# KiFramework — A Federated Learning Platform with an Innovative Communication Protocol

Jakub Michalek, Vaclav Oujezsky, and Vladislav Skorpil

Original scientific article

***Abstract*—This paper presents a newly developed framework for federated learning adapted to crisis management applications on the Android platform. This framework focuses on solving client communication problems during the federated learning process. KiFramework introduces an innovative communication protocol and framework that enhances data exchange reliability and supports efficient collaboration in multi-server environments, addressing key challenges in traditional centralized server architectures. Our research provides a comprehensive solution to optimize communication and ensure seamless collaboration among multiple servers, improving applications' efficiency using federated learning.**

***Index Terms*—Android, communication protocol, federated learning, framework, machine learning.**

## I. Introduction

In recent years, there has been an unprecedented increase in the use of technologies related to artificial intelligence and machine learning across all industries, and it appears that this trend will continue to proliferate upwards in the foreseeable future [1], [2]. Traditional machine learning paradigms in the case of collaborations between multiple distributed entities may pose some threats in terms of privacy, data ownership, or data protection violations, as the movement of this data from one location to another, as well as its centralization, may pose some security risks [3]. In recent years, the use of Federated Learning [4], [5] has emerged as a promising technique to address the challenges to improve security.

Critical infrastructure applications play a vital role in safeguarding the essential services that modern society relies on. These tools enable real-time monitoring, threat detection, incident response, and compliance tracking to maintain the security and reliability of key systems. Leveraging advanced algorithms, federated learning, and machine learning, they can proactively identify risks, autonomously respond to cyberattacks or physical threats, and optimize resource distribution. Built with resilience, these applications support backup system deployment and enhance infrastructure's ability to endure and recover from unexpected events, ensuring the uninterrupted delivery of critical services to millions of people [6].

Manuscript received October 21, 2024; revised November 22, 2024. Date of publication December 10, 2024. Date of current version December 10, 2024. The associate editor prof. Maja Braović has been coordinating the review of this manuscript and approved it for publication.

Authors are with the Department of Telecommunications, Brno University of Technology, Brno, Czech Republic (e-mails: 186140@vut.cz, oujezsky@vut.cz, skorpil@vut.cz).

Digital Object Identifier (DOI): 10.24138/jcomss-2024-0091

Federated learning [7] offers a cutting-edge approach to data processing, allowing models to be trained on decentralized data sources without transferring data to a central server. This approach significantly enhances privacy by ensuring data remains on the devices where it is generated. However, the use of federated learning in crisis management on Android platforms has yet to be explored.

This paper introduces KiFramework, an innovative framework for implementing federated learning techniques on Android devices. It addresses key challenges such as limited computational power, data heterogeneity, and stringent security requirements for mobile platforms.

KiFramework addresses key challenges of centralized architectures, such as mitigating server overload from excessive client requests and reducing the risk of Single Point of Failure (SPOF). It provides secure communication and efficient data exchange between system components through a custom communication protocol and ensures the resulting models are accurate and effective for real-world crisis scenarios.

This research advances the application of federated learning in critical infrastructure by offering a secure and practical solution specifically designed for Android platform. The framework paves the way for improved performance and security of these systems, enhancing their reliability and resilience during critical operations. This paper is an extended version of our conference paper [8] in which we focus on improving and modifying the communication protocol, and the final concept is presented here.

The main contributions of this work are:

- We introduce KiFramework, a federated learning platform specifically designed for crisis management applications on Android devices. Our framework prioritizes reliable communication between clients and servers, supporting decentralized model training in critical infrastructure systems with limited computational resources.
- We present a novel communication protocol optimized for federated learning, ensuring secure and efficient data exchange. This protocol supports multi-server deployments and addresses key challenges such as the SPOF, improving system reliability.
- We integrate Firebase Realtime Database for real-time synchronization with Apache Kafka for backup communication. This dual integration ensures uninterrupted data flow, providing real-time updates and resilience against data loss during network disruptions.
- We extend the use of the traditional Federated Averaging

(FedAvg) algorithm by implementing Weighted FedAvg, which incorporates local model losses to prioritize clients with more accurate models in the global aggregation process, enhancing overall model performance.

Section II summarizes the current knowledge regarding existing frameworks and tools used in federated learning. Following this, Section III introduces the Developed framework for federated learning on the Android platform for critical infrastructure systems, focusing mainly on the individual components of the framework and their contribution to reliable communication during the federated learning process. The principles of the developed communication protocol are presented in section IV, followed by the results of the framework's implementation and discussion in section V. Finally, the paper concludes with section VI.

## II. The State of the Art

Frameworks provide essential tools for developers and researchers to implement federated learning algorithms and conduct experiments in this rapidly evolving field. A comprehensive review of the most prominent open-source federated learning frameworks was presented in [9], outlining the foundational concepts of federated learning.

One of the most notable frameworks is TensorFlow Federated (TFF) [10], developed by Google for federated learning applications, particularly for tasks like mobile keyboard prediction on distributed mobile data. TFF offers two Application Programming Interface (API) layers and includes implementations of popular algorithms such as FedAvg [11] and Federated Stochastic Gradient Descent (FedSGD), along with the ability to create custom algorithms. TFF offers robust tools for simulation-based federated learning.

Another framework is Federated Artificial Intelligence Technology Enabler (FATE) [12], created by WeBank. FATE offers extensive customization for federated learning algorithms and supports vertical and horizontal data partitioning. A key advantage of FATE over TFF is its ability to be tested in simulated and real-world environments.

PySyft [13] is a framework that prioritizes security in federated learning, integrating privacy-preserving techniques. It supports static and dynamic computations and is compatible with popular deep-learning libraries like PyTorch [14] and TensorFlow [15]. PySyft stands out for its emphasis on secure, privacy-aware, federated learning workflows.

The Flower framework [16] is another significant player, known for its flexibility in deploying federated learning across a wide range of edge devices. Flower is designed for heterogeneous environments and directly supports federated learning algorithms on devices, making it highly suitable for large-scale, decentralized operations.

