ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG ĐIỆN-ĐIỆN TỬ

BÁO CÁO ĐỒ ÁN II

TRỢ LÝ AI SỬ DỤNG GIỌNG NÓI

Giảng viên hướng dẫn: **Tào Văn Cường**

Tên sinh viên: **Chu Quang Khải** **20210457**

Trần Huyền Nhi **20210659**

Mã lớp: **748500**

Hà Nội, tháng 1 năm 2025

LỜI NÓI ĐẦU

Trong thời đại cách mạng công nghiệp 4.0, trí tuệ nhân tạo (AI) và Internet vạn vật (IoT) đã trở thành hai lĩnh vực tiên phong, thay đổi cách chúng ta sống và làm việc. Các ứng dụng thông minh, từ nhà ở thông minh đến trợ lý ảo, ngày càng phổ biến và đóng vai trò quan trọng trong việc tối ưu hóa cuộc sống hàng ngày.

Đồ án **"Trợ lý AI sử dụng giọng nói"** được thực hiện với mục tiêu xây dựng một hệ thống trợ lý giọng nói thông minh, tích hợp giữa AI và IoT, dựa trên khả năng xử lý ngôn ngữ tự nhiên của các mô hình AI sẵn có và vi điều khiển ESP32. Hệ thống này không chỉ giúp người dùng giao tiếp dễ dàng thông qua giọng nói mà còn hỗ trợ điều khiển các thiết bị IoT, mang lại sự tiện lợi và hiệu quả cao.

Mục lục

I Tổng quan đề tài

1.1 Đặt vấn đề

Trong bối cảnh công nghệ phát triển vượt bậc, các giải pháp thông minh ngày càng được ứng dụng rộng rãi trong đời sống, từ nhà thông minh (Smart Home), thiết bị đeo tay thông minh (Wearable Devices), đến các hệ thống tự động hóa công nghiệp. Một trong những yếu tố quan trọng tạo nên sự phổ biến của các giải pháp này chính là khả năng tương tác tự nhiên giữa con người và thiết bị thông qua giọng nói.

Hệ thống trợ lý giọng nói (Voice Assistant) là một minh chứng điển hình cho sự tiến bộ này, khi cho phép người dùng thực hiện các tác vụ hàng ngày như điều khiển thiết bị, tìm kiếm thông tin, hoặc thậm chí trò chuyện, chỉ bằng cách sử dụng giọng nói. Tuy nhiên, hầu hết các hệ thống hiện tại như Google Assistant, Siri, hay Alexa đều yêu cầu phần cứng phức tạp và kết nối với nền tảng đám mây đắt đỏ.

Trong khi đó, **ESP32** – một vi điều khiển mạnh mẽ, giá thành thấp, tích hợp Wi-Fi và Bluetooth, đã trở thành lựa chọn lý tưởng trong các ứng dụng IoT. Khi kết hợp với công nghệ xử lý ngôn ngữ tự nhiên, ESP32 có thể được biến đổi thành một hệ thống trợ lý giọng nói hiệu quả, thông minh và tiết kiệm chi phí.

Vấn đề được đặt ra:

1. Làm thế nào để xây dựng một hệ thống trợ lý giọng nói sử dụng ESP32 với khả năng xử lý ngôn ngữ tự nhiên và tương tác theo ngữ cảnh?
2. Làm thế nào để hệ thống này có thể tích hợp điều khiển thiết bị IoT, đáp ứng yêu cầu tự động hóa trong đời sống, nhưng vẫn đảm bảo chi phí thấp, cấu hình đơn giản và dễ sử dụng?
3. Làm thế nào để tối ưu hóa hiệu năng xử lý trên phần cứng giới hạn của ESP32 trong khi vẫn đảm bảo trải nghiệm người dùng mượt mà?

Mục tiêu của đề tài: Đề tài này tập trung vào nghiên cứu và phát triển một hệ thống trợ lý giọng nói thông minh dựa trên ESP32, kết hợp với nền tảng sẵn có để xử lý ngôn ngữ tự nhiên, đồng thời tích hợp khả năng điều khiển thiết bị IoT. Sản phẩm này hướng đến:

- Cung cấp khả năng tương tác giọng nói tự nhiên, chính xác và thân thiện với người dùng.
- Tích hợp điều khiển các thiết bị IoT như đèn, quạt, cảm biến, tạo nên môi trường nhà thông minh.
- Thiết kế một giải pháp tiết kiệm chi phí, dễ triển khai và mở rộng.

Với tiềm năng ứng dụng cao và khả năng cá nhân hóa, hệ thống này không chỉ góp phần nâng cao trải nghiệm người dùng mà còn mở ra những cơ hội mới trong việc triển khai các giải pháp công nghệ thông minh ở nhiều lĩnh vực khác nhau.

1.2 Giải pháp

1.2.1 Kiến trúc hệ thống:

- **Đầu vào:** Micro INMP441 thu âm giọng nói, ESP32 xử lý tín hiệu và gửi đến API chuyển đổi giọng nói thành văn bản (Speech-to-Text - STT).
- **Xử lý trung tâm:** ESP32 kết nối với ChatGPT API để phân tích ngữ cảnh và sinh phản hồi.
- **Đầu ra:** Phản hồi qua loa bằng Text-to-Speech (TTS) hoặc điều khiển thiết bị loa thông qua Wi-Fi.

1.2.2 Cách thực hiện:

- **ESP32:** Là trung tâm xử lý và giao tiếp với Server API thông qua Wi-Fi.
- **Micro INMP441:** Thu giọng nói và giao tiếp qua I2S.
- **MAX98357:** Khuếch đại âm thanh từ phản hồi và phát qua loa.
- **Điều khiển IoT:** Kích hoạt thiết bị qua relay hoặc giao thức MQTT/HTTP.

