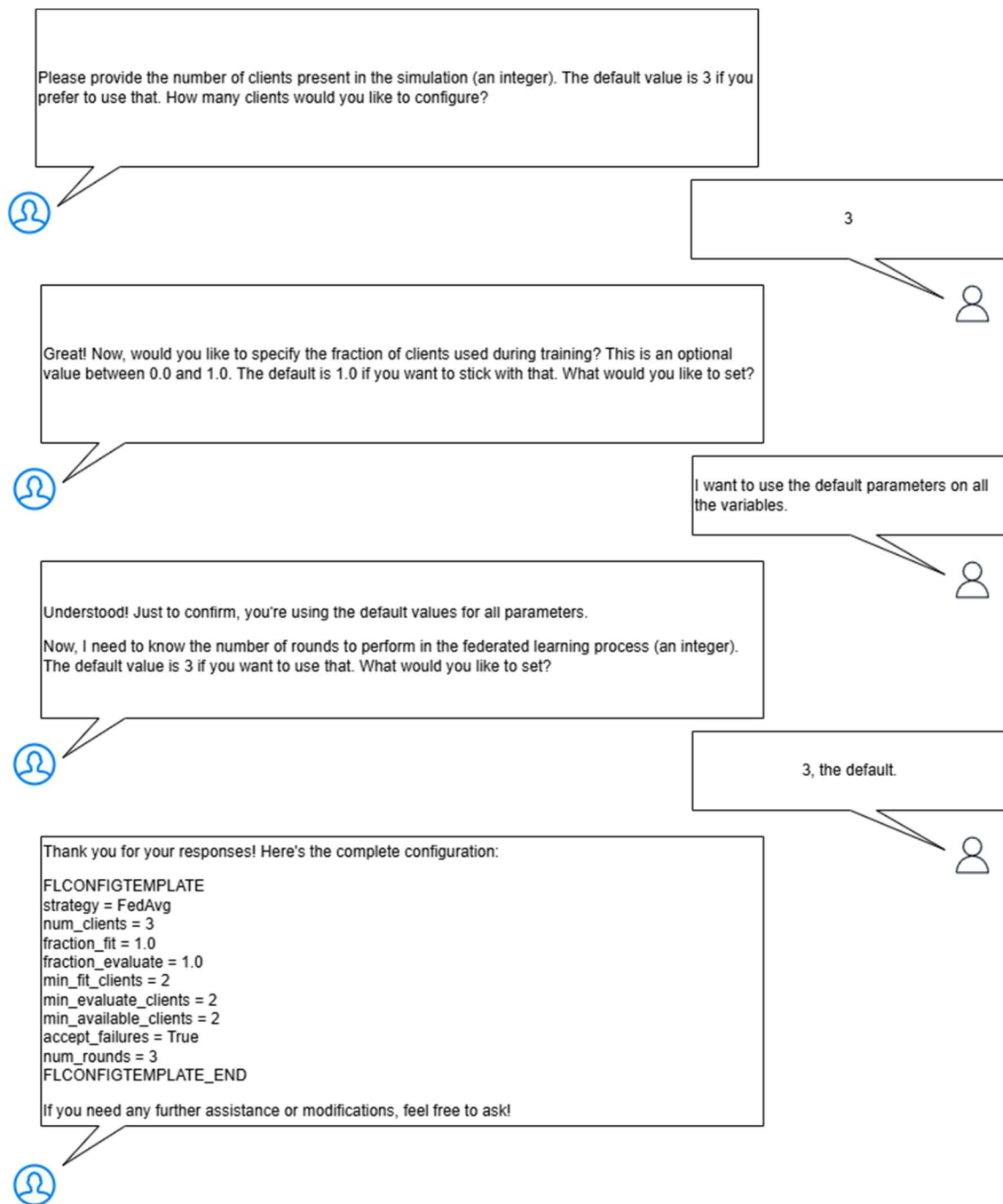


Figure 40. Example of FL making dialog with the user

To test this component, three FL federations were created. The first one used the default parameters, defined in Section 4.2.3, the FedAvg strategy, and 20 rounds of training:

- Num_clients, the number of clients present in the simulation, defined as 3;
- Fraction_fit, the fraction of clients used during training, set as 1;
- Fraction_evaluate, the fraction of clients used during validation, set as 1;
- Min_fit_clients, the minimum number of clients used during training, defined as 2;
- In_evaluate_clients, the minimum number of clients used during validation, set as 2;

- `Min_available_clients`, the minimum number of available clients, defined as 2;
- `Accept_failures`, whether to accept rounds containing failures; set as Yes;
- `Num_rounds`, the number of rounds to perform in the FL training process, is defined as 20.

The second FL federation also was trained for 20 rounds, but the FedMedian strategy was the one employed. Like the previous case studies, the energy consumption dataset from Section 3.2.1 was used to train the LSTM model. Likewise, to test the FL model only the last week of data was considered, and the results were measured using the MSE and RMSE metrics. Table 18 provides an overview of the FL federations.

Table 18. FL training results

Strategy used	Number of clients	Number of training rounds	MSE	RMSE
FedAvg	3	20	75602,63	274,96
FedMedian	3	20	75821,07	275,36
FedAvg	10	200	76570,65	276,71

The group chat responsible for creating the FL federation had agents utilizing the GPT-4o-mini LLMs. One of the biggest concerns with FL is resource usage due to the distributed nature of the training. However, even in a simulation, such as the ones used to test this case study, higher hardware requirements are needed, in comparison to a normal ML model, due to the additional need to simulate multiple clients' training models and the aggregation process done by the server.

The results obtained indicate that by training a model via FL, even with a small number of clients, for 20 rounds, the metrics achieved are competitive with Case Study 1 and Case Study 2. When compared to the first case study, both results can surpass GPT-4o, which achieved an RMSE metric of 289,12. On the other hand, when compared to Case Study 2, the training result metrics were able to surpass all ML model-making MAS group chats minus the one that also consisted of agents using the GPT-4o-mini LLM, which achieved an average RMSE of 152,6.

The third and final result consisted of an increase in the number of clients to 10, and the number of training rounds was multiplied tenfold, yet the error metrics were slightly worse than the previous training results, which might indicate the model converged earlier and no significant improvement was made during the remaining rounds. Figure 41 showcases the MSE results obtained by all rounds minus the first one, which was removed from the plot due to the difference in values, which would hamper the visualization.

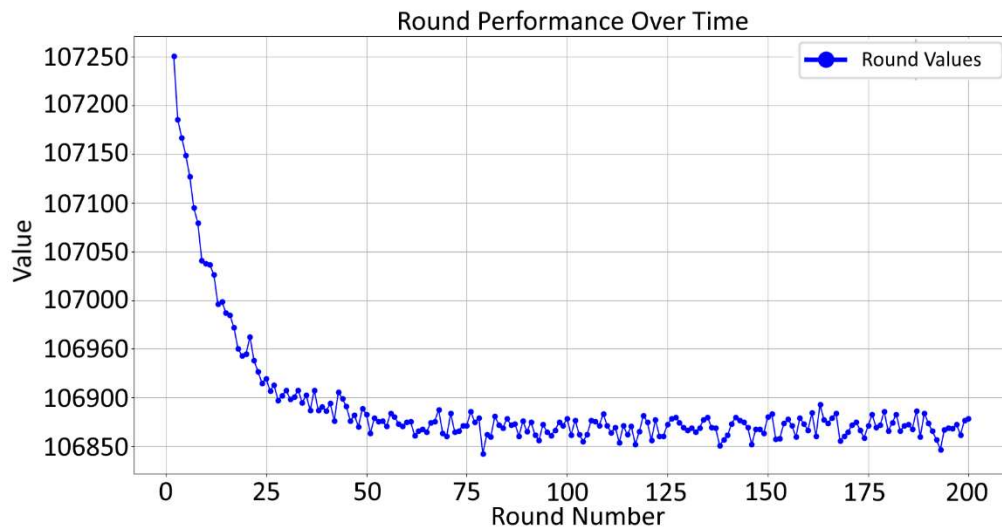


Figure 41. Results from 200 rounds of FL training

5.4 Case Study 4 – FL training logs analyzer

The objective of the case study was to test the full system when composed of all the developed components, namely the ML model creation, the integration of the ML model in the Flower Client, the FL federation creation component, and the FL training logs analyzer. For the testing of this case study, it was considered that the user started by creating an ML model, which would be integrated into a Flower Client and used in FL training. Afterward, the user would talk to the LLM that is responsible for analyzing the logs.

Considering that the ML model creation and FL federation creation components testing results have already been shown in Section 5.2 and 5.3 respectively, this section will demonstrate the results of the testing of the integration of ML in Flower Client, as described in Section 4.2.2 and the FL training logs analyzer, presented in Section 4.2.4.

The system was tasked with creating an LSTM model, which uses the TensorFlow framework, and the dataset the model was trained on was the community dataset described in Section 3.2.2.

After the system created both the ML model and the FL model, and the latter was run in a federated training simulation, the training logs were sent to the LLM equipped with RAG, which was tasked to analyze the logs and provide answers to the questions asked by the user. The FL training simulation ran for 3 rounds, and the logs generated can be seen in the Annex, Figure 51.

To start the dialog, the LLM starts by sending the user an overview of the FL training logs automatically, as seen in Figure 42 the overview provided by the LLM wasn't too specific to the

log data, and explained the general process of federated learning with some examples from the training logs, to which the user asked for a summary of the federated training.

Here's an overview of the federated learning training process, as illustrated by the log you provided:

Initialization: The process begins with the initialization of the federated learning server, which is responsible for coordinating the training process. In this case, the Flower framework is used, and the server is configured to run for a specified number of rounds (3 rounds in this example).

Client Selection: The server selects a subset of clients to participate in each training round. In the log, 3 clients are sampled for each round, which is the total number of available clients.

Parameter Request: Initially, the server requests the initial model parameters from one random client. This step is crucial to ensure that the server has a starting point for the global model.

Training Rounds: Each round consists of two main phases:

Fit Phase: The selected clients train the model on their local data and send the updated model parameters back to the server. The server aggregates these updates to form a new global model. In the log, all 3 clients successfully return their results without any failures.

Evaluate Phase: The server evaluates the updated global model by sending it to the clients, who then evaluate it on their local data and return the results. Again, all clients successfully return their evaluation results.

Aggregation: The server aggregates the results from the clients during both the fit and evaluate phases. This aggregation is crucial for updating the global model and assessing its performance.

Completion: After completing the specified number of rounds, the server summarizes the training process, including the loss values for each round. In this example, the loss values slightly decrease over the rounds, indicating some level of model improvement.

Termination: Once the training rounds are completed, the server and any associated resources (like actors in a Ray backend) are terminated, concluding the federated learning process.