Among these frameworks, PySyft and Flower are particularly well-suited for implementing federated learning on mobile devices [17]. KiFramework introduces innovative solutions based on the strengths of the mentioned frameworks and is specifically designed to address the unique challenges of crisis management on mobile platforms. By combining the best practices of simulation-oriented frameworks like TFF, enterprise-focused solutions like FATE, and privacy-preserving techniques inspired by PySyft, KiFramework addresses the unique challenges of federated learning in mobile environments. By focusing on communication reliability, adaptive learning, and real-time deployment, KiFramework [18] delivers an optimized approach to improving critical infrastructure systems' performance, security, and resilience. A detailed description of our novel communication protocol and the full KiFramework implementation is provided in the following sections.

## III. The Developed Framework

The KiFramework is a specialized framework designed to facilitate the implementation of federated learning on Android mobile devices with multi-server deployment. The framework strongly emphasizes ensuring reliable communication between mobile clients and servers involved in the federated learning process. This reliable communication is achieved through a custom communication protocol that defines the structure, types, and sequence of messages. This protocol is primarily used for transmitting control messages during the learning process and transferring loss values and weight updates for individual machine learning models. A detailed description of the communication protocol is provided in Chapter IV.

The framework is built around three key components: the server, mobile client devices, and real-time database services. KiFramework simplifies implementation, requiring minimal configuration on both server and client sides, making it a practical choice for real-world federated learning deployments. This simplicity allows developers to focus on customizing machine learning models and aggregation techniques while the underlying infrastructure handles communication and synchronization seamlessly.

The framework also supports multiple servers deployment to train a global model during federated learning. This feature reduces the risk of model aggregation failure if one of the servers becomes unreachable, allowing clients to continue contributing to improving the global data model. The conceptual connection between the servers and clients via the Firebase

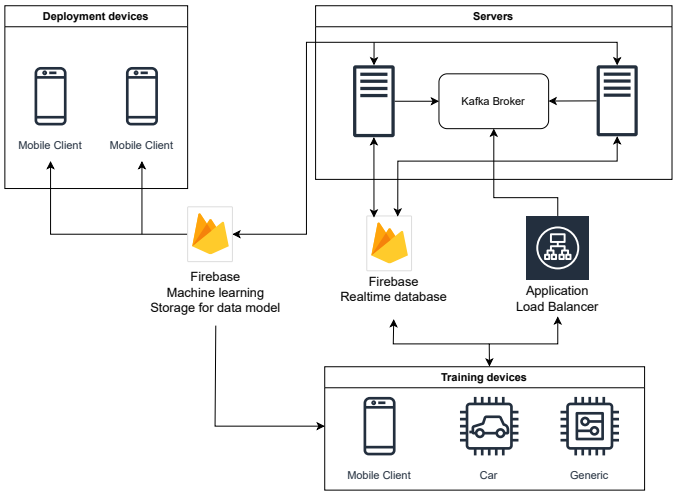


Fig. 1. The conceptual scheme of the framework.

Realtime Database [19] and Apache Kafka [20] is illustrated in Figure 1, which also shows the Firebase Datastore used for storing and distributing global models.

### A. The Server Component

The server is a critical component of the framework, responsible for aggregating local weights from clients into a global data model. The server is divided into two parts: the backend and the frontend.

The backend, written in Python, forms the core of the federated learning system. It encompasses the overall system logic, including orchestration, processing, and communication with mobile client devices enabled through Firebase services accessed by the backend. Additionally, it handles requests from the frontend via the application interface, providing real-time updates on the system's operations and status, as well as the condition of individual clients.

The backend collects local weight updates and aggregates them using the Weighted FedAvg algorithm, which enhances the traditional FedAvg [11] by incorporating local model losses into the averaging process. The key process performed by the backend for federated learning is the collection of local weight updates and their aggregation using the Weighted FedAvg algorithm. The formula for updating the global model is:

$$w^{t+1} = \sum_{k=1}^{K} \frac{\frac{1}{L_k}}{\sum_{j=1}^{K} \frac{1}{L_j}} w_k^{t+1}$$

where:

- $w^{t+1}$ are the updated weights of the global model at round $t+1$,
- $w_k^{t+1}$ are the model weights from client $k$ after local training at round $t+1$,
- $L_k$ is the loss of the local model for client $k$,
- $K$ is the total number of clients participating in the training process.

This algorithm ensures that clients with lower losses significantly influence the global model update, resulting in a more robust and accurate global model. The algorithm is implemented on the server using the TensorFlow framework [15].

The frontend is represented by a web application developed using React. This application provides a user interface for managing and monitoring server and client statuses, shown in Figure 2.

Upon successful login, users are redirected to the dashboard page, which displays comprehensive information about the server, including the current version of the global model, the number of training rounds completed, and the status of the server and client devices involved in the federated learning process. Additionally, the dashboard includes a button to initiate the servers, allowing administrators to start the servers directly from the interface.

### B. The Client Component

The client component is designed to enable decentralized model training on mobile devices. It supports client-server communication, local model training, real-time synchronization, and dynamic state management, ensuring a robust federated learning process. The system integrates Firebase for model download and server communication, TensorFlow Lite [21] for on-device training, and Kotlin Coroutines for handling asynchronous tasks.

The core functionality is driven by the three main state machines, Algorithms [1 2 3], defined for managing the

**Algorithm 1:** KeepAlive State Machine Mechanism

**State:**
- RUN(value: Int) **Action:**
  - Send keep-alive signal to server every $t$ seconds
- STOP **Action:**
  - Terminate keep-alive signals

**Algorithm 2:** Client State Machine Mechanism

**State:**
- INIT **Action:**
  - Trigger Alg. 1 + Retrieve configuration from the server; Transition to CONFIGURATION
- CONFIGURATION(data: ResponseConfiguration) **Action:**
  - Trigger Alg. 1 + Prepare client with server settings; Transition to READY
- READY **Action:**
  - Trigger Alg. 1 + Gather training data; Transition to DATA
- DATA **Action:**
  - Trigger Alg. 1 + Verify data readiness; Transition to TRAINING
- TRAINING **Action:**
  - Trigger Alg. 1 + Trigger Alg. 3; Wait for completion; Transition to FINISH
- FINISH **Action:**
  - Trigger Alg. 1 + Wait for next round; Transition to INIT or TERMINATE
- ERROR(errorMessage: String) **Action:**
  - Trigger Alg. 1 + Handle error; Retry or Transition to RESTORE
- RESTORE **Action:**
  - Trigger Alg. 1 + Restore last saved state; Retry TRAINING

**Algorithm 3:** Learning State Machine Mechanism

**State:**
- INIT **Action:**
  - Initialize training session; Transition to WAITING
- WAITING(data: LearningExchangeServerResponse) **Action:**
  - Receive global model update from server; Transition to RESTORE or TRAINING
- RESTORE(averageWeights: String) **Action:**
  - Apply global weights; Transition to TRAINING
- TRAINING **Action:**
  - Train local model on received data; Transition to SEND
- SEND **Action:**
  - Send local model weights to server; Transition to WAITING or FINISH
- FINISH **Action:**
  - End current training round; Transition to INIT or STOP, Trigger Alg. 2.