1.3 Các sản phẩm hiện có trên thị trường

1 Google Assistant:

- Hỗ trợ giọng nói và điều khiển thiết bị qua Google Home.
- Tích hợp đa nền tảng (Android, iOS).
- Hạn chế: Yêu cầu kết nối Google Cloud.

2 Apple Siri:

- Tương tác giọng nói trên iPhone, iPad, HomePod.
- Điều khiển thiết bị trong hệ sinh thái HomeKit.
- Hạn chế: Chỉ hoạt động trong môi trường Apple.

3 Mycroft AI:

- Trợ lý giọng nói mã nguồn mở.
- Tùy chỉnh và triển khai trên phần cứng riêng.
- Hạn chế: Cộng đồng hỗ trợ hạn chế, phức tạp hơn để triển khai.

4 Xiaomi Mi AI Speaker:

- Loa thông minh điều khiển các thiết bị IoT của Xiaomi.
- Hạn chế: Tập trung vào hệ sinh thái Xiaomi.

II Phân tích và thiết kế hệ thống

2.1 Mô tả hệ thống

Hệ thống trợ lý giọng nói sử dụng ESP32, kết hợp các API xử lý ngôn ngữ và các thiết bị IoT, được thiết kế để nhận lệnh giọng nói từ người dùng, xử lý qua các API xử lý ngôn ngữ và thực hiện điều khiển thiết bị IoT.

Chức năng chính:

- Nhận lệnh giọng nói từ người dùng.
- Gửi dữ liệu giọng nói đến dịch vụ **Speech-to-Text** để chuyển thành văn bản.
- Sử dụng các API xử lý ngôn ngữ để xử lý và tạo phản hồi thông minh.
- Chuyển đổi phản hồi văn bản của các API đó thành giọng nói (**Text-to-Speech).**

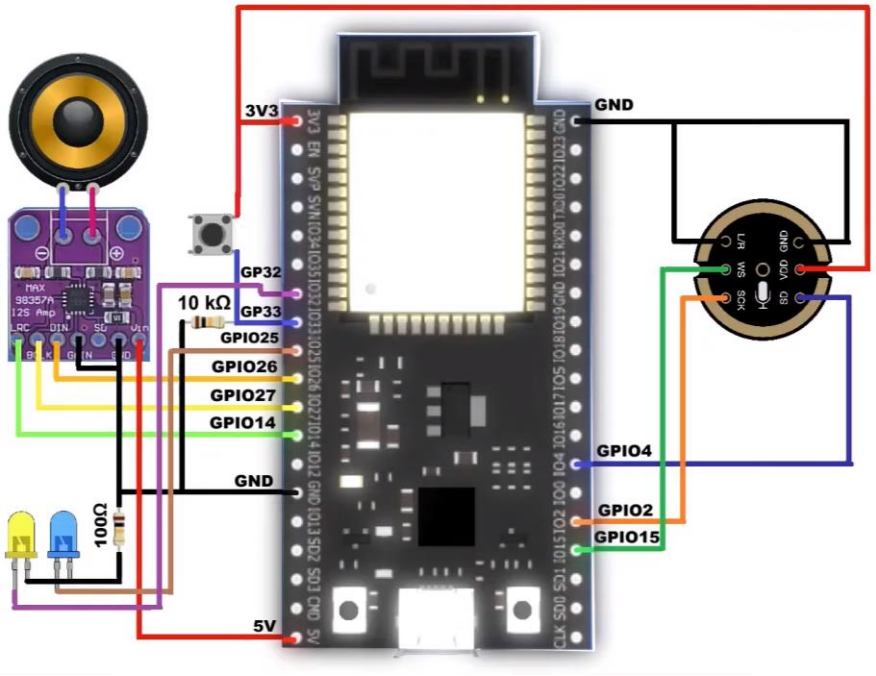
2.2 Sơ đồ kiến trúc hệ thống

Hệ thống được chia thành các thành phần chính sau:

1. **Người dùng**: Tương tác với hệ thống qua giọng nói.
2. **ESP32**: Thiết bị xử lý trung tâm, thực hiện các nhiệm vụ:
    - Thu âm giọng nói từ cảm biến âm thanh
    - Gửi dữ liệu đến server để xử lý.
3. **Máy chủ server**:
    - Nhận dữ liệu âm thanh từ ESP32, xử lý và phân tích lệnh giọng nói.
    - Trả về phản hồi thông minh từ dịch vụ AI (Google Gemini hoặc các dịch vụ tương tự).
    - Chuyển đổi phản hồi văn bản thành giọng nói và gửi lại cho ESP32 để phát lại qua loa.