This process allows for decentralized training, where client data remains local, enhancing privacy and reducing the need for data transfer to a central server. If you have any specific questions about any part of this process, feel free to ask!



Figure 42. FL logs analyzer dialog – part 1

As a reply, the LLM provides a concise summary of the federated learning training process, yet this time the LLM also explains how the simulation was run software-wise, as it mentions various components of the simulation software. This response can be seen in Figure 43.

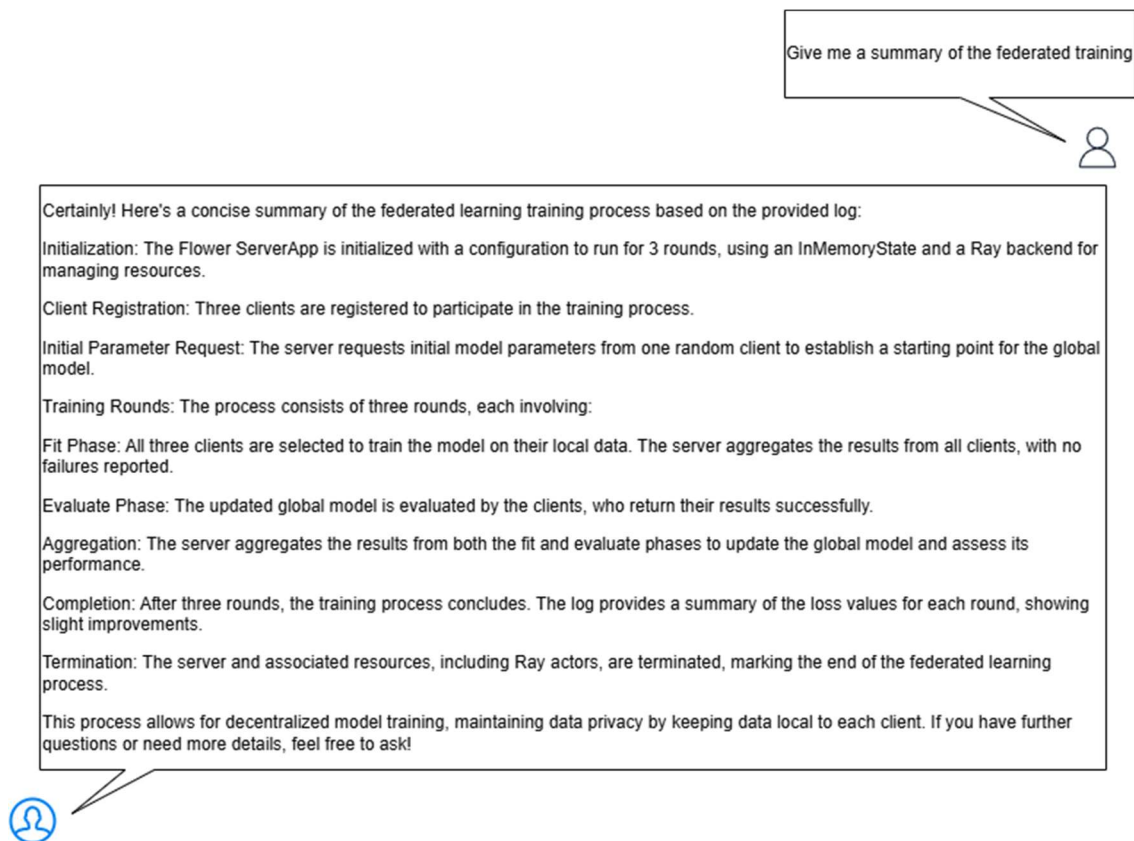


Figure 43. FL logs analyzer dialog – part 2

In order to generate the summary, the system had to retrieve the information from the logs. The following log rows, in Figure 44, show from where the initialization information was retrieved, as well as the number of clients (nodes) that were registered.

```

cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:15:15,222 | state_factory.py:50 | Using InMemoryState
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:15:15,225 | run_simulation.py:423 | Buffer time delay: 5s
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:15:15,224 | app.py:74 | Starting Flower ServerApp, config: num_rounds=3, round_timeout=999s

[...]

cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:15:20,227 | vce_api.py:59 | Registered 3 nodes
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:15:20,228 | vce_api.py:321 | Supported backends: ['ray']
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:15:20,228 | raybackend.py:46 | Initialising: RayBackend

```

Figure 44. FL training logs - initialization and client registration information

For the training process, the component retrieved the information from the logs generated by each round, which indicate client sampling, the aggregation of the training results and the evaluation losses, Figure 45 shows the logs for the second round.

```

cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:21:15,904 | server.py:113 | [ROUND 2]
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:21:15,904 | server.py:226 | configure_fit:
strategy sampled 3 clients (out of 3)
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:23:38,525 | server.py:240 | aggregate_fit:
received 3 results and 0 failures
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:23:38,528 | server.py:178 | configure_evaluate:
strategy sampled 3 clients (out of 3)
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:25:49,444 | server.py:192 |
aggregate_evaluate: received 3 results and 0 failures
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:25:49,446 | server.py:112 |

```

Figure 45. FL training logs – second round of training.

For the termination information, the logs retrieved to serve as LLM context are at the tail end of the file and can be seen in Figure 46 .

```

cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:30:35,119 | run_serverapp.py:89 |
ServerApp finished running.
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:30:35,119 | run_simulation.py:349 |
ServerApp finished running.
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:30:35,120 | run_simulation.py:353 |
Triggered stop event for Simulation Engine.
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:30:36,106 | ray_actor.py:450 | Terminated
6 actors
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:30:36,942 | raybackend.py:193 |
Terminated RayBackend
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:30:36,943 | run_simulation.py:455 |
Stopping Simulation Engine now.

```

Figure 46 FL training logs – termination information

Afterwards, the user asks the LLM for the results, and the loss, measured in each round of the federated training process. This information can be seen in the summary section of the training logs, as shown in Figure 47.

```

cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:30:35,112 | server.py:497 | [SUMMARY]
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:30:35,112 | server.py:498 | Run finished 3
round(s) in 796.70s
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:30:35,113 | server.py:500 | History (loss,
distributed):
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:30:35,114 | server.py:500 | round 1:
0.04595337063074112
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:30:35,116 | server.py:500 | round 2:
0.045863907784223557
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:30:35,116 | server.py:500 | round 3:
0.04587125778198242

```

Figure 47. FL training logs – summary section

The LLM can reply accurately per the information present in the logs, as shown by the response in Figure 48. It was able to retrieve the losses of the rounds and then perform a small analysis of the results.

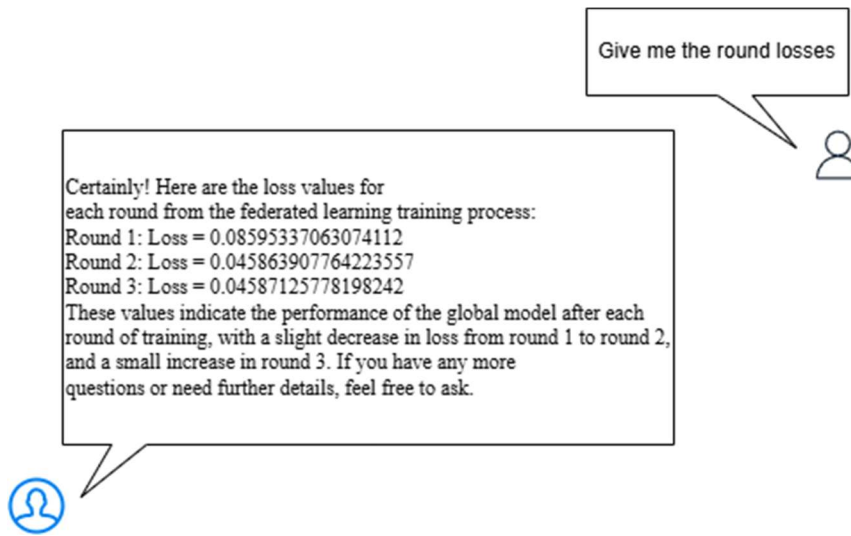


Figure 48. FL logs analyzer dialog – part 3

The log analyzer LLM also demonstrated some mathematical capability when asked for the average loss value of the three rounds. The expected value of the average loss value is 0.0458961787323157, which becomes 0.0459 when rounded to four decimal places. The LLM responded with a different unrounded value, 0.0458961780656497. However, the difference is very minimal, at 6.67×10^{-10} , which is negligible for most practical purposes. Additionally, the LLM replied with the LaTeX notation that represents the mathematical function used to calculate the average value, which was unparsed by the UI, as shown in Figure 49.

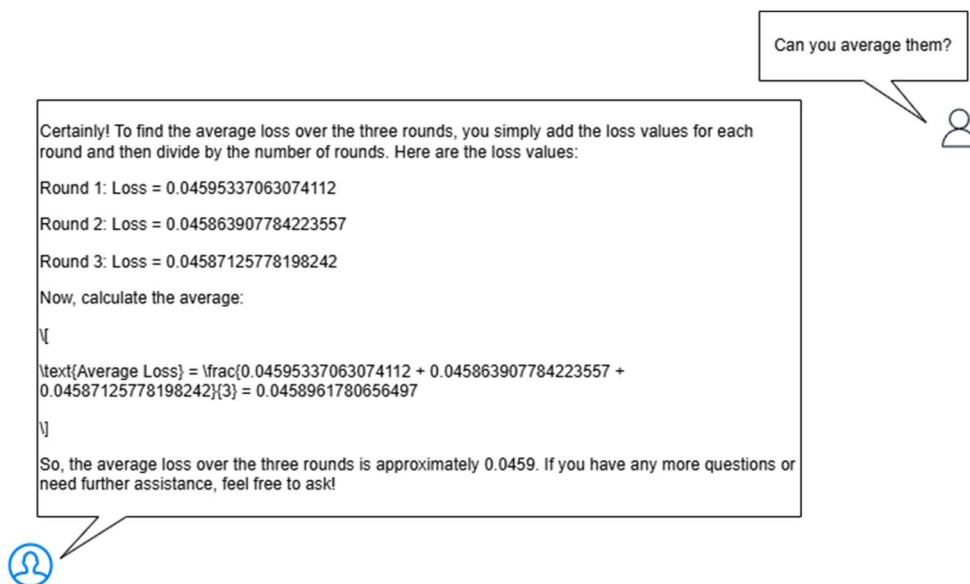


Figure 49. FL logs analyzer dialog – part 4

To test the analytical capacity of the LLM, and not just if the system was able to retrieve the correct information from the vector database, the user asked the LLM about the improvement of the model during the FL training rounds, as shown in Figure 50.

