Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
Consiglio Nazionale delle Ricerche

ISTI Technical Reports

HumAInFlow: a No-Code Platform for Modelling and Simulating Human AI Workflows

Giovanna Broccia, ISTI-CNR, Pisa, Italy
Roberto Cirillo, ISTI-CNR, Pisa, Italy
Alessio Ferrari, ISTI-CNR, Pisa, Italy
Lucio Lelii, ISTI-CNR, Pisa, Italy
Giorgio O. Spagnolo, ISTI-CNR, Pisa, Italy

ISTI-TR-2025/011

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
Consiglio Nazionale delle Ricerche

HumAInFlow : a no-code platform for modelling and simulating human-AI workflows

Giovanna Broccia; Roberto Cirillo; Alessio Ferrari; Lucio Lelii; Giorgio O. Spagnolo.
ISTI-TR-2025/011

The increasing integration of Generative AI (GenAI) agents into socio-technical systems, calls for platforms that can model, simulate, and analyse workflows involving both automated and human tasks. Existing agentic AI frameworks largely focus on automation and remain tightly bound to specific SDKs, often lacking structured support for human-in-the-loop modelling and simulation. To address these limitations, we introduce HumAInFlow, a no-code platform for modelling and simulating socio-technical workflows that explicitly integrates human roles as first-class nodes and supports their simulation via large language models (LLMs). The platform is SDK-agnostic, supports both local and remote LLM execution, and integrates the Model Context Protocol (MCP), ensuring interoperability and extensibility. Comparative analysis shows that HumAInFlow advances the state of the art by combining privacy-preserving deployment, execution monitoring, reproducibility, and explicit support for human–AI collaboration.

Keywords: Large Language Models (LLMs), Workflow Orchestration, Socio- Technical Systems, Human-AI Collaboration, Agentic AI Frameworks.

Citation
Broccia G., Cirillo R., Ferrari A., Lelii L., Spagnolo G.O. *HumAInFlow : a no-code platform for modelling and simulating human-AI workflows*. Technical Reports 2025/011. DOI: 10.32079/ISTI-TR-2025/011.

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
Area della Ricerca CNR di Pisa
Via G. Moruzzi 1
56124 Pisa Italy
http://www.isti.cnr.it

HumAInFlow: A No-Code Platform for Modelling and Simulating Human-AI Workflows

Giovanna Broccia[1], Roberto Cirillo[1], Alessio Ferrari [1,2], Lucio Lelii[1], and Giorgio Oronzo Spagnolo[1]

[1]Istituto di Scienza e Tecnologie dell'Informazione 'A. Faedo' – CNR, Pisa, Italy
[2] University College Dublin, Ireland

September 2025

Abstract

The increasing integration of Generative AI (GenAI) agents into socio-technical systems, calls for platforms that can model, simulate, and analyse workflows involving both automated and human tasks. Existing agentic AI frameworks largely focus on automation and remain tightly bound to specific SDKs, often lacking structured support for human-in-the-loop modelling and simulation. To address these limitations, we introduce HumAInFlow, a no-code platform for modelling and simulating socio-technical workflows that explicitly integrates human roles as first-class nodes and supports their simulation via large language models (LLMs). The platform is SDK-agnostic, supports both local and remote LLM execution, and integrates the Model Context Protocol (MCP), ensuring interoperability and extensibility. Comparative analysis shows that HumAInFlow advances the state of the art by combining privacy-preserving deployment, execution monitoring, reproducibility, and explicit support for human–AI collaboration.

Keywords— Large Language Models (LLMs), Workflow Orchestration, Socio-Technical Systems, Human-AI Collaboration, Agentic AI Frameworks

1 Introduction

Generative AI (GenAI) agents are autonomous systems that combine generative models—such as large language models (LLMs)—with goal-directed reasoning to perform complex tasks, generate content, and interact with digital or physical environments [10]. These agents are increasingly being integrated into decision-making processes across a wide range of domains—including healthcare, finance, and public services—where they are embedded in socio-technical systems involving interactions among humans, organisations, and technologies. This growing integration raises the need for tools that

can model, simulate, and analyse such complex human–AI workflows, allowing both technical and non-technical stakeholders to design and evaluate processes that include both automated and human tasks.

Despite the emergence of several Agentic AI frameworks—such as Langflow, Flowise AI, and AutoGen Studio—existing tools primarily focus on fully automated pipelines, often lacking structured support for human-in-the-loop modelling and role simulation. Additionally, these platforms are often tightly coupled to specific SDKs, limiting their long-term adaptability and interoperability.

To address these gaps, we introduce *HumAInFlow*, a no-code modelling and simulation platform that allows users to design workflows involving both GenAI agents and human roles. The platform enables graphical modelling of task sequences, communication flows, and decision logic through directed graphs, where nodes represent the actors involved in the processes and edges capture information flow and dependencies. Human roles can also be simulated using LLMs, allowing process testing in the absence of real users.

HumAInFlow is implemented in pure Python, is SDK-agnostic, supports both remote and local LLMs, and is released under a permissive MIT license. It is evolving toward compatibility with the Model Context Protocol (MCP) and supports containerised deployment (e.g., via Docker or Kubernetes) to enable reuse and integration in distributed environments.

2 Supported Features

The tool is designed to support the visual modelling and simulation of socio-technical workflows in which genAI agents and human actors collaboratively perform complex, interdependent tasks. Each task is represented as a node in the workflow, capturing both the actor responsible for execution (e.g., an AI agent or a human user) and the expected input–output relationships. This abstraction allows users to model realistic processes where automation and human judgement are intertwined.

The platform enables users to define not only automated steps driven by large language models, but also moments of human involvement (e.g review, confirmation, or decision-making) which are essential in domains where full automation is either infeasible or undesirable. These human nodes can also be simulated using LLMs, providing a means to test and iterate on workflow design even in the absence of actual users. Nodes are connected through directed edges that establish the flow of information and control across the workflow. This allows for dynamic process execution, where the output of one node feeds into the next, reflecting realistic dependencies and transitions between human and AI contributions. As workflows evolve, users can simulate their execution using real or simulated agents, examine intermediate states, and inspect the flow of data and decisions through the process.

Users interact with the platform through a no-code graphical interface, which simplifies the creation, modification, and simulation of nodes and workflows without requiring programming expertise. Each node represents a distinct actor or component within a socio-technical process. When creating a node (see Figure 1), users can specify its actor type (i.e. whether the node is executed by a human, an AI agent, or a simulated human via LLM), its task (i.e. the function or activity it is intended to perform), its input (i.e. the data or conditions required to activate the node), its output (i.e. the structure and type of information it will produce, to be passed to subsequent nodes), and—where applicable—its prompt and model configuration (e.g.

