Agentic Runtime Reconfiguration of Architectural Patterns in Federated Learning

Ivan Compagnucci[a,*], Qinghua Lu[b], Catia Trubiani[a]

[a] *Gran Sasso Science Institute, Via Michele Iacobucci 2, 67100, L'Aquila, Italy*
[b] *Data61, CSIRO, Level 5/13 Garden St, Eveleigh, Australia*

Abstract

Federated Learning (FL) allows multiple clients to collaboratively train a global model without sharing their local data, thereby preserving privacy. However, architecting FL systems involves complex architectural design choices that depend on both intrinsic system characteristics (e.g., the number of available clients) and evolving runtime conditions, such as network instability or new data arriving between training rounds. While architectural patterns can address these concerns, existing approaches typically keep them static throughout the entire FL execution. This static activation poses a significant limitation, as a pattern that provides benefits in early training rounds may later introduce performance drawbacks as operational conditions evolve. For instance, a pattern intended to improve model accuracy might eventually introduce more time and computational overhead if it is not deactivated when no longer needed. This paper introduces an agentic approach that treats architectural patterns as runtime-reconfigurable entities. We extend the standard FL workflow with a Multiple AI-Agent layer that leverages Large Language Models (LLMs) to dynamically (de)activate patterns after each training round. These decisions are driven by real-time performance metrics and evolving system characteristics. To coordinate the agents, we implement and compare three distinct coordination strategies: Voting-based, Role-based, and Debate-based, evaluating them against three baselines. Experiments are conducted by emulating clients with heterogeneous characteristics and varying system operational conditions to test the effectiveness of our approach. Results examine different datasets and show that, by adaptively (de)selecting architectural patterns across FL rounds, agentic approaches improve learning accuracy and can reduce execution time, at the cost of a moderate overhead.

Keywords: Architectural Patterns, Federated Learning, AI Agents, Large Language Models

1. Introduction

Federated Learning (FL) has emerged as a novel paradigm in distributed machine learning, enabling the orchestration of global model training without sharing raw data [67, 42, 31]. By decentralizing the learning process, FL allows multiple clients to collaboratively train a model on their local data, addressing critical privacy concerns. This way, each client shares only the trained model weights with a centralized server, which then aggregates them into a global model using information fusion algorithms (e.g. FedAvg [42]).

In the software architecture community, designing FL systems is recognized as a primary challenge, with *performance optimization* being a central concern [31, 67]. This is coherent with studies on distributed computing environments, which show that dynamic workloads and resource heterogeneity can affect execution efficiency [55, 53]. While significant research in FL has focused on refining training algorithms and global model aggregation techniques, the role of the underlying architecture has received less attention. A seminal

*Corresponding author
Email addresses: ivan.compagnucci@gssi.it (Ivan Compagnucci), qinghua.lu@data61.csiro.au (Qinghua Lu), catia.trubiani@gssi.it (Catia Trubiani)

contribution in this context is the FLRA reference architecture [39], which promotes the use of architectural patterns to encapsulate best practices for recurring problems. Building upon this reference architecture, the same authors further identified a catalog of 15 architectural patterns specifically tailored for FL systems [40]. Concrete examples of such patterns include: (*i*) the *Client Selector*, which restricts training participation to clients meeting specific criteria; (*ii*) the *Heterogeneous Data Handler*, which employs data augmentation to mitigate training data heterogeneity; and (*iii*) the *Message Compressor*, which compresses the size of client-server messages to minimize communication latency.

Traditionally, architectural patterns in FL are implemented statically at design time, meaning that the list of active patterns remains fixed throughout the entire training process [66, 40]. Our previous works [19, 18] show that these patterns exhibit both benefits and drawbacks depending on system settings (e.g., the number of clients and the efficiency of the communication network). However, since these settings can change at runtime (e.g., the communication network may become unstable), a static architectural pattern setup fails to maximize the effectiveness of the FL system. In fact, a pattern that is beneficial in the initial rounds may introduce drawbacks later. To provide a practical example, consider a *Client Selector* pattern whose selection criterion is based on computational power. In the early rounds, including all clients may be ideal to maximize data coverage and model generalization. If network conditions deteriorate in later rounds, dynamically activating this pattern to exclude low-power clients can prevent bottlenecks and significantly reduce the total time required to complete the FL execution.

Based on these premises, we envision architectural patterns as *runtime-reconfigurable entities* that can be adaptively (de)activated at each round of the FL execution. This implies monitoring the actual system's configuration and deciding which patterns contribute to enact the most effective design alternative at runtime. However, managing such complex decisions (i.e., which pattern (de)activate and why) poses significant challenges. The decision-making process must account for unpredictable factors, such as fluctuating network latency and varying data availability, which can change throughout the training process. Therefore, this requires a continuous, round-by-round evaluation that correlates the actual system settings with performance metrics gathered from previous rounds. This need aligns with a growing trend in the software architecture community, where Large Language Models (LLMs) are increasingly used to support complex architectural decisions [50, 33, 44, 45, 22, 23, 20]. Beyond decision support, LLM-based agents can be integrated into software systems to monitor runtime behavior, learn from historical outcomes, and suggest real-time interventions. This integration is particularly valuable in the FL context, enabling systems to autonomously self-optimize and adapt to evolving operational contexts [8, 58].

This work presents an agentic approach for the dynamic runtime management of architectural patterns in FL systems. To achieve this, we first instantiate the FLRA reference architecture by integrating three specific patterns from [40], namely Client Selector, Heterogeneous Data Handler, and Message Compressor. We then extend this architecture by introducing a Multiple AI-Agent layer. At the conclusion of each FL round, this layer evaluates the current system settings and performance metrics collected so far to decide the configuration of active patterns expected to be most effective. To govern this decision-making process, we implement and compare three coordination strategies proposed in the AgentOps catalog from [38]: (*i*) Voting-based, where agents submit preferences and a coordinator takes the final decision; (*ii*) Role-based, where agents are assigned specific roles, such as specialist for a specific pattern, to reach a conclusion; and (*iii*) Debate-based, where agents provide arguments to motivate their opinions and interact to converge toward a consensus. The effectiveness of our approach is validated through an extensive experimental campaign designed to grasp the inherent complexity and heterogeneity of FL systems. Specifically, we conduct experiments in an FL environment characterized by high system and data heterogeneity, where clients exhibit diverse computational capacities, skewed data distributions, and different data-inflow regimes, all under stochastic network instability. Results show that architectural decisions, i.e., the choice of whether to activate or deactivate a pattern, can be transformed into effective runtime control mechanisms. By introducing an agentic approach to coordinate the activation of architectural patterns, the system effectively turns static design choices into active entities, enabling the architecture to autonomously adapt and optimize its efficiency in real-time.

In summary, the main contributions of this paper are: (i) we extend the Federated Learning Reference Architecture (FLRA) by introducing a Multiple AI-Agents layer that treats architectural patterns as *runtime*

configurable entities. The system dynamically (de)activates patterns by reasoning over evolving operational conditions and real-time performance metrics; (ii) we implement three distinct agentic coordination strategies to automate the selection of three actionable architectural patterns; (iii) we conduct an extensive experimental campaign that provides quantitative evidence of the Multiple AI-Agents layer effectiveness in representative FL scenarios. To ensure reproducibility, all artifacts are publicly available [17].

2. Preliminaries

2.1. Federated Learning in a Nutshell

In traditional Machine Learning (ML), training data is typically centralized on cloud servers to enable the development of a single predictive model [2]. However, rising privacy concerns and regulatory constraints have reduced the willingness to share sensitive information [31, 48], resulting in data silos where data are isolated for protection [64]. Federated Learning (FL) offers a privacy-preserving alternative by enabling collaborative model training across distributed data sources without transferring raw data [31]. Due to these advantages, FL has experienced rapid adoption in recent years, with successful applications across various domains, including healthcare, finance, and industry [67].

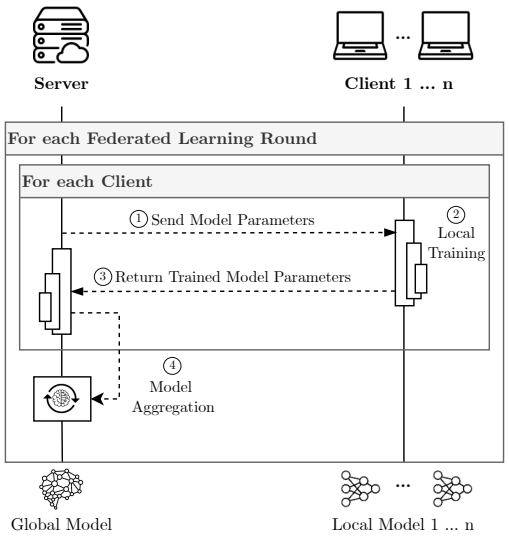


Figure 1: Federated Learning Workflow.

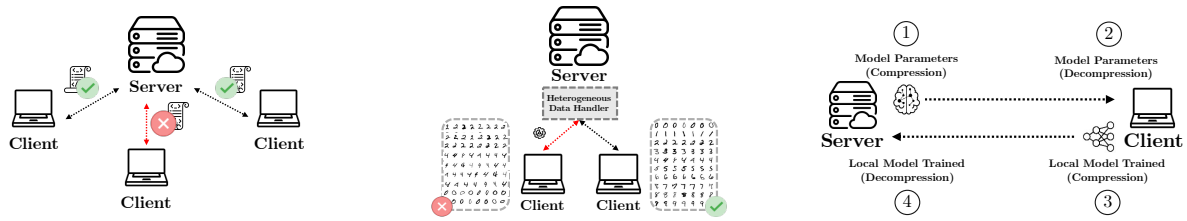
The main steps of the FL workflow are illustrated in Figure 1. The workflow starts with a central server broadcasting the initial global model parameters (e.g., model weights) to all participating clients ①. Each client then trains the model locally using its private data ② and returns only the updated model weights to the server ③. The server aggregates these updates to obtain a new global model ④, which is subsequently redistributed to the clients for the next training round. This cycle constitutes a single FL round and is repeated for a fixed number of rounds or until a predefined stopping criterion is met (e.g., convergence of the global model, stabilization of validation performance).

2.2. Architectural Patterns in Federated Learning Systems

Architectural patterns offer reusable solutions for common design challenges in complex systems [46]. Lo et al. [40] propose a set of patterns tailored to FL, addressing macro areas such as client management, model management, and model training, covering these system dimensions with architectural solutions. In our study, we consider different dimensions, focusing on clients, data, and communication, which are targeted by the following patterns: the Client Selector, the Heterogeneous Data Handler, and the Message Compressor [40], as shown in Figure 2. It is worth recalling that patterns are evaluated (and possibly selected) at each round of FL execution, thereby accounting for dynamic changes during operation.

2.2.1. Client Selector

The Client Selector is depicted in Figure 2a. It determines the subset of clients that will participate in the learning process according to predefined criteria [40]. Selection criteria can be: (*i*) *data-centric* (e.g., dataset size, heterogeneity, quality), (*ii*) *resource-centric* (e.g. computation and network capacity), and (*iii*) *performance-centric* (i.e., recent contribution to the global model) [14, 40]. Our current implementation includes the *resource-centric* selection criterion: clients are excluded if the number of physical CPU cores per client is below a predefined threshold.



(a) Client Selector. (b) Heterogeneous Data Handler. (c) Message Compressor.

Figure 2: Architectural Patterns Candidates for Selection.

Performance Implications. Client Selector can reduce both training and total round time, by excluding clients with low computational power, thereby mitigating bottlenecks during global model aggregation [40, 19, 60, 18]. On the other hand, it may slow model convergence or degrade model accuracy, because excluding clients from training prevents the model from incorporating weight updates computed from their local data.

2.2.2. Heterogeneous Data Handler

Figure 2b depicts the goal of the Heterogeneous Data Handler, which is to mitigate issues arising from non-IID client data (e.g., local datasets with different label proportions or feature distributions) that may negatively affect the global model's accuracy. In our implementation, the pattern applies *data augmentation* to generate synthetic samples and increase the diversity and size of the local dataset using Generative Adversarial Networks (GANs) [25, 65, 70]. When active, a generator produces synthetic samples while a discriminator learns to distinguish them from real ones; as training progresses, the generator improves and is used to create class-conditional samples that populate underrepresented classes. Non-IID conditions are detected through the Jensen–Shannon Divergence (JSD) metric [28] computed locally on each client and shared with the server as a single scalar $JSD_i^r = JSD(D_i^r) \in [0, 1]$ for round r, where D_i^r is the client dataset and $JSD_i^r = 0$ denotes a perfectly balanced distribution. This supports server-side decision making without exposing raw data or dataset statistics [28].