2.3 Thiết kế phần cứng

2.3.1. Sơ đồ mạch



2.3.2. Danh sách linh kiện

1. Vi xử lý ESP32

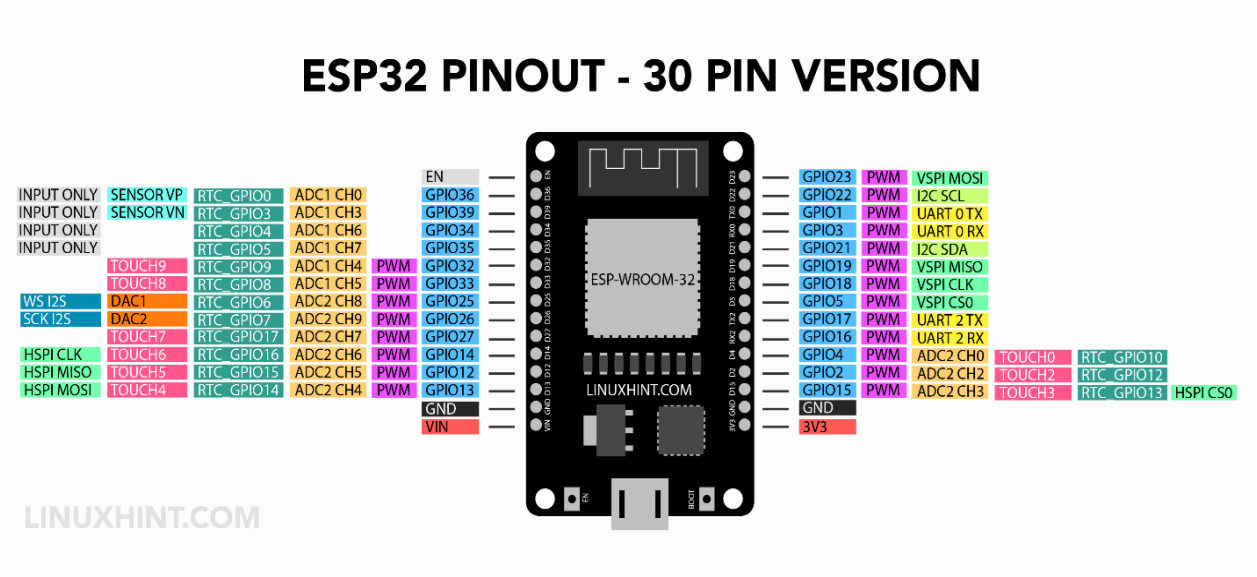
ESP32 là một module IoT(internet of thing) Hệ thống trên chip (SoC) có chi phí thấp và công suất thấp do Espressif phát triển. ESP32 được thiết kế trên kiến trúc 32bit với 2 core CPU hoạt động 1 cách độc lập có thể dễ dàng điều khiển.

Bộ xử lý	• CPU: Bộ vi xử lý Xtensa lõi kép (hoặc lõi đơn) 32-bit LX6, hoạt động ở tần số 160 hoặc 240 MHz và hoạt động ở tối đa 600 DMIPS • Bộ đồng xử lý (co-processor) công suất cực thấp (Ultra-low power)
Bộ nhớ	• 448 KB bộ nhớ ROM cho việc booting và các tính năng lõi • 520 KB bộ nhớ SRAM trên chip cho dữ liệu và tập lệnh
Kết nối không dây	• Wi-Fi: 802.11 b/g/n/e/i • Bluetooth: v4.2 BR/EDR and BLE

Ngoại vi INPUT/OUTPUT	• ADC SAR 12 bit, 18 kênh (ADC1: GPIO32-GPIO39; ADC2: GPIO0-GPIO15)
	• DAC 2 kênh, 8-bit (GPIO25, GPIO26)
	• 10 chân cảm ứng chạm: GPIO4, GPIO0, GPIO2, GPIO15, GPIO13, GPIO12, GPIO14, GPIO27, GPIO33, GPIO32
	• UART (Giao tiếp nối tiếp):
	  o TX (UART0_TXD): GPIO1.
	  o RX (UART0_RXD): GPIO3.
	• I2C (Giao tiếp 2 dây):
	  o SDA: GPIO21.
	  o SCL: GPIO22.
	• SPI (Giao tiếp tốc độ cao):
	  o MOSI: GPIO23.
	  o MISO: GPIO19.
	  o SCK: GPIO18.
	  o CS: GPIO5.
	• 2 giao diện I²S
	• Bộ điều khiển hồng ngoại từ xa (TX/RX, lên đến 8 kênh)
	• Tất cả các chân GPIO hỗ trợ PWM, dùng để điều khiển thiết bị như LED hoặc động cơ.
	• Một số chân GPIO như GPIO0, GPIO2, GPIO4, GPIO12, GPIO15… hỗ trợ chức năng RTC.

Bảo mật	• Hỗ trợ tất cả các tính năng bảo mật chuẩn IEEE 802.11, bao gồm WFA, WPA/WPA2 và WAPI • Secure boot • Mã hoá flash • 1024-bit OTP, lên đến 768-bit cho khách hàng • Tăng tốc mã hóa phần cứng: AES, SHA-2, RSA, elliptic curve cryptography (ECC, tạm dịch: mật mã đường cong ellip), trình tạo số ngẫu nhiên (random number generator)
Quản lý năng lượng	• Bộ ổn áp nội với điện áp rơi thấp (internal low-dropout regulator) • Miền nguồn riêng (individual power domain) cho RTC • Dòng 5 μA cho chế độ deep sleep

	• Trở lại hoạt động từ ngắt GPIO, timer, đo ADC, ngắt với cảm ứng điện dung



Hình 1: Sơ đồ chân vi xử lí ESP32

❖ Ứng dụng

ESP32 được ứng dụng trong các dự án IoT. Một lợi ích đáng kể của ESP32 là **mức tiêu thụ điện năng thấp** . ESP32 có thể được cấp nguồn bằng một cục pin nhỏ. Nó cũng đi kèm với kết nối Wi-Fi và Bluetooth tích hợp, giúp ESP32 dễ dàng thiết lập kết nối với internet và các thiết bị khác.

- Thiết bị IoT.
- SmartHome.
- Thiết bị đeo theo dõi.
- Robotics.
- Điều khiển từ xa.
- Automotive.

❖ Môi trường lập trình

- ESP-IDF (Espressif for IoT Development Framework): là môi trường lập trình do chính Espressif Systems cung cấp cho việc phát triển ứng dụng trên ESP32. ESP-IDF cung cấp các API và thư viện hỗ trợ các tính năng của ESP32 như Wifi, Bluetooth, GPIO, UART, I2C, I2S, SPI và hơn thế nữa. ESP-IDF được viết bằng ngôn ngữ C và hỗ trợ bởi các trình biên dịch GCC và Clang.

- Arduino IDE: là một môi trường lập trình phổ biến cho việc phát triển các ứng dụng IoT. Arduino IDE hỗ trợ các loại board khác nhau bao gồm cả ESP32. Nó cung cấp các thư viện hỗ trợ cho ESP32 và cho phép các nhà phát triển viết mã nguồn bằng C/C++.
- MicroPython: là một phiên bản rút gọn của python, được tối ưu cho việc chạy các thiết bị nhúng như ESP32. Micropython cho phép các nhà phát triển viết mã bằng python và sử dụng các tính năng của ESP32 thông qua các thư viện hỗ trợ.

