Desenvolvimento de um Assistente Doméstico Baseado em IA Generativa na Borda (Edge AI)

Ricardo Magno do Carmo Junior, Universidade Federal de Itajubá

***Resumo*—Este relatório apresenta o desenvolvimento, implementação e análise técnica do "Delta", um assistente doméstico inteligente projetado para operar inteiramente na borda (Edge), utilizando um Raspberry Pi 5 como unidade central de processamento. O projeto representa um avanço significativo em relação aos sistemas de automação convencionais baseados em nuvem, priorizando a privacidade do usuário e a autonomia operacional. O sistema integra reconhecimento de fala offline através da biblioteca VOSK, processamento de linguagem natural e raciocínio lógico via Small Language Model (SLM) Llama 3.2:3b executado no framework Ollama, e controle físico de dispositivos IoT (ar condicionado, ventilador, iluminação) através do protocolo Tuya com chaves locais. A arquitetura de hardware incorpora um conjunto de sensores redundantes (DHT22, AHT20, BMP280) para fusão de dados ambientais, permitindo decisões autônomas de climatização e iluminação. O relatório detalha a engenharia de software modular, a estratégia de *function calling* para tradução de intenção em ação, e avalia o desempenho do sistema em termos de latência de inferência, gerenciamento térmico e robustez da comunicação local.**

***Index Terms*—Edge AI, Small Language Models, Raspberry Pi 5, IoT, Automação Residencial, Privacidade.**

CONTEÚDO

I. Introdução

A. Contextualização e Motivação

A evolução da Internet das Coisas (IoT) transformou residências em ambientes conectados, onde dispositivos inteligentes prometem conveniência e eficiência energética. No entanto, a arquitetura predominante desses sistemas depende pesadamente da computação em nuvem. Assistentes de voz comerciais, como Alexa e Google Assistant, operam transmitindo áudio continuamente para servidores remotos para processamento de Linguagem Natural (NLP) e tomada de decisão. Este modelo apresenta vulnerabilidades críticas: dependência de conectividade constante à internet e, fundamentalmente, riscos severos à privacidade e soberania dos dados do usuário [2].

Além disso, dispositivos como a Alexa apresentam dificuldade em tomar decisões ou acionar mais de um dispositivo. Ela se limita aos comandos pré-definidos pela ”Alexa Skill” do dispositivo e não consegue interpretar quando o usuário ordena a ativação de dois ou mais dispositivos no mesmo comando.

Paralelamente, o campo da Inteligência Artificial testemunhou uma mudança de paradigma com o surgimento de Modelos de Linguagem Pequenos (SLMs). Diferente de seus predecessores massivos (LLMs), os SLMs são otimizados para execução em hardware com recursos limitados, mantendo capacidades surpreendentes de raciocínio e seguimento de instruções. O lançamento de modelos como o Llama 3.2 da Meta, quantizados para execução eficiente em CPUs ARM, abriu a possibilidade de trazer a ”agentes de IA” para a borda [3].

O Projeto Delta nasce na interseção dessas tecnologias. Ele propõe uma solução onde a inteligência não é apenas um comando remoto, mas uma propriedade intrínseca do ambiente local. Ao utilizar um Raspberry Pi 5, o sistema Delta elimina a necessidade de nuvem para suas funções core, processando voz e raciocínio lógico localmente.

Figura 1: Logo

```
[STATUS] Aguardando palavra-chave: 'delta'
[STATUS] Palavra-chave detectada. Fale o comando...
[USER] quem é você e o que você faz
[DELTA] Sou Delta, uma IA residencial brasileira. Minha função é
 controlar e gerenciar o ambiente em casa, garantindo conforto e
 segurança para os ocupantes.
```

Figura 2: Delta se identifica

B. Objetivos do Projeto

O objetivo primário é desenvolver um protótipo funcional de um assistente doméstico offline que demonstre os princípios de *Generative AI at the Edge*. Os objetivos específicos incluem:

1) **Processamento Local:** Implementar reconhecimento de fala (ASR) e inferência de LLM inteiramente no dispositivo.
2) **Controle Contextual:** Utilizar sensores ambientais (temperatura, umidade, pressão) não apenas para monitoramento, mas como vetores de entrada para o raciocínio da IA, permitindo automação baseada em conforto térmico real.
3) **Interoperabilidade de Protocolos:** Desenvolver uma ponte de software entre a saída não estruturada de um modelo de linguagem e os comandos estruturados exigidos pelo protocolo IoT Tuya.
4) **Interface Natural:** Permitir controle por voz fluido e feedback visual intuitivo através de códigos de cores em LED, mitigando a ausência de interface gráfica tradicional.