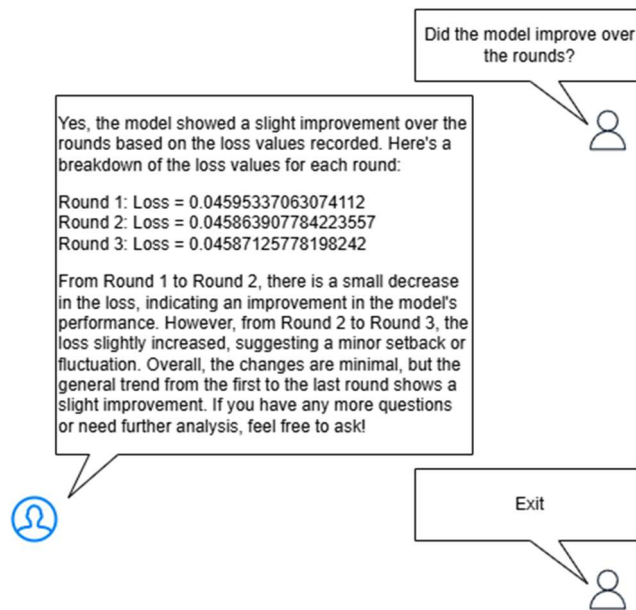


Figure 50. FL logs analyzer dialog – part 5

The response showed that the LLM was capable of realizing that the loss decreased from Round 1 to Round 2 but there was an increase from Round 2 to Round 3. The former indicates an improvement in the model's performance while the latter indicates the opposite. However, the LLM was able to conclude that the increase in loss represented a minor setback and that the general trend over all the rounds showed an improvement in the model. Finally, the user can stop talking with the log analyzer LLM by typing exit into the chat.

6 Conclusions and Future Work

This dissertation has demonstrated that the MASTerFlow system, a chain of MAS powered by LLMs, is capable of automating the creation of ML and FL pipelines with success. For log analysis, the system also integrates LLMs with RAG technology to enhance the system's capacity for accurate log interpretation and answer generation. Additionally, the system also provided the users with a web-based chat interface for seamless and easier interaction with MASTerFlow agents.

The case studies conducted not only validated the system's architecture but also provided knowledge on how MAS can handle complex tasks such as automation of pipeline creation, and how LLMs can handle log analysis and user interaction.

Additionally, Section 4 and the case studies demonstrated how the system was capable of addressing the objectives outlined in Section 1.3

The first objective, **OB1** (determining the state-of-the-art regarding MAS frameworks, and the most adequate to be used for the implementation of the chain of MAS), was answered in Section 2.3, where the MAS frameworks state-of-the-art is presented and in Section 3.1.2 where the ones considered to be used during the development are present. Additionally, Case Study 1 provided a comprehensive benchmark of various MAS frameworks, evaluating their suitability for the implementation of MASTerFlow. The study demonstrated that the AutoGen framework is the most appropriate for fulfilling the system's requirements, offering flexibility, ease of integration, and adaptability for ML and FL pipeline creation, addressing the first objective.

For the second objective, **OB2** (implementation of the MAS powered by LLM agents to assist users in creating ML models), MASTerFlow's ML model creation component, shown in Section 4.2.1, was developed. Case Study 2 showcases how the system was capable of preprocessing datasets, defining model architectures, and executing training and testing phases, while aligned with user-defined configurations. This case study confirms that the MAS framework, powered by LLM agents, is effective in assisting users in automating the creation of ML models, which addresses the second objective.

The results of the third objective, **OB3** (implementation of the MAS powered by LLM agents to assist users in creating a FL Federation in the Flower framework) are the components defined in Section 4.2.2 "Integration of ML model in Flower Client component" and in Section 4.2.3 "FL federation creation component". Case Study 3 demonstrated how the MAS agents successfully configured the FL federation based on user specifications, integrated existing ML models, and executed the FL simulation. Additionally, an LLM was able to determine the user's level of expertise in FL, which influenced the technologies and tools available to them. This user expertise determination allowed for a more personalized approach to the FL federation creation, ensuring that the system adapted to the user's needs. This validated the

capability of MAS agents to assist users in creating FL federations while considering their configurations and expertise, which addresses the third objective.

The fourth objective, **OB4** (development of an interactive system that employs an LLM and RAG to allow users to perform FL training log analysis), was addressed by the component in Section 4.2.4 and validated in Case Study 4, which showed how the system allowed users to interact with the logs, ask questions, and receive answers based on the analysis of the FL training results. The application of RAG technology significantly improved the accuracy of log analysis and user query response, which addresses the fourth objective.

Finally, regarding the fifth objective, **OB5** (test and validate the solution using realistic datasets and scenarios), all case studies (Case Study 1 to Case Study 4) contributed to testing and validating the solution using realistic datasets and scenarios. As described in Section 3.2, the datasets used were created from real-world data. This contributed to assess the system's ability to handle complex tasks like ML model creation, FL federation setup, and FL training log analysis while taking into consideration the configurations defined by the users and their varying levels of expertise.

In addition to the technical achievements, several publications were written during the course of this research, contributing to the academic discussion on MAS, LLMs, and FL.

In conclusion, MASTerFlow successfully met all the key objectives outlined in Section 1.3. The system demonstrated that the AutoGen framework can be effectively used to automate the creation of both ML and FL pipelines. Additionally, the integration of LLMs and RAG technology with the system enabled users to create ML models, FL federations, and perform training log analysis. While these accomplishments confirm the potential of MASTerFlow, further research is needed to refine the system's scalability, enhance user interface customization, and address the challenges associated with open-source model integration.

However, while this dissertation has demonstrated a proof-of-concept implementation, there are still several areas where the system can be further developed to enhance its applicability and efficiency in real-world scenarios.

Enhancing the system's scalability to handle larger, more complex ML and FL models is crucial for real-world applications. Additionally, adapting the system to handle multiple users is an important task. As the user base grows, MASTerFlow must be capable of managing concurrent users, their configurations, and resource allocation, while maintaining strict data privacy.

The current web-based UI, though functional, could benefit from expanded customization options. Enhancing the UI to make it more flexible and user-friendly will help bridge the gap between non-expert users and the system's full capabilities. This improvement could allow a broader audience to fully leverage the system's power.

The system has primarily been tested using energy-sector datasets. A valuable next step would be to evaluate MASTerFlow with datasets from other industries, assessing how well it performs

in different contexts and real-world scenarios. This would help determine the system's robustness and adaptability.

During the development of the system, one of the challenges observed was that MAS frameworks like AutoGen perform well with powerful closed-sourced models like ChatGPT, but open-source models face challenges, particularly in tasks requiring complex reasoning. This issue likely stems from the reliance on zero-shot learning, which many open-source models often struggle with. To address this, a potential solution would be to provide examples of previously generated FL federations and ML code to the LLM. By providing context on how to structure the code and best practices, the LLM could generate more accurate responses. Introducing a degree of rigidity in the LLM's generation, by limiting the models it can generate and focusing on predefined examples and configurations, could improve accuracy without overly restricting flexibility. Additionally, the use of closed-source models raises concerns regarding data privacy and cost, making the integration of open-source models a critical future direction.

In summary, the proposed improvements could significantly enhance the system's robustness and efficiency. By expanding its scalability, refining the UI, broadening the testing scope, and improving the integration of open-source models, MASTerFlow has the potential to become a valuable tool for automating ML and FL pipelines across a range of industries.

References

Abu-Salih, B. (2021). *Domain-specific Knowledge Graphs: A survey* (arXiv:2011.00235). arXiv.

<https://doi.org/10.48550/arXiv.2011.00235>

Adetunji, A. B., Akande, O. N., Ajala, F. A., Oyewo, O., Akande, Y. F., & Oluwadara, G. (2022).

House Price Prediction using Random Forest Machine Learning Technique. *Procedia*

Computer Science, 199, 806–813. <https://doi.org/10.1016/j.procs.2022.01.100>

Agarwal, L., & Nasim, A. (2024). *Comparison and Analysis of Large Language Models (LLMs)*

(SSRN Scholarly Paper 4939534). Social Science Research Network.

<https://doi.org/10.2139/ssrn.4939534>

Alduailij, M., Khan, Q. W., Tahir, M., Sardaraz, M., Alduailij, M., & Malik, F. (2022). Machine-

Learning-Based DDoS Attack Detection Using Mutual Information and Random Forest

Feature Importance Method. *Symmetry*, 14(6), Article 6.

<https://doi.org/10.3390/sym14061095>

Al-Fraihat, D., Sharrab, Y., Alzyoud, F., Qahmash, A., & Maaita, A. (2024). Speech Recognition

Utilizing Deep Learning: A Systematic Review of the Latest Developments. *Human-*

Centric Computing and Information Sciences, 15.

<https://doi.org/10.22967/HGIS.2024.14.015>

- Almeida, J., Barbarroxa, R., Lezama, F., Soares, J., Gomes, L., Oliveira, F., & Vale, Z. (2023). *Meta-ERM: Metaheuristic Optimization Platform for Energy Resource Management in the Smart Grid* (p. 62). <https://doi.org/10.1109/CAI54212.2023.00034>
- Anusha, G., Malisetty, N., & Mannazhath, G. (2022). *Classification of Toxicity in Comments using NLP and LSTM* (p. 21). <https://doi.org/10.1109/ICACCS54159.2022.9785067>
- Banerjee, S., & Pedersen, T. (2003). The Design, Implementation, and Use of the Ngram Statistics Package. In A. Gelbukh (Ed.), *Computational Linguistics and Intelligent Text Processing* (pp. 370–381). Springer. https://doi.org/10.1007/3-540-36456-0_38
- Bansal, M., Goyal, A., & Choudhary, A. (2022). A comparative analysis of K-Nearest Neighbor, Genetic, Support Vector Machine, Decision Tree, and Long Short Term Memory algorithms in machine learning. *Decision Analytics Journal*, 3, 100071. <https://doi.org/10.1016/j.dajour.2022.100071>
- Barbarroxa, R., Gomes, L., & Vale, Z. (2025). Benchmarking Large Language Models for Multi-agent Systems: A Comparative Analysis of AutoGen, CrewAI, and TaskWeaver. In P. Mathieu & F. De la Prieta (Eds.), *Advances in Practical Applications of Agents, Multi-Agent Systems, and Digital Twins: The PAAMS Collection* (pp. 39–48). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-70415-4_4
- Barbarroxa, R., Ribeiro, B., Gomes, L., & Vale, Z. (2024). Benchmarking AutoGen with different large language models. *2024 IEEE Conference on Artificial Intelligence (CAI)*, 263–264. <https://doi.org/10.1109/CAI59869.2024.00058>
- Barbarroxa, R., Silva, J., Gomes, L., Lezama, F., Ribeiro, B., & Vale, Z. (2023). *FeDis: Federated Learning Framework Supported by Distributed Ledger* (pp. 32–41). https://doi.org/10.1007/978-3-031-45155-3_4
- BEKKERMAN, R., & Allan, J. (2003). *Using Bigrams in Text Categorization*.