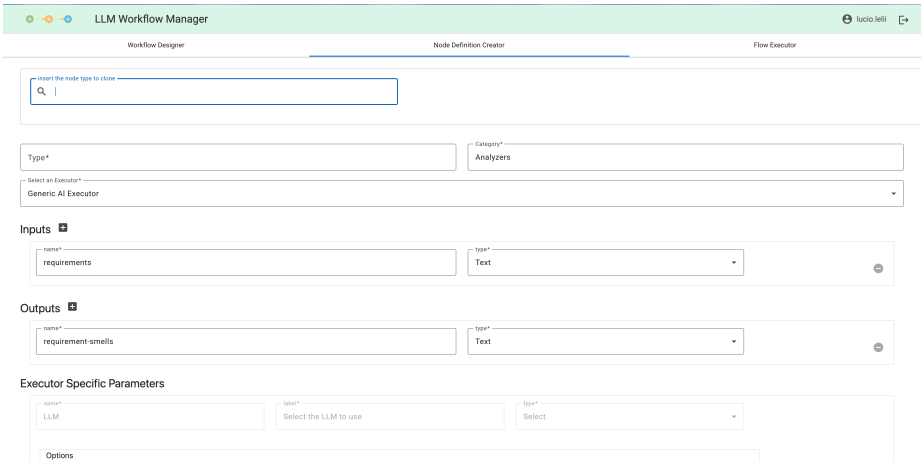

Figure 1: Node creation interface.

LLM prompt, model name, temperature, and other generation settings for AI-based nodes).

Once defined, nodes can be assembled into a complete workflow through the no-code editor by dragging and dropping them onto the canvas and drawing connections between them (see Figure 2). These directed edges represent the flow of information, decisions, or control throughout the process.

Workflows can be executed directly within the platform (see Figure 3); after execution, users can view and inspect the results through the interface. Each execution preserves the full system state—including all inputs, node configurations, and model settings—at the moment of execution. This ensures reproducibility, meaning that other users can obtain the same results by re-running the workflow under identical conditions, and replicability, allowing others to apply the same method to new data while expecting similar outcomes. Even if the user later modifies the workflow, the executed version is retained, along with all relevant metadata. This separation ensures that changes to the model do not affect past executions, enabling transparent comparison, iterative refinement, and rigorous validation of socio-technical processes.

The platform also supports collaborative design, allowing users to share workflows within the same platform instance. Nodes and workflows can be imported and exported individually, enabling reuse and exchange across different projects or local installations.

The tool supports local and remote LLM execution, offering flexibility in deployment and facilitating privacy-sensitive use cases where on-premise models are preferred. Workflows can be exported as standalone services or containerised components, ensuring integration with existing infrastructures. The platform further supports team collaboration through workflow sharing and role-based access control, allowing multiple users to co-design, edit, and test workflows.

As an example scenario, the platform can be applied to the early phases of software development, where users model workflows involving collaboration between humans and GenAI agents. In this case (further detailed in Section 4), the platform automates the transformation of a natural language stakeholder interview into structured artefacts — such as textual requirements, UML class and sequence diagrams — and

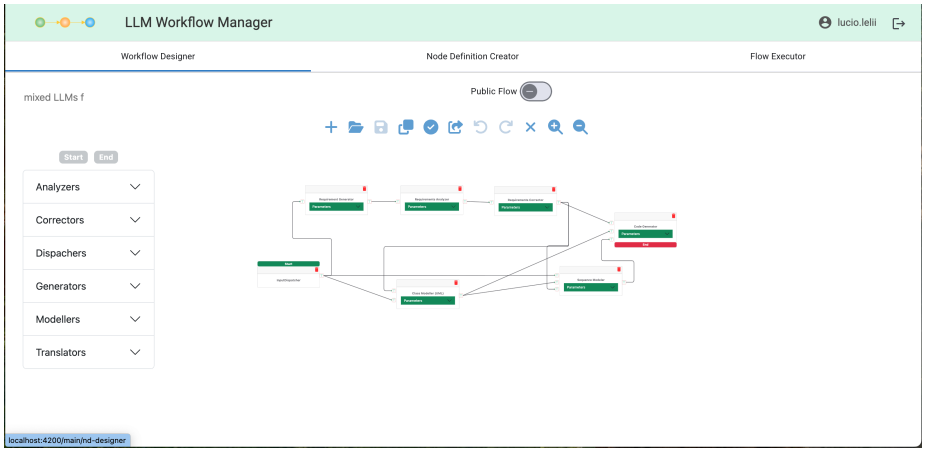

Figure 2: Workflow design editor.

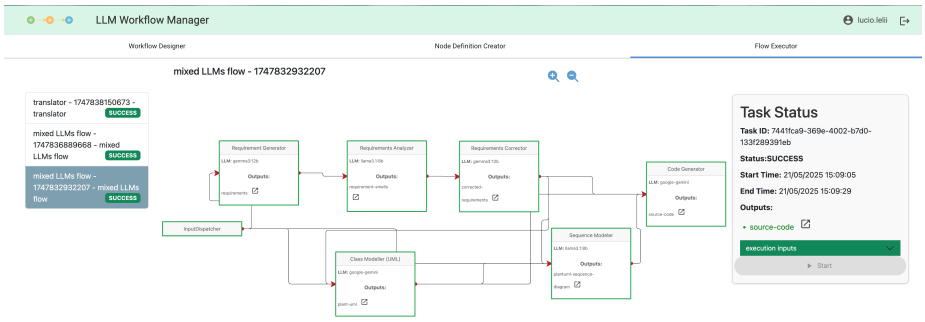

Figure 3: Workflow design editor.

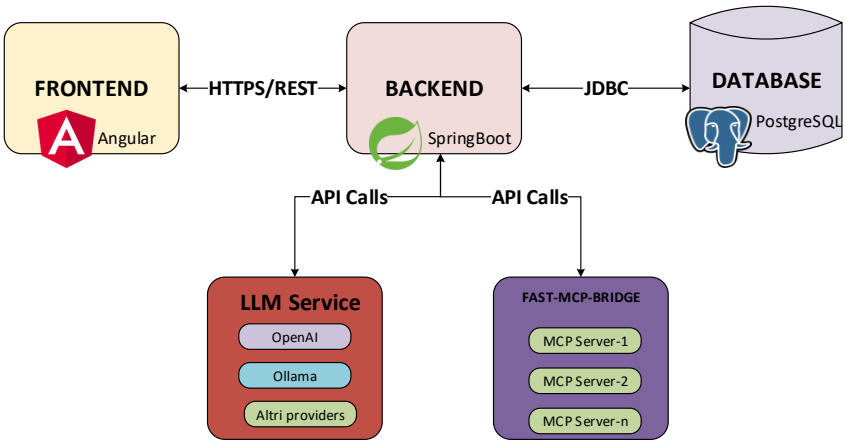

Figure 4: HumAInFlow system architecture

ultimately into executable code.