C. Escopo e Estrutura do Documento

Este relatório abrange desde a fundamentação teórica até a implementação prática. A Seção 2 revisa a literatura sobre Edge AI e protocolos IoT. A Seção 3 detalha a arquitetura de hardware e a seleção de componentes. A Seção 4 explora a arquitetura de software, focando na integração VOSK-Ollama. A Seção 5 aprofunda-se na lógica de controle e automação. Finalmente, as Seções 6 e 7 apresentam a avaliação de desempenho e conclusões sobre a viabilidade do sistema.

II. Fundamentação Teórica e Estado da Arte

A. Inteligência Artificial na Borda (Edge AI)

A computação de borda refere-se ao processamento de dados próximo à fonte de sua geração, em vez de enviá-los para data centers centralizados. A aplicação de algoritmos de IA neste contexto, denominada Edge AI, enfrenta desafios de restrição de energia, latência de processamento capacidade térmica e memória.

B. Small Language Models (SLMs) e Quantização

Para viabilizar a execução de modelos generativos em dispositivos como o Raspberry Pi 5 (8GB RAM), a quantização é essencial. O projeto utiliza o modelo Llama 3.2 com 3 bilhões de parâmetros. Em sua precisão original (float16), este modelo exigiria cerca de 6GB de VRAM apenas para os pesos, sem contar o contexto KV cache. A técnica de quantização utilizada, com precisão de 4-bits, reduz a representação de cada peso para 4 bits. Estudos indicam que a quantização Q4 mantém a maior parte da coerência semântica e capacidade de raciocínio do modelo original, enquanto reduz o consumo de memória para menos de 2.5GB, tornando-o viável para a arquitetura de memória unificada do Raspberry Pi 5 [5].

C. Reconhecimento Automático de Fala (ASR) Offline

A interface primária do Delta é a voz. Sistemas ASR modernos baseados em Transformers (como o Whisper da OpenAI) oferecem precisão excepcional, mas frequentemente exigem aceleração por GPU para latência em tempo real. Para este projeto, optou-se pelo VOSK. O VOSK é baseado no toolkit Kaldi e utiliza modelos acústicos baseados em Redes Neurais (DNN) combinados com modelos de linguagem baseados em n-gramas ou grafos de decodificação finitos. Esta arquitetura é significativamente mais leve que os modelos end-to-end baseados em attention, permitindo execução em tempo real na CPU do Raspberry Pi com latência mínima e baixo overhead computacional, crucial para deixar recursos livres para o SLM e não forçar as capacidades do processador [7].

D. Heurística de Detecção de Atividade de Voz (VAD)

Diferente de abordagens que delegam a detecção de silêncio inteiramente ao motor de reconhecimento, o sistema Delta implementa um pré-processamento de sinal baseado em energia (RMS - Root Mean Square). Utilizando a biblioteca nativa `audioop`, o fluxo de áudio PCM de 16-bit é analisado em janelas de 4096 bytes. O sistema mantém um estado de escuta ativa apenas quando a amplitude do sinal supera um limiar empírico (*threshold*) de 300 unidades. Esta abordagem híbrida, que combina a máquina de estados do VOSK para detecção de palavras-chave (*keyword spotting*) com uma verificação de energia bruta, reduz significativamente o consumo de CPU em estados de repouso e previne o processamento desnecessário de ruídos ambientais estacionários, otimizando o ciclo de vida da bateria e a temperatura do processador.

E. Protocolos de IoT e Controle Local: O Ecossistema Tuya

A plataforma Tuya é onipresente em dispositivos de casa inteligente, utilizando o protocolo MQTT sobre TCP/IP. No entanto, sua implementação padrão é "cloud-first". Para garantir a premissa de operação offline e privada do Delta, utiliza-se o controle local via *Local Keys*. Cada dispositivo Tuya possui um `device_id` (público) e uma `local_key` (privada, 16 caracteres). Com essa chave, é possível criptografar payloads JSON contendo comandos de Data Point System (DPS) e enviá-los diretamente ao IP do dispositivo na rede local (LAN), contornando completamente os servidores da Tuya. Isso reduz a latência de comando de ≈500ms (nuvem) para <50ms (local) e elimina a dependência de internet [9].