Performance Implications. This pattern improves model accuracy by mitigating the effects of non-IID data through augmentation. However, it introduces additional overhead due to the extra steps required by the GAN augmentation mechanism. In our setting, this overhead is included in the training time because, when enabled, GAN-based augmentation runs as an additional sub-phase before the client's local training [40, 19].

2.2.3. Message Compressor

The Message Compressor aims to reduce the communication time overhead by compressing the model parameters exchanged between the server and clients. Figure 2c illustrates the four phases [40]: ① the server compresses the model weights and sends them to clients; ② clients decompress the received parameters for local training; ③ after training, clients compress their updated parameters and return them to the server; ④ the server decompresses incoming updates for aggregation. Weight compression can reduce communication overhead when the client-server payload exchanged is large, e.g., models with a large number of parameters [40, 67]. Compression yields benefits only if the data exchanged is sufficiently large; otherwise, the compression and decompression overhead outweigh the benefits [18].

Performance Implications. By reducing the volume of data transmitted, this pattern can decrease Client–Server communication time and save bandwidth, which is particularly beneficial for bandwidth-constrained clients [40]. However, the extra computation required for compression and decompression may outweigh the communication-time savings, especially when the exchanged payload is small [67, 40, 18].

3. Related Work

In this section we review three complementary lines of work linked to our contribution: (i) the use of LLMs to support software-architecture tasks and architectural decision making, (ii) architectural solutions

for FL, including reference architectures and catalogs of FL architectural patterns, and (iii) dynamic and self-adaptive FL approaches where the *FL system* adapts its architecture or behavior on the fly. Building on these three macro areas, we then position our contribution with respect to prior work by clarifying the novelty of our approach.

3.1. LLMs for Software Architecture Design

Software engineers increasingly adopt LLMs for a variety of development activities, and software architects have begun to investigate their suitability for supporting architectural design decisions. LLMs are increasingly used to support architectural design decisions [44, 50, 33, 5]. Soliman et al. [50] evaluate seven LLMs on questions about the architectural knowledge of existing systems, showing that LLMs encode useful architectural information but suffer from precision and reliability issues. Dhar et al. [22] conduct an empirical study on LLMs generating architectural design decisions as architecture decision records. Preliminary analyses suggest that such LLM-generated decisions can support software architects, although further research is needed. Pace et al. [44] introduce an LLM-based assistant that guides novice architects through interactive decision-making workflows that combine architectural knowledge and quality attributes. Dhar et al. [23] study whether LLMs can generate Architecture Decision Records and later propose DRAFT, a retrieval-augmented and few-shot fine-tuned approach for more effective and efficient ADR drafting. Arun et al. [5] investigate the effectiveness of LLMs for generating architectural components by comparing LLM-produced code with human-developed implementations.

3.2. Architectural Solutions in Federated Learning

Prior work has proposed several architectural solutions for FL systems. `FLRA` [39] defines a pattern-oriented reference architecture spanning the main FL phases, from job creation to monitoring, and provides the foundation for our Multiple AI-Agents layer. Baresi et al. [6] present a requirement-driven reference architecture that integrates node selection and configuration strategies and empirically compares alternative configurations under varying conditions. Zhang et al. [67] analyze four FL system architectures (centralized, hierarchical, regional, decentralized) and compare them in terms of communication overhead, convergence speed, and scalability. Lo et al. [40] complement these contributions with a catalog of FL architectural patterns (e.g., the Client Selector adopted in this work), each providing actionable design strategies.

Many research works benchmark the performance of FL systems. FedScale [32] offers realistic datasets, a scalable runtime, and high-level APIs to support and benchmark FL experiments. Li et al. [35] survey FL optimization issues, highlighting the impact of device constraints, non-IID data, and privacy. Client selection strategies have been systematically analyzed [24, 41], addressing data heterogeneity, hardware limitations, and scheduling. Compagnucci et al. [19] empirically evaluate four FL architectural patterns and their combinations from Lo et al. [40], quantifying the trade-offs between computational overhead and resulting performance gains.

3.3. Architectural Adaptation in Federated Learning

Dynamic FL focuses on developing methods or strategies to improve the efficiency of FL executions at each round [47]. However, these methods are typically driven by runtime observations (e.g., accuracy trends, resource availability, and data heterogeneity), while keeping the underlying system architecture fixed [47]. As a result, architectural variability is not treated as a first-class adaptation mechanism, and the architecture itself is not leveraged as a runtime optimization knob. Baresi et al. [7] frame FL as a self-adaptation problem: given a target accuracy for the next round, the system estimates the number of local epochs based on clients' resource constraints and the accuracy observed in the previous two rounds. Wang et al. [61] introduce a control scheme that, under a fixed resource budget, tunes the balance between local computation and global aggregation across rounds to reduce training loss. Li et al. [34] propose a server-side approach that exploits information from past executions to predict, for each client, the workload it can realistically sustain in subsequent rounds. Zhang et al. [68] mitigate the degradation caused by non-IID data by adjusting each client's batch size at every round, tailoring local training to the observed heterogeneity.

Ilhan et al. [29] address model downsizing by adding early-exit classifiers, thereby improving training cost-effectiveness for clients with limited resources. Singh et al. [49] address the problem of data heterogeneity via an FL strategy that applies adaptive masking to unlabeled data, thereby reducing prediction entropy and increasing confidence. These contributions confirm the usefulness of self-adaptation in FL. However, they mostly target training-level levers (e.g., the number of local epochs, batch size, aggregation frequency) or the learning model itself (e.g., architectural variants or hyperparameter settings), rather than the FL solution's system architecture. In contrast, our work applies self-adaptation to architectural decisions by selecting and dynamically toggling FL architectural patterns, which provides a distinct and complementary optimization dimension for FL systems.

3.4. Research Gap and Positioning

Existing research on FL architectures provides reference models and catalogs of architectural patterns, but these patterns are typically treated as static design-time decisions [40, 39]. Once selected, they remain statically configured throughout the training process, even though system conditions (e.g., client heterogeneity, network instability, and evolving data availability) may change across rounds. At the same time, dynamic FL approaches focus primarily on adapting training-level parameters (e.g., local epochs, aggregation frequency, batch size), while leaving the architectural structure unchanged. As a result, architectural variability has not been systematically exploited as a runtime optimization mechanism.

Our work addresses this gap by elevating architectural pattern activation to a first-class runtime architectural decision. We leverage LLM-based agents not merely as advisory tools, but as decision-making components that reason over system settings and historical evaluation metrics to determine, at each round, the most suitable architectural setup. By dynamically reconfiguring the system architecture itself, rather than only tuning training parameters, we introduce a complementary adaptation dimension for FL systems. This positions our contribution at the intersection of FL architectural design and agent-based runtime adaptation, enabling architectural decisions to be treated as systematic and context-aware optimization entities.

4. Our Approach

In this section, we present the rationale for our work and describe the architectural design of our approach, which instantiates `FLRA` [39] and augments it with a Multi AI-Agents layer. We then present three coordination strategies implemented to manage agent decisions, explaining how this layer integrates into the standard FL workflow.

4.1. Motivation

Let us consider the goal of pursuing the *performance optimization* in FL systems. A traditional static configuration of architectural patterns acts like a fixed checklist, applying the same setup each FL round, regardless of context. In contrast, an AI-based agent may act like an *intelligent system architect*, analyzing past and actual performance metrics and dynamically selecting architectural patterns for the next FL round. The operational logic of this agentic layer can be grounded in the MAPE-K (Monitor-Analyze-Plan-Execute over Knowledge) reference model for self-adaptive systems [52, 57]. From this perspective, the agent realizes a structured feedback loop that *monitors* the FL environment, *analyzes* performance and system metrics, *plans* the possible architectural alternatives, and *executes* the selection of the most appropriate pattern(s) for the subsequent round. By placing AI agents at the core of this loop, the framework leverages their reasoning capabilities as the *Knowledge* component, thus the system can autonomously manage complex trade-offs and adapt to changing operational conditions, thereby continuously monitoring the FL system.

Consider the use case reported by Dayan et al [21], where a group of researchers deployed a FL system for the COVID-19 emergency, where 20 hospitals collaboratively trained a classification model without moving patient data, using only the first emergency-department measurements available at presentation (i.e., vital signs, laboratory values, and the initial chest X-ray) to predict patients' oxygen needs at 24 and 72 hours. In such settings, early rounds may benefit from selecting a larger set of hospitals (*Client Selector*) to maximize data coverage and train the global model with a larger dataset. As training progresses,

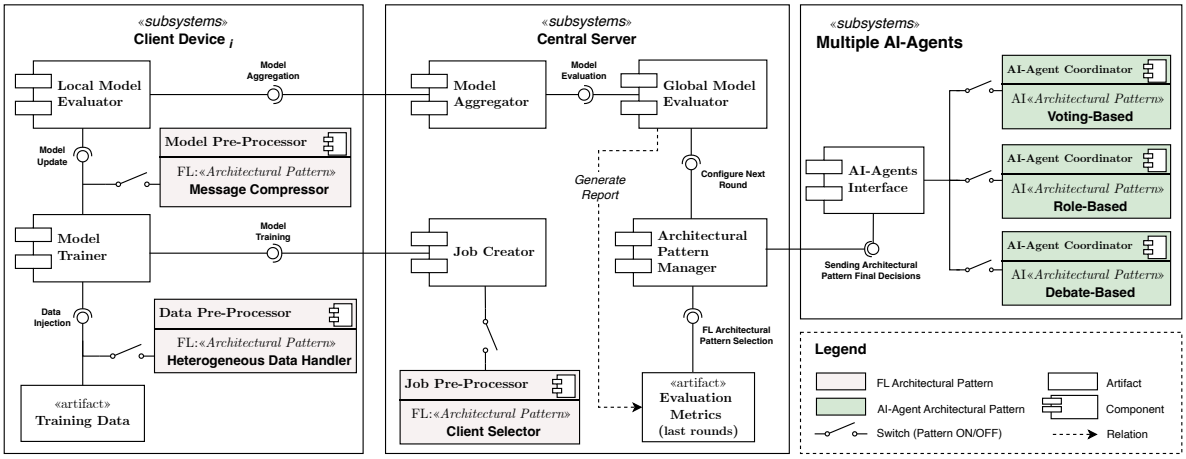


Figure 3: Component Diagram Representing the FLRA Extended with the Multiple AI-Agents Layer.

however, including every hospital in every round can make the overall process unnecessarily slow: clients with limited computational resources or poor connectivity may consistently take longer to complete local training, becoming stragglers that delay aggregation and turn each round into a bottleneck. In later rounds, the client selector can therefore prioritize clients with higher computational capacity, reducing round duration and accelerating convergence. Likewise, when the local datasets follow different data distributions across hospitals, enabling a *Heterogeneous Data Handler* can mitigate non-IID effects (e.g., skewed case severity across hospitals) and improve generalization; and when network latency becomes a bottleneck, activating a *Message Compressor* can reduce communication time by exchanging compressed updates. The key advantage is that these patterns can be (de)activated on the fly, making the FL process dynamic and context-aware, thereby improving overall system performance.

4.2. Extending the FLRA with a Multiple AI-Agents Layer

Our approach extends the Federated Learning Reference Architecture (FLRA) [39] by introducing a Multiple AI-Agents layer. The rationale for choosing this reference architecture is that, to the best of our knowledge, it is the only one that enables modular integration of architectural patterns while preserving the standard FL workflow (see Figure 1). Moreover, FLRA is grounded in a systematic literature review and qualitatively validated using evidence from both the literature and industrial implementations [39]. FLRA's explicit modularization of architectural pattern components enables runtime reconfiguration, allowing us to (de)activate and combine patterns on the fly without affecting the rest of the system. This choice also supports our optimization goals, as the architectural patterns implemented in FLRA affect system architecture (e.g., client participation, local training configuration, and communication policies), which, in turn, influence system evaluation metrics such as accuracy, training time, and communication time.