3 HumAInFlow Architecture

The system architecture is designed as a multi-layered structure, enabling efficient system management, secure user authentication, and flexible integration with different LLMs. The key components of this architecture (see Figure 4) are: a Frontend interface, an application Backend, a database management system, and an integration layer with LLM services. This integration layer is supported by the *fast-mcp-bridge*, which provides a modular interface to both local and remote LLM providers and ensures interoperability through the Model Context Protocol (MCP).

Frontend The Frontend module, developed in Angular[1], is responsible for the user interface (UI), providing an intuitive and responsive experience. Its main functions include:

- Display, creation, and modification of workflows.
- Management of user settings.

[1] Angular[1] is an open-source framework based on TypeScript for building dynamic and high-performance single-page web applications (SPAs). Developed by Google, it provides a complete platform with a robust structure and a set of tools for managing the development of complex frontend applications.

- Handling of login and registration forms, sending credentials to the Backend for verification.
- Communication with the Backend via HTTP/S requests (RESTful APIs) to retrieve data, save changes, and start processes.
- Management of the local application state (e.g., information about the authenticated user, temporary form data).

Backend The Backend module, developed with Spring Boot[2], acts as the central core of the system, managing business logic, security, and the orchestration of data and services. Its responsibilities include:

- Exposing secure RESTful APIs for consumption by the Frontend.
- Implementing business logic related to workflows (creation, execution, monitoring).
- Handling user authentication and authorisation, verifying credentials and managing permissions.
- Communicating with the PostgreSQL database for data persistence, including the storage of user information and workflow details.
- Interfacing with external LLM services.

Database For the secure storage of user data and the preservation of workflow definitions, their instances, states, and execution history, a relational database (PostgreSQL) has been chosen.

Integration with LLM Services (Managed by the Backend) The Backend includes dedicated components for direct interfacing with both local and remote LLM providers, such as Ollama[12] and OpenAI[13], as well as indirect integration via the *fast-mcp-bridge* component, which facilitates the use of AI agents through MCP protocol support. This integration is designed to be modular, allowing the system to:

- Send requests (prompts) and receive responses from language models or AI agents.
- Easily incorporate support for additional LLM providers in the future.
- Manage provider-specific configurations (e.g., API keys, endpoints).

Fast-MCP-Bridge Thanks to support for the MCP (Model Context Protocol) [8], the fast-mcp-bridge enables the combination of local or remote MCP servers with different LLMs, whether local (hosted via Ollama) or remote (via OpenAI APIs) (see Figure 5). Integration with the MCP protocol is implemented through the *fastmcp* framework [19], which provides compatibility and extensibility by allowing heterogeneous services and models to interoperate seamlessly.

[2]Spring Boot[16] is a Java framework that simplifies and accelerates the development of Spring applications, making them standalone (directly executable) and production-ready with minimal configuration.

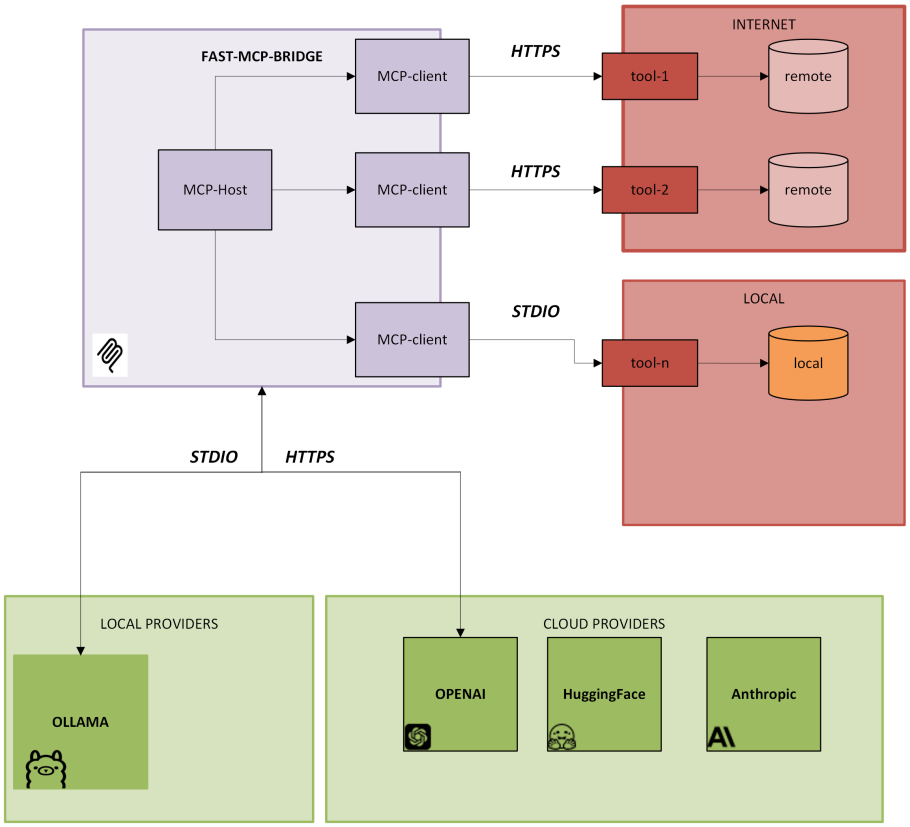

Figure 5: Architecture diagram of fast-mcp-bridge component

Connectivity with LLM providers is achieved through adapters that dynamically translate the MCP data model into the specific data model required by each provider. Currently, local integration is supported exclusively via Ollama, while remote integration is enabled through OpenAI; however, the modular design makes it straightforward to extend support to additional providers (e.g., HuggingFace, Anthropic). Using Ollama offers important benefits in terms of privacy—keeping data fully on-premise—and cost reduction by avoiding dependence on commercial APIs.

To interact with *HumAInFlow*, the fast-mcp-bridge exposes a REST interface, implemented with FastAPI. This interface allows users to specify the LLM to be used, one or more MCP servers with which the model will interact, and the prompts with their associated configuration parameters (e.g., temperature, max tokens). In this way, *HumAInFlow* can flexibly combine models and tools while managing the logic of their interactions.

This architecture is designed to separate concerns, thereby improving the maintainability, scalability, and testability of the overall system.