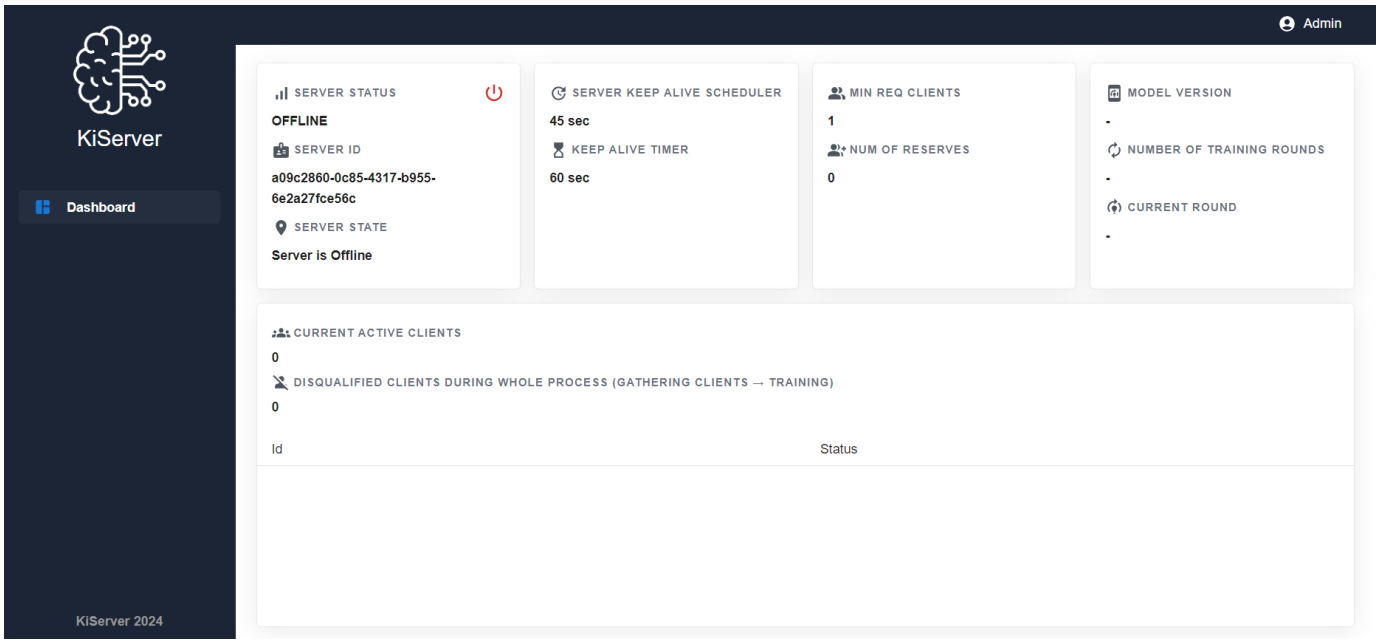


Fig. 2. Frontend view on the dashboard.

federated learning workflow and auxiliary processes like keep-alive signaling using the Hybrid Automata Library (HAL) state machine framework [22]. The training process follows the proposed state machine architecture previously published in [23].

The Algorithm 2 governs the primary lifecycle stages and transitioning between states. It is integrated into the *MainFragment* class within the user interface package. This fragment is responsible for handling User Interface (UI) events, client-server communication, and state management.

The functionality of federated learning, Algorithm 3 is encapsulated in the *KiClientTraining* class, presented in simplified diagram in Figure 3, which manages the training lifecycle on client devices. This class is responsible for downloading models, conducting local training, and returning updated model weights to the server. Initially, the system downloads the custom machine learning model using Firebase Model Downloader [24] to ensure the most up-to-date version is deployed. TensorFlow Lite, optimized for mobile platforms, runs and trains the model on the device. Local training is performed through TensorFlow Lite's Interpreter, which processes batches of data over multiple epochs, and after training, the updated model weights are exported and sent back to the server for aggregation with the global model.

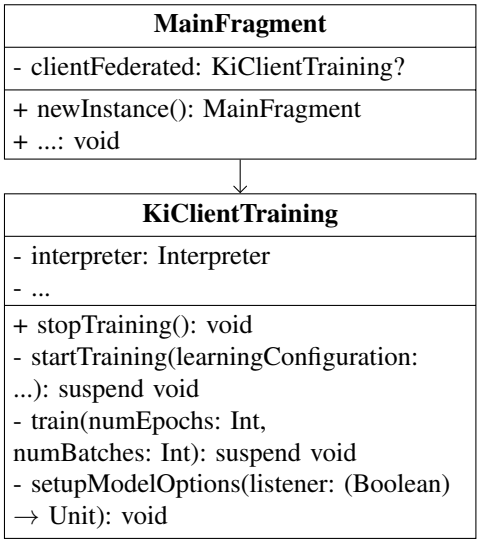


Fig. 3. Simplified UML diagram.

To ensure resilience in the case of interruptions, the system supports checkpoint management, allowing the model to resume training from the last saved state. Federated learning is conducted over multiple rounds, with each client participating in several training rounds before sending updated weights back to the server. This process is configured through the *learningConfiguration* value, which dictates the number of training rounds and other settings.

Firebase's real-time database synchronizes the client's states with the server, ensuring proper alignment during learning. The fragment listens for server commands and triggers actions such as data gathering, model training, or error handling. The user interface is dynamically updated based on the client's current state in the federated learning process, with messages displayed through a *RecyclerView*, Figure 4, which logs essential events such as the number of training rounds and server selections.

Additionally, the system includes a developed Keep-Alive mechanism, Algorithm 1, implemented by using view model architecture, which sends periodic signals to the server to maintain active communication during the learning process. This ensures that the client remains responsive during long training or data-gathering phases. The federated client is instantiated within the fragment, and it directly controls the


Fig. 4. The output from the KiClient mobile application during one training round.

lifecycle of the learning process, triggering local training sessions and managing communication with the server once the training is complete.