- Beutel, D. J., Topal, T., Mathur, A., Qiu, X., Fernandez-Marques, J., Gao, Y., Sani, L., Li, K. H., Parcollet, T., de Gusmão, P. P. B., & Lane, N. D. (2022). *Flower: A Friendly Federated Learning Research Framework* (arXiv:2007.14390). arXiv.
<https://doi.org/10.48550/arXiv.2007.14390>
- Bratman, M. (1987). *Intention, Plans, and Practical Reason*. Cambridge, MA: Harvard University Press.
- Bretas, A. M. C., Mendes, A., Jackson, M., Clement, R., Sanhueza, C., & Chalup, S. (2023). A decentralised multi-agent system for rail freight traffic management. *Annals of Operations Research*, 320(2), 631–661. <https://doi.org/10.1007/s10479-021-04178-x>
- Brienza, M., Argenziano, F., Suriani, V., Bloisi, D. D., & Nardi, D. (2024). *Multi-Agent Planning Using Visual Language Models*. <https://doi.org/10.3233/FAIA240916>
- Bücker, M., Szepannek, G., Gosiewska, A., & Biecek, P. (2022). Transparency, auditability, and explainability of machine learning models in credit scoring. *Journal of the Operational Research Society*, 73(1), 70–90. <https://doi.org/10.1080/01605682.2021.1922098>
- Carrera, Á., & Iglesias, C. A. (2015). A systematic review of argumentation techniques for multi-agent systems research. *Artificial Intelligence Review*, 44(4), 509–535.
<https://doi.org/10.1007/s10462-015-9435-9>
- Casimiro-Artés, M. Á., Hileno, R., & Garcia-de-Alcaraz, A. (2024). Applying Unsupervised Machine Learning Models to Identify Serve Performance Related Indicators in Women's Volleyball. *Research Quarterly for Exercise and Sport*, 95(1), 47–53.
<https://doi.org/10.1080/02701367.2022.2142494>
- Chen, B., Zhang, Z., Langrené, N., & Zhu, S. (2024). *Unleashing the potential of prompt engineering in Large Language Models: A comprehensive review* (arXiv:2310.14735). arXiv. <https://doi.org/10.48550/arXiv.2310.14735>

- Chen, H., Ji, W., Xu, L., & Zhao, S. (2025). *Multi-Agent Consensus Seeking via Large Language Models* (arXiv:2310.20151). arXiv. <https://doi.org/10.48550/arXiv.2310.20151>
- Chen, J., Wang, Y., Zhang, Y., Lu, Y., Shu, Q., & Hu, Y. (2025). Extrinsic-and-Intrinsic Reward-Based Multi-Agent Reinforcement Learning for Multi-UAV Cooperative Target Encirclement. *IEEE Transactions on Intelligent Transportation Systems*, 1–13. IEEE Transactions on Intelligent Transportation Systems. <https://doi.org/10.1109/TITS.2024.3524562>
- Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794. <https://doi.org/10.1145/2939672.2939785>
- Chuor, P., Ittoo, A., & Heng, S. (2024). User Story Classification with Machine Learning and LLMs. In C. Cao, H. Chen, L. Zhao, J. Arshad, T. Asyhari, & Y. Wang (Eds.), *Knowledge Science, Engineering and Management* (pp. 161–175). Springer Nature. https://doi.org/10.1007/978-981-97-5492-2_13
- Costa, V. G., & Pedreira, C. E. (2023). Recent advances in decision trees: An updated survey. *Artificial Intelligence Review*, 56(5), 4765–4800. <https://doi.org/10.1007/s10462-022-10275-5>
- Dam, S. K., Hong, C. S., Qiao, Y., & Zhang, C. (2024). *A Complete Survey on LLM-based AI Chatbots* (arXiv:2406.16937). arXiv. <https://doi.org/10.48550/arXiv.2406.16937>
- Dastres, R., & Soori, M. (2021). Artificial Neural Network Systems. *International Journal of Imaging and Robotics (IJIR)*, 21(2), 13–25.
- Ding, W., Cao, Y., Zhao, D., Xiao, C., & Pavone, M. (2024). *RealGen: Retrieval Augmented Generation for Controllable Traffic Scenarios* (arXiv:2312.13303). arXiv. <https://doi.org/10.48550/arXiv.2312.13303>

- Draxler, F., Buschek, D., Tavast, M., Hämäläinen, P., Schmidt, A., Kulshrestha, J., & Welsch, R. (2023). *Gender, Age, and Technology Education Influence the Adoption and Appropriation of LLMs* (arXiv:2310.06556). arXiv. <https://doi.org/10.48550/arXiv.2310.06556>
- Du, H., Thudumu, S., Vasa, R., & Mouzakis, K. (2024). *A Survey on Context-Aware Multi-Agent Systems: Techniques, Challenges and Future Directions* (arXiv:2402.01968). arXiv. <https://doi.org/10.48550/arXiv.2402.01968>
- Emezue, C. (2024). *Beyond MLE: Investigating SEARNN for Low-Resourced Neural Machine Translation* (arXiv:2405.11819). arXiv. <https://doi.org/10.48550/arXiv.2405.11819>
- Es, S., James, J., Espinosa-Anke, L., & Schockaert, S. (2023). *RAGAS: Automated Evaluation of Retrieval Augmented Generation* (arXiv:2309.15217). arXiv. <https://doi.org/10.48550/arXiv.2309.15217>
- Fernandez-Quilez, A. (2023). Deep learning in radiology: Ethics of data and on the value of algorithm transparency, interpretability and explainability. *AI and Ethics*, 3(1), 257–265. <https://doi.org/10.1007/s43681-022-00161-9>
- File System API - Web APIs | MDN*. (2024, July 29). https://developer.mozilla.org/en-US/docs/Web/API/File_System_API
- FIPA, A. (2004). Fipa agent management specification. *Standard Component SC00023K, Foundation for Intelligent Physical Agents, 2*.
- Fluetsch, A., Di Lascio, E., Gerebtzoff, G., & Rodríguez-Pérez, R. (2024). Adapting Deep Learning QSPR Models to Specific Drug Discovery Projects. *Molecular Pharmaceutics*, 21(4), 1817–1826. <https://doi.org/10.1021/acs.molpharmaceut.3c01124>
- Fourrier, C., Habib, N., Lozovskaya, A., Szafer, K., & Wolf, T. (2024). *Open LLM Leaderboard v2*. Hugging Face. https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard

- Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., & Wang, H. (2024). *Retrieval-Augmented Generation for Large Language Models: A Survey* (arXiv:2312.10997). arXiv. <https://doi.org/10.48550/arXiv.2312.10997>
- Gezici, K., & Sengul, S. (2023). Estimation and analysis of missing temperature data in high altitude and snow-dominated regions using various machine learning methods. *Environmental Monitoring and Assessment*, 195. <https://doi.org/10.1007/s10661-023-11143-7>
- Ghimire, A., & Edwards, J. (2024). Generative AI Adoption in the Classroom: A Contextual Exploration Using the Technology Acceptance Model (TAM) and the Innovation Diffusion Theory (IDT). *2024 Intermountain Engineering, Technology and Computing (IETC)*, 129–134. <https://doi.org/10.1109/IETC61393.2024.10564292>
- Goncalves, C., Barreto, R., Faria, P., Gomes, L., & Vale, Z. (2022). Dataset of an energy community's generation and consumption with appliance allocation. *Data in Brief*, 45, 108590. <https://doi.org/10.1016/j.dib.2022.108590>
- Goonatilleke, M. A. S. T., Jayampath, M. W. G., & Hettige, B. (2019). Rice Express: A Communication Platform for Rice Production Industry. In J. Hemanth, T. Silva, & A. Karunananda (Eds.), *Artificial Intelligence* (pp. 269–277). Springer. https://doi.org/10.1007/978-981-13-9129-3_19
- Guan, W., Cao, J., Qian, S., & Gao, J. (2024). *LogLLM: Log-based Anomaly Detection Using Large Language Models* (arXiv:2411.08561). arXiv. <https://doi.org/10.48550/arXiv.2411.08561>
- Guha, N., Nyarko, J., Ho, D., Ré, C., Chilton, A., K, A., Chohlas-Wood, A., Peters, A., Waldon, B., Rockmore, D., Zambrano, D., Talisman, D., Hoque, E., Surani, F., Fagan, F., Sarfaty, G., Dickinson, G., Porat, H., Hegland, J., ... Li, Z. (2023). LegalBench: A Collaboratively Built

- Benchmark for Measuring Legal Reasoning in Large Language Models. *Advances in Neural Information Processing Systems*, 36, 44123–44279.
- Guo, D., Zhu, Q., Yang, D., Xie, Z., Dong, K., Zhang, W., Chen, G., Bi, X., Wu, Y., Li, Y. K., Luo, F., Xiong, Y., & Liang, W. (2024). *DeepSeek-Coder: When the Large Language Model Meets Programming -- The Rise of Code Intelligence* (arXiv:2401.14196). arXiv. <https://doi.org/10.48550/arXiv.2401.14196>
- Guo, T., Chen, X., Wang, Y., Chang, R., Pei, S., Chawla, N. V., Wiest, O., & Zhang, X. (2024). *Large Language Model based Multi-Agents: A Survey of Progress and Challenges* (arXiv:2402.01680; Version 2). arXiv. <https://doi.org/10.48550/arXiv.2402.01680>
- Gyawali, A., Adhikari, H., Aalto, M., & Ranta, T. (2024). From simple linear regression to machine learning methods: Canopy cover modelling of a young forest using planet data. *Ecological Informatics*, 82, 102706. <https://doi.org/10.1016/j.ecoinf.2024.102706>
- Hadi, M. U., Tashi, Q. A., Qureshi, R., Shah, A., Muneer, A., Irfan, M., Zafar, A., Shaikh, M. B., Akhtar, N., Wu, J., & Mirjalili, S. (2023). *A Survey on Large Language Models: Applications, Challenges, Limitations, and Practical Usage*. <https://doi.org/10.36227/techrxiv.23589741.v1>
- Han, S., Zhang, Q., Yao, Y., Jin, W., Xu, Z., & He, C. (2024). *LLM Multi-Agent Systems: Challenges and Open Problems* (arXiv:2402.03578; Version 1). arXiv. <https://doi.org/10.48550/arXiv.2402.03578>
- Han, Z., Gao, C., Liu, J., Zhang, J., & Zhang, S. Q. (2024). *Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey* (arXiv:2403.14608). arXiv. <https://doi.org/10.48550/arXiv.2403.14608>
- Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M.