The adoption of specialised technologies for each layer (Angular for a dynamic user interface, Spring Boot for robust and secure business logic, PostgreSQL for reliable data management) leverages the strengths of each, optimising performance and user experience. Furthermore, the well-defined APIs exposed by the backend not only serve the current Angular frontend but also enable future reusability, allowing integration with other client applications (e.g., mobile apps) or external systems with minimal effort.

From a security perspective, the backend acts as a central control point, protecting the database and managing access to LLM services. Finally, this layered design promotes system resilience: potential issues in one component, such as the temporary unavailability of an LLM service, can be more effectively managed, minimising the impact on the overall application and ensuring greater reliability.

4 Example Scenario

This section illustrates a possible usage of the *HumAInFlow* platform through a worked-out example scenario. The idea behind the scenario shown in Figure 6, is to automate the transformation of a natural language interview with stakeholders — who intend to develop a Calculator application — into executable code. This is achieved step by step through the generation of system requirements, UML diagrams (Class and Sequence Diagrams), and finally source code. Each block in Figure 6 corresponds to a specialised prompt designed to perform a specific task.

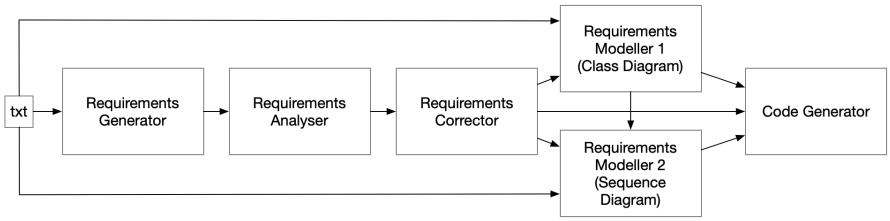

Figure 6: High-level workflow of the example scenario

The workflow consists of a sequence of interconnected nodes (see Figure 7).In the following, we describe each node in more detail, specifying its inputs, outputs, and, where applicable, the prompt and an example of the generated output.

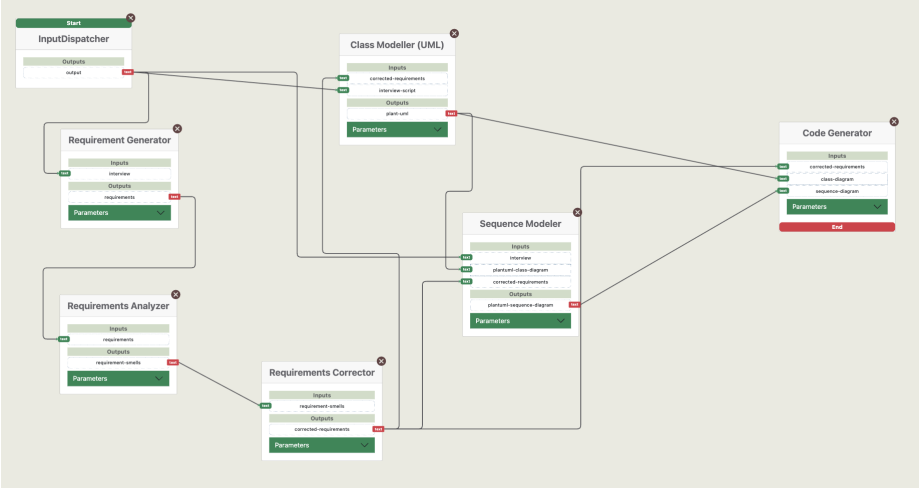

Figure 7: Workflow representation in *HumAInFlow*

Input Dispatcher Initial module that collects and routes stakeholder interview transcripts.

Input

Textual transcription of stakeholder interviews (`.txt` format), either pasted directly or uploaded from a file.

Output

The raw transcript, structured as the initial input artefact, passed downstream to the *Requirements Generator* and the two requirements modellers (i.e., *Class Modeller (UML)* and *Sequence Modeller*).

Requirements Generator Extracts functional requirements from interview transcripts.

Input

Interview transcript, provided in `.txt` format (from the Input Dispatcher node).

Output

CSV file where each line contains: *ID; requirement text; customer quote supporting requirement.*

Prompt

Analyze the following interview transcript and extract a list of functional requirements.
Format the output as a CSV with the following columns:

1. **ID**: Unique identifier (e.g., F1, F2, F3...).
2. **Requirement Text**: Clear description of what the system must do (in plain language).
3. **Customer Quote**: Exact phrase from the transcript that justifies the requirement.

Rules:

- Include **only functional requirements** (no qualities or non-functional requirements).
- Keep requirements concise and actionable.
- List requirements in logical order.
- Make sure requirements are free of smells.
- Output only the CSV, do not add any text.

CSV Format Example:

- F1; The system shall ask users to select the arithmetic operation to perform. The possible arithmetic operations are addition, subtraction, multiplication, division; "I need some support to perform different arithmetic operations".
- F2: The system shall ask users to select the first operand of the operation; "I want to express the entire operation verbally, and get the results".
- F3: The system shall ask users to select the second operand of the operation; "I want to express the entire operation verbally, and get the results".

Transcript to Analyze: {Input Transcript}
Output Requirements (CSV Format): {Output Requirements}

Example Output

F1; The system shall allow users to create a task; "I need to keep track of what I have to do"
F2; The system shall allow users to delete an existing task; "Sometimes I complete things and want them removed"

Requirements Analyser Analyse the requirements to detect potential smells.

Input

CSV file containing the list of functional requirements, as produced by the Requirements Generator. Each row has the structure: *ID; requirement text; customer quote supporting requirement.*

Output

CSV file containing the analysed requirements with potential smells. Each row has the structure: *ID; requirement text; customer quote supporting requirement; smell description.* If no smell is detected, the *smell description* field is left empty.

Prompt

Act as a **Requirements Analyst** tasked with reviewing a set of functional requirements derived from a customer interview. Analyze each requirement for potential **requirement smells** (e.g., ambiguity, incompleteness, inconsistency, or vagueness) and generate a CSV report with the following columns:

1. **ID**: Unique identifier of the requirement (e.g., F1, F2).
2. **Requirement**: The exact text of the requirement.
3. **Customer Quote**: Supporting quote from the stakeholder.
4. **Smell Description**:
    - If a smell is detected, briefly describe it (e.g., "Ambiguous: 'user-friendly' is subjective").
    - If no smell is found, leave this field empty (null).