2. Cảm biến âm thanh INMP441

Cảm biến âm thanh INMP441 là một loại micro kỹ thuật số MEMS (Micro-Electro-Mechanical System) có độ nhạy cao và khả năng thu âm chính xác, được sử dụng phổ biến trong các dự án âm thanh và nhận diện giọng nói. Dưới đây là các thông tin chi tiết về cảm biến này:

Thông số kỹ thuật cơ bản của INMP441:

1. **Giao tiếp:** I2S (Inter-IC Sound) – chuẩn giao tiếp kỹ thuật số phổ biến trong xử lý âm thanh.
2. **Điện áp hoạt động:** 1.8V – 3.3V.
3. **Dải tần số:** 60Hz – 15kHz.
4. **Độ nhạy:** -26 dBFS (±1dB).
5. **Tín hiệu đầu ra:** Âm thanh số dạng PCM (Pulse Code Modulation).
6. **Kích thước nhỏ gọn:** Dễ dàng tích hợp vào các dự án IoT và mạch điện tử.

3. Mạch khuếch đại âm thanh Max98357

Mạch khuếch đại âm thanh MAX98357 là một bộ khuếch đại âm thanh kỹ thuật số nhỏ gọn, hiệu quả cao, có khả năng chuyển đổi tín hiệu âm thanh số I2S thành tín hiệu âm thanh analog, được sử dụng để điều khiển loa. Đây là một giải pháp lý tưởng trong các ứng dụng âm thanh nhúng và IoT nhờ tiêu thụ năng lượng thấp và chất lượng âm thanh tốt.

Thông số kỹ thuật cơ bản của MAX98357:

1. **Giao tiếp đầu vào:** Tín hiệu âm thanh số I2S.
2. **Đầu ra:** Tín hiệu khuếch đại công suất cao cho loa (mono, Class D).
3. **Điện áp hoạt động:**
    - VDD: 2.5V - 5.5V.
    - PVDD (điện áp cấp cho loa): 2.5V - 5.5V.

4. **Công suất đầu ra:**
    - Lên đến 3.2W (ở 4Ω, nguồn 5V).
5. **Dải tần số:** 20Hz - 20kHz.
6. **Hiệu suất:** Lên đến 92%, nhờ thiết kế khuếch đại Class D.
7. **Cấu hình mono:** Hỗ trợ chế độ phát mono trực tiếp qua loa.
8. **Tự động phát hiện định dạng I2S:** Hỗ trợ nhiều tốc độ mẫu (8kHz đến 96kHz).

4. Giao thức I2C

I2C (Inter-Integrated Circuit) là một giao thức giao tiếp nối tiếp phổ biến được sử dụng để kết nối các vi điều khiển với các thiết bị ngoại vi (như cảm biến, bộ nhớ, màn hình, ADC, DAC, v.v.).

Đặc điểm chính của giao thức I2C

Chế độ giao tiếp	I2C là giao thức đa thiết bị và nhiều master-slave, nghĩa là: • Có thể có một hoặc nhiều master và một hoặc nhiều slave trên cùng một bus. • Master là thiết bị điều khiển bus (thường là vi điều khiển). • Slave là thiết bị ngoại vi như cảm biến hoặc module khác.
Tốc độ truyền dữ liệu:	• Standard mode: 100 kbps. • Fast mode: 400 kbps. • Fast mode plus: 1 Mbps. • High-speed mode: 3.4 Mbps.
Số lượng dây cần thiết:	• SCL (Serial Clock Line): Đồng hồ tín hiệu do master tạo ra. • SDA (Serial Data Line): Đường dữ liệu để truyền thông tin giữa master và slave.
Điện trở Pull-up:	Cả SCL và SDA đều cần điện trở pull-up (thường là 4.7kΩ) để hoạt động ổn định.

Cách hoạt động của giao thức I2C

Giao thức I2C hoạt động dựa trên mô hình master-slave, trong đó master điều khiển tín hiệu SCL và khởi tạo giao tiếp trên SDA

Địa chỉ hóa thiết bị	• Mỗi thiết bị slave trên bus I2C có một địa chỉ duy nhất (7-bit hoặc 10-bit). • Master sử dụng địa chỉ này để chọn slave cụ thể trong quá trình truyền dữ liệu.
Chu trình giao tiếp	Giao tiếp I2C bao gồm 3 giai đoạn chính: • Start Condition (Điều kiện bắt đầu): o Master kéo SDA từ mức cao xuống mức thấp trong khi SCL vẫn ở mức cao, báo hiệu bắt đầu giao tiếp. • Truyền dữ liệu: o Dữ liệu được truyền qua SDA, đồng bộ với xung nhịp trên SCL. o Mỗi byte (8 bit) được truyền, sau đó slave gửi một bit ACK (Acknowledge) để xác nhận đã nhận được dữ liệu. • Stop Condition (Điều kiện kết thúc): o Master kéo SDA từ mức thấp lên mức cao trong khi SCL vẫn ở mức cao, báo hiệu kết thúc giao tiếp.
Frame dữ liệu I2C	Dữ liệu trong giao thức I2C được tổ chức thành các frame như sau: • Start Condition • Địa chỉ slave (7-bit) + Bit đọc/ghi (1-bit): Bit cuối xác định đọc (1) hoặc ghi (0). • ACK/NACK từ slave (1-bit) • Dữ liệu (8-bit): Được truyền byte từng byte. • ACK/NACK sau mỗi byte. • Stop Condition