Figure 3 depicts a component diagram representing the FLRA architecture extended with the Multiple AI-Agents layer. It consists of three components: the *Central Server*, the *i-th Client Device* (i.e., multiple instances may be present; only one is illustrated for simplicity), and the *Multiple AI-Agents* layer. Note that FLRA includes the FL architectural patterns analyzed in this study (i.e., Client Selector, Message Compressor, and Heterogeneous Data Handler), which represent design strategies of the FL system itself. The *Multiple AI-Agents* layer includes three coordination strategies (i.e., Voting-based, Role-based, and Debate-based) to manage and coordinate the AI agents' decision-making processes. Each architectural pattern and coordination strategy is modeled as a configurable component that can be activated (ON) or deactivated (OFF). Importantly, while all patterns can be switched OFF, at least one coordination strategy must be enabled to ensure agentic decision-making.

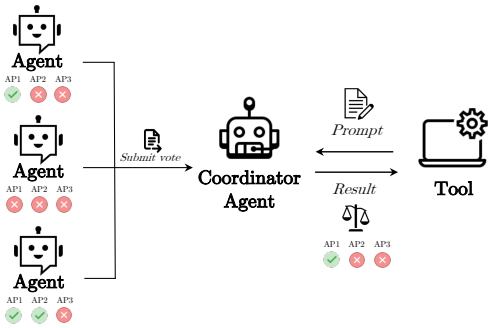


Figure 4: Voting-based Coordination Strategy.

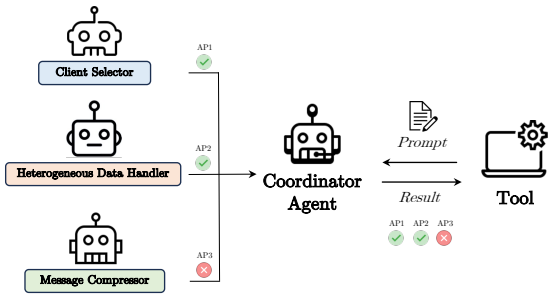


Figure 5: Role-based Coordination Strategy.

Central Server. It initializes the ML task, coordinates federated training, collects client updates, aggregates the global model, and computes core metrics. The server includes the ARCHITECTURAL PATTERN MANAGER to determine which patterns to enable, and applies this decision by toggling the pattern switches, thereby configuring the patterns enabled for the next FL round. Specifically, the component manager receives a binary array from the agentic layer, where each value indicates whether a target pattern is active.

Client Device $_i$. Each client prepares its local data and model, trains and evaluates locally, and sends its update to the server. At the beginning of every FL round, the ARCHITECTURAL PATTERN MANAGER, guided by the *Multiple AI-Agents* layer, broadcasts a binary array to each client specifying which patterns to activate or not. Patterns apply to three stages of the workflow: (i) selecting which clients join the round (*Client Selector*); (ii) augmenting client training data (*Heterogeneous Data Handler*); and (iii) compressing client-server messages (*Message Compressor*).

Multiple AI-Agents. This layer consists of a set of AI agents that leverage LLMs to analyze the current FL system configuration and the evaluation metrics collected so far, identifying the pattern combination they deem optimal for the next round. Note that both architecture pattern activation and deactivation do not require additional client-side infrastructure; clients simply execute the corresponding modules as part of the standard FL workflow. This avoids compatibility issues across heterogeneous clients and prevents additional runtime overhead beyond the execution of the selected architectural pattern modules.

The agentic recommendation process employs three coordination strategies drawn from the AgentOps catalog [38], specifically: Voting-based, Role-based, and Debate-based Cooperation. Additional details on these coordination strategies and their implementation are provided hereafter.

4.3. Multiple AI-Agents Coordination Strategies

4.3.1. Voting-based Cooperation

This strategy introduces a decision mechanism in which multiple AI agents independently form opinions and then cast votes to a centralized coordinator agent, which then elaborates on and delivers a verdict [38]. Typical voting rules include (*i*) *democratic majority* or (*ii*) *weighted voting*, where weights may reflect agent roles or reliability. As depicted in Figure 4, a central coordinator agent receives a prompt incorporating a task from a tool. It then broadcasts the request to a set of agents, who reason locally and return their votes. The coordinator evaluates the votes and, according to the chosen voting rule, sends the final decision, i.e., the set of patterns to be activated in the next FL round. This strategy preserves fairness and accountability by logging each agent's vote, allowing for the aggregation of collective intelligence to enhance decision quality.

Our Implementation. We implement the Voting-based coordination strategy by deploying multiple AI agents alongside a coordinator agent. At the end of each FL round, each agent, together with a textual explanation, can express a binary preference (ON or OFF) for each pattern, which will then be compared with the preferences expressed by other agents on the same pattern. The coordinator collects these choices and decides whether to set the pattern to ON or OFF based on the democratic majority, i.e., the pattern endorsed by receiving >50% of votes from agents is selected.

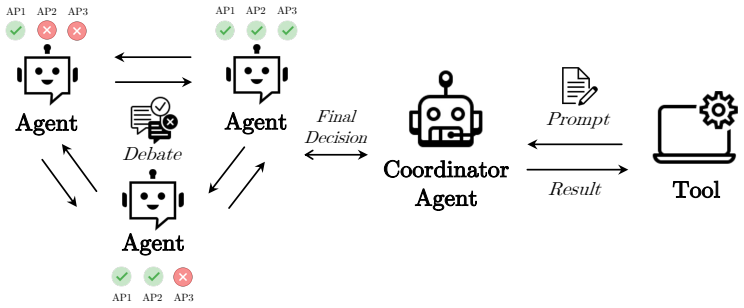


Figure 6: Debate-based Coordination Strategy.

4.3.2. Role-based Cooperation

This strategy assigns explicit roles to the agents, each with specific responsibilities [38]. Typical roles include (*i*) *coordinator* to orchestrate other agents; (*ii*) a *specialist*, which performs predefined and tailored tasks; and (*iii*) *evaluator/auditor* to assess outcomes and compliance. As depicted in Figure 5, a central coordinator agent receives a task, applies the role policy, and dispatches tasks to agents according to their roles; agents act within their corresponding role boundaries and return outputs accordingly. The coordinator then merges and organizes the results. This strategy improves accountability and safety by enforcing least privilege, separating concerns, and making each action attributable to an agent, which simplifies auditing [38].

Our Implementation. We implement the Role-based coordination strategy by defining two main agent roles: the *Coordinator* and the *Pattern Architect*. Each Pattern Architect is "specialized" in a specific architectural pattern under analysis (i.e., Client Selector, Heterogeneous Data Handler, and Message Compressor) and implemented as an AI agent. The task of each Pattern Architect is to recommend whether the corresponding pattern should be selected in the next round, and justify the decision with a rationale. The Coordinator then aggregates all recommendations and applies the resulting configuration, that is, the combination of ON/OFF pattern states for the next FL round.

4.3.3. Debate-based Cooperation

The Debate-based cooperation strategy lets agents present proposals and challenge each other's reasons under a coordinator agent [38]. It is useful when trade-offs are complex and evidence is uncertain, as arguments must be explicit, comparable, and auditable. Typical rules include time-limited turns, a single critique round, and scoring mechanisms to select the winning proposal. As depicted in Figure 6, the coordinator proposes a task along with its corresponding context. Based on this information, agents submit a proposal accompanied by a rationale. Subsequently, agents criticize and may revise their proposals. After this revision, the coordinator applies the scoring rule to select the best-supported option and outputs the decision. This strategy enhances transparency and decision quality by requiring agents to provide reasons tied to evidence, thereby reducing single-heuristic bias and making the outcome easier to audit [38].

Our Implementation. The Debate-based coordination strategy is implemented by deploying multiple AI agents and a coordinator agent. At the end of each FL round, the coordinator triggers a debate among the AI agents. Each agent first drafts its ON/OFF proposal for all patterns with a brief rationale. Then, agents exchange their positions and iteratively revise both proposal and critique in reply to other agents' proposals. A lightweight shared debate memory records arguments and refinements. The debate stops when agents converge on the same proposal or when a maximum of n turns is reached. As suggested in [38], in our experimentation, we set a maximum number of debate turns (i.e., n=5) to avoid the overhead introduced due to the high reasoning time. If no consensus emerges, the current configuration is carried over to the next round, with convergence defined as stable proposals or a strict majority agreeing on each pattern.

4.4. Integrating the Multiple AI-Agents Layer into the Federated Learning Workflow

Figure 7 depicts the extended FL workflow, introducing the "Architectural Pattern Selection" phase ⑤. After the "Model Aggregation" step ④, the server maintains a **system configuration file** reflecting the actual system configuration (e.g., number of clients, their computing capabilities, the list of currently active patterns) and an **evaluation metrics report** containing the evaluation metrics history of previous rounds (e.g., model accuracy, training time). These files are updated round-by-round to provide a detailed snapshot of the system state and process performance. Leveraging these files, the server initiates the "Architectural Pattern Selection" phase ⑤ by invoking the Multiple AI-Agents layer to determine the updated pattern configuration for the next round.

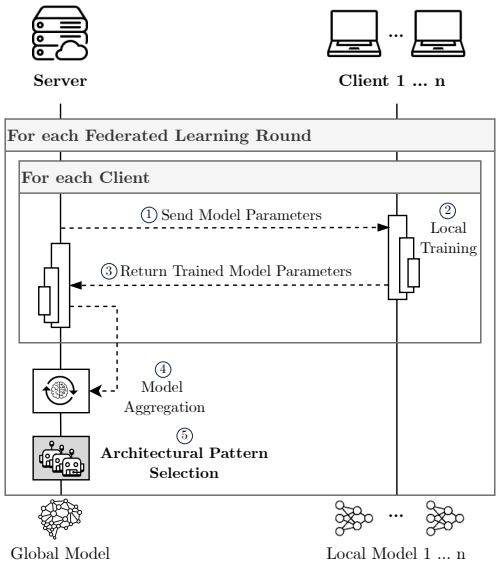


Figure 7: Federated Learning Workflow including the Multiple AI-Agents Layer.

To augment each AI agent's decision-making process, a prompt must be carefully designed. A prompt is a textual input given to an LLM that defines its role, specifies the task, provides context if needed, and instructs the desired output format [15, 37, 27, 59]. Following the guidelines in [8], we structure the prompt into four parts: (*i*) *Role* (i.e., the role that the agent must assume), (*ii*) *Task* (i.e., the task that the agent must perform), (*iii*) *Guardrails* (i.e., guidelines and rules to avoid failures), (*iv*) *Output* (i.e., the type of output format expected). To further enhance reasoning capabilities, the agents operate under a Retrieval-Augmented Generation (RAG) configuration. The RAG setup enables LLMs to integrate relevant external knowledge (e.g., database data) at inference time [4]. In our implementation, RAG provides access to two key information sources: (*i*) the system configuration file (see Table 1 for a concrete example), which specifies all the system configuration parameters governing the FL process, and (*ii*) the evaluation metrics report listing the metrics from all previous rounds. The description of the considered evaluation metrics is reported in Table 2. Both files contain key information that supports the agentic decision to select architectural patterns, ensuring it is driven by runtime contextual parameters and performance metrics.

The final decision is encoded as an updated binary array of ON/OFF flags, which overwrites the corresponding entries in the system configuration file. Once these changes are committed, the server starts a new FL round and propagates the updated configuration to all participants. Consequently, before beginning local training ②, each client applies the received settings by toggling its internal pattern-specific components according to the binary array. This ensures that the subsequent execution of steps ①–④ is performed under a dynamically optimized architectural setup, fully aligned with the agentic decision.

5. Experiments Design

We aim to address three research questions. We first identify the optimal LLM configurations and prompting strategies to ensure reliable architectural decisions (RQ1). We then evaluate how this orchestration impacts FL system evaluation metrics (RQ2). Finally, we quantify the computational overhead introduced by the Multiple AI-Agents layer to assess the practical sustainability of our approach (RQ3).

The research questions are listed as follows:

RQ1 *Which LLM configurations are most suitable for deploying the Multiple AI-Agents layer?*
We evaluate diverse LLM families to identify which model provides the most effective architectural decisions for our use case.

Table 1: System Input Parameters.

