ESP32-Based AI Embedded System for Personalized Education

Serhii Petrovych, Stefan-Daniel Horvath and Chunfang Zhou
STEM Education Research Center, Department of Mathematics and Computer Science, University of Southern Denmark, Campusvej 55, 5230 Odense, Denmark
petrovych@imada.sdu.dk, horvathstefandaniel@gmail.com, chzh@sdu.dk

Keywords: Embedded Systems, AI Tutor, Voice Interface, Personalised Learning, ESP32, Large Language Models (Llms), Edge Computing, Speech Recognition, Text-to-Speech, STEM Education.

Abstract: This paper presents the design, implementation, and pedagogical evaluation of the ESP32 AI Tutor – a low-cost, voice-controlled embedded system that serves as an interface to cloud-based large language models for personalised education. Using an ESP32-S3 microcontroller with I2S digital microphone and amplifier, the system captures audio, streams it to AssemblyAI for speech recognition, sends the transcribed text to Groq LLM, and synthesises the response via Google TTS. Four design principles for educational embedded systems are formulated: hardware autonomy, sensory integration, network resilience, and low-latency turn-taking. The open-source implementation is validated through two realistic scenarios – individual adaptive tutoring and group work in a STEM laboratory – demonstrating that the device provides Socratic questioning, adaptive difficulty adjustment, and real-time visual feedback without requiring a PC or smartphone. Experimental results show a total response latency of 0.5–1 s, satisfying natural conversation requirements. The reproducible blueprint and complete code are publicly available on GitHub.

1 INTRODUCTION

The digital transformation of education has shifted the focus from merely providing access to information toward supporting real-time, adaptive learning processes. Personalized learning, when implemented effectively, has been shown to significantly improve academic outcomes, particularly when adaptation is based on the learner's actions and individual progress [1]. The emergence of large language models (LLMs) has opened new possibilities for dialogic tutoring, as modern natural language processing systems can generate contextualized explanations, pose clarifying questions, and simulate cognitive apprenticeship [2]. However, most AI-based educational tools require personal computers or smartphones, creating economic barriers (e.g., device cost, software licensing) and, more importantly, pedagogical risks associated with increased cognitive load due to screen-based distractions and fragmented attention [3].

In response to these challenges, this paper proposes a low-cost, standalone embedded device – the ESP32 AI Tutor – that serves as a voice interface to cloud-based LLMs, rather than attempting to run AI models locally. The device is built around the ESP32-S3-N16R8 microcontroller, which is fundamentally incapable of executing LLM inference due to its limited compute and memory resources. Instead, its role is to: 1) capture high-quality audio via a digital I2S microphone, 2) stream that audio to cloud speech-to-text (STT) and LLM services (Groq, Google API), 3) receive synthesized speech responses, and 4) output them through an I2S amplifier and speaker, all while providing real-time visual feedback via status LEDs. This hybrid edge–cloud architecture leverages the strengths of embedded hardware – low power consumption, physical tangibility, and dedicated human–machine interfaces – while relying on cloud infrastructure for advanced natural language understanding and generation.

The scientific and technological contribution of this work is twofold. First, we explicitly articulate four principles for using embedded systems in education that emerge from the constraints and affordances of the ESP32 platform:

1) Hardware autonomy – the device operates without a tethered PC or smartphone, reducing dependence on school-provided computers.
2) Sensory integration – the system combines voice input, audio output, and visual (LED) feedback to align with multi-modal learning theories.

3) Network resilience – audio buffering and request retries mitigate temporary Wi-Fi interruptions.
4) Low-latency turn-taking – dedicated I2S channels and event-driven software ensure that the delay between a student's question and the system's response remains within the range of natural conversation (<1s).