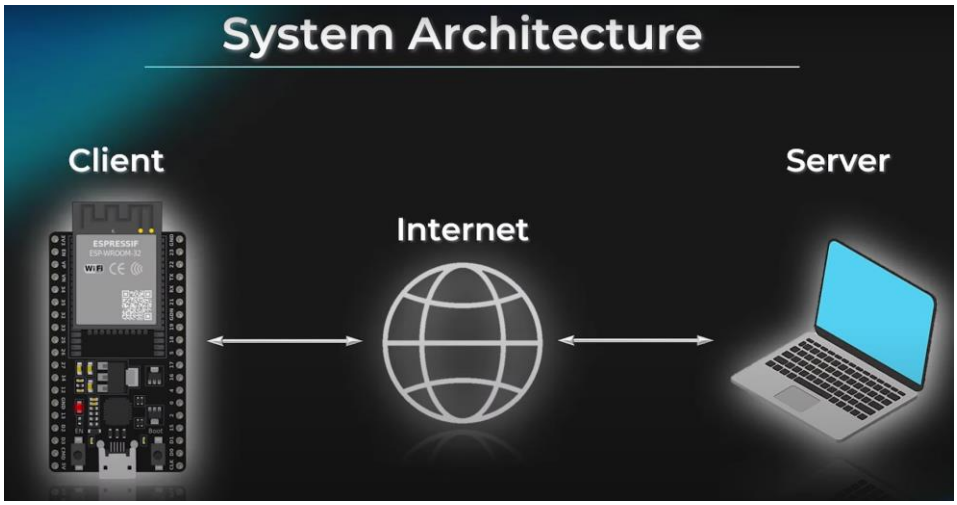
Ưu nhược điểm của giao thức I2C

Ưu	Nhược
• Tiết kiệm số lượng dây: Chỉ cần 2 dây cho cả truyền và nhận dữ liệu. • Hỗ trợ nhiều thiết bị trên cùng bus: Có thể kết nối nhiều slave với một master. • Độ ổn định: Giao tiếp đồng bộ nhờ SCL.	• Tốc độ không cao bằng SPI: I2C chậm hơn SPI do cần xử lý ACK/NACK và Start/Stop Condition. • Cần điện trở pull-up: Dễ gặp vấn đề nếu không cấu hình đúng.

• Tốc độ linh hoạt: Hỗ trợ nhiều chế độ tốc độ, từ 100 kbps đến 3.4 Mbps.	• Địa chỉ thiết bị bị giới hạn: Địa chỉ 7-bit chỉ cho phép kết nối tối đa 127 thiết bị (10-bit có thể dùng nhiều hơn).

2.4 Thiết kế phần mềm

2.4.1 Tổng quan



Hệ thống ESP32 Voice Assistant được thiết kế để xử lý thu âm, phát âm thanh, và giao tiếp với máy chủ. Chương trình được phát triển trên nền tảng ESP-IDF với các thành phần chính như sau:

- I2S: Giao tiếp với microphone INMP441 để thu âm và loa MAX98357A để phát âm thanh.
- SPIFFS: Hệ thống tệp nhúng để lưu trữ các tệp âm thanh.
    - Lưu trữ các tệp âm thanh ghi âm từ microphone.
    - Hỗ trợ xóa và ghi đè các tệp âm thanh khi cần thiết.
- WiFi: Kết nối mạng không dây để gửi và nhận dữ liệu với máy chủ. Kiểm tra trạng thái kết nối mạng, đảm bảo quá trình truyền dữ liệu không bị gián đoạn.
- RTOS Tasks: Tổ chức mã nguồn theo mô hình đa luồng để tăng hiệu quả xử lý. Hệ thống cần thực hiện nhiều chức năng cùng lúc, như:
    - Ghi âm từ microphone (INMP441).
    - Lưu trữ dữ liệu âm thanh vào SPIFFS.
    - Phát lại âm thanh qua DAC hoặc loa (MAX98357A).

- Duy trì kết nối Wi-Fi để truyền dữ liệu lên máy chủ.

2.4.2 Các hàm chính trong hệ thống.

1. Khởi tạo và quản lý hệ thống tệp SPIFFS

Hàm SPIFFSInit()

- Mục đích:
  - Khởi tạo hệ thống tệp SPIFFS để lưu trữ các tệp âm thanh ghi âm và phát lại.
  - Kiểm tra và xóa các tệp cũ nếu tồn tại, đảm bảo tệp mới sẽ không bị ghi đè dữ liệu không mong muốn.
- Chi tiết hoạt động:
  - Gọi hàm SPIFFS.begin() để khởi động hệ thống SPIFFS.
  - Xóa các tệp âm thanh cũ như audioRecordfile và audioResponsefile nếu chúng tồn tại.
  - Tạo một tệp mới với tiêu đề WAV được thiết lập thông qua hàm wavHeader().
- Vai trò:
  - Đảm bảo không gian lưu trữ được tối ưu hóa và định dạng tệp WAV chính xác trước khi ghi dữ liệu âm thanh.

Hàm printSpaceInfo()

- Mục đích:
  - Hiển thị thông tin về không gian sử dụng và còn trống trong hệ thống SPIFFS.
- Vai trò:
  - Hỗ trợ kiểm tra và giám sát dung lượng lưu trữ, đảm bảo việc ghi và đọc tệp không gặp lỗi.

2. Khởi tạo giao tiếp I2S

Hàm i2sInitINMP441()

- Mục đích:

    - Thiết lập giao tiếp I2S với microphone INMP441 để thu âm.
- Chi tiết hoạt động:
    - Cấu hình chế độ I2S là Master | RX (chỉ nhận dữ liệu).
    - Định nghĩa tần số mẫu, độ phân giải, và định dạng dữ liệu âm thanh (16-bit, mono).
    - Gán chân giao tiếp (clock, data, word select) tương ứng với phần cứng.
    - Cài đặt driver I2S với các thông số đã cấu hình.
- Vai trò:
    - Đảm bảo dữ liệu âm thanh từ microphone được đọc chính xác và đúng định dạng.