Parameter	**Value**
Model Architecture	{CNN, MLP}
Dataset	{FashionMNIST, AG_NEWS}
Data Distribution Type	{IID, non-IID ($\alpha = 0.5$)}
Client Data Inflow	{One-shot, Batched}
Network Condition	{Stable, Unstable}
Experiments Configuration Parameters	
(C1) Client 1 - High-Spec.	5 CPU; non-IID; One-shot; Stable
(C2) Client 2 - High-Spec.	5 CPU; non-IID; Batched; Stable
(C3) Client 3 - High-Spec.	5 CPU; IID; Batched; Unstable
(C4) Client 4 - Low-Spec.	3 CPU; non-IID; Batched; Unstable
(C5) Client 5 - Low-Spec.	3 CPU; IID; One-shot; Stable

Table 2: System Evaluation Metrics.

Evaluation Metric	**Description**
Model Accuracy	Global Model Accuracy (F1 Score)
Training Time	Local Model Training Time per Client
Communication Time	Client–Server Communication Time
Total Round Time	Time to Complete a FL Round
Agent Reasoning Time	Overhead Introduced by AI-Agents
Total FL Time	Time to Complete an entire FL Execution

RQ2 *How effective is the Multiple AI-Agents layer in improving FL system evaluation metrics?*
We quantify the impact of the Multiple AI-Agents layer on the FL system's evaluation metrics.

RQ3 *What is the computational overhead introduced by the Multiple AI-Agents layer?*
We measure the overhead introduced by the Multiple AI-Agents layer on the FL process.

Subject Systems. Table 1 reports the input parameters chosen for our experimentation. To evaluate our approach, we consider two different ML tasks for training the global model: image and text classification. For the image classification task, we run 10 FL rounds by training a Convolutional Neural Network (CNN) as the global model on FashionMNIST [63], which contains 70,000 28×28 images (60,000 for training and 10,000 for testing) across 10 clothing categories (e.g., T-shirt/top, trouser, dress, sneaker). For the Text Classification task, we also run 10 FL rounds and train a Multi-Layer Perceptron (MLP) on AG_NEWS, a benchmark dataset of text news articles labeled by topic (e.g., World, Sport, Business). AG_NEWS contains 120,000 training samples and 7,600 testing samples. Each sample includes a news title and a short description, which are jointly used as input for text classification. We emulate a heterogeneous federation of five clients with two hardware profiles: *High-Spec.* clients utilize 5 CPU cores, and *Low-Spec.* clients utilize 3 CPU cores. This setup exposes the "Straggler" effect [60], where lower-spec clients delay the completion of the round and higher-spec clients are forced to wait idle for them. Client data distribution mixes IID and non-IID using a Dirichlet sampler with α=0.5. Data inflow is either *one-shot* (all training data available from round 1) or *batched* (an equal fraction released each round so that by round 10 the client has its full local dataset). Network conditions are *stable* or *unstable*; in the unstable setting, we inject a random client-to-server latency of 20-50 seconds during model-parameter uploads each round. The concrete per-client setup is: C1 (5 CPU, non-IID, one-shot, stable), C2 (5 CPU, non-IID, batched, stable), C3 (5 CPU, IID, batched, unstable), C4 (3 CPU, non-IID, batched, unstable), and C5 (3 CPU, IID, one-shot, stable). Note that, rather than relying on a synthetic simulation, we use Docker Compose to emulate the FL environment;

by containerizing the server and each client with dedicated physical resources, we ensure a more realistic assessment of system evaluation metrics.

Evaluation Metrics. Table 2 describes the evaluation metrics used in our experiments, building on the indicators proposed in [30] to assess FL systems. Model Accuracy captures the predictive effectiveness of the global model and is reported as F1 score [62]. Training Time measures the average duration of local model training on the clients, while Communication Time quantifies the time spent exchanging messages between clients and the server. Total Round Time denotes the time required to complete one FL round, and Total FL Time measures the duration of the entire FL process, i.e., considering all the rounds. The Agent Reasoning Time measures the additional time required by the Multiple AI-Agents layer to reason over the current FL state and select the architectural patterns for the next round.

Evaluation Baselines. To the best of our knowledge, the current literature lacks approaches for the intelligent and dynamic adaptation of FL architectural patterns at runtime. Consequently, to rigorously evaluate the effectiveness of our solution, we define three distinct baseline strategies for comparative analysis. In the (*i*) Never configuration, all architectural patterns remain disabled throughout the entire training process, serving as a static solution to measure the performance of the FL system in the absence of any architectural pattern. In (*ii*) Random, each pattern is stochastically activated at each round with a fixed probability of 50%, providing a benchmark to determine whether our approach yields a significant advantage over a non-deterministic adaptation strategy. Finally, (*iii*) Expert-Driven strategy relies on deterministic activation conditions derived from the quantitative evidence reported in prior empirical studies on FL architectural patterns [19, 40, 18]. Instead of using fixed absolute thresholds, the activation logic is driven by observable variations in model accuracy, data distribution, and communication overhead. The activation rules at round r are defined as follows:

$$\text{Pattern Activation}^{(r)} = \begin{cases} \frac{\text{Model Accuracy}_r}{\text{Total Round Time}_r} < \frac{\text{Model Accuracy}_{r-1}}{\text{Total Round Time}_{r-1}} & \text{Client Selector} \\ JSD_r > JSD_{r-1} \ \wedge \ \text{Model Accuracy}_r < \text{Model Accuracy}_{r-1} & \text{Heterogeneous Data Handler} \\ \text{Communication Time}_r > \text{Communication Time}_{r-1} & \text{Message Compressor} \end{cases}$$

The *Client Selector* is activated when the ratio between model accuracy and total round time begins to decline; in this case, the pattern discards *Low-Spec* clients to prevent them from creating bottlenecks in the FL process. The *Heterogeneous Data Handler* is triggered when an increase in the JSD metric correlates with a decrease in model accuracy, addressing statistical heterogeneity only when it seems to affect model accuracy. Finally, the Message Compressor is activated when Communication Time increases, specifically when network bottlenecks become an issue on the total round duration.

Hardware Setup. Experiments are conducted using a workstation machine with an Intel Xeon W5-2445 chip featuring a 24 Core CPU @3.1GHz and 64 GB of RAM. Resource allocation across containers is managed via Docker Compose [9] to emulate a controlled system heterogeneity. For instance, in the *Client Selector* pattern, clients are assigned different CPU quotas to assess performance under varied computational capacities. It is worth noting that the maximum number of containers running at the same time is bounded by the capacity of the host machine (i.e., 24 cores) to avoid CPU overcommitment. This constraint is motivated by the need to avoid exceeding available processing capacity, which may lead to resource contention, affecting execution conditions and compromising the validity of the experiment results [16].

Statistical Analysis. To compare the performance of our approach against the baselines, we follow the guidelines proposed by Arcuri and Briand [3]. We repeat each experiment 10× and apply the Mann–Whitney U test to assess whether the observed differences are statistically significant. We then compute Vargha and Delaney's A_{12} statistic to quantify the effect size of the difference between samples [56]. We interpret the effect size using the standard thresholds adopted in the literature: *small* (✓) for values greater than 0.55, *medium* (✓✓) for values greater than 0.63, and *large* (✓✓✓) for values greater than 0.70. If no statistically significant difference is detected (i.e., $0.45 \leq A_{12} \leq 0.55$), we use the $\equiv$ symbol. Note that for metrics where "lower is better" (e.g., execution time), we invert the A_{12} calculation to ensure that a *Large* effect size consistently represents a significant performance improvement in favor of our approach. Note that the statistical analyses are reported for each RQ and discussed in the corresponding section.

Prompt 1: Simplified Example of the Few-Shot Prompt

Contextual Prompt

Role: "You are an expert software architect advising a Federated Learning system [...]"

Task: "Recommend which architectural patterns to activate or deactivate for the next round [...]"

Guardrails: "At least 2 clients must always participate in the FL round otherwise [...]"

Output: "Return only a Rationale text and a JSON object with the following format [...]"

Retrieval-Augmented Generation (RAG)

- From the **system configuration file**: extract the actual system parameters (e.g., dataset, CPU/RAM per client, data distr. type)
- From the **evaluation metrics report**: extract the actual system performance (e.g., model accuracy, total round time, training time).

Examples

{Example 1:"In an FL system with 4 High-Spec. and 1 Low-Spec. clients, the Low-Spec. client slows down the round, leaving others idle while waiting for synchronization [...]" → Example 1 Decision: "Enable Client Selector (CS=ON) and exclude the Low-Spec. client [...]"}

{Example 2:"CPUs [5, 5, 5, 5, 1]; CS=OFF; high Total Round Time;" → Example 2 Decision: "CS=ON with selection_value>1; 9× lower Total Round Time;")

{Example n: ...} → {Example n Decision: ...}

LLM Output

Decision:{"CS=OFF; HDH=OFF; MC=ON; Rationale=Given the system configuration and the evaluation metrics, I decided to activate the message compressor pattern only because [...]"}

5.1. RQ1: Multiple AI-Agents Layer Configuration

This research question aims to explore which configuration settings of AI agents (e.g., LLM type, prompt structure) contribute to optimizing FL systems. We use this experiment to determine which settings yield the optimal configuration. The identified configuration will serve as the LLM baseline model for the subsequent research questions.

We evaluate the agents using three open-source LLMs from the Llama [51], DeepSeek [26], and OpenAI [1] families. The choice falls on these model families because, according to the study by Soliman et al. [50], they have proven particularly effective in providing accurate and consistent answers to queries involving architectural design decisions. Specifically, the selected models belong to three families: `Llama`, `DeepSeek`, and `OpenAI`. For each family, we consider one representative version, i.e., `3.2` for `Llama`, `r1` for `DeepSeek`, and `gpt-oss` for `OpenAI`. As shown in Table 3, these models also cover different scales in terms of parameter count and storage requirements, ranging from 3B parameters and 1.3 GB for `Llama 3.2`, to 8B parameters and 5.2 GB for `DeepSeek r1`, up to 20B parameters and 14 GB for `OpenAI gpt-oss`. All models share the same 128K-token context length and are evaluated with a temperature of 1.0, following the baseline configuration used in a recent study that investigates the impact of temperature on LLM performance [54]. This supports a consistent setup for comparing different model families and sizes.

We implement and test the LLMs' reasoning ability by considering two inference strategies that reflect different levels of contextual guidance: (*i*) the *zero-shot* and (*ii*) the *few-shot* configurations. In the zero-

Table 3: LLMs Considered in this Study.

Model Family	Model Version	Model Parameters	Model Size	Context Length
`Llama`	`3.2`	3B	1.3 GB	128K
`DeepSeek`	`r1`	8B	5.2 GB	128K
`OpenAI`	`gpt-oss`	20B	14 GB	128K

Table 4: Never, Random, Expert-Driven, and LLM-based configurations benchmark, together with the statistical analysis for each LLM configuration against the baselines on Model Accuracy and Total FL Time. (ZS) and (FS) indicate Zero-Shot and Few-Shot prompting strategies adopted by the LLM, respectively.

Configurations	Evaluation Metrics			A_{12} on Model Accuracy			A_{12} on Total FL Time		
	Model Accuracy	Total FL Time	Agent Overhead	Never	Random	Expert-Driven	Never	Random	Expert-Driven
Never	0.74 ± 0.00	1666 ± 124	-	-	-	-	-	-	-
Random	0.74 ± 0.01	3796 ± 977	-	-	-	-	-	-	-
Expert-Driven	0.76 ± 0.01	1619 ± 291	-	-	-	-	-	-	-
`llama3.2` (ZS)	0.77 ± 0.01	1422 ± 66	7 ± 0	✓✓✓	✓✓✓	✓✓	✓✓✓	✓✓✓	✓✓✓
`deepseek-r1` (ZS)	0.80 ± 0.00	1870 ± 240	19 ± 4	✓✓✓	✓✓✓	✓✓✓	✗✗✗	✓✓✓	✗✗✗
`gpt-oss` (ZS)	0.79 ± 0.01	2276 ± 665	176 ± 6	✓✓✓	✓✓✓	✓✓✓	✗✗✗	✓✓✓	✗✗✗
`llama3.2` (FS)	0.82 ± 0.02	**1372 ± 151**	**5 ± 1**	✓✓✓	✓✓✓	✓✓✓	✓✓✓	✓✓✓	✓✓✓
`deepseek-r1` (FS)	**0.84 ± 0.01**	1893 ± 188	18 ± 1	✓✓✓	✓✓✓	✓✓✓	✗✗✗	✓✓✓	✗✗✗
`gpt-oss` (FS)	0.83 ± 0.01	1765 ± 627	180 ± 0	✓✓✓	✓✓✓	✓✓✓	✗✗✗	✓✓✓	✗✗✗

shot setting, the model relies exclusively on the provided context prompt [15, 43]. Conversely, the few-shot setting augments the same contextual prompt with a small set of representative examples [37]. Prompt 1 shows the structure of the few-shot configuration, whereas the zero-shot setting follows the same structure but omits the Examples part[1]. For instance, the contextual prompt includes as guardrails the requirement of having at least two clients participating in each FL round, otherwise the learning is not performed in a federated fashion. As another example, the RAG instructs verification of data from the system configuration file (e.g., CPU/RAM per client contributes to agents' decisions) and the evaluation metrics report (e.g., the current model accuracy is compared to the previous round to evaluate its variation). Examples include brief explanations of the context and the subsequent decision to be taken. For instance, in Prompt 1 we can see: *Example 1:* "4 High-Spec + 1 Low-Spec" implies the straggler effect [60], and the corresponding decision is the following: "CS=ON; exclude Low-Spec. client"). It is worth discussing that this scenario can be supported by providing another example that numerically quantifies such system behavior. For instance, Prompt 1 reports, under *Example 2*, the numerical values of CPUs: four clients have 5 CPUs, whereas one client has 1 CPU. The decision is to activate the Client Selector and exclude the client with 1 CPU, resulting in a 9× reduction in Total Round Time. The outcome of the example in Prompt 1 is that LLM decides to select the Message Compressor pattern only.