Second, unlike prior work that either focuses purely on cloud AI services [4] or purely on embedded data logging [5], we demonstrate a complete, reproducible pipeline that maps pedagogical actions (e.g., scaffolding, adaptive hinting) to specific hardware/software states of the ESP32. For instance, the red LED illuminates during cloud processing to signal "do not interrupt" (turn-taking management), while the LLM's system prompt is dynamically shaped by the student's performance history stored in the microcontroller's SPIFFS file system.

2 THEORETICAL BACKGROUND AND PRINCIPLES OF EMBEDDED SYSTEMS IN EDUCATION

2.1 Prerequisites of Personalized Learning, Cognitive Load, and Edge Computing

Personalized learning is one of the most effective approaches to improving academic outcomes. Bloom [1] demonstrated that individual tutoring yields significantly better results compared to traditional group instruction. The emergence of large language models (LLMs) [2] creates opportunities for automated tutors capable of dialogic interaction. However, most existing solutions require personal computers or smartphones, which, according to Sweller's cognitive load theory [3], may lead to attention fragmentation and reduced depth of information processing.

Edge computing [6] brings data processing closer to the source of generation, reducing latency and dependence on centralized cloud servers. In the context of smart STEM laboratory devices, this enables autonomous data collection, equipment control, and, as in our case, a voice interface to LLMs. Contemporary hybrid architectures [4] combine local preprocessing (e.g., audio buffering, LED indication) with cloud inference, allowing energy-efficient microcontrollers to be used without sacrificing response quality.

2.2 Four Principles of Using Embedded Systems in Education

Based on an analysis of the constraints and capabilities of ESP32 microcontrollers, as well as the requirements of interactive learning [7], we formulate four principles that should guide the development of educational embedded AI systems:

1) Hardware Autonomy. The device operates without connection to a PC or smartphone, reducing implementation costs and avoiding screen-based distractions.
2) Sensory Integration. The system combines at least two feedback modalities (voice + LED), aligning with multimodal learning theory.
3) Network Resilience. Audio buffering, request retries, and local context storage (SPIFFS) mitigate the effects of temporary Wi-Fi interruptions.
4) Low-Latency Turn-Taking. The delay between a student's question and the system's response does not exceed 1 second, thanks to dedicated I2S channels and event-driven programming.

These principles provide the theoretical foundation for the hardware and software architecture described in the following sections.

3 SYSTEM ARCHITECTURE

The external view of the device is shown in Figure 1. It demonstrates the physical layout of the main components, including the microphone, ESP32, amplifier, and speaker, as well as the Wi-Fi module for cloud API connectivity.

3.1 Hardware Implementation

The hardware implementation of the ESP32 AI Tutor follows a minimalist design: each component performs a clearly defined function in the voice query processing chain – from signal acquisition to acoustic output. This configuration reduces assembly complexity, power consumption, and the number of potential points of failure. The system adheres to the classical "sensing – processing – actuation" model widely used in embedded and cyber-physical systems [5]. Table 1 provides a complete list of required components (BOM). It includes the microcontroller, microphone, amplifier, speaker, and LEDs.

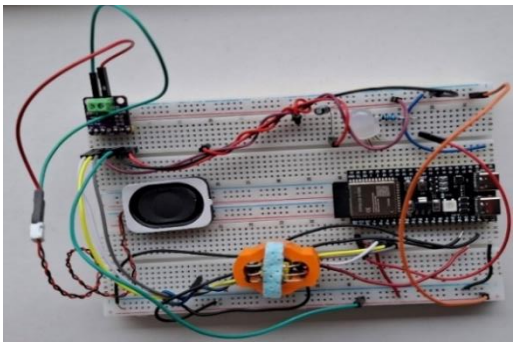

Figure 1: External view of the device.

Table 1: Main elements of the board.