Hàm i2sInitMax98357A()

- Mục đích:
    - Thiết lập giao tiếp I2S với DAC MAX98357A để phát âm thanh.
- Chi tiết hoạt động:
    - Cấu hình chế độ I2S là Master | TX (chỉ truyền dữ liệu).
    - Định nghĩa tần số mẫu và độ phân giải tương thích với phần cứng.
    - Gán chân giao tiếp (BCLK, LRCLK, DATA_OUT) với DAC.
    - Cài đặt driver I2S và xóa DMA buffer để tránh dữ liệu dư thừa.
- Vai trò:
    - Chuyển đổi và phát dữ liệu âm thanh từ ESP32 ra loa thông qua DAC.

3. Xử lý ghi âm thanh

Hàm I2SAudioRecord(void *arg)

- Mục đích:
    - Thu thập dữ liệu âm thanh từ microphone INMP441 và ghi vào tệp WAV trên SPIFFS.
- Chi tiết hoạt động:
    - Khởi tạo buffer đọc dữ liệu I2S và buffer lưu trữ dữ liệu vào SPIFFS.

    - Đọc liên tục dữ liệu từ microphone bằng i2s_read().
    - Gọi hàm I2SAudioRecord_dataScale() để xử lý biên độ dữ liệu trước khi lưu vào tệp.
    - Theo dõi kích thước dữ liệu đã ghi để dừng khi đạt đủ thời gian ghi âm yêu cầu.
    - Sau khi ghi âm xong:
        - Tắt đèn LED báo hiệu.
        - Gửi dữ liệu lên máy chủ nếu Wi-Fi được kết nối.
- Vai trò:
    - Cung cấp khả năng ghi âm liên tục và lưu trữ tệp âm thanh trong bộ nhớ SPIFFS.

Hàm I2SAudioRecord_dataScale(uint8_t *d_buff, uint8_t *s_buff, uint32_t len)

- Mục đích:
    - Chuẩn hóa và nén biên độ dữ liệu âm thanh đọc từ microphone để phù hợp với định dạng WAV.
- Chi tiết hoạt động:
    - Chuyển đổi dữ liệu 16-bit từ microphone thành dạng phù hợp trước khi lưu vào SPIFFS.
- Vai trò:
    - Đảm bảo dữ liệu được xử lý đúng cách để đạt chất lượng âm thanh tốt nhất.

4. Quản lý kết nối Wi-Fi

Hàm wifiConnect(void *pvParameters)

- Mục đích:
    - Kết nối ESP32 với mạng Wi-Fi được định nghĩa trước.
- Chi tiết hoạt động:
    - Gọi hàm WiFi.begin() với tên và mật khẩu Wi-Fi.
    - Duy trì kết nối Wi-Fi và báo hiệu trạng thái kết nối qua đèn LED.
    - Giữ kết nối trong suốt quá trình hoạt động của ESP32.

- Vai trò:
    - Cung cấp liên lạc mạng để gửi và nhận dữ liệu từ máy chủ.

5. Xử lý giao tiếp với máy chủ

Hàm uploadFile()

- Mục đích:
    - Gửi tệp âm thanh ghi âm lên máy chủ Node.js.
- Chi tiết hoạt động:
    - Mở tệp ghi âm từ SPIFFS.
    - Sử dụng HTTP POST để gửi tệp với định dạng audio/wav.
    - Xử lý phản hồi từ máy chủ, in thông tin hoặc chuyển ESP32 vào chế độ ngủ sâu nếu cần.
- Vai trò:
    - Cung cấp chức năng upload dữ liệu lên máy chủ để xử lý thêm, như nhận diện giọng nói.

Hàm broadcastAudio(void *arg)

- Mục đích:
    - Nhận dữ liệu âm thanh từ máy chủ và phát lại qua DAC MAX98357A.
- Chi tiết hoạt động:
    - Khởi tạo giao tiếp I2S với DAC.
    - Tạo kết nối HTTP với máy chủ và đọc luồng dữ liệu âm thanh.
    - Phát dữ liệu âm thanh qua loa, đồng thời nhân đôi biên độ để tăng âm lượng.
    - Sau khi phát xong, đưa ESP32 vào chế độ ngủ sâu.
- Vai trò:
    - Tái hiện nội dung âm thanh từ máy chủ, hoàn thiện chức năng trợ lý giọng nói.

6. Đồng bộ hóa và quản lý đa luồng

Hàm semaphoreWait(void *arg)

- Mục đích:
    - Đồng bộ hóa giữa các tác vụ, chờ tín hiệu hoàn tất ghi âm trước khi kiểm tra trạng thái máy chủ.
- Chi tiết hoạt động:
    - Sử dụng semaphore để kiểm soát luồng dữ liệu, đảm bảo chỉ phát âm thanh khi việc ghi âm đã hoàn tất và máy chủ đã sẵn sàng.
- Vai trò:
    - Tăng độ tin cậy và hiệu quả xử lý khi làm việc với nhiều luồng.

2.5 Thiết kế Server máy chủ

Hệ thống server sử dụng **Node.js** kết hợp với thư viện **Express** để thiết lập một server mạnh mẽ, hỗ trợ xử lý các chức năng cốt lõi như nhận diện giọng nói, tích hợp trí tuệ nhân tạo, chuyển đổi văn bản thành giọng nói và giao tiếp với client. Phần dưới đây mô tả chi tiết cách xây dựng server, các chức năng chính và những thách thức trong quá trình triển khai.