Results. Table 4 shows the results of the LLMs benchmark against baseline configurations. We assess the agent's performance based on the global model accuracy, the total time required to complete the FL process (i.e., total FL time), and the additional overhead introduced by the LLM's reasoning, in order to identify an effective trade-off between computational efficiency and predictive accuracy. Regarding the baseline results, Never and Random both yield an accuracy of 0.74, but show a large difference in execution time: while Never completes the process in 1666 seconds, Random has the highest total FL time (3796 seconds), with very high variance. The Expert-Driven baseline, as expected, shows a slight improvement, reaching an accuracy of 0.76 ± 0.01 and a shorter total FL time of 1619 seconds.

On the other hand, LLM configurations surpass the baselines in model accuracy, but they do not always achieve shorter total FL times. While agentic approaches are significantly faster than the random baseline (Random), they exhibit a higher total FL time across nearly all configurations compared to static baselines

[1]All the text prompts used in this work are available at the following link: `https://anonymous.4open.science/r/Agentic-FL-Prompts/README.md`

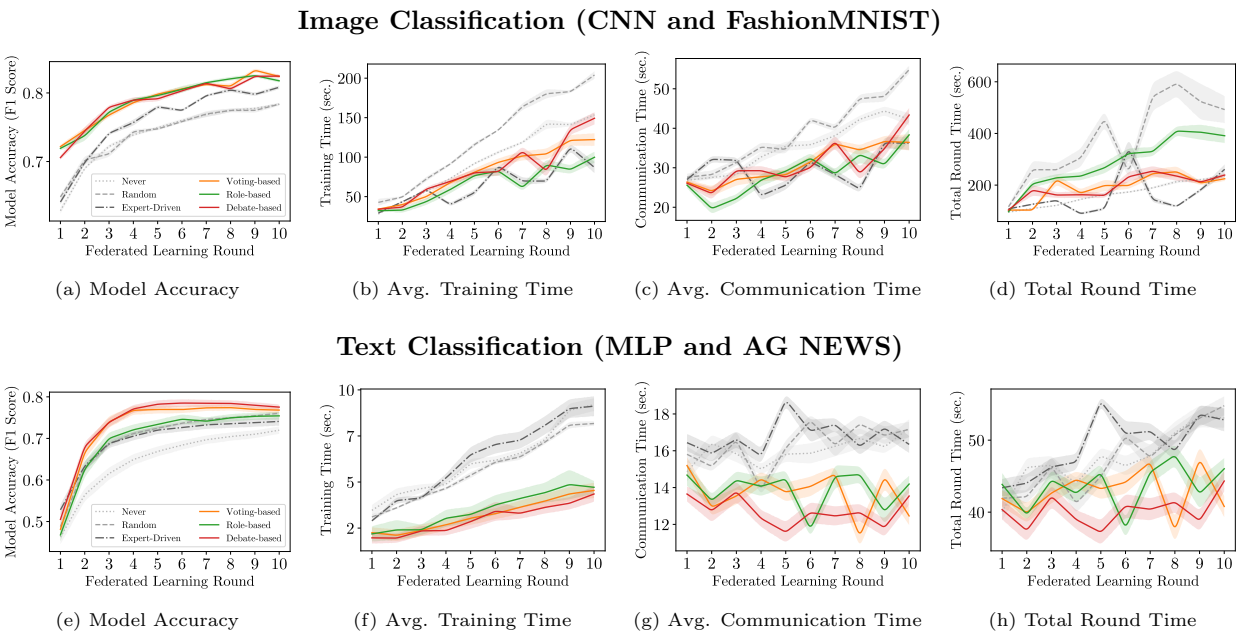


Figure 8: Performance analysis of the Multiple AI-Agents coordination strategies.

(Never and Expert-Driven), except for `llama3.2` models, which achieve a lower TOTAL FL TIME. Zero-Shot (ZS) approaches bring moderate improvements, whereas Few-Shot (FS) settings generally achieve higher accuracy; however, their impact on TOTAL FL TIME depends on the model family. Among them, `deepseek-r1` (FS) achieves the highest accuracy, reaching 0.84, with an average total FL process time of 1893 seconds and an agent reasoning overhead of 18 seconds. By contrast, `llama3.2` (FS) achieves lowest accuracy (i.e., 0.82), completing the entire FL process in an average of 1372 seconds, with only a minimal reasoning overhead (e.g., 5 seconds), representing the fastest configuration but with fewer accuracy gains. Finally, `gpt-oss` (FS) achieves 0.83 accuracy with 1765 seconds TOTAL FL TIME and 180 seconds overhead, indicating very high reasoning cost despite a good model accuracy. Statistical analyses of the collected results confirm that all LLM-based configurations significantly outperform the baselines in terms of MODEL ACCURACY, with consistently large effect sizes. For TOTAL FL TIME, only `llama3.2` (ZS) and (FS) outperforms the Never and Expert-Driven baselines, whereas the other LLM configurations improve over Random but are slower than Never and Expert-Driven.

In light of these results, we confirm the findings of [50] and select `deepseek-r1` with the Few-Shot approach as the baseline LLM configuration for the Multiple AI-Agents layer. Although `llama3.2` (FS) is faster, we prioritize the higher accuracy of `deepseek-r1` (FS), as even small gains in accuracy are relevant in FL processes [66, 31, 35].

5.2. RQ2: Multiple AI-Agents Layer Effectiveness

In this research question, we evaluate the effectiveness of the Multiple AI-Agents layer in optimizing the FL system's evaluation metrics compared to non-adaptive configurations. After fixing the LLM configuration identified in RQ1, we compare the three coordination strategies (Voting-based, Role-based, and Debate-based) against three baselines (Never, Random, and Expert-Driven). The analysis examines, for both ML tasks, the round-by-round evolution of MODEL ACCURACY, TRAINING TIME, COMMUNICATION TIME, and TOTAL ROUND TIME, thereby assessing both predictive performance and execution efficiency.

Results. The performance outcomes are depicted in Figure 8. Note that all time-related metrics show an upward trend across rounds. This trend is driven by the batched data-inflow regime adopted by some clients: as additional data becomes available at each round, the local dataset grows progressively. Consequently, clients require more time for training, leading to a natural increase in both training and total round time.

Table 5: Statistical analysis for RQ2: Agentic configurations against baselines.

(a) Image Classification (CNN and FashionMNIST).

Evaluation Metric	Configuration	Never	Random	Expert-Driven
Model Accuracy	Voting-based	✓✓✓	✓✓✓	✓✓✓
	Role-based	✓✓✓	✓✓✓	≡
	Debate-based	✓✓✓	✓✓✓	≡
Training Time	Voting-based	✓✓✓	✓✓✓	✗✗✗
	Role-based	✓✓✓	✓✓✓	≡
	Debate-based	✓✓✓	✓✓✓	✗✗✗
Communication Time	Voting-based	✓✓✓	✓✓✓	≡
	Role-based	✓✓✓	✓✓✓	≡
	Debate-based	✓✓✓	✓✓✓	≡
Total Round Time	Voting-based	✗✗✗	✓✓✓	✗✗✗
	Role-based	✗✗✗	✓✓✓	✗✗✗
	Debate-based	≡	✓✓✓	✗✗✗

(b) Text Classification (MLP and AG NEWS).

Evaluation Metric	Configuration	Never	Random	Expert-Driven
Model Accuracy	Voting-based	≡	≡	≡
	Role-based	≡	≡	≡
	Debate-based	✓✓✓	≡	≡
Training Time	Voting-based	✓✓✓	✓✓✓	✓✓✓
	Role-based	✓✓✓	✓✓✓	✓✓✓
	Debate-based	✓✓✓	✓✓✓	✓✓✓
Communication Time	Voting-based	✓✓✓	✓✓✓	✓✓✓
	Role-based	✓✓✓	✓✓✓	✓✓✓
	Debate-based	✓✓✓	✓✓✓	✓✓✓
Total Round Time	Voting-based	≡	✓✓✓	✓✓✓
	Role-based	✓✓✓	✓✓✓	✓✓✓
	Debate-based	✓✓✓	✓✓✓	✓✓✓

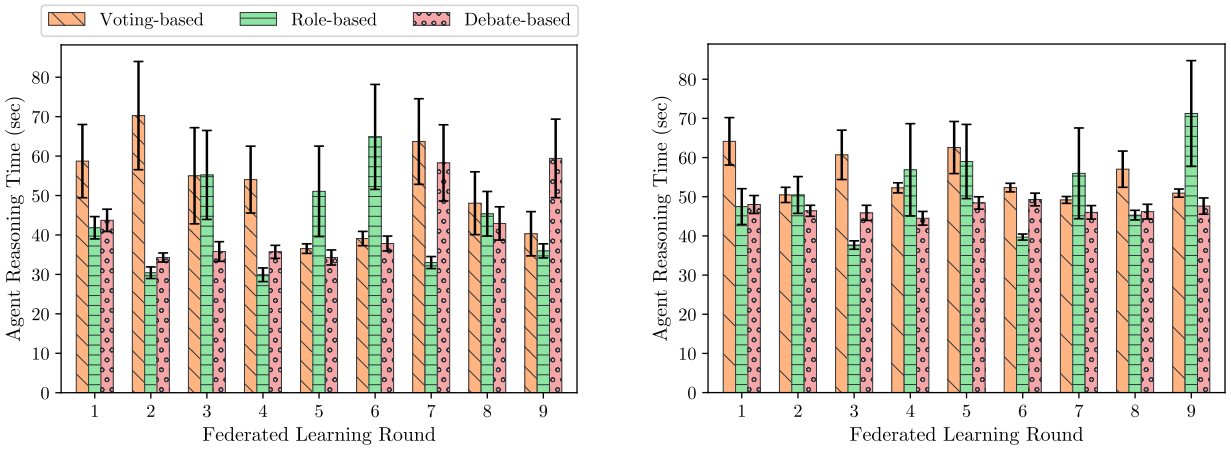
Figure 8a shows the global model accuracy for each configuration in image classification. From the early rounds, the coordination strategies reach higher accuracy and faster convergence than the baselines. The Expert-Driven configuration is the strongest baseline, achieving an F1 score of about 0.79, whereas the agentic strategies reach final values of 0.84-0.85. The three coordination strategies follow similar trends, suggesting that the Multiple AI-Agents layer consistently improves predictive performance in this task. Figure 8b reports the average client training time. The agentic configurations reduce training time with respect to Never and Random. Role-based shows the clearest reduction from round 6 onward, which is consistent with the frequent activation of the Client Selector pattern. However, Expert-Driven remains competitive, maintaining a relatively short training time. Figure 8c reports the average communication time. The agentic strategies reduce communication time compared to Random, whose trend increases across rounds. Voting-based and Role-based remain close to Expert-Driven, while Debate-based is closer to Never and achieves among the lowest communication times. Figure 8d reports the total round time. All agentic strategies improve over Random, but they are not always faster than Never and Expert-Driven. This is expected because Never avoids architectural-pattern overhead, while Expert-Driven activates patterns only under predefined conditions. Among the agentic strategies, Role-based shows the highest total round time, whereas Voting-based and Debate-based follow more similar trends.