- H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., ... Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362.
<https://doi.org/10.1038/s41586-020-2649-2>
- Hasan, M. R., & Ferdous, J. (2024). Dominance of AI and Machine Learning Techniques in Hybrid Movie Recommendation System Applying Text-to-number Conversion and Cosine Similarity Approaches. *Journal of Computer Science and Technology Studies*, *6*(1), Article 1. <https://doi.org/10.32996/jcsts.2024.6.1.10>
- Hassija, V., Chamola, V., Mahapatra, A., Singal, A., Goel, D., Huang, K., Scardapane, S., Spinelli, I., Mahmud, M., & Hussain, A. (2024). Interpreting Black-Box Models: A Review on Explainable Artificial Intelligence. *Cognitive Computation*, *16*(1), 45–74.
<https://doi.org/10.1007/s12559-023-10179-8>
- He, X., Tian, Y., Sun, Y., Chawla, N. V., Laurent, T., LeCun, Y., Bresson, X., & Hooi, B. (2024). *G-Retriever: Retrieval-Augmented Generation for Textual Graph Understanding and Question Answering* (arXiv:2402.07630). arXiv.
<https://doi.org/10.48550/arXiv.2402.07630>
- He, Z., Liang, T., Jiao, W., Zhang, Z., Yang, Y., Wang, R., Tu, Z., Shi, S., & Wang, X. (2024). Exploring Human-Like Translation Strategy with Large Language Models. *Transactions of the Association for Computational Linguistics*, *12*, 229–246.
https://doi.org/10.1162/tacl_a_00642
- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*, 1735–1780. <https://doi.org/10.1162/neco.1997.9.8.1735>
- Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Zhang, C., Wang, J., Wang, Z., Yau, S. K. S., Lin, Z., Zhou, L., Ran, C., Xiao, L., Wu, C., & Schmidhuber, J. (2024). *MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework* (arXiv:2308.00352). arXiv.
<https://doi.org/10.48550/arXiv.2308.00352>

- Hu, Y., Lei, Z., Zhang, Z., Pan, B., Ling, C., & Zhao, L. (2024). *GRAG: Graph Retrieval-Augmented Generation* (arXiv:2405.16506). arXiv. <https://doi.org/10.48550/arXiv.2405.16506>
- Huang, Z., Zheng, H., Li, C., & Che, C. (2024). Application of Machine Learning-Based K-means Clustering for Financial Fraud Detection. *Academic Journal of Science and Technology*, 10(1), Article 1. <https://doi.org/10.54097/74414c90>
- Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3), 90–95. <https://doi.org/10.1109/MCSE.2007.55>
- Ibrahim, I. M., & Al, E. (2021). Task Scheduling Algorithms in Cloud Computing: A Review. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, 12(4), Article 4.
- Ikotun, A. M., Ezugwu, A. E., Abualigah, L., Abuhaija, B., & Heming, J. (2023). K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data. *Information Sciences*, 622, 178–210. <https://doi.org/10.1016/j.ins.2022.11.139>
- Ismail, Z., & Sariff, N. (2018). *A Survey and Analysis of Cooperative Multi-Agent Robot Systems: Challenges and Directions*. <https://doi.org/10.5772/intechopen.79337>
- Jain, N., Zhang, T., Chiang, W.-L., Gonzalez, J. E., Sen, K., & Stoica, I. (2023). *LLM-Assisted Code Cleaning For Training Accurate Code Generators* (arXiv:2311.14904). arXiv. <https://doi.org/10.48550/arXiv.2311.14904>
- Jebli, I., Belouadha, F.-Z., Kabbaj, M. I., & Tilioua, A. (2021). Prediction of solar energy guided by pearson correlation using machine learning. *Energy*, 224, 120109. <https://doi.org/10.1016/j.energy.2021.120109>
- Ji, Y., Liu, Y., Yao, F., He, M., Tao, S., Zhao, X., Chang, S., Yang, X., Meng, W., Xie, Y., Chen, B., & Yang, H. (2024). *Adapting Large Language Models to Log Analysis with Interpretable*

- Domain Knowledge* (arXiv:2412.01377). arXiv.
<https://doi.org/10.48550/arXiv.2412.01377>
- Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. de las, Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., & Sayed, W. E. (2023). *Mistral 7B* (arXiv:2310.06825). arXiv. <https://doi.org/10.48550/arXiv.2310.06825>
- Jiang, F., Peng, Y., Dong, L., Wang, K., Yang, K., Pan, C., Niyato, D., & Dobre, O. A. (2024). Large Language Model Enhanced Multi-Agent Systems for 6G Communications. *IEEE Wireless Communications*, 31(6), 48–55. IEEE Wireless Communications. <https://doi.org/10.1109/MWC.016.2300600>
- Jiang, S., Dong, R., Wang, J., & Xia, M. (2023). Credit Card Fraud Detection Based on Unsupervised Attentional Anomaly Detection Network. *Systems*, 11(6), Article 6. <https://doi.org/10.3390/systems11060305>
- Jiang, S., Wang, Y., & Wang, Y. (2023). *SelfEvolve: A Code Evolution Framework via Large Language Models* (arXiv:2306.02907). arXiv. <https://doi.org/10.48550/arXiv.2306.02907>
- Jie, Z., Chen, S., Lai, J., Arif, M., & He, Z. (2023). Personalized federated recommendation system with historical parameter clustering. *Journal of Ambient Intelligence and Humanized Computing*, 14(8), 10555–10565. <https://doi.org/10.1007/s12652-022-03709-z>
- Ke, Y., Jin, L., Elangovan, K., Abdullah, H. R., Liu, N., Sia, A. T. H., Soh, C. R., Tung, J. Y. M., Ong, J. C. L., & Ting, D. S. W. (2024). *Development and Testing of Retrieval Augmented Generation in Large Language Models—A Case Study Report* (arXiv:2402.01733). arXiv. <https://doi.org/10.48550/arXiv.2402.01733>

- Khalil, R. A., Safelnasr, Z., Yemane, N., Kedir, M., Shafiqurrahman, A., & SAEED, N. (2024). Advanced Learning Technologies for Intelligent Transportation Systems: Prospects and Challenges. *IEEE Open Journal of Vehicular Technology*, 5, 397–427. IEEE Open Journal of Vehicular Technology. <https://doi.org/10.1109/OJVT.2024.3369691>
- Khan, F. H., Pasha, M. A., & Masud, S. (2021). Advancements in Microprocessor Architecture for Ubiquitous AI—An Overview on History, Evolution, and Upcoming Challenges in AI Implementation. *Micromachines*, 12(6), Article 6. <https://doi.org/10.3390/mi12060665>
- Khan, P. W., Byun, Y.-C., Lee, S.-J., Kang, D.-H., Kang, J.-Y., & Park, H.-S. (2020). Machine Learning-Based Approach to Predict Energy Consumption of Renewable and Nonrenewable Power Sources. *Energies*, 13(18), Article 18. <https://doi.org/10.3390/en13184870>
- Kiasari, M., Ghaffari, M., & Aly, H. H. (2024). A Comprehensive Review of the Current Status of Smart Grid Technologies for Renewable Energies Integration and Future Trends: The Role of Machine Learning and Energy Storage Systems. *Energies*, 17(16), Article 16. <https://doi.org/10.3390/en17164128>
- Kone, M., Akira, S., & Nakajima, T. (2000). The State of the Art in Agent Communication Languages. *Knowledge and Information Systems*, Vol. 2, pp.1-26. <https://doi.org/10.1007/PL00013712>
- Kumar, V., Gleyzer, L., Kahana, A., Shukla, K., & Karniadakis, G. E. (2023). MYCRUNCHGPT: A LLM ASSISTED FRAMEWORK FOR SCIENTIFIC MACHINE LEARNING. *Journal of Machine Learning for Modeling and Computing*, 4(4). <https://doi.org/10.1615/JMachLearnModelComput.2023049518>
- Lavanya, PM., & Sasikala, E. (2021). Deep Learning Techniques on Text Classification Using Natural Language Processing (NLP) In Social Healthcare Network: A Comprehensive

- Survey. *2021 3rd International Conference on Signal Processing and Communication (ICPSC)*, 603–609. <https://doi.org/10.1109/ICSPC51351.2021.9451752>
- Li, H., Xu, T., Chang, E., & Wen, Q. (2024). *Knowledge Tagging with Large Language Model based Multi-Agent System* (arXiv:2409.08406). arXiv. <https://doi.org/10.48550/arXiv.2409.08406>
- Li, Q., Wen, Z., Wu, Z., Hu, S., Wang, N., Li, Y., Liu, X., & He, B. (2023). A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection. *IEEE Transactions on Knowledge and Data Engineering*, *35*(4), 3347–3366. IEEE Transactions on Knowledge and Data Engineering. <https://doi.org/10.1109/TKDE.2021.3124599>
- Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., & Smith, V. (2018, December 14). *Federated Optimization in Heterogeneous Networks*. arXiv.Org. <https://arxiv.org/abs/1812.06127v5>
- Liang, Y., & Samavi, R. (2020). Optimization-based k-anonymity algorithms. *Computers & Security*, *93*, 101753. <https://doi.org/10.1016/j.cose.2020.101753>
- Liaw, A., & Wiener, M. (2001). Classification and Regression by RandomForest. *Forest*, *23*.
- Lim, M. K., Tan, K., & Leung, S. C. H. (2013). Using a multi-agent system to optimise resource utilisation in multi-site manufacturing facilities. *International Journal of Production Research*, *51*(9), 2620–2638. <https://doi.org/10.1080/00207543.2012.737953>
- Liu, J., Keqin, L., Zhu, A., Hong, B., Zhao, P., Dai, S., Wei, C., Huang, W., & Su, H. (2024). Application of Deep Learning-Based Natural Language Processing in Multilingual Sentiment Analysis. *Mediterranean Journal of Basic and Applied Sciences*, *08*, 243–260. <https://doi.org/10.46382/MJBAS.2024.8219>