2.5.1. Cấu hình cơ bản của server

Server được khởi tạo trên cổng 3000, đóng vai trò trung tâm xử lý dữ liệu giữa client và các dịch vụ trí tuệ nhân tạo bên thứ ba. Server được thiết kế để xử lý hiệu quả các yêu cầu của client thông qua giao thức HTTP và WebSocket. Hai loại request chính được hỗ trợ:

- **POST**: Dùng để nhận dữ liệu âm thanh từ client và xử lý qua API nhận diện giọng nói.
- **GET**: Dùng để kiểm tra trạng thái hệ thống và trả về kết quả âm thanh đã xử lý.

Server được cấu hình với middleware của **Express** để xử lý các yêu cầu JSON và multipart/form-data với dung lượng tối đa 50MB, đảm bảo phù hợp với dữ liệu âm thanh lớn.

2.5.2 Chức năng chính của server

2.5.2.1. Nhận file âm thanh từ client

- API: /uploadAudio
    - Mô tả: API này nhận dữ liệu âm thanh từ client qua phương thức POST. Dữ liệu được truyền dưới dạng stream và lưu trữ tạm thời trên server với tên file recording.wav.
    - Luồng xử lý chi tiết:

1. Dữ liệu âm thanh từ client được nhận qua stream và ghi vào file recording.wav sử dụng thư viện fs.
2. Sau khi file được lưu, server gọi đến API nhận diện giọng nói (Speech-to-Text API) để trích xuất nội dung văn bản từ âm thanh.
3. Kết quả văn bản trả về được gửi lại client và đồng thời được sử dụng để gọi API trí tuệ nhân tạo của Google Gemini.

2.5.2.2. Chuyển đổi giọng nói thành văn bản (Speech-to-Text)

- Phương pháp: Sử dụng dịch vụ API của Google để thực hiện nhận diện giọng nói.
- Quy trình:
    1. File âm thanh recording.wav được đọc và chuyển đổi thành định dạng Base64.
    2. Nội dung Base64 được gửi đến API Speech-to-Text với yêu cầu tạo transcript.
    3. Server nhận kết quả transcript, xử lý và chuyển tiếp đến Google Gemini API để tạo phản hồi AI.

2.5.2.3. Tích hợp trí tuệ nhân tạo Google Gemini

- Chức năng: Tận dụng sức mạnh của mô hình trí tuệ nhân tạo Gemini để phân tích và phản hồi dựa trên nội dung văn bản.
- Cách hoạt động:
    - Văn bản từ API Speech-to-Text được truyền đến Google Gemini API
    - Google Gemini trả về một đoạn văn bản phản hồi. Kết quả này tiếp tục được chuyển đổi thành giọng nói.

2.5.2.4. Chuyển đổi văn bản thành giọng nói (Text-to-Speech - TTS)

- Mô tả: Server tích hợp WebSocket để sử dụng dịch vụ TTS của bên thứ ba là API TTS của Xunfei.
- Quy trình xử lý chi tiết:
    1. Văn bản phản hồi từ Google Gemini được gửi qua WebSocket đến API TTS.
    2. API TTS trả về dữ liệu âm thanh theo từng khung nhỏ (frame) dưới dạng Base64.
    3. Server kết hợp các frame thành một buffer âm thanh hoàn chỉnh và lưu dưới định dạng file WAV (voicedby.wav).

4. File WAV được chuẩn hóa bằng cách tạo header WAV chuẩn để đảm bảo khả năng phát lại trên các thiết bị khác nhau.

2.5.2.5. Cung cấp file âm thanh đã xử lý cho client

- API: /broadcastAudio
    - Mô tả: API này cung cấp file âm thanh kết quả (voicedby.wav) cho client.
    - Cách hoạt động:
        1. Kiểm tra sự tồn tại của file WAV trên server.
        2. Nếu file tồn tại, server truyền file đến client dưới dạng stream, đảm bảo hiệu suất cao khi xử lý dữ liệu lớn.

2.5.2.6. Kiểm tra trạng thái hệ thống

- API: /checkVariable
    - Mô tả: API này cho phép client kiểm tra xem file âm thanh đã được xử lý xong hay chưa thông qua biến trạng thái shouldDownloadFile.
    - Ứng dụng: Hỗ trợ đồng bộ hóa quá trình xử lý giữa client và server.

2.6 Quy trình tổng quát của hệ thống

1. Upload file âm thanh: Client gửi file âm thanh đến server thông qua API /uploadAudio.
2. Xử lý âm thanh: File được chuyển đổi thành văn bản bằng Speech-to-Text API.
3. Phân tích nội dung: Văn bản được gửi đến Google Gemini để nhận phản hồi.
4. Chuyển đổi văn bản thành âm thanh: Phản hồi từ Gemini được chuyển đổi thành giọng nói bằng API TTS.
5. Trả về kết quả: File âm thanh kết quả (voicedby.wav) được lưu trữ và trả về client qua API /broadcastAudio.

III Tổng kết

3.1. Kết quả đạt được

- Hệ thống hoạt động ổn định, có thể nhận diện và phản hồi lệnh bằng giọng nói một cách chính xác.
- Tích hợp điều khiển các thiết bị IoT như đèn, quạt, cảm biến nhiệt độ, giúp tăng tính tiện lợi và tự động hóa trong môi trường gia đình thông minh.

- Mô hình mang tính linh hoạt và có thể mở rộng thêm nhiều tính năng khác, như học hỏi từ người dùng hoặc tích hợp các dịch vụ thông minh khác.