The federated learning system also handles model versioning and server selection, ensuring that the client is connected to the appropriate federated learning server. Server selection is based on the client's unique Identifier (ID), and the system includes safeguards against waiting indefinitely for server responses by using timeouts. If errors occur during training or communication, the system transitions into an error state, and the user is notified. The *MainFragment* also includes a reset button, allowing the user to reinitialize the training process and reset the client state. The federated learning process starts with the INIT state, where the client synchronizes local data (such as the user ID) with the server. The client then transitions to the CONFIGURATION state, where it retrieves necessary settings from the server, such as the number of training rounds and the model version. After this, the client enters the SERVER_SELECTION state, listening for server selection instructions and ensuring it is connected to the appropriate server node.

The client gathers training data on the device in the DATA state and enters the TRAINING state. During training, the client runs local training sessions using TensorFlow Lite, after which the updated model weights are exported and sent back to the federated server for aggregation. The process concludes with the FINISH state, signaling the end of the training round and preparing the client for the next cycle of federated learning, if necessary.

### C. The Firebase Component

The framework utilizes two key Firebase services: the real-time database and cloud storage. The real-time database serves as a communication channel between clients and servers, with its main advantage being the ability to synchronize data instantly across all clients. Any change made by one client is automatically propagated to others. Additionally, the database is optimized for cases of connection loss—data is temporarily stored in the client's local cache and synchronized upon reconnection, which minimizes the risk of losing local model weights and allows for the seamless continuation of training. A detailed description of message exchange, structure, and meaning is provided in Section IV, which discusses the communication protocol.

Firebase cloud storage stores and distributes global models, which are generated by aggregating local weights on the server side. Its advantage lies in its ease of access for all clients through a single authentication process and a high level of security that ensures access is restricted to authenticated users only. Global models are then integrated into Firebase Machine Learning and deployed as custom models, ensuring that mobile clients always utilize the most up-to-date version of the global model.

## IV. The Developed Communication Protocol

The communication protocol serves as a fundamental framework component, defining message formats, sequences, and interactions between clients and servers. Its design prioritizes reliability and efficient data exchange, focusing on transmitting model weights and loss metrics critical for federated learning. Additionally, it defines control messages to coordinate the training process and monitor system components. The protocol is compatible with both Firebase Realtime Database, which serves as the primary communication medium, and Apache Kafka [20], a secondary option designed to take over in case of primary channel failure. All messages are structured using JavaScript Object Notation (JSON) to support both communication platforms.

A key feature of the protocol is its seamless operation in environments with multiple server instances. This multi-server architecture mitigates the risk of overloading a single server by distributing client communications across available servers, ensuring continued operation even under high demand. Each client and server in the system is assigned a unique identifier, which facilitates correct message routing by allowing senders to target specific recipients and recipients to process only messages addressed to them. This enhanced protocol version builds on the initial design, detailed in [23], with improvements derived from extensive testing and simulations.

The protocol defines four types of messages: `configuration`, `server selection`, `status keepalive`, and `learning exchange`. The structure of these messages is illustrated in Figure 5.

The `configuration` message sets the configuration parameters for servers and clients. This message is always stored in the Firebase Realtime Database. It contains information about the current version of the global model stored in the Firebase database and the number of training rounds to be completed during federated learning.

Compared to the original version, only minor adjustments have been made. Since the model is stored on Firebase, the *ipv4AddressServer*, previously used for storing the model, is no longer needed. However, *ModelVersion* remains due to the need for synchronization, and a new field specifying the number of training rounds has been added. The newly designed version of the message is shown in Listing 1.

The `server selection` message is responsible for registering clients with a specific available server. It is divided into two sub-nodes: `clients` and `servers`. In the `clients` sub-node, clients signal to the servers that they are ready for the training process. Each client is identified by its unique ID and a timestamp, which informs the server whether the registration is still valid. The `servers` sub-node contains messages from servers responding to clients who have expressed interest in joining the learning process. Each entry is identified by a unique `commonUID`, a combination of the server and client IDs, which allows the client to identify the message addressed to it and the server with which it should communicate during training. Additionally, each entry includes a `type` field, where the server instructs the client to wait until sufficient clients are gathered for training. The newly designed structure and message types are illustrated in Listing 2.

An essential message of the protocol is `learning exchange`, which is pivotal for the exchange of local model weights and losses between clients and the server and for sending aggregated global model weights back to the clients.

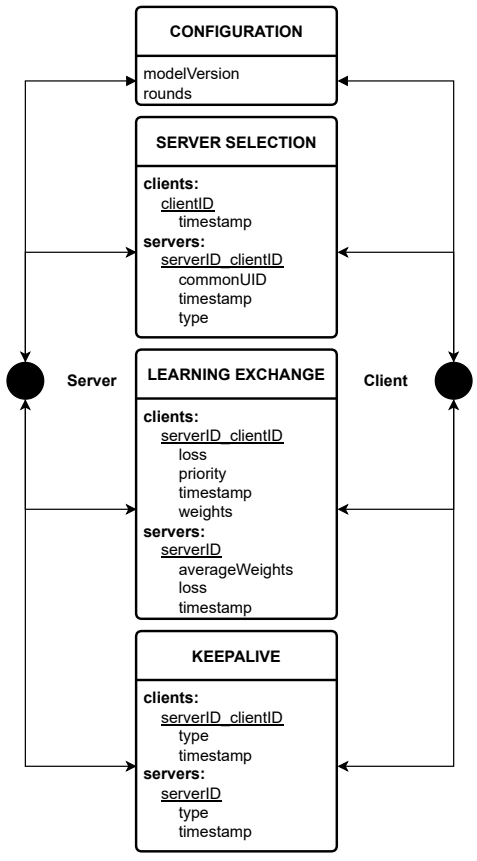


Fig. 5. Messages of the communication protocol.

```
{
  "configuration": {
    "modelVersion": ...,
    "rounds": ...
  },
  ... //other protocol nodes
}
```