Rules:

- Read the requirement carefully, and think whether it can be tested
- If it cannot be tested, understand what the smell is that makes it not testable.
- Typical smells are: Ambiguity: Subjective terms (e.g., "easy," "fast"). Incompleteness: Missing steps/triggers (e.g., "remind users" but no timing specified). Inconsistency: Conflicts with other requirements. Vagueness: Unquantifiable metrics (e.g., "often," "sometimes").
- Output only the CSV, do not add any text.

Output CSV Format Example:

- F1; The system shall ask users to select the arithmetic operation to perform. The possible arithmetic operations are addition, subtraction, multiplication, division; "I need some support to perform different arithmetic operations"; null
- F2; The system shall ask users to select the first operand of the operation; "I want to express the entire operation verbally, and get the results"; users is plural, can be ambiguous

Input Requirements (CSV Format): {Input Requirements}
Output Requirements with Smells (CSV Format): {Output Requirements with Smells}

Requirements Corrector Correct requirements where smells have been identified.

Input

CSV file containing the analysed requirements as produced by the Requirements Analyser. Each row has the structure: *ID; requirement text; customer quote supporting requirement; smell description.*

Output

CSV file containing corrected requirements. Each row has the structure: *ID; corrected requirement text; customer quote supporting requirement.*

Prompt

Act as a **Requirements Corrector**. Your task is to revise requirements containing smells (ambiguity, vagueness, inconsistency, incompleteness) into **clear, actionable, and testable** functional requirements.
Input: A CSV of requirements with identified smells (columns: `ID`, `Requirement`, `Customer Quote`, `Smell Description`).
Output Rules:

- Generate a new CSV with the following columns:
  1. **ID**: Original requirement identifier (e.g., F1, F2).
  2. **Corrected Requirement**: Rewritten version that resolves the smell.
  3. **Customer Quote**: Original supporting quote from the transcript.
- Remove the *Smell Description* column (corrections should speak for themselves).
- Preserve intent: ensure the corrected requirement aligns with the stakeholder's original need.
- For non-smelly requirements: copy them verbatim into the new CSV.
- Output only the CSV, do not add any text.

Output CSV Format Example:

- F1; The system shall ask users to select the arithmetic operation to perform. The possible arithmetic operations are addition, subtraction, multiplication, division; "I need some support to perform different arithmetic operations"
- F2; The system shall ask the user to select the first operand of the operation; "I want to express the entire operation verbally, and get the results"

Input Requirements with Smells (CSV Format): {Input Requirements with Smells}
Output Corrected Requirements (CSV Format): {Corrected Requirements}

Class Modeller (UML) Generate a UML Class Diagram in textual PlantUML syntax.

Input

Two inputs:

1. Corrected requirements (CSV format with *ID; Corrected Requirement; Customer Quote*).
2. Original interview transcript (for contextual information).

Output

Textual UML Class Diagram written in `PlantUML` syntax. The output can be directly visualised using tools such as: `https://www.plantuml.com/plantuml/uml/`.

Prompt

Act as a **Requirements Modeller** specialising in structural design. Create a PlantUML class diagram that captures the core domain model implied by the functional requirements and interview transcript.
Input:

1. Corrected requirements (CSV with `ID`, `Corrected Requirement`, `Customer Quote`).
2. Interview transcript for context.

Output Rules:

- Generate one concise class diagram in textual PlantUML syntax.
- Focus on:
  - Key entities (classes) and their attributes.
  - Critical relationships (associations, compositions).
  - Basic operations/methods implied by requirements.
- Output only the model; do not add any explanatory text.

Input Data: {Corrected Requirements} {Interview Transcript}
Output: {Class Diagram}

Example Output

```
@startuml Calculator_Class_Diagram

class Calculator {
  + currentInput: Number
  + storedValue: Number
  + selectedOperation: OperationType
  + calculate(): Number
```

```
  + clear(): void
}

class InputHandler {
  + handleDigitPress(digit: Integer): void
  + handleOperator(op: OperationType): void
  + handleEquals(): void
}

class Display {
  + updateScreen(value: String): void
}

enum OperationType <<enum>> {
  ADD
  SUBTRACT
  MULTIPLY
  DIVIDE
}

Calculator "1" *-- "1" OperationType
Calculator --> Display : updates
InputHandler --> Calculator : commands

note top of Calculator : "Must support basic
arithmetic\noperations (F2)"
note right of InputHandler : "User quote: 'I need buttons
for numbers and functions...'"
@enduml
```

Sequence Modeller Generate a UML Sequence Diagram in textual PlantUML syntax.

Input

Three inputs:

1. Corrected requirements (CSV format with *ID; Corrected Requirement; Customer Quote*).
2. Original interview transcript (for contextual information).
3. UML Class Diagram (PlantUML format) as generated by the Class Modeller.

Output

Textual UML Sequence Diagram written in `PlantUML` syntax. The output can be directly visualised using tools such as: `https://www.plantuml.com/plantuml/uml/`.

Prompt

Act as a **Requirements Modeller** specialising in interaction design. Create a PlantUML sequence diagram that visually captures the key system interactions described in the functional requirements and interview transcript.

Input:

1. Corrected requirements (CSV with `ID`, `Corrected Requirement`, `Customer Quote`).
2. Original interview transcript for context.
3. Class diagram (PlantUML format).

Output Rules:

- Generate one comprehensive sequence diagram in textual PlantUML syntax.
- Use only the classes from the input class diagram.
- Focus on:
  - Primary user-to-system interactions.
  - Critical system responses.
  - Important timing constraints (if specified).
- Include only the most essential interactions (5–10 steps maximum).
- Output only the model; do not add explanatory text.

Input Data: {Corrected Requirements} {Interview Transcript} {Class Diagram}
Output: {Sequence Diagram}

Example Output

```
@startuml Calculator_Sequence_Diagram

actor User
participant InputHandler
participant Calculator
participant Display

User -> InputHandler : Press digit "5"
note right of User: "I need buttons for numbers..." (Requirement F1)
InputHandler -> Calculator : Store input (currentInput = 5)
Calculator -> Display : Update screen ("5")

User -> InputHandler : Press operator "+"
note right of User: "Basic arithmetic is essential" (F3)
InputHandler -> Calculator : Set operation (ADD)

User -> InputHandler : Press digit "3"
InputHandler -> Calculator : Store input (currentInput = 3)
```

```
Calculator -> Display : Update screen ("3")

User -> InputHandler : Press "="
InputHandler -> Calculator : calculate()
Calculator -> Calculator : Compute 5 + 3
Calculator -> Display : Update screen ("8")

@enduml
```

Code Generator Generate executable Python code from requirements and UML diagrams.

Input

Three inputs:

1. Corrected requirements (CSV format with *ID; Corrected Requirement; Customer Quote*).
2. UML Class Diagram (PlantUML format) from the Class Modeller.
3. UML Sequence Diagram (PlantUML format) from the Sequence Modeller.