For text classification, Figure 8e shows that the differences in model accuracy are less evident. All configurations converge to a similar range, with Debate-based showing a slight advantage. The main benefit of the Multiple AI-Agents layer appears in the time-related metrics. As shown in Figures 8f and 8g, the coordination strategies reduce both training and communication time with respect to the baselines. For total round time, Figure 8h shows that Role-based and Debate-based improve over all baselines, while Voting-based remains comparable to Never and improves over Random and Expert-Driven.

Table 5 reports the statistical analysis. For image classification, Voting-based significantly improves Model Accuracy over all baselines, while Role-based and Debate-based improve over Never and Random but are statistically equivalent to Expert-Driven. For Training Time and Communication Time, the agentic strategies generally improve over Never and Random, although Expert-Driven remains competitive. For Total Round Time, all coordination strategies improve over Random, but Expert-Driven remains stronger. For text classification, accuracy differences are limited, whereas all agentic strategies significantly improve Training Time and Communication Time. Role-based and Debate-based also improve Total Round Time over all baselines. Overall, the results show that coordinated runtime pattern activation is more effective than random activation.

5.3. RQ3: Multiple AI-Agents Layer Overhead

This research question accounts for the overhead introduced by the Multiple AI-Agents layer. We measure the time required for the agents to reason and elaborate on which pattern or combination should be activated, along with the coordinator (if present), which takes the final decision. Importantly, the Agent Reasoning Time is not included in the Total FL time metric and analyzed in RQ2. We report this overhead for



(a) Image Classification (CNN and FashionMNIST).

(b) Text Classification (MLP and AG_NEWS).

Figure 9: Multiple AI-Agents Layer: Average Agent Reasoning Time Overhead.

Table 6: Statistical analysis for the Multiple AI-Agents Layer Reasoning Time Overhead.

(a) CNN and FashionMNIST.

	Voting-based	Role-based	Debate-based
Voting-based	–	≡	✗✗✗
Role-based	≡	–	≡
Debate-based	✗✗✗	≡	–

(b) MLP and AG_NEWS.

	Voting-based	Role-based	Debate-based
Voting-based	–	≡	✗✗✗
Role-based	≡	–	≡
Debate-based	✗✗✗	≡	–

both ML tasks considered in this study: image and text classification. We also perform a statistical analysis to assess whether the observed differences in agent reasoning time among the coordination mechanisms are statistically significant.

Results. Figure 9 depicts the time overhead introduced by the Multiple AI-Agents layer for both ML tasks in rounds 1–9, since the FL process concludes at the 10th round. For each round, the overhead is computed as the average reasoning time over the 10 repetitions executed with the same configuration and coordination mechanism. Overall, across both ML tasks, the agents' reasoning time per FL round ranges from 30 to 70 seconds, showing noticeable variability across coordination strategies. On average, for the image recognition task, the Voting-based strategy requires about 51.7 seconds per round (approximately 465.7 seconds over 9 rounds), the Role-based strategy about 43.1 seconds per round (approximately 387.6 seconds in total), and the Debate-based strategy about 42.2 seconds per round (approximately 372.0 seconds over 9 rounds). For text classification, the Voting-based strategy requires about 55.5 seconds per round (approximately 499.6 seconds over 9 rounds), the Role-based strategy about 51.5 seconds per round (approximately 463.7 seconds in total), and the Debate-based strategy about 46.9 seconds per round (approximately 422.3 seconds over 9 rounds). Overall, our experimentation points out that the Debate-based strategy is slightly faster than the others, whereas the Voting-based strategy is the slowest.

Table 6 reports the pairwise statistical analysis of the agent reasoning time for both ML tasks. The results show a consistent trend across image and text classification tasks. In both cases, Voting-based and Role-based coordination do not show a statistically significant difference. Similarly, Role-based and Debate-based coordination do not show a statistically significant difference. Interestingly, Voting-based coordination is significantly worse than Debate-based coordination with a large effect size in both tasks. This result is consistent with the descriptive results reported above: Voting-based coordination shows the highest average reasoning time, whereas Debate-based coordination shows the lowest.

6. Discussion

This section discusses the main findings of our study and reflects on the overall effectiveness of our approach. We analyze how the different coordination strategies influence the FL evaluation metrics and what these results imply for adopting dynamic, agent-driven architectural decisions in practice.

6.1. Configuration Efficiency Score

We evaluate and compare the coordination strategies against the considered baselines using a weighted composite score, denoted by $\mathcal{S}$. This metric is computed as:

$$\mathcal{S} = w_1 \cdot \frac{\textsc{Model Accuracy} - A_{\min}}{A_{\max} - A_{\min}} + w_2 \cdot \frac{T_{\max} - \textsc{Total FL Time}}{T_{\max} - T_{\min}} \tag{1}$$

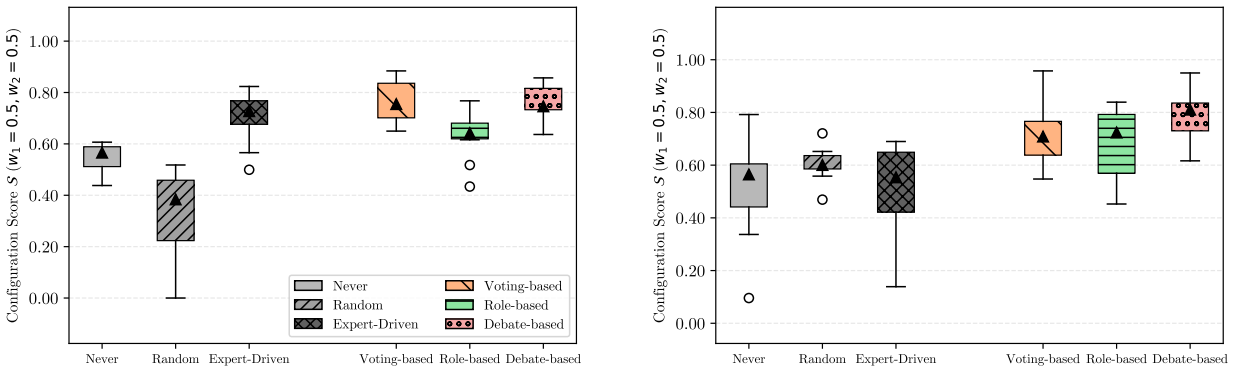
where $w_1, w_2 \in [0, 1]$ and $w_1 + w_2 = 1$. The weight w_1 determines the contribution of Model Accuracy, whereas w_2 determines the complementary contribution of Total FL Time. Both terms are normalized in the range $[0, 1]$. The second term is inverted so that lower Total FL Time yields a higher score. Therefore, larger values of $\mathcal{S}$ indicate a more favorable trade-off between predictive quality and execution time. For example, setting $w_1 = 0.7$ and $w_2 = 0.3$ means that 70% of the final score is determined by Model Accuracy, whereas the remaining 30% is determined by Total FL Time. In our evaluation, we adopt the balanced setting with $w_1 = 0.5$ and $w_2 = 0.5$.

The values assumed by this ratio are shown in Figure 10. Contrary to our initial expectation, not all coordination strategies outperform every baseline. In the image classification task (Figure 10a), Random obtains the lowest efficiency score, indicating that purely stochastic activation of architectural patterns does not yield a favorable trade-off between accuracy and execution time. In the text classification task (Figure 10b), Random remains among the weakest configurations, although the differences among strategies are less marked. In contrast, both Never and Expert-Driven achieve competitive efficiency values in some cases. In particular, Expert-Driven stands out among the baselines, confirming that rule-based activation grounded in empirical evidence can still serve as a strong reference point. At the same time, Never benefits from architectural simplicity, avoiding the additional overhead introduced by pattern management and reducing execution time. Among the agentic approaches, the Role-based strategy yields a less favorable trade-off in the image classification task, while remaining closer to the other agentic configurations in the text classification task. This behavior is mainly explained by its coordination structure: each agent acts as a specialist for a single architectural pattern and recommends its activation independently from the others. As a result, Role-based tends to activate individual patterns more often, which can increase the Total FL Time and reduce the configuration efficiency score when the additional execution cost is not compensated by proportional accuracy gains. In contrast, Voting-based and Debate-based evaluate pattern configurations more collectively, leading to more competitive median values of $\mathcal{S}$ across the two tasks. Voting-based is particularly effective in image classification (Figure 10a), whereas Debate-based achieves the highest median in text classification (Figure 10b).

Although the configuration efficiency score $\mathcal{S}$ does not imply the existence of a *silver bullet* strategy, the results show that agentic approaches generally achieve higher efficiency, except for the Role-based approach. This confirms that treating architectural pattern activation as a runtime, systematically coordinated decision process can significantly improve the balance between predictive performance and total FL time. At the same time, the magnitude of these improvements depends on the specific system settings (e.g., client heterogeneity, data distribution, and network conditions). In other words, while the agentic approach proves advantageous in our experimental scenarios, its effectiveness may vary across operational conditions and should therefore not be considered always superior in all possible deployment contexts.

6.2. Architectural Implications: Baseline Strategies

The Never configuration represents the simplest architectural setup, where no patterns are activated throughout the FL process. Its main advantage lies in its stability: by avoiding additional processing related to pattern activation, it achieves consistently low total round times. However, this simplicity comes



(a) Image Classification (CNN and FashionMNIST).

(b) Text Classification (MLP and AG_NEWS)

Figure 10: Comparative analysis of the Configuration Efficiency Score ($\mathcal{S}$): Agentic strategies versus baseline configurations.

at the cost of limited adaptability, preventing the system from reacting to runtime heterogeneity in data distribution, client resources, or network conditions.

The Random configuration highlights the risks of uncoordinated adaptation. While patterns are dynamically activated, the absence of a decision rationale leads to unstable behavior and the lowest efficiency score. This suggests that architectural variability alone is insufficient; without systematic coordination, dynamic activation may introduce overhead (e.g., by activating architectural patterns when unnecessary) without delivering proportional performance gains.

The Expert-Driven configuration provides a more elaborate alternative. By grounding activation rules in empirical evidence [19, 18], it achieves competitive efficiency scores and a balanced trade-off between accuracy and total execution time. From an architectural perspective, this confirms that rule-based logic derived from prior studies remains a solid baseline. However, fixed activation rules based only on the metrics from the previous round may limit responsiveness to broader runtime dynamics, especially when decisions would benefit from observing trends over multiple rounds or combining several contextual factors.

6.3. Architectural Implications: Agentic Strategies

The **Voting-based** strategy achieves a competitive median score $\mathcal{S}$, especially in image classification, although the Debate-based strategy achieves a higher median in text classification. It also reduces total FL time compared to the Random configuration and avoids the higher activation cost observed in the Role-based configuration. In our setting, each agent reasons about the list of architectural patterns and their impact on the FL metrics, and the voting step aggregates all proposals into a single decision. This behavior aligns with practitioners' expectations for Voting-based cooperation, where decisions made by multiple agents are more accurate and reliable than those made by a single agent [38]. This strategy leverages *Collective Intelligence*, which filters out locally optimal but globally harmful suggestions and favors pattern combinations that multiple agents independently regard as beneficial, rather than optimizing a single pattern in isolation [38].