Listing 1. JSON format of the Firebase Realtime Database with the *Configuration* node.

```
{
  "serverSelection": {
    "clients": {
      "<clientID-1>": {
        "timestamp": ...
      },
      "<clientID-2>": ...,
      ...
    },
    "servers": {
      "<serverID-1>_<clientID-1>": {
        "commonUID": ...,
        "timestamp": ...,
        "type": ...
      },
      "<serverID-1>_<clientID-2>": ...,
      ...
    }
  },
  ... //other protocol nodes
}
```

Listing 2. JSON format of the Firebase Realtime Database with the *Server Selection* node.

```
  "learningExchange": {
    "clients": {
      "<serverID-1>_<clientID-1>": {
        "loss": ...,
        "priority": ...,
        "timestamp": ...,
        "weights": ...,
      },
      "<serverID-1>_<clientID-2>": ...,
      ...
    },
    "servers": {
      "<serverID-1>": {
        "averageWeights": ...,
        "loss": ...,
        "timestamp": ...
      },
      "<serverID-2>": ...,
      ...
    }
  },  ... //other protocol nodes
```

Listing 3. JSON format of the Firebase Realtime Database with the *Learning Exchange* node.

This message is divided into two sub-nodes.

The `clients` node contains the local model weights and their loss, as well as a priority value, which accounts for the priority of individual clients, determined, for instance, by the amount of data each client uses to train its local model. The `servers` node includes information that notifies all clients of a given server about the training results performed on the aggregated set of client weights. Specifically, it contains the global weights clients utilize to update their local models. The newly designed structure and message types are shown in Listing 3. The final type of message is the `status keepalive (SKA)`, which is used for regularly updating the database about clients' and servers' current state, availability, and system control purposes. The newly designed structure and message types are shown in Listing 4.

The mechanism of periodic updates from clients and servers aids in system monitoring, maintaining stability, and ensuring quick response times. This message is also divided into two sub-nodes. The `clients` node contains records of individual clients that report their status and ongoing processes through the `type` parameter. These parameters' values and explanations are listed in Table I. The `servers` node contains records that inform about the status and processes of individual servers, also using the `type` parameter, as shown in Table II.

A critical factor is the scalability and efficiency of the protocol, which is tied to the communication overhead. Communication overhead refers to the total amount of data that needs to be exchanged between the clients and the server during the training rounds in a federated learning setup. This overhead impacts the overall performance and feasibility of federated learning, especially in environments with limited bandwidth or network reliability. The most straightforward formula for the communication cost, which often depends on the size of the messages, the number of rounds, and number of clients, can be approximated as follows:

$$C = 2R \cdot K \cdot \text{size}(w)$$

```
{
  "statusKeepAlive": {
    "clients": {
      "<serverID-1>_<clientID-1>": {
        "timestamp": ...,
        "type": ...,
      },
      "<serverID-1>_<clientID-2>": ...,
      ...
    },
    "servers": {
      "<serverID-1>": {
        "timestamp": ...,
        "type": ...,
      },
      "<serverID-2>": ...,
      ...
    }
  },
  ... //other protocol nodes
}
```

Listing 4. JSON format of the Firebase Realtime Database with the *Status Keep Alive* node.

TABLE I
TYPES OF CLIENT *Status Keep Alive* MESSAGES AND EXPLANATIONS OF THEIR VALUES.

| Value | Type | Explanation |
|---|---|---|
| 1 | O | *SERVER_ACKNOWLEDGE* – confirmation that the client has accepted registration by the server. |
| 2 | P | *DATA* – the client informs about its availability during data collection. |
| 3 | O | *DATA_COLLECTED* – the client reports the completion of data collection. |
| 4 | P | *TRAINING* – the client informs about its availability during training. This message is sent even when the client is waiting for the server to finish training. |
| 5 | P | *READY* – the client reports its availability and waits for instructions from the server. |

O – one-time message. P – periodic message.

TABLE II
TYPES OF SERVER *Status Keep Alive* MESSAGES AND EXPLANATIONS OF THEIR VALUES.

| Value | Type | Explanation |
|---|---|---|
| 1 | O | *DATA* – the server issues a command to start data collection. |
| 4 | O | *TRAINING* – the server instructs clients to start training. |
| 5 | P | *ALIVE* – the server reports its availability during client data collection and training. |
| 7 | O | *CANCELLED* – the server informs clients to stop the current process and search for a new server. |

O – one-time message. P – periodic message.

where $R$ is the number of communication rounds, $K$ is the number of participating clients, and size$(w)$ is the size of the messages. The 2 is the factor. Each round of federated learning involves two-way communication. The communication overhead is defined for the `learning exchange` message. In the case of the `keep alive` messages, for the keep-alive mechanism respectively, if the message is expected every $\tau$ seconds, then the total number of messages exchanged during the training period $T$ is $M = T/\tau$. Figure 6 combines the communication overhead for weight exchanges and keep-alive

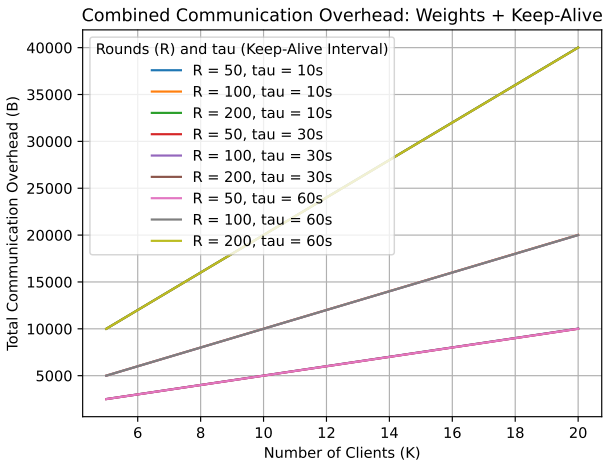


Fig. 6. Combined communication overhead of the protocol.