Output

Executable source code in Python. The generated code reflects the structure of the UML class diagram and the interaction flows from the UML sequence diagram, while implementing the validated requirements.

Prompt

Act as a **Senior Software Engineer** specialising in **model-driven development**. Generate clean, maintainable code that:

1. Implements all validated requirements (from CSV).
2. Matches the structure of the provided class diagram.
3. Follows the interaction flows from the sequence diagram.

Input:

- `requirements.csv` (ID, Corrected Requirement, Customer Quote)
- `class_diagram` (PlantUML)
- `sequence_diagram` (PlantUML)

Output Rules:

- **Language:** Python.
- **Architecture:**
  - 1:1 mapping of PlantUML classes to Python classes.
  - Sequence diagram steps → method logic.

- **Quality:**
  - Type hints (Python).
  - Docstrings referencing requirement IDs (e.g., # F1).
  - Appropriate error handling.
- **Format:** Output only executable Python code, with comments if needed. No additional explanatory text.

Input Data: {Corrected Requirements} {Class Diagram} {Sequence Diagram}
Output: {Python Source Code}

Example Output

```
from enum import Enum, auto

class OperationType(Enum):
    """Supported operations (matches class diagram <<enum>>)"""
    ADD = auto()      # F3: Basic arithmetic
    SUBTRACT = auto() # F3
    MULTIPLY = auto() # F3
    DIVIDE = auto()   # F3

class Calculator:
    """Core calculator logic (from class diagram)"""
    def __init__(self):
        self.current_input = 0      # F1: Digit entry
        self.stored_value = None    # For multi-step operations
        self.selected_op = None     # F3: Operation selection

    def calculate(self) -> float:
        """Execute the current operation (sequence diagram '=' press)"""
        if not self.selected_op:
            return self.current_input

        # Sequence diagram compute step
        match self.selected_op:
            case OperationType.ADD:
                result = self.stored_value + self.current_input
            case OperationType.SUBTRACT:
                result = self.stored_value - self.current_input
            # ... other operations (F3)

        self.current_input = result
        return result

class InputHandler:
    """Manages user input flow (sequence diagram interactions)"""
```

```
    def __init__(self, calculator: Calculator, display: Display):
        self.calc = calculator
        self.display = display

    def handle_digit(self, digit: int):  # F1
        """Process digit press (sequence diagram step 1)"""
        self.calc.current_input = self.calc.current_input * 10 + digit
        self.display.update(str(self.calc.current_input))

    def handle_operator(self, op: OperationType):  # F3
        """Process +,-,*,/ press (sequence diagram step 2)"""
        self.calc.stored_value = self.calc.current_input
        self.calc.selected_op = op
        self.calc.current_input = 0
```

5 State of the Art

The growing adoption of Generative AI (GenAI) in socio-technical systems has driven the emergence of various platforms aimed at designing, orchestrating, and deploying AI-enhanced workflows. In line with Yu et al. [20], we analyse these tools by considering both *system-level features* (which determine backend architecture, extensibility, and execution) and *interaction-level features* (which focus on user interaction, collaboration, and human-AI integration).

For this study, we selected a representative set of platforms that are either widely adopted in practice or actively discussed in the research and developer community: *Langflow* [3], *Flowise AI* [2], *n8n* [11], *Dive* [5], *Dify* [4], *Haystack UI* [7], *AgenticFlow* [17], and *AutoGen Studio* [6]. These tools were chosen because they illustrate different design choices and trade-offs in GenAI workflow orchestration. Some (e.g., Langflow, Flowise AI) are strongly tied to popular orchestration frameworks such as LangChain, while others (e.g., n8n, Dify) position themselves as more general-purpose workflow automation platforms with GenAI extensions. Research-driven solutions such as *Haystack UI* and *Dive* highlight community efforts toward open experimentation, while emerging tools such as *AgenticFlow* and *AutoGen Studio* explore advanced agent-based and multi-LLM orchestration. Together, this set of tools offers a balanced view of the current landscape, covering both open-source and proprietary models, code-based and no-code approaches, and varying levels of support for extensibility, collaboration, and human involvement. This diversity makes them suitable benchmarks against which to position *HumAInFlow*.

The two-dimensional perspective is consistent with the main challenges identified in the literature, namely: ensuring extensibility and interoperability (e.g., MCP support, SDK independence) [20], maintaining flexible and reliable execution (monitoring, logging, error handling) [10], and lowering barriers to adoption via accessible interfaces and explicit support for human-AI collaboration [14, 18].

5.1 System-level features

System-level features capture architectural and operational dimensions, including support for local and remote LLMs, protocol compatibility, reliance on SDKs, licensing,

extensibility, and execution monitoring.

Table 1: System-level features of GenAI-enabled workflow tools.

Tool	**MCP**	**Local LLM**	**Remote LLM**	**SDK Dependence**	**License**	**Extensibility**	**Exec. Monitoring**
Langflow	No	Partial	OpenAI, Anthropic, etc.	LangChain	MIT	Limited (LangChain)	Limited (basic logs)
Flowise AI	Yes	Yes	OpenAI, Claude	LangChain-based	MIT	Moderate (via LangChain)	Limited (basic logs)
n8n	Yes	Yes	OpenAI, Claude, etc.	None	Fair-code	High (plugin-based)	Partial (logging)
Dive	Yes	Yes	OpenAI, Anthropic	None (built-in orchestration)	MIT	Moderate (tool toggling, model switching)	Partial (logging)
Dify	Partial	Yes	OpenAI, HuggingFace	Internal SDK	MIT	Moderate (custom LLMs, APIs)	Limited (basic logs)
Haystack UI	No	Yes	HuggingFace, OpenAI	Haystack	Apache 2.0	Moderate (via Haystack)	Partial (traces, logs)
AgenticFlow	Yes	Partial	OpenAI, Claude, etc.	Proprietary stack	Proprietary	Unspecified	Unspecified
AutoGen Studio	Yes	Yes	OpenAI, Azure	AutoGen	Apache, MIT	High (AutoGen plugins)	Partial (Azure logs)
HumAInFlow	Yes	Yes	OpenAI, extensible	None	MIT	High (SDK-agnostic)	High

Table 1 compares the main tools in this respect. From this comparison, only a few platforms lack full MCP support – Langflow and Haystack UI – while Dify provides only partial compatibility limited to HTTP. MCP is a key step toward interoperability and standardisation in agent workflows, as it allows heterogeneous tools and LLM applications to interact seamlessly within distributed environments [20]. Support for local LLM execution is common among the surveyed platforms, often via environments such as Ollama. Local execution is crucial in privacy-sensitive contexts and in scenarios where reducing reliance on commercial APIs lowers operational costs [9]. Nevertheless, many tools still prioritise remote LLM integration (e.g., OpenAI, Anthropic, HuggingFace), which remains attractive due to the broader availability of powerful models and faster setup for experimentation. Some platforms remain tightly coupled to specific SDKs or frameworks: Langflow and Flowise AI are bound to LangChain,

while Haystack UI depends on the Haystack SDK. Such dependencies can accelerate prototyping and simplify workflow construction thanks to pre-built components, but they also risk reduced adaptability and long-term flexibility. Framework evolution, breaking changes, or shifts in community support can directly constrain the platform's capabilities, making SDK independence or modular design a more future-proof approach.