- Liu, Y., Cao, J., Liu, C., Ding, K., & Jin, L. (2024). *Datasets for Large Language Models: A Comprehensive Survey* (arXiv:2402.18041). arXiv.
<https://doi.org/10.48550/arXiv.2402.18041>
- Liu, Y., Yan, S., Zhao, Y., Song, C., & Li, F. (2022). Improved Dyna-Q: A Reinforcement Learning Method Focused via Heuristic Graph for AGV Path Planning in Dynamic Environments. *Drones*, 6(11), Article 11. <https://doi.org/10.3390/drones6110365>
- Liu, Z., Yang, Q., & Zou, J. (2024). Lowering Costs and Increasing Benefits Through the Ensemble of LLMs and Machine Learning Models. In D.-S. Huang, C. Zhang, & Y. Pan (Eds.), *Advanced Intelligent Computing Technology and Applications* (pp. 368–379). Springer Nature. https://doi.org/10.1007/978-981-97-5666-7_31
- Long, T., Zhang, D., Li, G., Taraif, B., Menon, S., Smith, K. S., Wang, S., Gero, K. I., & Chilton, L. B. (2023). *Tweetorial Hooks: Generative AI Tools to Motivate Science on Social Media* (arXiv:2305.12265). arXiv. <http://arxiv.org/abs/2305.12265>
- Luo, F.-M., Xu, T., Lai, H., Chen, X.-H., Zhang, W., & Yu, Y. (2024). A survey on model-based reinforcement learning. *Science China Information Sciences*, 67(2), 121101.
<https://doi.org/10.1007/s11432-022-3696-5>
- Lyu, J., Ma, X., Li, X., & Lu, Z. (2022). Mildly Conservative Q-Learning for Offline Reinforcement Learning. *Advances in Neural Information Processing Systems*, 35, 1711–1724.
- Lyu, Y., Li, Z., Niu, S., Xiong, F., Tang, B., Wang, W., Wu, H., Liu, H., Xu, T., & Chen, E. (2025). CRUD-RAG: A Comprehensive Chinese Benchmark for Retrieval-Augmented Generation of Large Language Models. *ACM Trans. Inf. Syst.*, 43(2), 41:1-41:32.
<https://doi.org/10.1145/3701228>
- Ma, L., Yang, W., Jiang, S., Fei, B., Zhou, M., Li, S., Xu, B., & Xiao, Y. (2024). *LUK: Empowering Log Understanding with Expert Knowledge from Large Language Models* (arXiv:2409.01909). arXiv. <https://doi.org/10.48550/arXiv.2409.01909>

- Mahmud, K., Azam, S., Karim, A., Zobaed, S., Shanmugam, B., & Mathur, D. (2021). Machine Learning Based PV Power Generation Forecasting in Alice Springs. *IEEE Access*, 9, 46117–46128. IEEE Access. <https://doi.org/10.1109/ACCESS.2021.3066494>
- Mandal, S., Chethan, A., Janfaza, V., Mahmud, S. M. F., Anderson, T. A., Turek, J., Tithi, J. J., & Muzahid, A. (2023). *Large Language Models Based Automatic Synthesis of Software Specifications* (arXiv:2304.09181). arXiv. <https://doi.org/10.48550/arXiv.2304.09181>
- Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, ... Xiaoqiang Zheng. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. <https://www.tensorflow.org/>
- Marukat, S. (2023). Tutorial on PCA and approximate PCA and approximate kernel PCA. *Artificial Intelligence Review*, 56(6), 5445–5477. <https://doi.org/10.1007/s10462-022-10297-z>
- McKinney, W. (2010). Data Structures for Statistical Computing in Python. In S. van der Walt & J. Millman (Eds.), *Proceedings of the 9th Python in Science Conference* (pp. 56–61). <https://doi.org/10.25080/Majora-92bf1922-00a>
- Mei, T., Zi, Y., Cheng, X., Gao, Z., Wang, Q., & Yang, H. (2024). *Efficiency optimization of large-scale language models based on deep learning in natural language processing tasks* (arXiv:2405.11704). arXiv. <https://doi.org/10.48550/arXiv.2405.11704>
- Menghani, G. (2023). Efficient Deep Learning: A Survey on Making Deep Learning Models Smaller, Faster, and Better. *ACM Comput. Surv.*, 55(12), 259:1-259:37. <https://doi.org/10.1145/3578938>

- Mhammedi, Z., Block, A., Foster, D. J., & Rakhlin, A. (2023). Efficient Model-Free Exploration in Low-Rank MDPs. *Advances in Neural Information Processing Systems*, 36, 66782–66817.
- Miessler, D. (2024). *Danielmiessler/fabric* [Python]. <https://github.com/danielmiessler/fabric> (Original work published 2024)
- Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., & Gao, J. (2024). *Large Language Models: A Survey* (arXiv:2402.06196). arXiv. <https://doi.org/10.48550/arXiv.2402.06196>
- Mohr, F., & van Rijn, J. N. (2024). Learning curves for decision making in supervised machine learning: A survey. *Machine Learning*, 113(11), 8371–8425. <https://doi.org/10.1007/s10994-024-06619-7>
- Moura, J. (2023). *crewAI*. GitHub. <https://github.com/crewAIInc/crewAI>
- Muhammad, K., Wang, Q., O'Reilly-Morgan, D., Tragos, E., Smyth, B., Hurley, N., Geraci, J., & Lawlor, A. (2020). FedFast: Going Beyond Average for Faster Training of Federated Recommender Systems. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1234–1242. <https://doi.org/10.1145/3394486.3403176>
- Naeem, S., Ali, A., Anam, S., & Ahmed, M. (2023). An Unsupervised Machine Learning Algorithms: Comprehensive Review. *IJCDS Journal*, 13, 911–921. <https://doi.org/10.12785/ijcds/130172>
- Niknam, S., Dhillon, H. S., & Reed, J. H. (2020). Federated Learning for Wireless Communications: Motivation, Opportunities, and Challenges. *IEEE Communications Magazine*, 58(6), 46–51. IEEE Communications Magazine. <https://doi.org/10.1109/MCOM.001.1900461>

- Noh, S.-H. (2021). Analysis of Gradient Vanishing of RNNs and Performance Comparison. *Information*, 12(11), Article 11. <https://doi.org/10.3390/info12110442>
- Oliveira, F., Lezama, F., Gomes, L., Soares, J., & Vale, Z. (2023). Risk assessment model based on centrifugal governors and artificial neural networks. *2023 IEEE Conference on Artificial Intelligence (CAI)*, 218–219. <https://doi.org/10.1109/CAI54212.2023.00101>
- Olu-Ajayi, R., Alaka, H., Sulaimon, I., Sunmola, F., & Ajayi, S. (2022). Building energy consumption prediction for residential buildings using deep learning and other machine learning techniques. *Journal of Building Engineering*, 45, 103406. <https://doi.org/10.1016/j.jobbe.2021.103406>
- OpenAI. (2024). *Swarm*. GitHub. <https://github.com/openai/swarm>
- Pal, M., Mittal, M. L., Soni, G., Chouhan, S. S., & Kumar, M. (2023). A multi-agent system for FJSP with setup and transportation times. *Expert Systems with Applications*, 216, 119474. <https://doi.org/10.1016/j.eswa.2022.119474>
- Palanca, J., Rincon, J. A., Carrascosa, C., Julian, V. J., & Terrasa, A. (2023). Flexible Agent Architecture: Mixing Reactive and Deliberative Behaviors in SPADE. *Electronics*, 12(3), Article 3. <https://doi.org/10.3390/electronics12030659>
- Paleyas, A., Urma, R.-G., & Lawrence, N. D. (2022). Challenges in Deploying Machine Learning: A Survey of Case Studies. *ACM Comput. Surv.*, 55(6), 114:1-114:29. <https://doi.org/10.1145/3533378>
- Pan, J., Liang, W. S., & Yidi, Y. (2024). RAGLog: Log Anomaly Detection using Retrieval Augmented Generation. *2024 IEEE World Forum on Public Safety Technology (WFPST)*, 169–174. <https://doi.org/10.1109/WFPST58552.2024.00034>
- Pan, J., Sadé, A., Kim, J., Soriano, E., Sole, G., & Flamant, S. (2023). *SteloCoder: A Decoder-Only LLM for Multi-Language to Python Code Translation* (arXiv:2310.15539). arXiv. <https://doi.org/10.48550/arXiv.2310.15539>

- Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). *Generative Agents: Interactive Simulacra of Human Behavior* (arXiv:2304.03442). arXiv.
<https://doi.org/10.48550/arXiv.2304.03442>
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library* (arXiv:1912.01703). arXiv.
<https://doi.org/10.48550/arXiv.1912.01703>
- Patankar, S., Usakoyal, C., Patil, P., & Raut, K. (2024). A Survey of Deep Reinforcement Learning in Game Playing. *2024 MIT Art, Design and Technology School of Computing International Conference (MITADTSoCiCon)*, 1–5.
<https://doi.org/10.1109/MITADTSoCiCon60330.2024.10575819>
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(85), 2825–2830.
- Peng, B., Zhu, Y., Liu, Y., Bo, X., Shi, H., Hong, C., Zhang, Y., & Tang, S. (2024). *Graph Retrieval-Augmented Generation: A Survey* (arXiv:2408.08921). arXiv.
<https://doi.org/10.48550/arXiv.2408.08921>
- Pythagora-io/gpt-pilot*. (2024). [Python]. Pythagora-io. <https://github.com/Pythagora-io/gpt-pilot> (Original work published 2023)
- Qian, C., Cong, X., Liu, W., Yang, C., Chen, W., Su, Y., Dang, Y., Li, J., Xu, J., Li, D., Liu, Z., & Sun, M. (2023). *Communicative Agents for Software Development* (arXiv:2307.07924). arXiv. <http://arxiv.org/abs/2307.07924>

- Qian, Y., Hu, L., Chen, J., Guan, X., Hassan, M. M., & Alelaiwi, A. (2019). Privacy-aware service placement for mobile edge computing via federated learning. *Information Sciences*, 505, 562–570. <https://doi.org/10.1016/j.ins.2019.07.069>
- Qiao, B., Li, L., Zhang, X., He, S., Kang, Y., Zhang, C., Yang, F., Dong, H., Zhang, J., Wang, L., Ma, M., Zhao, P., Qin, S., Qin, X., Du, C., Xu, Y., Lin, Q., Rajmohan, S., & Zhang, D. (2024). *TaskWeaver: A Code-First Agent Framework* (arXiv:2311.17541). arXiv. <https://doi.org/10.48550/arXiv.2311.17541>
- Quan, Y., & Liu, Z. (2024). *InvAgent: A Large Language Model based Multi-Agent System for Inventory Management in Supply Chains* (arXiv:2407.11384). arXiv. <https://doi.org/10.48550/arXiv.2407.11384>
- Ramos, D., Faria, P., & Vale, Z. (2024). Analysis of the Consumption and Sensors Features Contribution to the Consumption Forecast Using Explainable AI in Buildings. *2024 22nd International Conference on Intelligent Systems Applications to Power Systems (ISAP)*, 1–5. <https://doi.org/10.1109/ISAP63260.2024.10744322>
- Rao, A. S., & Georgeff, M. P. (1995). *BDI Agents: From Theory to Practice*.
- Ribeiro, B., Dias, D., Gomes, L., & Vale, Z. (2025). PEAK: Python-based framework for heterogeneous agent communities. *SoftwareX*, 30, 102190. <https://doi.org/10.1016/j.softx.2025.102190>
- Ribeiro, B., Gomes, L., & Vale, Z. (2025). Retrieval-Augmented Generation Based Assistant: A Smart Home Case Study. In M. F. Santos, J. Machado, P. Novais, P. Cortez, & P. M. Moreira (Eds.), *Progress in Artificial Intelligence* (pp. 160–171). Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-73500-4_14
- Safyari, Y., Mahdianpari, M., & Shiri, H. (2024). A Review of Vision-Based Pothole Detection Methods Using Computer Vision and Machine Learning. *Sensors*, 24(17), Article 17. <https://doi.org/10.3390/s24175652>