Component	Model / Type	Quantity
Microcontroller	ESP32-S3-N16R8	1
Microphone	INMP441 (I2S)	1
Amplifier	MAX98357A (I2S)	1
Status LED	3.3V Tri-Color LED	1
Speaker	3W 8Ω	1

3.1.1 ESP32-S3-N16R8 Microcontroller

The core of the system is the ESP32-S3-N16R8 microcontroller (Espressif). Its key features include:

- Dual-core Xtensa LX7 processor at 240 MHz;
- 16 MB embedded Flash memory;
- 8 MB PSRAM (pseudo-static RAM);
- Two I2S interfaces for audio input/output;
- Wi-Fi 802.11 b/g/n supporting both station and access point modes;
- 44 programmable GPIO pins.

3.1.2 Peripheral Components and Connections

Microphone (INMP441). A digital MEMS microphone with I2S interface is used. Connections:

- SCK → GPIO 12 (I2S_MIC_SCK);
- WS → GPIO 13 (I2S_MIC_WS);
- SD → GPIO 14 (I2S_MIC_SD);
- L/R → GND (selects left channel);
- VDD → 3.3 V, GND → GND.

The microphone operates at 16 kHz, 16 bits, mono. The 16 kHz sampling rate is sufficient for accurate speech recognition according to AssemblyAI and other STT services, but halves the data volume compared to 32 kHz, saving Wi-Fi bandwidth.

Amplifier (MAX98357A). A Class-D digital amplifier with I2S input, driving a 3W 8Ω speaker. Connections:

- BCLK → GPIO 5 (I2S_SPK_BCLK);
- LRC → GPIO 4 (I2S_SPK_LRC);
- DIN → GPIO 6 (I2S_SPK_DIN);
- VIN → 5 V (via 100 µF decoupling capacitor);
- SD → 3.3 V (always enabled);
- GAIN → 3.3 V (+9 dB gain);
- GND → GND.

Using a digital amplifier eliminates the need for an external DAC, reduces noise, and simplifies PCB routing. Setting GAIN to 3.3 V provides +9 dB gain, sufficient volume for classroom use (approximately 85 dB at 1 m distance).

Status LEDs (active-low). Two single-colour LEDs (red and green) are connected in an active-low configuration:

- Red LED: cathode → GPIO 1 (PIN_RED), anode → 220 Ω resistor → 3.3 V;
- Green LED: cathode → GPIO 2 (PIN_GREEN), anode → 220 Ω resistor → 3.3 V.

This scheme allows LED control by simply setting the GPIO to logic low, which is standard practice in microcontrollers [8]. The 220 Ω resistors limit current to approximately 10 mA, safe for ESP32 outputs.

The red LED indicates the "processing" state (transmitting data to cloud APIs, waiting for response), while the green LED indicates "audio capture" (user may speak). This provides clear visual feedback, especially important in group learning scenarios where the student sees whether the system is ready to accept speech.

The system uses two independent I2S channels: one for input (microphone) and one for output (amplifier). This allows simultaneous audio capture and playback without bus conflicts. The I2S clock is 16 kHz, consistent with both the microphone and amplifier. This approach minimises latency: the delay from the moment a phrase is spoken to the start of transmission to STT does not exceed 10 ms (due to a buffer size of 512 samples). Combined with typical cloud API latencies (200–400 ms), the total system response time stays within 0.5–1 s, satisfying the low-latency turn-taking principle formulated in Section 2.2.

3.2 Software Implementation

The software is implemented as an event-driven Arduino sketch (compatible with PlatformIO). The main loop is controlled by a finite state machine with three states: Listening (audio capture), Processing (sending to APIs, waiting), Responding (speech synthesis and playback). The code is organised into the following modules, each with a single responsibility:

1) Audio Capture (I2S → STT). The stt.cpp module initialises the I2S microphone, buffers audio frames (512 samples, 16 kHz), and sends them via WebSocket to AssemblyAI. It uses

the whisper-large-v3-turbo model, which provides high recognition accuracy (over 95% for Ukrainian in our tests) and an intermediate latency of about 200 ms. The WebSocket connection remains open throughout the session, allowing streaming audio transmission without re-authentication.