Licensing is generally permissive (MIT, Apache 2.0), with the notable exception of n8n's *Fair-code*, which restricts certain commercial uses and may hinder openness in research settings. By contrast, AgenticFlow follows a proprietary model (with a free plan available), which limits transparency, self-hosting, and community-driven extensibility compared to open-source alternatives. Extensibility is a critical requirement for GenAI workflow platforms, as it determines whether developers can adapt the tool to evolving models, APIs, and organisational needs. Platforms tightly bound to specific SDKs often face limitations in this regard, since their flexibility depends on the roadmap of the underlying framework. For example, Langflow is constrained by LangChain, meaning new integrations are possible only when supported within the LangChain ecosystem, while Flowise provides slightly broader integration but still relies heavily on LangChain connectors. Tools like Dive and Dify offer moderate extensibility: Dive enables switching between supported tools and models but does not provide a general plugin framework, whereas Dify allows custom LLMs and APIs but only within the constraints of its internal SDK. By contrast, SDK-agnostic platforms such as n8n (plugin-based) and HumAInFlow allow broader integration possibilities, enabling the addition of new LLM providers, plugins, or monitoring capabilities without structural changes.

Execution monitoring refers to the ability to observe and trace workflow execution at runtime (e.g., highlighting active nodes, recording start and end times, logging success or failure, and capturing traces for debugging). These features are essential for ensuring reproducibility, diagnosing errors, and providing transparency in complex agentic workflows, but they remain open challenges in the current landscape, with most tools offering only partial support rather than full-fledged observability [20]. For instance, n8n and Dive provide only basic logging, allowing users to check whether nodes have run but offering little in terms of traceability or error analysis. Haystack UI extends this slightly by exposing traces and logs for debugging, but still lacks interactive monitoring features. AutoGen Studio integrates Azure logs, enabling developers to inspect execution traces at the infrastructure level, though not directly within the workflow editor. In contrast, HumAInFlow offers high-level observability: it highlights executed nodes in green, records start and end times, tags success or failure, and crucially, preserves the full execution state (inputs, configurations, and outputs). This ensures not only transparency but also reproducibility and replicability across runs.

5.2 Interaction-level features

Interaction-level features capture the collaborative dimensions of workflow tools, including accessibility of the interface, support for collaboration, explicit representation of human nodes, and simulation of human nodes through LLMs. Table 2 provides an overview.

Table 2 shows a comparison of the tools w.r.t. these features. No-code interfaces, as in Flowise AI, n8n, and Dify, play a crucial role in lowering entry barriers by allowing non-technical users to design and execute workflows through graphical editors. By contrast, low-code platforms such as Langflow and Haystack UI offer greater flexibility

Table 2: Interaction-level features of GenAI-enabled workflow tools.

Tool	**Code Level**	**Team Collaboration**	**Human Nodes**	**Human Simulation**
Langflow	Low-code	Limited	No	No
Flowise AI	No-code	Limited	Partial (pause nodes)	No
n8n	No-code	Shared workflows	Partial (manual input)	No
Dive	Low-code	Multi-user	No	No
Dify	No-code	Multi-user	No	No
Haystack UI	Low-code	Basic (API only)	No	No
AgenticFlow	Low-code	UI sync	No	No
AutoGen Studio	Low-code	GitHub integration	No	No
HumAInFlow	No-code	GitHub, shared workflows	Yes	Yes (LLM-based simulation)

but still require some programming expertise, which can hinder adoption by wider stakeholder groups.

Easy and transparent collaboration is a critical challenge within the landscape of this kind of tools, as workflows increasingly become collective artefacts maintained by distributed teams [20] Some tools, such as AutoGen Studio, integrate with GitHub, enabling robust version control, co-development, and transparent workflow sharing. Others offer more limited or ad-hoc collaboration features, often relying on local sharing or basic UI synchronisation.

Modelling and simulating human–AI collaboration is a key requirement for the advancement of agent workflow systems. This includes representing human contributions as first-class elements in workflows and enabling their simulation to study socio-technical dynamics. Such an approach aligns with recent perspectives that emphasise the centrality of human-AI coevolution, where humans and AI systems continuously adapt to each other through feedback loops and joint decision-making processes [20, 14]. Critically, human-in-the-loop functionality remains limited across most tools. While platforms such as n8n and Flowise AI allow pausing workflow execution for manual approval or input, these capabilities are not modelled as first-class elements of the workflow itself, restricting their expressiveness. By contrast, HumAInFlow explicitly supports *human nodes*, enabling workflows to include tasks that require review, decision-making, or validation by human users. Additionally, HumAInFlow introduces the ability to simulate human roles via LLMs, bridging availability gaps during prototyping and allowing experimentation with socio-technical processes at design time.

5.3 Progress Beyond the State of the Art

HumAInFlow advances the landscape of GenAI-enabled workflow tools in several ways. It explicitly supports human nodes as first-class workflow components, allowing the inclusion of tasks that require human judgment or validation. These human nodes can also be simulated by LLMs, enabling prototyping and testing even in the absence of real users. Architecturally, it is implemented in pure Python, SDK-agnostic, and supports both local and remote LLMs, allowing privacy-preserving deployments. Its

MCP compatibility positions it as an interoperable and extensible platform.

In addition, by decoupling from specific orchestration frameworks and adopting permissive MIT licensing, HumAInFlow promotes maintainability, reproducibility, and openness, making it suitable for research and collaborative environments. Unlike most surveyed tools, it explicitly addresses the challenge of human-AI collaboration through simulation, marking a step beyond the current state of the art.

6 Validation Plan

The validation of *HumAInFlow* will follow an iterative, user-centred design cycle, where evaluation and development proceed hand in hand. Rather than being conceived as a one-off exercise, validation is treated as a continuous process in which successive usability studies provide concrete feedback to guide improvements to the platform. In this way, the tool will progressively evolve to meet the needs of its intended users and to remain aligned with its overall objectives.