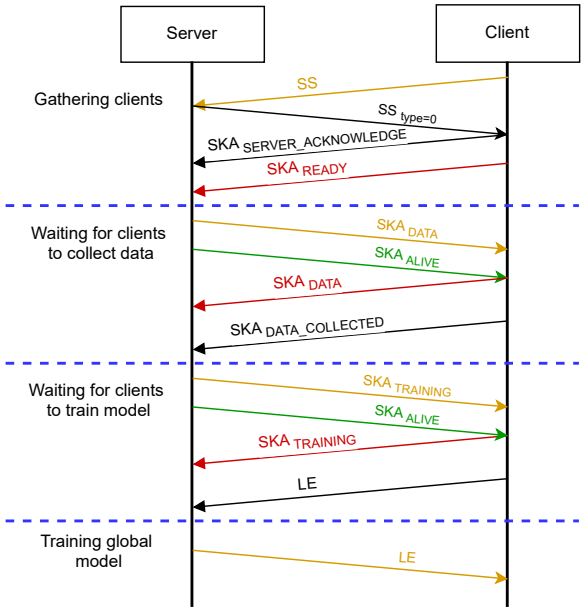


Fig. 7. Scenario of successful communication between client and server during the training process.

messages. It includes different intervals ($\tau$) for keep-alive messages and various communication rounds ($R$). The total communication overhead is displayed in $B$, considering both the size of the weight updates and the periodic keep-alive signals.

### A. Training Process

The training process for one round between the client and server is illustrated in Figure 7 and is described from the server's perspective, which oversees federated learning.

Before initiating the federated learning process, both the client and server load the initial configuration from the Firebase database, which includes the current version of the model stored in the database and the number of rounds for training the data model on the client's data. Both client and server check whether they have the current version of the model, and if not, they download it from Firebase storage. The server then transitions to the `GATHERING CLIENTS` state, which is used for selecting clients for the federated learning process. The client sends a `server selection` message with its ID to the Firebase database and waits for a response from the server. If the server has available space for the client, it responds with a `server selection` message with a value of 0. The client then confirms acceptance of the registration by sending an `SKA SERVER_ACKNOWLEDGE` message. The server selects clients until a sufficient number of them are reached to start federated learning.

After selecting clients, the server moves to the `COLLECT DATA` state. The server sends a `SKA DATA` message to all its clients, informing them to start collecting data required for training. During data collection, clients continuously send `SKA DATA` messages to the server, and once they have sufficient data for training, they inform the server with a `SKA DATA_COLLECTED` message. After receiving this message from all selected clients, the server transitions to the `TRAIN_MODEL` state.

In the `TRAIN_MODEL` state, the server sends a `SKA TRAINING` message to clients, instructing them to start training the current model on their data. During training, clients inform the server of progress with `SKA TRAINING` messages. After completing training, the client sends a `learning exchange` message to the server containing the weights of the newly trained model and its loss.

The final state of the server is `TRAIN_GLOBAL_MODEL`, which the server transitions to once it receives `learning exchange` messages from all clients. In this state, the server aggregates local model weights with consideration of their losses using the Weighted FedAvg algorithm. The following steps are determined after aggregating the weights, and whether or not it is the last round of federated learning is determined. If not, the server returns the aggregated weights to the clients for further training rounds. If it is the final round, the server creates a new global model from the aggregated weights, stores it in Firebase storage, and updates the current model version in the configuration message. This step completes the federated learning process, and the cycle starts a new one.

During communication between the server and client, periodic availability messages are also sent to inform both parties that the counterpart is active. The server sends `SKA ALIVE` messages, and the client sends `SKA READY` messages. If any error occurs during training, the server informs clients with a `SKA CANCELLED` message. Upon receiving this message, clients terminate the current activity and return to searching for a server for training.

## V. Results and Discussion

All components of the framework were tested locally during the development phase. For the final testing, the complete implementation was deployed in a real-world environment, with the server hosted within the infrastructure of the Brno University of Technology. This deployment enabled us to observe the system's behavior under real conditions, providing opportunities for further development, testing, optimization, and fine-tuning communication protocol when interacting with mobile clients. Testing in a real-world environment verifies the framework's functionality. It is a foundation for monitoring system operations, tracking communication between framework components, analyzing resource utilization, and enabling potential optimization.

The testing aimed to verify the functionality of both the server and clients from various perspectives. For the server, the reliability of communication between the backend and frontend was tested, along with the security of its components against misuse and the restriction of access to sensitive data for unauthorized users. Additionally, the correct display of information about the status of individual system components and the ability to reliably shut down and restart the server

were verified. A crucial part of the testing involved validating communication functionality using the protocol, specifically whether the server could correctly receive client messages and respond appropriately. The ability to aggregate received local weights using the FedAvg algorithm into global weights and subsequently create and upload the global model to Firebase storage was also verified.

Simultaneously with the server, the mobile application *KiClient* was tested on real mobile devices. The application was evaluated for its reliability in handling state changes during communication with the server, correct implementation of the communication protocol, and the ability to retrieve the global model from Firebase storage and improve it using local data.

Testing confirmed the process's correctness, from client collection to model training. The server reliably registered clients, responded to their messages, executed required actions, and sent appropriate responses. Combining Apache Kafka and Firebase Realtime Database creates a versatile and reliable communication channel that addresses potential data loss. KiFramework's integration with TensorFlow and Firebase ensures seamless performance within the Android ecosystem while highlighting opportunities for future cross-platform adaptability. This setup is optimized for the Android ecosystem, offering a robust solution for organizations leveraging Google technologies while paving the way for broader compatibility in future iterations.

In future revisions or expansions of the communication protocol, it is important to remember that using the Firebase Realtime Database has certain constraints. These constraints relate to the maximum depth of nesting and characters for node names and values. The database allows nesting data up to 32 levels deep. However, deeply nested structures can lead to inefficiencies, as fetching data from a parent node retrieves all child nodes. It is recommended to keep data structure as flat as possible. Other constraints related to naming conventions in the Firebase Realtime Database were encountered during implementation. Ensuring compatibility requires adherence to the prescribed naming conventions for weights and biases (keys and values).

Each key can be up to 768 bytes in size and must be UTF-8 encoded and cannot contain certain characters. Individual string values can be up to 10 MB in size. Storing excessively large strings can impact performance and increase latency. In case of the size of data downloaded from the database at a single location, it should be less than 256 MB for each read operation [25]. Following these conventions helps manage the limitations and ensures the smooth operation of the communication protocol. As a mitigation strategy, KiFramework can alleviate data structure limitations by designing flatter and more efficient JSON structures. A middleware layer could take care of data transformation and validation, ensuring compatibility while optimizing database performance. To ensure compatibility, proper database functionality, and communication, it is essential to adhere to the prescribed naming conventions, assign appropriate values, and structure the data accordingly.