The **Role-based** strategy achieves a lower score $\mathcal{S}$ than the Voting- and Debate-based strategies in the image classification task, while remaining closer to the other agentic configurations in the text classification task. It also exhibits higher total FL round time, often close to the Random configuration and clearly higher than the Voting and Debate-based approaches, especially in the image classification task. Accuracy, however, remains broadly comparable to the other strategies; what deteriorates primarily is the total round time, which increases due to frequent pattern activations. This is particularly evident in Figure 11, where the Role-based strategy shows the highest number of individual FL pattern activations. This behavior stems from each role agent acting as a *Pattern Architect* for a single architectural pattern and reasoning primarily about its own activation, without a full view of how combinations of patterns affect the global FL metrics. Prior work on FL performance analysis [18] has shown that combining patterns can yield both benefits and degradations, depending on how they interact. When two or more patterns are activated simultaneously,

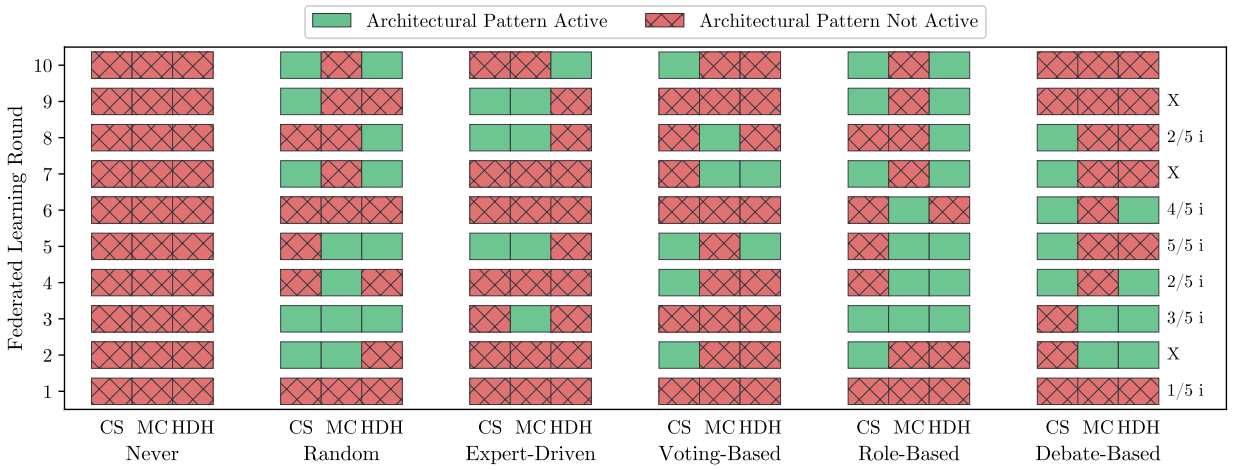


Figure 11: Architectural Patterns Activation by different strategies. In the Debate-based strategy, labels $n/5$ indicate the number of debate iterations required to reach consensus, while `X` marks rounds without consensus, in which the previous configuration of architectural pattern is reused.

they may interfere, resulting in degraded performance (e.g., longer total round time). This is why, in this context, software architects are encouraged to use a global coordination strategy that leverages collective intelligence across agents.

The **Debate-based** strategy delivers one of the highest scores $\mathcal{S}$ among the agentic coordination strategies, reaching values comparable to Voting-based in the image classification task and higher median values in the text classification task. This indicates that the explicit exchange of arguments among agents can help identify effective pattern combinations, especially when the decision involves multiple interacting architectural concerns. However, the iterative nature of the debate still introduces variability: when agents disagree for several turns, the system spends more time debating before making a decision, thereby directly increasing the total FL time. This behavior is consistent with practitioners' reports on Debate-based strategy [38], where richer debates among agents introduce extra communication and coordination time. In our setting, Debate-based is therefore a competitive coordination strategy rather than only an occasional best-case solution: it can occasionally outperform all other strategies when consensus is reached quickly; however, it can also take a long time to reach a decision when consensus is not reached quickly. Software architects should consider the Debate-based strategy for scenarios where higher decision quality is required and coordination variability is acceptable. Future work will investigate the Debate-based strategy with more capable LLMs (e.g., models with more parameters) to assess whether higher-capacity agents can shorten the time to reach consensus and, in turn, reduce variability, making this strategy a more reliable option.

6.4. Adaptation Overhead

The time overhead introduced by the Multiple AI-Agents layer is analyzed by measuring the reasoning cost required to decide which architectural patterns to activate or deactivate at runtime. It affects the FL execution in two different ways: (i) it can reduce the time required to complete the overall FL execution by avoiding unnecessary pattern activations, but (ii) it also introduces an additional cost due to LLM reasoning.

From the Total FL Time perspective, the agentic strategies are effective in reducing the time required to complete the overall FL execution by making pattern activation more selective, especially compared with the Random baseline. In the image classification task, Random requires, on average, about 379.1 seconds per round, whereas Voting-based and Debate-based coordination reduce this value to about 192.2 and 193.9 seconds, corresponding to reductions of approximately 49.3% and 48.8%, respectively. Role-based coordination also improves over Random, with an average round time of about 288.5 seconds, corresponding to a reduction of approximately 23.9%. A similar effect is observed in the text classification task: Voting-

based, Role-based, and Debate-based coordination require about 42.9, 43.6, and 40.2 seconds per round, respectively, compared to 47.3 seconds for Random and 49.3 seconds for Expert-Driven.

However, the reduction in Total FL Time comes with an additional cost. The LLM reasoning phase introduces extra execution time, since agents need to analyze the current FL state and decide the architectural configuration for the next round. Across the two ML tasks, the average reasoning overhead is about 53.6 seconds per round for Voting-based coordination, 47.3 seconds per round for Role-based coordination, and 44.6 seconds per round for Debate-based coordination. Therefore, although agentic configurations reduce the Total FL Time through more informed architectural decisions, accounting for the agentic adaptation phase increases overall execution time, adding an average cost of about 33% over the Total FL Time. However, this overhead should be interpreted together with the effectiveness gains of the Multiple AI-Agents layer. Across all experiments, agentic configurations improve average model accuracy relative to the baselines, indicating that the reasoning cost yields measurable predictive gains.

Overall, these results clarify the main trade-off introduced by the Multiple AI-Agents layer. On the one hand, the agents improve the quality of architectural decisions: they reduce unnecessary time overhead caused by architectural patterns and avoid ineffective pattern combinations. On the other hand, they introduce a new source of overhead due to LLM reasoning. Therefore, the Multiple AI-Agents layer shifts part of the execution cost from pattern execution to decision making. This cost can be justified when reasoning prevents the activation of expensive patterns that would not improve the FL process (e.g., avoiding the Heterogeneous Data Handler when synthetic data generation would increase training time without improving accuracy). In light of this, reasoning overhead should not be interpreted in isolation. The absence of large differences in reasoning time among coordination strategies suggests that this metric alone is insufficient to determine the most suitable strategy. Instead, the choice of the coordination mechanism should be considered together with the effectiveness results discussed in RQ2.

6.5. Threats to Validity

External Validity. Generalization of findings is not guaranteed since we present a specific experimental setup. Our current evaluation is limited to a restricted setup of the considered architectural patterns, the number of datasets, FL rounds, and clients. This setup reflects a methodological trade-off, since we prioritize: (i) the implementation of patterns contributing to different evaluation metrics; (ii) the datasets learning different data types; (iii) the hardware-level fidelity by assigning dedicated CPU resources to each client container and avoiding CPU overcommitment. Exceeding available processing capacity can lead to resource contention, potentially affecting execution conditions and compromising the correctness of the collected metrics [16]. While this limits the number of concurrent clients, it endorses the reliability of the performance metrics used by the agents for runtime architectural decisions. Yet, there is a combination of diverse LLMs, coordination strategies, and FL scenarios spanning multiple dimensions, including LLM families and prompting regimes (zero-shot and few-shot), datasets performing different classification tasks, heterogeneous client hardware profiles, and varying network conditions. Such variations span a broad range of variegate conditions and provide initial evidence that our approach can be applied beyond a single case. To further strengthen external validity, we plan to expand the experimentation by considering more architectural patterns, as well as further datasets and models, along with more FL rounds and clients.

Internal Validity. The settings and input parameters adopted for performance analysis can be vulnerable to potential threats, as they rely on numerical values that serve as evidence of performance variations. Yet, identifying appropriate configuration settings and parameter values remains a challenge in software performance [13], and additional alternatives could be explored in future work. To this end, we make the framework available to replicate the presented experimental results and potentially investigate additional variations of the input parameters. Besides, adopting computational power as a client selection criterion may represent a threat, since excluding low-power devices can improve overall system efficiency at the cost of unfairly excluding certain types of clients that may contain relevant training data [40], possibly adding bias to the overall model. As future work, we plan to develop different client selection strategies, e.g., by measuring the fairness of client selection during training without compromising their privacy.

Construct Validity. Threats to construct validity concern possible misinterpretation of the measured metrics. We assess FL systems using standard metrics, e.g., F1 score. We do not employ text generation

metrics such as ROUGE-1 [36] or BERTScore [69], as our setting lacks ground-truth outputs for comparison with model responses. Instead, we evaluate LLMs indirectly through FL metrics as proxies for their decision effectiveness. Moreover, a significant threat in this context is the inherent non-determinism of LLMs, which can introduce variability in the agents' decisions across different runs, potentially masking the actual impact of the coordination strategies. To mitigate this threat, we repeat each experiment 10× and assess the statistical significance of the observed variations using the Mann–Whitney U test together with Vargha and Delaney's $\hat{A}_{12}$ effect size.

7. Conclusion

This paper investigates how architectural decisions in FL systems can be elevated from static design-time choices to dynamic runtime mechanisms. We introduce an agentic framework in which multiple AI agents, coordinated through Voting-, Role-, and Debate-based strategies, dynamically (de)activate three architectural patterns: Client Selector, Heterogeneous Data Handler, and Message Compressor. Our results show that treating architectural pattern activation as a systematically coordinated runtime decision can improve the balance between predictive performance and total execution time. In particular, agentic coordination strategies achieve higher configuration efficiency scores in almost all cases in our experimental setting, showing that context-aware architectural reconfiguration improves the trade-off between predictive performance and core FL execution time, while the additional reasoning overhead remains a practical cost to be considered. These results highlight a complementary optimization dimension for FL systems: beyond tuning training parameters or global model aggregation strategies, the system architecture itself can be dynamically reconfigured based on evolving runtime conditions and historical performance metrics. While no single coordination strategy is a *silver bullet* across all scenarios, our findings provide quantitative evidence that agent-driven architectural adaptation can significantly improve the efficiency of FL systems under heterogeneous, dynamic operating conditions.

As future work, we plan to investigate additional agentic coordination strategies and experiment with new families of LLMs (e.g., Gemma 4 [10], Ministral-3 [11], and Qwen [12]) to further assess their reasoning capabilities. This is particularly relevant, as LLMs are advancing rapidly, with new models and updated versions released frequently. Moreover, we intend to integrate a broader set of FL architectural patterns in order to expand the space of runtime design alternatives.

Declaration of Generative AI use

During the preparation of this work, the authors used GPT-5.5 and Grammarly for grammar and spelling checks. After using these tools, the authors reviewed and edited the content as needed and took full responsibility for the publication's content. LLMs (i.e., `llama3.2`, `deepseek-r1`, and `gpt-oss`) are integrated into the Multiple AI-Agent layer as a core component of the proposed framework and empirically evaluated.

Acknowledgment

This work has been partially funded by the Italian Ministry of University and Research (MUR) Department of Excellence 2023 - 2027 for GSSI, and the MUR-PRIN project 20228FT78M DREAM (modular software Design to Reduce uncertainty in Ethics-based cyber-physicAl systeMs).

References

[1] Sandhini Agarwal and et al. 2025. gpt-oss-120b & gpt-oss-20b Model Card. *CoRR* abs/2508.10925 (2025).

[2] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software Engineering for Machine Learning: A Case Study. In *International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 291–300.

[3] Andrea Arcuri and Lionel Briand. 2011. A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering. In *International Conference on Software Engineering (ICSE)*. 1–10.