A central focus of validation is usability: the platform is designed to lower the barriers to modelling, simulating, and executing socio-technical workflows, including for non-technical users. The studies will therefore assess how effectively users can interact with the no-code interface, how efficiently they can complete tasks, and how easily they can learn to use the system. Particular attention will be paid to whether users feel able to represent socio-technical processes faithfully, and to whether they can rely on the platform to support their decision-making when working with both human and AI agents.

To capture these aspects in a systematic way, the UEQ+ (User Experience Questionnaire Plus) [15] will be employed. This modular instrument allows researchers to select scales that best reflect the goals of the platform. In the context of *HumAInFlow*, the chosen scales will measure key qualities such as transparency (whether users can understand how AI components behave), trustworthiness (whether they feel confident in relying on the results), efficiency (whether the system is responsive and productive), dependability (whether users feel in control of workflow execution), and perspicuity (whether the platform is easy to learn and understand). These quantitative measures will be complemented by qualitative insights gathered through scenario-based walkthroughs and think-aloud protocols, which will allow participants to articulate their reasoning and difficulties while using the system.

By combining structured questionnaires with direct observation and feedback, the validation will provide both breadth and depth of evidence on the platform's performance. This approach ensures that *HumAInFlow* is not only technically robust, but also usable, transparent, and supportive of human–AI collaboration in realistic socio-technical contexts. Ultimately, the validation plan is designed to ensure that the platform can achieve its purpose: to make complex workflows involving humans and GenAI agents understandable, reliable, and accessible to a broad spectrum of users.

7 Conclusion

The present report presented *HumAInFlow*, a novel platform for designing, simulating, and executing GenAI-enabled socio-technical workflows. Unlike existing solutions, the platform explicitly integrates human nodes and supports their simulation through LLMs, thereby enabling the representation of realistic human–AI collabora-

tion processes. This functionality, together with features such as local LLM support, SDK-agnostic implementation, and emerging compatibility with MCP, distinguishes *HumAInFlow* from other tools in the current landscape.

The comparative analysis with state-of-the-art workflow platforms highlighted that while several solutions provide no-code interfaces and remote LLM integration, very few combine interoperability, extensibility, and explicit human-in-the-loop modelling. *HumAInFlow* addresses this gap by offering a no-code environment that is both accessible to non-technical users and flexible enough to support advanced experimentation, including privacy-preserving local deployments.

A validation plan has also been outlined, based on iterative usability studies and the use of UEQ+ questionnaires, complemented by scenario-based walkthroughs and think-aloud protocols. This strategy ensures that the platform will evolve in close alignment with user needs and will be evaluated not only for technical robustness but also for transparency, trustworthiness, and learnability.

Taken together, these contributions position *HumAInFlow* as a step forward in the development of human-centred, extensible, and trustworthy workflow platforms. Future work will extend its capabilities with enhanced monitoring, richer collaboration features, and deeper integration with standardised protocols, consolidating its role as a reference tool for the design and analysis of socio-technical systems augmented by Generative AI.

References

[1] Angular Team. *Angular Documentation*. 2025. URL: `https://angular.dev/` (visited on 05/22/2025).

[2] FlowiseAI Contributors. *Flowise: Drag & Drop UI to Build LLM Apps with LangChain*. `https://github.com/FlowiseAI/Flowise`. Accessed: 2025-09-18. 2024.

[3] Langflow Contributors. *Langflow: A Visual Framework for Building LLM Applications*. `https://github.com/langflow-ai/langflow`. Accessed: 2025-09-18. 2025.

[4] LangGenius Contributors. *Dify: Open-source llm application development platform*. `https://github.com/langgenius/dify`. Accessed: 2025-09-18. 2025.

[5] OpenAgent Contributors. *Dive AI Agent*. `https://github.com/OpenAgentPlatform/Dive`. Accessed: 2025-09-18. 2025.

[6] Victor Dibia et al. "Autogen studio: A no-code developer tool for building and debugging multi-agent systems". In: *arXiv preprint arXiv:2408.15247* (2024).

[7] ExpediaDotCom Contributors. *Haystack-UI: Visualization dashboard for traces, trends, and service graphs*. `https://github.com/ExpediaDotCom/haystack-ui`. Accessed: 2025-09-18. 2025.

[8] Xinyi Hou et al. "Model context protocol (mcp): Landscape, security threats, and future research directions". In: *arXiv preprint arXiv:2503.23278* (2025).

[9] A Dorca Josa and Marc Bleda-Bejar. "Local LLMS: Safeguarding Data Privacy in the Age of Generative AI. A Case Study at the University of Andorra". In: *ICERI2024 Proceedings*. IATED. 2024, pp. 7879–7888.

[10] Xinyi Li et al. "A survey on LLM-based multi-agent systems: workflow, infrastructure, and challenges". In: *Vicinagearth* 1.1 (2024), p. 9.

[11] n8n-io. *n8n: Fair-code workflow automation platform with powerful ui and integrations*. `https://github.com/n8n-io/n8n`. Accessed: 2025-09-18. 2025.

[12] Ollama Team. *Ollama GitHub Repository*. GitHub. 2025. URL: `https://github.com/ollama/ollama` (visited on 05/22/2025).

[13] OpenAI Team. *OpenAI platform*. AI platform. 2025. URL: `https://openai.com/` (visited on 07/17/2025).

[14] Dino Pedreschi et al. "Human-AI coevolution". In: *Artificial Intelligence* 339 (2025), p. 104244.

[15] Martin Schrepp. "Measuring user experience with modular questionnaires". In: *2021 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*. IEEE. 2021, pp. 1–6.

[16] Spring Team. *Spring Boot Reference Documentation*. 2025. URL: `https://docs.spring.io/spring-boot/docs/current/reference/html/` (visited on 05/22/2025).

[17] AgenticFlow Team. *AgenticFlow: AI-Powered Workflow Automation*. `https://agenticflow.ai/`. Accessed: 2025-09-18. 2025.

[18] Vamsi Viswanadhapalli. "The Future of Intelligent Automation: How Low-Code/No-Code Platforms are Transforming AI Decisioning". In: *International Journal Of Engineering And Computer Science* 14.1 (2025), pp. 26803–26825.

[19] Emily Xie. "FastMCP: A Simplified Framework for Model Context Protocol". In: *Twelfth MICCAI Educational Challenge*.

[20] Chaojia Yu et al. "A survey on agent workflow–status and future". In: *2025 8th International Conference on Artificial Intelligence and Big Data (ICAIBD)*. IEEE. 2025, pp. 770–781.