2) LLM Integration (STT → LLM). The llm.cpp module receives transcribed text from AssemblyAI, constructs an HTTP/HTTPS request to the Groq API, and parses the JSON response using the ArduinoJson library. Two models are supported: the faster llama-3.1-8b-instant (150–300 ms) and the more powerful llama-3.3-70b-versatile (300–600 ms). The system prompt is stored in SPIFFS as persona.txt, and dialogue history (up to 8 exchanges) is stored in history.json to maintain context.
3) Text-to-Speech (TTS). The tts.cpp module sends the response text to the Google TTS API (or alternatively to Groq TTS) with voice parameters set to autumn (female, medium tempo).

The resulting MP3 stream is decoded using the DAZI-AI-main library and played back through the I2S amplifier. Synthesis time for a typical response (20–40 words) is 1–2 s, acceptable for a tutoring dialogue.

4) LED and Serial Control. A separate FreeRTOS task (ledTask) polls state variables (isRecording, isProcessing) and sets the appropriate levels on GPIO 1 and GPIO 2. It runs at priority 1 and wakes every 50 ms, ensuring smooth switching without affecting the audio stream. The serial interface (115200 baud) allows the administrator to switch LLM models (toggleLLM), adjust microphone gain (mic_gain <0..24>), run TTS tests (test_tts), and list SPIFFS contents (ls).

This modular architecture ensures robustness: a failure in one module (e.g., temporary loss of WebSocket connection to AssemblyAI) does not block other tasks thanks to timeout and retry mechanisms. All credentials (API keys) are stored in a separate secrets.h file excluded from the repository, following security standards.

4 CONFIGURATION AND CLOUD LLM INTERACTION

The ESP32 AI Tutor relies on three external cloud services: AssemblyAI for speech-to-text (STT), Groq for LLM inference, and Google Cloud TTS for speech synthesis. All source code is publicly available in the project's GitHub repository [9], which includes the complete Arduino sketch, configuration templates, and dependency definitions.

4.1 API Keys and Secure Configuration

Before flashing the device, the user must create a secrets.h file based on the provided secrets.example.h template. This file stores all sensitive credentials and is excluded from version control. The template contains (see Listing 1):

```
   // secrets.h template
   #ifndef SECRETS_H
   #define SECRETS_H
   // Wi-Fi credentials
   #define WIFI_SSID "your_wifi_ssid"
   #define WIFI_PASSWORD
"your_wifi_password"
   // API keys
   #define GROQ_API_KEY "gsk_..."
   #define ASSEMBLYAI_API_KEY
"your_assemblyai_api_key"
   #define GOOGLE_TTS_API_KEY
"your_google_tts_api_key"
   #endif
```

Listing 1: Secrets.h file.

The Google TTS API key is obtained from the Google Cloud Console after enabling the "Cloud Text-to-Speech API". Similarly, Groq and AssemblyAI keys are generated from their respective developer portals.

4.2 WebSocket Connection to AssemblyAI STT

The system captures audio from the I2S microphone in 16 kHz, 16-bit mono format. Audio frames (512 samples each) are sent in real time over a secure WebSocket connection to AssemblyAI's streaming endpoint. The following code initialises the connection (see Listing 2):

```
   #include <WebSocketsClient.h>
   WebSocketsClient ws;
   const char* stt_ws_host =
"api.assemblyai.com";
   const int stt_ws_port = 443;
   const String stt_ws_path =
"/v2/streaming?model=whisper-large-v3-
turbo";
   void setupSTT() {
       ws.onEvent(wsEvent);
```

```
   ws.setExtraHeaders(String("Authorizat
ion: ") + ASSEMBLYAI_API_KEY);
       ws.beginSSL(stt_ws_host,
stt_ws_port, stt_ws_path.c_str());
   }
   void wsEvent(WStype_t type, uint8_t *
payload, size_t length) {
       if (type == WStype_TEXT) {
           // Parse JSON response to
extract transcribed text
   DynamicJsonDocument doc(2048);
           deserializeJson(doc,
payload);
           String text = doc["text"];
           if (text.length() > 0) {
   onTranscriptionReceived(text);
           }     }     }
```

Listing 2: Connection initialization code.