[4] Muhammad Arslan, Hussam Ghanem, Saba Munawar, and Christophe Cruz. 2024. A Survey on RAG with LLMs. In *Knowledge-Based and Intelligent Information & Engineering Systems*, Vol. 246. 3781–3790.
[5] Shrikara Arun, Meghana Tedla, and Karthik Vaidhyanathan. 2025. LLMs for Generation of Architectural Components: An Exploratory Empirical Study in the Serverless World. In *International Conference on Software Architecture (ICSA)*. 25–36.
[6] Luciano Baresi, Livia Lestingi, and Iyad Wehbe. 2025. Architecting Federated Learning Systems: A Requirement-Driven Approach. In *European Conference on Software Architecture (ECSA)*, Vol. 15929. Springer, 224–239.
[7] Luciano Baresi, Giovanni Quattrocchi, and Nicholas Rasi. 2021. Federated Machine Learning as a Self-Adaptive Problem. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE, 41–47.
[8] Len Bass, Qinghua Lu, Ingo Weber, and Liming Zhu. 2025. *Engineering AI Systems: Architecture and DevOps Essentials*.
[9] Docker, Inc. 2026. Docker. `https://www.docker.com/`. Accessed: 2026-05-09.
[10] Google DeepMind. 2026. Gemma 4 Model Card. `https://ai.google.dev/gemma/docs/core/model_card_4`. Accessed: 2026-05-09.
[11] Mistral AI. 2026. Ministral 3. *CoRR* abs/2601.08584 (2026).
[12] Qwen Team. 2025. Qwen3 Technical Report. *CoRR* abs/2505.09388 (2025).
[13] André B Bondi. 2015. *Foundations of Software and System Performance Engineering: Process, Performance Modeling, Requirements, Testing, Scalability, and Practice*.
[14] Christopher Briggs, Zhong Fan, and Peter Andras. 2020. Federated Learning with Hierarchical Clustering of Local Updates to Improve Training on Non-IID Data. In *International Conference on Neural Network (IJCNN)*. 1–9.
[15] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Conference on Neural Information Processing Systems, (NeurIPS)*.
[16] Maxime C. Cohen, Philipp W. Keller, Vahab S. Mirrokni, and Morteza Zadimoghaddam. 2019. Overcommitment in Cloud Services: Bin Packing with Chance Constraints. *Management Science* 65, 7 (2019), 3255–3271.
[17] Ivan Compagnucci, Qinghua Lu, and Catia Trubiani. 2026. *Open Science Artifact: Agentic Runtime Reconfiguration of Architectural Patterns in Federated Learning*. doi:`10.5281/zenodo.20006358`
[18] Ivan Compagnucci, Riccardo Pinciroli, and Catia Trubiani. 2025. Performance Analysis of Architectural Patterns for Federated Learning Systems. In *International Conference on Software Architecture (ICSA)*. 289–300.
[19] Ivan Compagnucci, Riccardo Pinciroli, and Catia Trubiani. 2026. Experimenting Architectural Patterns in Federated Learning Systems. *Journal of Systems and Software* 232 (2026), 112655.
[20] Ivan Compagnucci and Catia Trubiani. 2025. Towards AI Agents for Selecting Architectural Patterns in Federated Learning Systems. In *Proceedings of the Fourth Conference on System and Service Quality (QualITA 2025)*, Vol. 4080.
[21] Ittai Dayan et al. 2021. Federated learning for predicting clinical outcomes in patients with COVID-19. *Nature Medicine* 27, 10 (2021), 1735–1743.
[22] Rudra Dhar, Karthik Vaidhyanathan, and Vasudeva Varma. 2024. Can LLMs Generate Architectural Design Decisions? - An Exploratory Empirical Study. In *International Conference on Software Architecture (ICSA)*. 79–89.
[23] Rudra Dhar, Karthik Vaidhyanathan, and Vasudeva Varma. 2024. Leveraging Generative AI for Architecture Knowledge Management. In *International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 163–166.
[24] Lei Fu, Huanle Zhang, Ge Gao, Mi Zhang, and Xin Liu. 2023. Client selection in federated learning: Principles, challenges, and opportunities. *IEEE Internet of Things Journal* 10, 24 (2023), 21811–21819.
[25] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Conference on Neural Information Processing Systems (NeurIPS)*. 2672–2680.
[26] Daya Guo and et al. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *CoRR* abs/2501.12948 (2025).
[27] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Transaction on Software Engineering Methodology* 33, 8 (2024), 220:1–220:79.
[28] Zhiyao Hu, Dongsheng Li, Ke Yang, Ying Xu, and Baoyun Peng. 2025. Optimizing Data Distributions Based on Jensen-Shannon Divergence for Federated Learning. *Tsinghua Science and Technology* 30, 2 (2025), 670–681.
[29] Fatih Ilhan, Gong Su, and Ling Liu. 2023. Scalefl: Resource-adaptive federated learning with heterogeneous clients. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 24532–24541.
[30] Yuang Jiang, Shiqiang Wang, Victor Valls, Bong Jun Ko, Wei-Han Lee, Kin K Leung, and Leandros Tassiulas. 2022. Model Pruning Enables Efficient Federated Learning on Edge Devices. *IEEE Transactions on Neural Networks and Learning Systems* 34, 12 (2022), 10374–10386.
[31] Peter Kairouz et al. 2021. Advances and Open Problems in Federated Learning. *Foundations and Trends in Machine Learning* 14, 1-2 (2021), 1–210.
[32] Fan Lai, Yinwei Dai, Sanjay Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha Madhyastha, and Mosharaf Chowdhury. 2022. FedScale: Benchmarking Model and System Performance of Federated Learning at Scale. In *Proceedings of Machine Learning Research (PMLR)*. 11814–11827.
[33] Jialong Li, Mingyue Zhang, Nianyu Li, Danny Weyns, Zhi Jin, and Kenji Tei. 2024. Exploring the Potential of Large Language Models in Self-adaptive Systems. In *Proceedings of the International Symposium on Software Engineering for*

Adaptive and Self-Managing Systems (SEAMS). 77–83.
[34] Li Li, Moming Duan, Duo Liu, Yu Zhang, Ao Ren, Xianzhang Chen, Yujuan Tan, and Chengliang Wang. 2021. FedSAE: A novel self-adaptive federated learning framework in heterogeneous systems. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*. 1–10.
[35] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Processing Magazine* 37, 3 (2020), 50–60.
[36] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*. 74–81.
[37] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *Comput. Surveys* 55, 9 (2023), 195:1–195:35.
[38] Yue Liu, Sin Kit Lo, Qinghua Lu, Liming Zhu, Dehai Zhao, Xiwei Xu, Stefan Harrer, and Jon Whittle. 2025. Agent Design Pattern Catalogue: A Collection of Architectural Patterns for Foundation Model Based Agents. *Journal of Systems and Software* 220 (2025), 112278.
[39] Sin Kit Lo, Qinghua Lu, Hye-Young Paik, and Liming Zhu. 2021. FLRA: A reference architecture for federated learning systems. In *European Conference on Software Architecture*. Springer, 83–98.
[40] Sin Kit Lo, Qinghua Lu, Liming Zhu, Hye-Young Paik, Xiwei Xu, and Chen Wang. 2022. Architectural Patterns for the Design of Federated Learning Systems. *Journal of Systems and Software* 191 (2022), 111357.
[41] Samara Mayhoub and Tareq M. Shami. 2024. A Review of Client Selection Methods in Federated Learning. *Archives of Computational Methods in Engineering* 31, 2 (2024), 1129–1152.
[42] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *International Conference on Artificial Intelligence and Statistics, AISTATS*, Vol. 54. PMLR, 1273–1282.
[43] Marius Mosbach, Tiago Pimentel, Shauli Ravfogel, Dietrich Klakow, and Yanai Elazar. 2023. Few-shot Fine-tuning vs. In-context Learning: A Fair Comparison and Evaluation. In *Findings of the Association for Computational Linguistics: ACL*. 12284–12314.
[44] Jorge Andrés Díaz Pace, Antonela Tommasel, and Rafael Capilla. 2024. Helping Novice Architects to Make Quality Design Decisions Using an LLM-Based Assistant. In *Proceedings of European Conference on Software Architecture (ECSA)*, Vol. 14889. 324–332.
[45] Jorge Andrés Díaz Pace, Antonela Tommasel, Rafael Capilla, and Yamid E. Ramírez. 2025. Architecture Exploration and Reflection Meet LLM-based Agents. In *International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 1–5.
[46] Mark Richards. 2015. *Software Architecture Patterns*. Vol. 4.
[47] Elsa Rizk, Stefan Vlaski, and Ali H Sayed. 2020. Dynamic Federated Learning. In *Proceedings of the International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. 1–5.
[48] Pedro Miguel Sánchez Sánchez, Alberto Huertas Celdrán, Ning Xie, Gérôme Bovet, Gregorio Martínez Pérez, and Burkhard Stiller. 2024. FederatedTrust: A solution for trustworthy federated learning. *Future Gener. Comput. Syst.* 152 (2024), 83–98.
[49] Neha Singh and Mainak Adhikari. 2025. SelfFed: Self-adaptive Federated Learning with Non-IID data on Heterogeneous Edge Devices for Bias Mitigation and Enhance Training Efficiency. *Information Fusion* 118 (2025), 102932.
[50] Mohamed Soliman and Jan Keim. 2025. Do Large Language Models Contain Software Architectural Knowledge? : An Exploratory Case Study with GPT. In *International Conference on Software Architecture (ICSA)*. IEEE, 13–24.
[51] Llama Team. 2024. The Llama 3 Herd of Models. *CoRR* abs/2407.21783 (2024).
[52] Jeffrey O. Kephart and David M. Chess. 2003. The Vision of Autonomic Computing. *Computer* 36, 1 (2003), 41–50.
[53] Joost Verbraeken and Matthijs Wolting and Jonathan Katzy and Jeroen Kloppenburg and Tim Verbelen and Jan S. Rellermeyer. 2021. A Survey on Distributed Machine Learning. *ACM Comput. Surv.* 53, 2 (2021), 30:1–30:33.
[54] Lujun Li and Lama Sleem and Niccolo' Gentile and Geoffrey Nichil and Radu State. 2025. Exploring the Impact of Temperature on Large Language Models: Hot or Cold? *Procedia Computer Science* 264 (2025), 242–251.
[55] Sindhu Menon and Santosh Reddy Addula and A. Parkavi and Ch. Subbalakshmi and V. Bala Dhandayuthapani and Kiran Sree Pokkuluri and Anita Soni. 2024. Streamlining Task Planning Systems for Improved Enactment in Contemporary Computing Surroundings. *SN Comput. Sci.* 5, 8 (2024), 993.
[56] Vargha, András and Delaney, Harold D. 2000. A critique and improvement of the CL common language effect size statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132.
[57] Yuriy Brun and Giovanna Di Marzo Serugendo and Cristina Gacek and Holger Giese and Holger M. Kienle and Marin Litoiu and Hausi A. Müller and Mauro Pezzè and Mary Shaw. 2009. Engineering Self-Adaptive Systems through Feedback Loops. In *Software Engineering for Self-Adaptive Systems*. 48–70.
[58] Karthik Vaidhyanathan and Henry Muccini. 2025. Software Architecture in the Age of Agentic AI. In *European Conference on Software Workshops (ECSA)*, Vol. 15982. Springer, 41–49.
[59] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Conference on Neural Information Processing Systems, (NeurIPS)*. 5998–6008.
[60] Tung T. Vu, Duy T. Ngo, Hien Quoc Ngo, Minh N. Dao, Nguyen H. Tran, and Richard H. Middleton. 2021. Straggler Effect Mitigation for Federated Learning in Cell-Free Massive MIMO. In *IEEE International Conference on Communications (ICC)*. 1–6.
[61] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K. Leung, Christian Makaya, Ting He, and Kevin Chan. 2019.

Adaptive Federated Learning in Resource Constrained Edge Computing Systems. *IEEE J. Sel. Areas Commun.* 37, 6 (2019), 1205–1221.

[62] Ni Wayan Surya Wardhani, Masithoh Yessi Rochayani, Atiek Iriany, Agus Dwi Sulistyono, and Prayudi Lestantyo. 2019. Cross-Validation Metrics for Evaluating Classification Performance on Imbalanced Data. In *International Conference on Computer, Control, Informatics and its Applications (IC3INA)*. 14–18.

[63] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *ArXiv* abs/1708.07747 (2017).

[64] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated Machine Learning: Concept and Applications. *ACM Transactions on Intelligent Systems and Technology* 10, 2, Article 12 (2019), 19 pages.

[65] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. In *Conference on Artificial Intelligence AAAI*. 2852–2858.

[66] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. 2021. A Survey on Federated Learning. *Knowledge-Based Systems* 216 (2021), 106775.

[67] Hongyi Zhang, Jan Bosch, and Helena Holmström Olsson. 2020. Federated Learning Systems: Architecture Alternatives. In *Asia-Pacific Software Engineering Conference (APSEC)*. 385–394.

[68] Jie Zhang, Song Guo, Zhihao Qu, Deze Zeng, Yufeng Zhan, Qifeng Liu, and Rajendra Akerkar. 2021. Adaptive federated learning on non-iid data with resource constraint. *IEEE Trans. Comput.* 71, 7 (2021), 1655–1667.

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

[70] Yizhe Zhang, Zhe Gan, Kai Fan, Zhi Chen, Ricardo Henao, Dinghan Shen, and Lawrence Carin. 2017. Adversarial Feature Matching for Text Generation. In *International Conference on Machine Learning, ICML*, Vol. 70. PMLR, 4006–4015.