Simulation results validate the communication protocol's effectiveness in controlled environments, though real-world deployments may bring additional complexities requiring ongoing evaluation and refinement. TensorFlow Lite is a powerful tool for deploying machine learning models on Android devices, offering optimized performance and a smaller footprint. However, its design prioritizes efficiency and mobile compatibility, which introduces specific considerations. These considerations are opportunities for adaptation and optimization to align with mobile constraints.

TensorFlow Lite is tailored for mobile and edge devices, which means it may not natively support all TensorFlow models and operations, particularly those that are complex or resource-intensive. It encourages the creation of models that are efficient and well-suited to the limitations of mobile hardware, ensuring smoother deployment and runtime performance.

Further, TensorFlow Lite is designed with model size restrictions to accommodate the memory and storage constraints typical of mobile devices. While this may limit the types and sizes of models that can be directly deployed, it also inspires innovation in model compression and optimization techniques. Techniques such as quantization, pruning, and knowledge distillation can be employed to significantly reduce model size while maintaining performance.

TensorFlow Lite provides a robust set of pre-defined operations, but its support for custom operations is limited [26]. When working with models that rely on specialized or less common operations, this presents an opportunity to explore alternatives, such as converting those operations into supported equivalents or implementing custom operators.

Models with dynamic inputs or architectures may require adjustments to work seamlessly with TensorFlow Lite, which generally favors static computation graphs for improved efficiency. Developers can address this by restructuring their models to use fixed input shapes where possible or leveraging TensorFlow Lite's flexibility to handle dynamic behavior with appropriate configurations. Mitigation tactics also depend on collaboration with Google developers and updates to the TensorFlow Lite used. There was a major update at the beginning of the year 2024 and there is currently a new version called LiteRT in the stable version that will need to be migrated to [27].

To decrease the communication overhead, here are several strategies that could be implemented without compromising the integrity of the federated learning process. These modifications focus on reducing the size and frequency of the data being exchanged and optimizing the communication process. Instead of sending the entire set of weights after each round, clients can send only the delta, the difference between the current weights and the weights from the previous round. This technique, called Model Delta Communication, can drastically reduce the amount of data transferred, as only the changes to the model parameters are transmitted. The other option is to use the Hierarchical Federated Learning technique. It introduces an intermediate aggregation layer between the clients and the central servers. These intermediate nodes can aggregate local updates from nearby clients, reducing the number of updates the central servers must handle.

Regarding the comparison between the current solutions and the solutions we have developed, we can compare in par-

ticular the functional characteristics. Our solutions are based on specific requirements. KiFramework improves scalability by leveraging a multi-server architecture. It also improves real-time communication by integrating Firebase for instant synchronization and Apache Kafka as a backup communication channel, making it more resilient in the real world. In addition, KiFramework is specifically designed for Android devices, allowing real deployment beyond simulation. KiFramework extends the capabilities of existing solutions like Flower and KafkaFed by incorporating real-time database integration through Firebase Realtime Database. While Flower and KafkaFed excel in their respective domains, they do not natively include real-time database capabilities. KiFramework's use of Firebase addresses this gap, enabling instant synchronization of data across multiple mobile devices. This feature is particularly valuable in scenarios requiring high responsiveness and consistency, such as emergency management or collaborative mobile applications. It also provides a two-layer communication approach, which increases reliability and resilience, especially in environments where network outages are common.

By using technologies as diverse as JSON, Firebase and Kafka, the protocol provides a strong foundation for cross-platform integration. Future enhancements such as protocol abstraction and platform-specific software development kits will further extend its usability. These include the transition of the framework to a cross-platform solution, as well as the use of the new LiteRT, which also supports cross-platform solutions to a greater extent.

## VI. Conclusion

This paper introduces KiFramework, a federated learning platform specifically designed for crisis management applications on Android devices. The framework addresses critical challenges in decentralized model training, such as limited computational power, data heterogeneity, and strict security requirements. By combining Apache Kafka and Firebase Realtime Database, KiFramework provides reliable communication, robust data exchange, and real-time synchronization among distributed components. The inclusion of TensorFlow Lite on client devices enables efficient model training. At the same time, adaptive learning algorithms adjust updates in response to infrastructure changes, ensuring models remain relevant and accurate in dynamic environments.

Despite some considerations, such as TensorFlow Lite's constraints on model size and compatibility with complex models, the framework demonstrates strong performance in real-time operations. Simulation results confirm the efficacy of the communication protocol, though real-world implementations may introduce complexities that require continuous monitoring and adaptation.

KiFramework stands out by combining Firebase and Apache Kafka, leveraging their strengths to provide seamless real-time synchronization and resilient fallback communication, ensuring uninterrupted operations. The structured communication protocol, utilizing JSON, enhances clarity and interoperability, making it suitable for Android's ecosystem and real-time decision-making scenarios.

The framework has been implemented and tested in controlled and real-world environments, confirming its readiness for deployment in critical infrastructure systems. Its ability to ensure privacy, maintain reliable communication, and deliver efficient model updates makes it a valuable solution for emergency management applications. Further development will focus on improving the web-based management of server configurations, enhancing monitoring capabilities for federated learning processes, reducing communication overhead and multi-platform migration.

to influence the work reported in this paper.

## Acknowledgement

This article is based upon the grant of the Ministry of the Interior of the Czech Republic, Open challenges in security research, VK01030152, "Android federated learning framework for emergency management applications".

## References

[1] P. Rao, "Charted: The exponential growth in ai computation," 2023. [Online]. Available: https://www.visualcapitalist.com/cp/charted-history-exponential-growth-in-ai-computation/ (Accessed 2024-03-31).

[2] "Artificial intelligence - worldwide," 2023. [Online]. Available: https://www.statista.com/outlook/tmo/artificial-intelligence/worldwide (Accessed 2024-03-31).

[3] R. Xu, N. Baracaldo, and J. Joshi, "Privacy-preserving machine learning: Methods, challenges and directions," 2021.

[4] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, B. McMahan, H., V. Overveldt, T., D. Petrou, D. Ramage, and J. Roselander, "Towards federated learning at scale: System design," 2019.

[5] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-efficient learning of deep networks from decentralized data," *CoRR*, 2016. [Online]. Available: https://arxiv.org/abs/1602.05629

[6] I. Siniosoglou, S. Bibi, K.-F. Kollias, G. Fragulis, P. Radoglou-Grammatikis, T. Lagkas, V. Argyriou, V. Vitsas, and P. Sarigiannidis, "Federated learning models in decentralized critical infrastructure," in *Shaping the Future of IoT with Edge Intelligence: How Edge Computing Enables the Next Generation of IoT Applications*. Abingdon (UK): River Publishers, Jan. 2024, pp. 95–115.