- Saxena, A., Agarwal, A., Pandey, B. K., & Pandey, D. (2024). Examination of the Criticality of Customer Segmentation Using Unsupervised Learning Methods. *Circular Economy and Sustainability*, 4(2), 1447–1460. <https://doi.org/10.1007/s43615-023-00336-4>
- Schmidhuber, J. (2015). Deep Learning in Neural Networks: An Overview. *Neural Networks*, 61, 85–117. <https://doi.org/10.1016/j.neunet.2014.09.003>
- Schneider, A., Hommel, G., & Blettner, M. (2010). Linear Regression Analysis. *Deutsches Ärzteblatt International*, 107(44), 776–782. <https://doi.org/10.3238/arztebl.2010.0776>
- Shanahan, M., McDonell, K., & Reynolds, L. (2023). Role play with large language models. *Nature*, 623(7987), 493–498. <https://doi.org/10.1038/s41586-023-06647-8>
- Shang, J., Yan, J., & Ren, F. (2024). BDI Agents Based Dynamic Resource Allocation in Emergency Scenarios. *2024 IEEE International Conference on Agents (ICA)*, 45–49. <https://doi.org/10.1109/ICA63002.2024.00018>
- Sharma, S., Singh, G., & Sharma, M. (2021). A comprehensive review and analysis of supervised-learning and soft computing techniques for stress diagnosis in humans. *Computers in Biology and Medicine*, 134, 104450. <https://doi.org/10.1016/j.combiomed.2021.104450>
- Singh, B., Kumar, R., & Singh, V. P. (2022). Reinforcement learning in robotic applications: A comprehensive survey. *Artificial Intelligence Review*, 55(2), 945–990. <https://doi.org/10.1007/s10462-021-09997-9>
- Singh, U., Rizwan, M., Alaraj, M., & Alsaidan, I. (2021). A Machine Learning-Based Gradient Boosting Regression Approach for Wind Power Production Forecasting: A Step towards Smart Grid Environments. *Energies*, 14(16), Article 16. <https://doi.org/10.3390/en14165196>

- Slimani, H., Mhamdi, J. E., & Jilbab, A. (2024). Deep Learning Structure for Real-time Crop Monitoring Based on Neural Architecture Search and UAV. *Brazilian Archives of Biology and Technology*, 67, e24231141. <https://doi.org/10.1590/1678-4324-2024231141>
- Smith. (1980). The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, C-29(12), 1104–1113. <https://doi.org/10.1109/TC.1980.1675516>
- Staudemeyer, R. C., & Morris, E. R. (2019, September 12). *Understanding LSTM -- a tutorial into Long Short-Term Memory Recurrent Neural Networks*. arXiv.Org. <https://arxiv.org/abs/1909.09586v1>
- Supriyono, Wibawa, A. P., Suyono, & Kurniawan, F. (2024). A survey of text summarization: Techniques, evaluation and challenges. *Natural Language Processing Journal*, 7, 100070. <https://doi.org/10.1016/j.nlp.2024.100070>
- Tabany, M., & Gueffal, M. (2024). Sentiment Analysis and Fake Amazon Reviews Classification Using SVM Supervised Machine Learning Model. *Journal of Advances in Information Technology*, 15(1), 49–58. <https://doi.org/10.12720/jait.15.1.49-58>
- Talebirad, Y., & Nadiri, A. (2023). *Multi-Agent Collaboration: Harnessing the Power of Intelligent LLM Agents* (arXiv:2306.03314). arXiv. <http://arxiv.org/abs/2306.03314>
- Tan, B., Liu, B., Zheng, V., & Yang, Q. (2020). A Federated Recommender System for Online Services. *Proceedings of the 14th ACM Conference on Recommender Systems*, 579–581. <https://doi.org/10.1145/3383313.3411528>
- Tan, N. (2024). *Advancements and Innovations in Recommendation Systems: From Traditional Algorithms to Deep Learning Evolution*. 638–644. https://doi.org/10.2991/978-94-6463-512-6_67

- Taranjeet Singh, D. Y. (2023). Embedchain: The Open Source RAG Framework. In *GitHub repository*. GitHub. <https://github.com/embedchain/embedchain>
- Teixeira, B., Valina, L., Pinto, T., Reis, A., Barroso, J., & Vale, Z. (2024). Exploring Clustering to Improve Interpretability in Complex Energy Forecasting Models. *2024 International Conference on Smart Energy Systems and Technologies (SEST)*, 1–6. <https://doi.org/10.1109/SEST61601.2024.10694413>
- Theocharides, S., Makrides, G., Livera, A., Theristis, M., Kaimakis, P., & Georghiou, G. E. (2020). Day-ahead photovoltaic power production forecasting methodology based on machine learning and statistical post-processing. *Applied Energy*, *268*, 115023. <https://doi.org/10.1016/j.apenergy.2020.115023>
- Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., Cola, G. D., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Tan, H., & Younis, O. G. (2024). *Gymnasium: A Standard Interface for Reinforcement Learning Environments* (arXiv:2407.17032). arXiv. <https://doi.org/10.48550/arXiv.2407.17032>
- Tsai, C.-Y., Kim, J., Jin, F., Jun, M., Cheong, M., & Yammarino, F. J. (2022). Polynomial regression analysis and response surface methodology in leadership research. *The Leadership Quarterly*, *33*(1), 101592. <https://doi.org/10.1016/j.leaqua.2021.101592>
- Tunstall, L., Beeching, E., Lambert, N., Rajani, N., Rasul, K., Belkada, Y., Huang, S., Werra, L. von, Fourier, C., Habib, N., Sarrazin, N., Sanseviero, O., Rush, A. M., & Wolf, T. (2023). *Zephyr: Direct Distillation of LM Alignment*.
- Valkenborg, D., Geubbelmans, M., Rousseau, A.-J., & Burzykowski, T. (2023). Supervised learning. *American Journal of Orthodontics and Dentofacial Orthopedics*, *164*(1), 146–149. <https://doi.org/10.1016/j.ajodo.2023.04.010>

- Värtinen, S., Hämäläinen, P., & Guckelsberger, C. (2024). Generating Role-Playing Game Quests With GPT Language Models. *IEEE Transactions on Games*, 16(1), 127–139. IEEE Transactions on Games. <https://doi.org/10.1109/TG.2022.3228480>
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023). *Attention Is All You Need* (arXiv:1706.03762). arXiv. <https://doi.org/10.48550/arXiv.1706.03762>
- Volum, R., Rao, S., Xu, M., DesGarennés, G., Brockett, C., Van Durme, B., Deng, O., Malhotra, A., & Dolan, B. (2022). Craft an Iron Sword: Dynamically Generating Interactive Game Characters by Prompting Large Language Models Tuned on Code. In M.-A. Côté, X. Yuan, & P. Ammanabrolu (Eds.), *Proceedings of the 3rd Wordplay: When Language Meets Games Workshop (Wordplay 2022)* (pp. 25–43). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2022.wordplay-1.3>
- Wang, J., Hong, Y., Wang, J., Xu, J., Tang, Y., Han, Q.-L., & Kurths, J. (2022). Cooperative and Competitive Multi-Agent Systems: From Optimization to Games. *IEEE/CAA Journal of Automatica Sinica*, 9(5), 763–783. <https://doi.org/10.1109/JAS.2022.105506>
- Wang, K., Lu, Y., Santacrose, M., Gong, Y., Zhang, C., & Shen, Y. (2024). *Adapting LLM Agents with Universal Feedback in Communication* (arXiv:2310.01444). arXiv. <https://doi.org/10.48550/arXiv.2310.01444>
- Wang, M., Guo, X., She, Y., Zhou, Y., Liang, M., & Chen, Z. S. (2024). Advancements in Deep Learning Techniques for Time Series Forecasting in Maritime Applications: A Comprehensive Review. *Information*, 15(8), Article 8. <https://doi.org/10.3390/info15080507>
- Wang, S., Khramtsova, E., Zhuang, S., & Zuccon, G. (2024). FeB4RAG: Evaluating Federated Search in the Context of Retrieval Augmented Generation. *Proceedings of the 47th*

- International ACM SIGIR Conference on Research and Development in Information Retrieval*, 763–773. <https://doi.org/10.1145/3626772.3657853>
- Wang, X., Kang, Y., Hyndman, R. J., & Li, F. (2023). Distributed ARIMA Models for Ultra-long Time Series. *International Journal of Forecasting*, 39(3), 1163–1184. <https://doi.org/10.1016/j.ijforecast.2022.05.001>
- Weerasinghe, S., Hettige, B., Kathriarachchi, R., & Karunananda, A. (2016). *ITray: Multi-agent Solution for LAN Based File Sharing*.
- Wei, K., Li, J., Ding, M., Ma, C., Yang, H. H., Farokhi, F., Jin, S., Quek, T. Q. S., & Vincent Poor, H. (2020). Federated Learning With Differential Privacy: Algorithms and Performance Analysis. *IEEE Transactions on Information Forensics and Security*, 15, 3454–3469. *IEEE Transactions on Information Forensics and Security*. <https://doi.org/10.1109/TIFS.2020.2988575>
- Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., & Wang, C. (2023). *AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation* (arXiv:2308.08155). arXiv. <https://doi.org/10.48550/arXiv.2308.08155>
- Wu, S., Irsoy, O., Lu, S., Dabrovolski, V., Dredze, M., Gehrmann, S., Kambadur, P., Rosenberg, D., & Mann, G. (2023). *BloombergGPT: A Large Language Model for Finance* (arXiv:2303.17564). arXiv. <https://doi.org/10.48550/arXiv.2303.17564>
- Wu, S., Xiong, Y., Cui, Y., Wu, H., Chen, C., Yuan, Y., Huang, L., Liu, X., Kuo, T.-W., Guan, N., & Xue, C. J. (2024). *Retrieval-Augmented Generation for Natural Language Processing: A Survey* (arXiv:2407.13193). arXiv. <https://doi.org/10.48550/arXiv.2407.13193>
- Xiao, W., Yang, Y., Mu, X., Xie, Y., Tang, X., Cao, D., & Liu, T. (2024). Decision-Making for Autonomous Vehicles in Random Task Scenarios at Unsignalized Intersection Using Deep Reinforcement Learning. *IEEE Transactions on Vehicular Technology*, 73(6),