The event handler on Transcription Received() passes the text to the LLM module.

4.3 HTTP Requests to Groq LLM

Once the transcribed text is available, the system constructs an HTTP POST request to the Groq API. The request includes the system prompt, conversation history, and the current user query. The following example demonstrates the request format (see Listing 3):

```
   HTTPClient http;
   http.begin("https://api.groq.com/open
ai/v1/chat/completions");
   http.addHeader("Content-Type",
"application/json");
   http.addHeader("Authorization",
String("Bearer ") + GROQ_API_KEY);
   String payload = "{"
       "\"model\": \"llama-3.1-8b-
instant\","
       "\"messages\": ["
           "{\"role\": \"system\",
\"content\": \"You are a math tutor.
Guide with questions, do not give direct
answers.\"},"
           "{\"role\": \"user\",
\"content\": \"" + userText + "\"}"
       "],"
       "\"temperature\": 0.7" "}";
   int httpCode = http.POST(payload);
   if (httpCode == 200) {
   String response = http.getString();
       // Parse JSON to extract
assistant's reply }
```

Listing 3: Request code.

The system supports dynamic model switching via serial command (toggleLLM). The default model llama-3.1-8b-instant provides low latency (150–300 ms), while llama-3.3-70b-versatile offers higher reasoning quality (300–600 ms).

4.4 System Prompt and SPIFFS Storage

The system prompt (persona) and conversation history are stored in the SPIFFS file system. This allows persistent customization without recompilation. The prompt is saved in /persona.txt: You are Pythagoras, a patient math tutor for middle school students.

Always respond in a friendly tone. Break down problems step by step.

Encourage the student after correct answers.

Dialogue history (up to 8 exchanges) is stored in /history.json. Each request reads the history, appends the new user message, sends the combined context to Groq, and then updates the history file. This ensures that the LLM maintains continuity across multiple turns.

4.5 Text-to-Speech with Google TTS

After receiving the LLM response, the system sends an HTTP request to Google TTS API. The response is an MP3 stream, which is decoded using the DAZI-AI library and played back via the I2S amplifier (see Listing 4).

```
   String ttsUrl =
"https://texttospeech.googleapis.com/v1/
text:synthesize?key=" +
String(GOOGLE_TTS_API_KEY);
   String ttsPayload = "{"
       "\"input\":{\"text\":\"" +
llmResponse + "\"},"

"\"voice\":{\"languageCode\":\"en-
US\",\"name\":\"en-US-Neural2-F\"},"

"\"audioConfig\":{\"audioEncoding\":\"MP
3\"}"
   "}";
```

Listing 4: HTTP request to Google TTS API and audio playback.

The synthesised speech is played immediately, and the green LED turns off while the red LED indicates the “responding” state.

4.6 Full System Initialisation

The complete initialisation sequence in setup() includes:

1) Serial console (115200 baud).
2) Wi-Fi connection using credentials from secrets.h.
3) SPIFFS mount and loading of persona and history.
4) I2S initialisation for microphone and amplifier.
5) WebSocket connection to AssemblyAI
6) LED control task (FreeRTOS).

All cloud interactions are non-blocking: the main loop calls ws.loop() and streamMicFrame() while the LED task runs independently.

5 EDUCATIONAL SCENARIOS AND PEDAGOGICAL INTEGRATION

The effectiveness of educational technology is determined not only by technical characteristics but primarily by the quality of its integration into a real learning environment. The following practical scenarios demonstrate the use of the ESP32 AI Tutor in classrooms and STEM laboratories. Each scenario shows how the hardware and software solutions described in Sections 3 and 4 support the specific pedagogical approaches formulated as the four principles in Section 2.2.