3.2. Phân tích ưu nhược điểm

3.1.1 Ưu điểm

- **Khả năng kết nối và mở rộng cao:** Hệ thống ESP32 Voice Assistant có khả năng kết nối mạng không dây qua Wi-Fi, cho phép người dùng điều khiển và nhận lệnh từ xa. Hệ thống có thể dễ dàng mở rộng để tích hợp với các thiết bị IoT khác thông qua giao thức MQTT, giúp tạo ra một hệ sinh thái nhà thông minh linh hoạt.
- **Xử lý thời gian thực:** Với khả năng thu âm và phát âm thanh qua các giao diện I2S, hệ thống có thể xử lý các lệnh giọng nói trong thời gian thực, tạo ra trải nghiệm người dùng mượt mà và không gián đoạn.
- **Tiết kiệm chi phí và tài nguyên:** ESP32 là một vi điều khiển mạnh mẽ nhưng tiết kiệm chi phí và tài nguyên. Điều này giúp giảm chi phí triển khai hệ thống mà vẫn đảm bảo hiệu năng xử lý tốt, đặc biệt là khi tích hợp với các dịch vụ AI trên máy chủ.
- **Dễ dàng cập nhật và bảo trì:** Hệ thống có thể được cập nhật và bảo trì từ xa qua kết nối mạng. Việc xử lý và lưu trữ dữ liệu trên server giúp giảm tải cho thiết bị phần cứng và dễ dàng cập nhật các mô hình AI mà không cần thay đổi phần cứng.
- **Tích hợp với các dịch vụ AI:** Hệ thống có thể kết nối và sử dụng các dịch vụ AI mạnh mẽ như Google Gemini hoặc các nền tảng trí tuệ nhân tạo khác để xử lý ngữ nghĩa và đưa ra phản hồi thông minh, tăng tính linh hoạt và khả năng đáp ứng của hệ thống.

3.1.2 Nhược điểm

- **Phụ thuộc vào kết nối mạng:** Hệ thống yêu cầu kết nối Wi-Fi ổn định để truyền tải dữ liệu giữa ESP32 và server. Nếu mạng gặp sự cố hoặc tốc độ truyền tải kém, sẽ ảnh hưởng đến khả năng phản hồi của hệ thống, gây gián đoạn trải nghiệm người dùng.
- **Tiêu thụ năng lượng cao khi hoạt động liên tục:** Mặc dù ESP32 có mức tiêu thụ năng lượng thấp, nhưng khi hệ thống phải liên tục thu âm, truyền tải và xử lý dữ liệu, nó vẫn có thể tiêu tốn một lượng năng lượng đáng kể. Điều này có thể gây khó khăn cho các ứng dụng yêu cầu hoạt động lâu dài hoặc trên các thiết bị không có nguồn điện ổn định.
- **Yêu cầu tài nguyên tính toán lớn trên máy chủ:** Việc sử dụng dịch vụ AI và xử lý âm thanh đòi hỏi server phải có khả năng tính toán mạnh mẽ và tài nguyên lớn. Điều này có thể làm tăng chi phí phần cứng và yêu cầu bảo trì máy chủ liên tục.
- **Chất lượng âm thanh có thể bị ảnh hưởng bởi phần cứng và môi trường:** Chất lượng ghi âm từ microphone INMP441 và khả năng phát âm thanh qua loa MAX98357A có thể bị ảnh hưởng bởi các yếu tố môi trường như tiếng ồn hoặc chất

lượng phần cứng. Điều này có thể làm giảm độ chính xác của quá trình nhận diện giọng nói và sự rõ ràng của phản hồi.

3.3. Phát triển phần mềm

Hệ thống ESP32 Voice Assistant có thể phát triển theo các hướng sau:

1. **Cải thiện chất lượng nhận diện giọng nói**: Tích hợp dịch vụ nhận diện giọng nói mạnh mẽ hơn và giảm tiếng ồn để nâng cao độ chính xác.
2. **Tăng cường trí tuệ nhân tạo**: Phát triển AI để phản hồi thông minh và tự động hóa các tác vụ phức tạp.
3. **Tiết kiệm năng lượng và tối ưu hóa hiệu suất**: Tối ưu phần mềm và phần cứng để giảm tiêu thụ năng lượng và cải thiện hiệu suất.
4. **Mở rộng kết nối IoT**: Điều khiển thêm các thiết bị IoT và tích hợp với hệ sinh thái nhà thông minh.
5. **Bảo mật và bảo vệ dữ liệu**: Cải thiện các phương thức xác thực và mã hóa dữ liệu để bảo vệ quyền riêng tư người dùng.

Kết luận

Trong đồ án này, chúng tôi đã thiết kế và phát triển một hệ thống trợ lý giọng nói dựa trên nền tảng ESP32 và các mô hình AI xử lý ngôn ngữ, kết hợp các tính năng của IoT để tạo ra một giải pháp thông minh và tiện lợi trong cuộc sống hàng ngày.

Hệ thống được xây dựng thành công với các chức năng chính bao gồm nhận diện giọng nói, xử lý ngôn ngữ tự nhiên và điều khiển các thiết bị IoT. ESP32 đóng vai trò làm bộ điều khiển trung tâm, kết nối với các cảm biến và thiết bị ngoại vi, trong khi server đảm nhiệm việc phân tích và phản hồi thông minh theo ngữ cảnh của người dùng.

Dự án đã chứng minh được tiềm năng của việc tích hợp công nghệ AI và IoT trong việc xây dựng các giải pháp thông minh phục vụ đời sống. Đây là nền tảng quan trọng để tiếp tục phát triển và ứng dụng công nghệ này vào các lĩnh vực khác như giáo dục, y tế, hoặc công nghiệp thông minh.

Quá trình thực hiện đồ án không chỉ là cơ hội để nhóm tôi áp dụng những kiến thức đã học mà còn là dịp khám phá và phát triển các kỹ năng mới trong lĩnh vực công nghệ. Trong suốt thời gian nghiên cứu và triển khai.

Chúng tôi xin chân thành cảm ơn sự hướng dẫn của Thầy **Tào Văn Cường** trong quá trình thực hiện đồ án. Bên cạnh đó, chúng tôi cũng ý thức được rằng, mặc dù đã cố gắng hoàn thiện, nhưng đồ án khó tránh khỏi những thiếu sót. Chúng tôi rất mong nhận được ý kiến đóng góp từ quý Thầy để tiếp tục cải thiện và nâng cao chất lượng công việc.