- 7812–7825. *IEEE Transactions on Vehicular Technology*.
- <https://doi.org/10.1109/TVT.2024.3360445>
- Xu, S., & Zhang, X. (2023). *Leveraging generative artificial intelligence to simulate student learning behavior* (arXiv:2310.19206). arXiv. <http://arxiv.org/abs/2310.19206>
- Xu, Y., Wang, S., Li, P., Luo, F., Wang, X., Liu, W., & Liu, Y. (2024). *Exploring Large Language Models for Communication Games: An Empirical Study on Werewolf* (arXiv:2309.04658). arXiv. <https://doi.org/10.48550/arXiv.2309.04658>
- Yan, B., Zhang, X., Zhang, L., Zhang, L., Zhou, Z., Miao, D., & Li, C. (2025). *Beyond Self-Talk: A Communication-Centric Survey of LLM-Based Multi-Agent Systems* (arXiv:2502.14321). arXiv. <https://doi.org/10.48550/arXiv.2502.14321>
- Yang, H., Liu, X.-Y., & Wang, C. D. (2023). *FinGPT: Open-Source Financial Large Language Models* (arXiv:2306.06031). arXiv. <https://doi.org/10.48550/arXiv.2306.06031>
- Yang, H., Yue, S., & He, Y. (2023). *Auto-GPT for Online Decision Making: Benchmarks and Additional Opinions* (arXiv:2306.02224). arXiv. <https://doi.org/10.48550/arXiv.2306.02224>
- Yang, S., Mao, X., Yang, S., & Liu, Z. (2017). Towards a hybrid software architecture and multi-agent approach for autonomous robot software. *International Journal of Advanced Robotic Systems*, 14(4), 1729881417716088. <https://doi.org/10.1177/1729881417716088>
- Yang, T., Andrew, G., Eichner, H., Sun, H., Li, W., Kong, N., Ramage, D., & Beaufays, F. (2018). *Applied Federated Learning: Improving Google Keyboard Query Suggestions* (arXiv:1812.02903). arXiv. <https://doi.org/10.48550/arXiv.1812.02903>
- Yu, Y., Ping, W., Liu, Z., Wang, B., You, J., Zhang, C., Shoeybi, M., & Catanzaro, B. (2024). *RankRAG: Unifying Context Ranking with Retrieval-Augmented Generation in LLMs* (arXiv:2407.02485). arXiv. <https://doi.org/10.48550/arXiv.2407.02485>

- Zhan, Y., Zhang, J., Hong, Z., Wu, L., Li, P., & Guo, S. (2022). A Survey of Incentive Mechanism Design for Federated Learning. *IEEE Transactions on Emerging Topics in Computing*, *10*(2), 1035–1044. *IEEE Transactions on Emerging Topics in Computing*.
<https://doi.org/10.1109/TETC.2021.3063517>
- Zhang, C., Sjarif, N. N. A., & Ibrahim, R. (2024). Deep learning models for price forecasting of financial time series: A review of recent advancements: 2020–2022. *WIREs Data Mining and Knowledge Discovery*, *14*(1), e1519. <https://doi.org/10.1002/widm.1519>
- Zhang, C., Xie, Y., Bai, H., Yu, B., Li, W., & Gao, Y. (2021). A survey on federated learning. *Knowledge-Based Systems*, *216*, 106775.
<https://doi.org/10.1016/j.knosys.2021.106775>
- Zhang, D., Feng, G., Shi, Y., & Srinivasan, D. (2021). Physical Safety and Cyber Security Analysis of Multi-Agent Systems: A Survey of Recent Advances. *IEEE/CAA Journal of Automatica Sinica*, *8*(2), 319–333. <https://doi.org/10.1109/JAS.2021.1003820>
- Zhang, W., Zhang, Q., Yu, E., Ren, Y., Meng, Y., Qiu, M., & Wang, J. (2024). LogRAG: Semi-Supervised Log-based Anomaly Detection with Retrieval-Augmented Generation. *2024 IEEE International Conference on Web Services (ICWS)*, 1100–1102.
<https://doi.org/10.1109/ICWS62655.2024.00129>
- Zhang, X., Chen, Z. Z., Ye, X., Yang, X., Chen, L., Wang, W. Y., & Petzold, L. R. (2024). *Unveiling the Impact of Coding Data Instruction Fine-Tuning on Large Language Models Reasoning* (arXiv:2405.20535). arXiv. <https://doi.org/10.48550/arXiv.2405.20535>
- Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., ... Wen, J.-R. (2024). *A Survey of Large Language Models* (arXiv:2303.18223). arXiv.
<https://doi.org/10.48550/arXiv.2303.18223>

- Zhao, Y., Zhao, J., Jiang, L., Tan, R., & Niyato, D. (2019). Mobile Edge Computing, Blockchain and Reputation-based Crowdsourcing IoT Federated Learning: A Secure, Decentralized and Privacy-preserving System. *ArXiv*.
<https://www.semanticscholar.org/paper/Mobile-Edge-Computing%2C-Blockchain-and-Crowdsourcing-Zhao-Zhao/7c7ccf8defa9c61eb2487af1249974b3f1213957>
- Zhou, Y., Liu, Y., Li, X., Jin, J., Qian, H., Liu, Z., Li, C., Dou, Z., Ho, T.-Y., & Yu, P. S. (2024). *Trustworthiness in Retrieval-Augmented Generation Systems: A Survey* (arXiv:2409.10102). arXiv. <https://doi.org/10.48550/arXiv.2409.10102>
- Zou, H., Zhao, Q., Bariah, L., Bennis, M., & Debbah, M. (2023). *Wireless Multi-Agent Generative AI: From Connected Intelligence to Collective Intelligence* (arXiv:2307.02757). arXiv. <https://doi.org/10.48550/arXiv.2307.02757>

Annex

```
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:15:15,221 |  
run_simulation.py:401 | Pre-registering run with id 4051440859517936841  
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:15:15,222 | state_factory.py:50 |  
Using InMemoryState  
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:15:15,222 | state_factory.py:50 |  
Using InMemoryState  
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:15:15,225 |  
run_simulation.py:423 | Buffer time delay: 5s  
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:15:15,224 | app.py:74 | Starting  
Flower ServerApp, config: num_rounds=3, round_timeout=999s  
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:15:15,228 | app.py:79 |  
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:15:15,229 | server.py:92 | [INIT]  
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:15:15,230 | server.py:281 |  
Requesting initial parameters from one random client  
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:15:20,227 | state_factory.py:50 |  
Using InMemoryState  
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:15:20,227 | vce_api.py:59 |  
Registered 3 nodes  
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:15:20,228 | vce_api.py:321 |  
Supported backends: ['ray']  
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:15:20,228 | raybackend.py:46 |  
Initialising: RayBackend  
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:15:20,228 | raybackend.py:47 |  
Backend config: {'init_args': {'logging_level': 30, 'log_to_driver': True}, 'client_resources': {'num_cpus':  
2, 'num_gpus': 0}, 'actor': {'tensorflow': 0}}  
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:15:33,042 | raybackend.py:141 |  
Constructed ActorPool with: 6 actors  
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:15:33,044 | state_factory.py:50 |  
Using InMemoryState  
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:17:18,406 | server.py:288 |  
Received initial parameters from one random client  
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:17:18,407 | server.py:94 | Starting  
evaluation of initial global parameters  
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:17:18,407 | server.py:106 |  
Evaluation returned no results ('None')  
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:17:18,408 | server.py:112 |
```

```

cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:17:18,409 | server.py:113 | [ROUND
1]
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:17:18,410 | server.py:226 |
configure_fit: strategy sampled 3 clients (out of 3)
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:19:30,120 | server.py:240 |
aggregate_fit: received 3 results and 0 failures
cacfe70b-2024_12_27_15_00_44-Test | WARNING flwr 2024-12-27 15:19:30,125 | fedavg.py:252 |
No fit_metrics_aggregation_fn provided
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:19:30,126 | server.py:178 |
configure_evaluate: strategy sampled 3 clients (out of 3)
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:21:15,901 | server.py:192 |
aggregate_evaluate: received 3 results and 0 failures
cacfe70b-2024_12_27_15_00_44-Test | WARNING flwr 2024-12-27 15:21:15,903 | fedavg.py:283 |
No evaluate_metrics_aggregation_fn provided
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:21:15,903 | server.py:112 |
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:21:15,904 | server.py:113 | [ROUND
2]
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:21:15,904 | server.py:226 |
configure_fit: strategy sampled 3 clients (out of 3)
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:23:38,525 | server.py:240 |
aggregate_fit: received 3 results and 0 failures
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:23:38,528 | server.py:178 |
configure_evaluate: strategy sampled 3 clients (out of 3)
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:25:49,444 | server.py:192 |
aggregate_evaluate: received 3 results and 0 failures
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:25:49,446 | server.py:112 |
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:25:49,446 | server.py:113 | [ROUND
3]
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:25:49,447 | server.py:226 |
configure_fit: strategy sampled 3 clients (out of 3)
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:28:26,579 | server.py:240 |
aggregate_fit: received 3 results and 0 failures
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:28:26,582 | server.py:178 |
configure_evaluate: strategy sampled 3 clients (out of 3)
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:30:35,109 | server.py:192 |
aggregate_evaluate: received 3 results and 0 failures
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:30:35,111 | server.py:496 |
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:30:35,112 | server.py:497 |
[SUMMARY]
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:30:35,112 | server.py:498 | Run
finished 3 round(s) in 796.70s
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:30:35,113 | server.py:500 |
History (loss, distributed):
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:30:35,114 | server.py:500 |
round 1: 0.04595337063074112
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:30:35,116 | server.py:500 |
round 2: 0.045863907784223557
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:30:35,116 | server.py:500 |
round 3: 0.04587125778198242
cacfe70b-2024_12_27_15_00_44-Test | INFO flwr 2024-12-27 15:30:35,117 | server.py:501 |
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:30:35,119 | run_serverapp.py:89
| ServerApp finished running.
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:30:35,119 |
run_simulation.py:349 | ServerApp finished running.

```

```

cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:30:35,120 |
run_simulation.py:353 | Triggered stop event for Simulation Engine.
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:30:36,106 | ray_actor.py:450 |
Terminated 6 actors
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:30:36,942 | raybackend.py:193 |
Terminated RayBackend
cacfe70b-2024_12_27_15_00_44-Test | DEBUG flwr 2024-12-27 15:30:36,943 |
run_simulation.py:455 | Stopping Simulation Engine now.

```

Figure 51. An example of generated FL training logs

Public Case Studies					Q Search	X	Download
Name	Type	Status	Uploader	File			
> UC-33Bus_2040_1800_EVs	Normal	public	Admin	DOWNLOAD			
> gecco15sc	Risk Based	public	Admin	DOWNLOAD			
> cec100sc	Normal	public	Admin	DOWNLOAD			
> cec200sc	Normal	public	Admin	DOWNLOAD			

Figure 52. Web Meta-ERM public case studies display