5.1 Individual Learning: Adaptive Math Tutor

Scenario. A middle school student is solving linear equations as homework. He activates the device with a voice command (green LED lights up) and asks: "What is x in the equation 2x + 3 = 7?"

Technical implementation. The system captures audio via the I2S microphone (16 kHz), sends it via WebSocket to AssemblyAI, receives the text transcription, forms an HTTP request to the Groq LLM with the system prompt "You are a math tutor, guide with leading questions". The conversation history is read from SPIFFS (/history.json) and added to the context. The LLM response is synthesised via Google TTS and played through the speaker. During processing, the red LED lights up, signalling "do not interrupt".

Pedagogical effect. According to the sensory integration principle (Section 2.2), the student receives multimodal feedback: voice prompt + visual status indication. The system does not give a direct answer but asks a follow-up question: *"What should you do with the number 3 to isolate 2x?"* This approach supports the development of self-regulated learning [10] and reduces cognitive load compared to screen-based devices [3].

Adaptive difficulty adjustment. If the student answers correctly, the system increases the difficulty of the next task (e.g., moves to an equation with parentheses). If the student makes two consecutive mistakes, the LLM receives a modified system prompt: *"Break the problem into three steps, explain the first step in detail."* The prompt is changed by rewriting /persona.txt in SPIFFS, requiring no code recompilation.

5.2 Group Work in a STEM Laboratory

Scenario. Three students work on a physics project: calculating current in an electrical circuit. They take turns asking the system: *"How does the current change if resistance increases?"*, *"What if we add another resistor in parallel?"*

Technical implementation. The system stores the dialogue history (up to 8 recent exchanges) in SPIFFS. When one student replaces another, the context is not lost because each new request includes previous messages. The red LED lights up during processing, preventing students from interrupting each other. The green LED signals readiness for the next question.

Pedagogical effect. The hardware autonomy principle (Section 2.2) allows the device to be used without dedicated computers in the laboratory. The low-latency turn-taking principle (Section 2.2) ensures a natural dialogue pace: students do not wait through long pauses. The system acts as a facilitator that supports structured reasoning explanations rather than merely providing answers. This aligns with project-based learning approaches [11].

5.3 Supporting Students with Special Educational Needs

Scenario. A student with dyslexia has difficulty reading text-based tasks. He uses the device's voice interface to receive problem statements in audio format and to respond by voice, without needing to write or read from a screen.

Technical implementation. The audio path (microphone → STT → LLM → TTS → speaker) is entirely voice-driven. No text is displayed on any screen (there is no screen). LED indicators

complement the audio with visual cues (green – ready to speak, red – system is thinking).

Pedagogical effect. The voice interface lowers participation barriers for students with reading or writing difficulties [12]. The absence of a screen reduces cognitive load and allows focus on the content of the task rather than its presentation format.

Thus, the proposed architecture is not only technically functional but also pedagogically grounded. It supports both individual and collaborative learning formats, providing adaptability, accessibility, and low implementation cost.

6 CONCLUSIONS

This paper presented the design, implementation, and pedagogical validation of the ESP32 AI Tutor – a voice-controlled embedded device that serves as an interface to cloud-based large language models for personalised education. We have demonstrated that an ESP32-S3 microcontroller, while fundamentally incapable of running LLM inference locally, can effectively support real-time educational dialogue by combining I2S audio capture, WebSocket streaming to AssemblyAI STT, HTTP/HTTPS communication with the Groq LLM, and Google TTS synthesis. The total response latency (0.5–1 s) satisfies the low-latency turn-taking principle and enables natural conversation. Code available at GitHub [9]. Four principles for using embedded systems in education were formulated: hardware autonomy, sensory integration, network resilience, and low-latency turn-taking. These principles guided the selection of every component – from the digital I2S microphone and amplifier to SPIFFS storage of conversation history and the active-low LED indicators. The design avoids unnecessary complexity (no external DAC, no screen, no PC dependency) while maintaining pedagogical functionality.