[7] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," *CoRR*, vol. abs/1602.05629, 2016. [Online]. Available: http://arxiv.org/abs/1602.05629

[8] J. Michalek, V. Oujezsky, and V. Skorpil, "An android federated learning framework for emergency management applications," in *2023 15th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, 2023. doi: 10.1109/ICUMT61075.2023.10333300 pp. 97–101.

[9] J. Michalek, V. Skorpil, and V. Oujezsky, "Federated learning on android - highlights from recent developments," in *2022 14th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*. Valencia, Spain: IEEE Computer Society, 2022. doi: 10.1109/ICUMT57764.2022.9943382. ISBN 979-8-3503-9866-3. ISSN 2157-023X pp. 27–30. [Online]. Available: https://ieeexplore.ieee.org/document/9943382

[10] Google, "Tensorflow federated: Machine learning on decentralized data," 2024. [Online]. Available: https://www.tensorflow.org/federated

[11] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Singh and J. Zhu, Eds., vol. 54. PMLR, 20–22 Apr 2017, pp. 1273–1282. [Online]. Available: https://proceedings.mlr.press/v54/mcmahan17a.html

[12] Y. Liu, T. Fan, T. Chen, Q. Xu, and Q. Yang, "Fate: An industrial grade platform for collaborative learning with data protection," *Journal of Machine Learning Research*, vol. 22, no. 226, pp. 1–6, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-815.html

[13] A. Ziller, A. Trask, A. Lopardo, B. Szymkow, B. Wagner, E. Bluemke, J.-M. Nounahon, J. Passerat-Palmbach, K. Prakash, N. Rose, T. Ryffel, Z. N. Reza, and G. Kaissis, *PySyft: A Library for Easy Federated Learning*. Cham: Springer International Publishing, 2021, pp. 111–139. ISBN 978-3-030-70604-3. [Online]. Available: https://doi.org/10.1007/978-3-030-70604-3_5

[14] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[16] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, T. Parcollet, and N. D. Lane, "Flower: A friendly federated learning research framework," *CoRR*, vol. abs/2007.14390, 2020. [Online]. Available: https://arxiv.org/abs/2007.14390

[17] J. Michalek, V. Skorpil, and V. Oujezsky, "Federated learning on android - highlights from recent developments," in *2022 14th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, 2022. doi: 10.1109/ICUMT57764.2022.9943382 pp. 27–30.

[18] J. Michalek, V. Oujezsky, and V. Skorpil, "Android federated learning framework for emergency management applications," https://nsr.utko.fekt.vut.cz/VK01030152.php, 2024, accessed: August 2024.

[19] L. Moroney, *The Firebase Realtime Database*. Berkeley, CA: Apress, 2017, pp. 51–71. ISBN 978-1-4842-2943-9. [Online]. Available: https://doi.org/10.1007/978-1-4842-2943-9_3

[20] Apache Software Foundation, "Kafka," 2024. [Online]. Available: https://kafka.apache.org

[21] Google, "TensorFlow Lite," 2024. [Online]. Available: https://www.tensorflow.org/lite

[22] A. Café, "Hal: Hybrid automata library," 2023, accessed: 21 September 2024. [Online]. Available: https://github.com/adrielcafe/hal

[23] J. Michalek, V. Oujezsky, M. Holik, and V. Skorpil, "A proposal for a federated learning protocol for mobile and management systems," *Applied Sciences*, vol. 14, no. 1, 2024. [Online]. Available: https://www.mdpi.com/2076-3417/14/1/101

[24] Firebase, *Manage Hosted Models with Firebase ML*, 2023, accessed: 21 September 2024. [Online]. Available: https://firebase.google.com/docs/ml/manage-hosted-models

[25] Firebase Documentation, "Realtime database usage limits," 2024, accessed: 2024-12-03. [Online]. Available: https://firebase.google.com/docs/database/usage/limits

[26] Google AI, "Litert operator compatibility guide," 2024, accessed: 2024-12-03. [Online]. Available: https://ai.google.dev/edge/litert/models/ops_compatibility.md

[27] Google Developers, "TensorFlow Lite is Now Liter!" Sep. 2024, accessed: 2024-11-24. [Online]. Available: https://developers.googleblog.com/en/tensorflow-lite-is-now-litert/

**Jakub Michalek** is an engineer currently pursuing a doctoral degree at the Department of Telecommunications, Faculty of Electrical Engineering and Communication, at Brno University of Technology. His research focuses on software development, with a particular emphasis on mobile and web applications. Passionate about creating innovative software, he has a strong interest in machine learning and exploring cutting-edge methodologies within the tech industry.

**Vaclav Oujezsky** studied teleinformatics at the Brno University of Technology, earning a Master's degree from 2008 to 2013. Between 2013 and 2017, he completed a Ph.D. in the same field. His professional experience includes working as a Network Engineer at T-Systems Czech Republic from 2006 to 2014, a Senior Network Engineer at T-Mobile CZ from 2014 to 2016, and a Senior Network Engineer at IBM Client Innovation Center Brno from 2016 to 2021. Simultaneously, he served as a Researcher in the Department of Telecommunications at the Brno University of Technology. Since 2021, he has held the position of Associate Professor at Masaryk University. Over the years, he has earned several certifications, contributed to numerous grant projects, and co-authored a wide range of scientific articles and conference papers. He is recognized as a leading expert in the field of modern information systems.

**Vladislav Skorpil** was born in Brno in 1955. He earned his MSc. and CSc. degrees from the Brno University of Technology (BUT). From 1980 to 1982, he worked as a designer at the telecommunication design office. In 1982, he returned to the Department of Telecommunications at BUT as a university teacher, where he has been working ever since. Currently, he holds the position of Associate Professor and is deeply interested in modern telecommunication systems. He is the author of approximately 65 international scientific papers and several manuals. Over the years, he has collaborated on numerous telecommunication projects, including those focused on digital transmission and switching systems, telecommunication broadband networks, data networks, neural networks, genetic algorithms, Quality of Service, data bit rate compression, IoT, 5G networks, and more.