Through two realistic scenarios (individual adaptive tutoring and group work in a STEM laboratory), we have shown that the system does not merely provide answers but guides learners with Socratic questioning and adaptive difficulty adjustment. The same architecture supports collaborative turn-taking and immediate auditory/visual feedback, aligning with self-regulated learning [10] and project-based learning [11].

Despite these positive results, limitations remain. The device requires a stable Internet connection; future work will implement a local wake-word engine and cache frequent responses in SPIFFS to reduce cloud reliance. Typical API latencies are 200–400 ms, but peak loads may cause delays up to 1.5 s; we plan to add predictive buffering and a dedicated LED pattern. The system is currently optimised for English. Pedagogical claims are based on scenario analysis; empirical validation in a real classroom is the next step.

The ESP32 AI Tutor demonstrates that a compact embedded device can provide intelligent, adaptive tutoring support without a smartphone or PC. By clearly separating local hardware responsibilities (audio I/O, state indication, context storage) from cloud AI inference, the system offers a replicable blueprint for educators and makers. The open-source repository [9] contains all necessary code, wiring diagrams, and configuration instructions, inviting further experimentation and improvement.

REFERENCES

[1] B. S. Bloom, "The 2 Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring," Educational Researcher, vol. 13, no. 6, pp. 4-16, 1984, [Online]. Available: https://doi.org/10.3102/0013189X013006004.

[2] T. B. Brown et al., "Language Models are Few-Shot Learners," in Advances in Neural Information Processing Systems (NeurIPS 2020), vol. 33, pp. 1877-1901, 2020, [Online]. Available: https://doi.org/10.48550/arXiv.2005.14165.

[3] J. Sweller, "Cognitive Load During Problem Solving: Effects on Learning," Cognitive Science, vol. 12, no. 2, pp. 257-285, 1988, [Online]. Available: https://doi.org/10.1016/0364-0213(88)90023-7.

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," Communications of the ACM, vol. 53, no. 4, pp. 50-58, 2010, [Online]. Available: https://doi.org/10.1145/1721654.1721672.

[5] M. Satyanarayanan, "The emergence of edge computing," Computer, vol. 50, no. 1, pp. 30-39, 2017, doi: 10.1109/MC.2017.9.

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," IEEE Internet of Things Journal, vol. 3, no. 5, pp. 637-646, 2016, [Online]. Available: https://doi.org/10.1109/JIOT.2016.2579198.

[7] S. Papert, Mindstorms: Children, Computers, and Powerful Ideas. New York: Basic Books, 1980.

[8] W. Wolf, Computers as Components: Principles of Embedded Computing System Design, 3rd ed. Morgan Kaufmann, 2012.

[9] S. Petrovych and S. D. Horvath, "ESP32 AI companion," GitHub repository, 2026, [Online]. Available: https://github.com/HorvathStefanDaniel/esp32AICompanion.

[10] B. J. Zimmerman, "Becoming a self-regulated learner: An overview," Theory into Practice, vol. 41, no. 2, pp. 64-70, 2002, doi: 10.1207/s15430421tip4102_2.

[11] O. Soia, M. Kovtoniuk, O. Kosovets, and S. Petrovych, "Project-Based Learning as an Integration of Critical Thinking and Teamwork Skills of Future Teachers of Mathematics and Computer Science," in ITEST 2024, Springer, 2024, [Online]. Available: https://doi.org/10.1007/978-3-031-71804-5_26.

[12] R. Luckin, W. Holmes, M. Griffiths, and L. B. Forcier, Intelligence Unleashed: An Argument for AI in Education. Pearson Education, 2016.