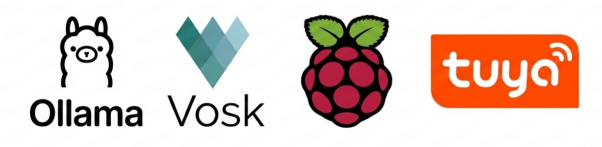

Figura 3: Sistemas que serão integrados

III. Arquitetura de Hardware e Integração de Sensores

A. A Plataforma Computacional: Raspberry Pi 5

O coração do sistema Delta é o Raspberry Pi 5. Esta escolha é estratégica devido à sua arquitetura atualizada:

- **SoC BCM2712:** Processador quad-core Arm Cortex-A76 de 64 bits rodando a 2.4GHz. O Cortex-A76 oferece instruções vetoriais aprimoradas e uma pipeline superscalar mais larga, resultando em um desempenho de inferência 2 a 3 vezes superior ao Pi 4.
- **Memória LPDDR4X-4267:** A largura de banda de memória é o principal gargalo para LLMs. A memória mais rápida do Pi 5 é crucial para alimentar os tensores do modelo Llama 3.2 durante a geração de tokens.
- **I/O Controller RP1:** O chip dedicado de I/O melhora a estabilidade dos barramentos I2C e GPIO, essencial para a leitura confiável dos múltiplos sensores ambientais do projeto.

B. Feedback de Interface Assíncrono (Multithreading)

Dada a latência inerente à inferência de modelos baseados em Transformers em CPUs ARM, a experiência do usuário (UX) torna-se um desafio crítico. Para mitigar a percepção de latência durante o processamento do Llama 3.2 (que pode durar entre 4 a 15 segundos), o sistema implementa o controle de feedback visual em uma *thread*

dedicada (`threading.Thread`), desacoplada do loop principal de processamento de áudio e inferência. A classe `GerenciadorLED` gerencia transições de estado (Ouvindo, Processando, Respondendo) através de padrões de cores RGB e efeitos como o ”Rainbow Cycle”. Esta arquitetura paralela assegura que o feedback visual permaneça fluido e responsivo, indicando atividade de processamento contínua mesmo quando o núcleo da aplicação está bloqueado aguardando a tokenização e geração de resposta do SLM, prevenindo a percepção de travamento do sistema pelo usuário.

C. Subsistema de Captura de Áudio (Microfone USB)

Para a interface de entrada de voz, o sistema utiliza um microfone com conexão USB. Esta escolha de design simplifica a arquitetura de hardware ao eliminar a necessidade de *HATs* de áudio ou conversores externos, visto que o Raspberry Pi 5 não possui entradas analógicas nativas para microfone. A solução apresenta as seguintes características técnicas:

- **Conversão AD Integrada:** O dispositivo USB possui seu próprio conversor Analógico-Digital (ADC) interno. Isso isola o sinal de áudio de ruídos eletromagnéticos que poderiam ser induzidos nas trilhas da PCB do Raspberry Pi, garantindo uma relação sinal-ruído (SNR) adequada para o pré-processamento.
- **Compatibilidade UAC (USB Audio Class):** O periférico opera como um dispositivo *plug-and-play* compatível com o driver ALSA (*Advanced Linux Sound Architecture*) do kernel Linux. Isso permite o acesso direto ao *stream* de áudio PCM sem a necessidade de drivers proprietários.
- **Taxa de Amostragem (Sampling Rate):** O sistema foi configurado para capturar áudio em canal mono a 16.000 Hz (16kHz). Esta taxa é escolhida propositalmente para otimizar o desempenho dos modelos de *Speech-to-Text* (como Whisper ou VOSK), que são tipicamente treinados nesta frequência, reduzindo a necessidade de *resampling* via software e poupando ciclos de CPU.

D. Matriz de Sensoriamento Ambiental

O Raspberry Pi 5 foi escolhido pelo seu processador Cortex-A76 e I/O aprimorado. O projeto emprega fusão de sensores para garantir precisão. Três sensores distintos são utilizados, conforme detalhado na Tabela I.

Tabela I: Matriz de Sensores Utilizada

Sensor	Protocolo	Variáveis	Função
DHT22	Digital	Temp, Umid	Backup de baixo custo.
AHT20	I2C (0x38)	Temp, Umid	Alta precisão, rápido.
BMP280	I2C (0x76)	Temp, Pressão	Barômetro de precisão.

A decisão de usar três sensores permite ao sistema calcular médias ponderadas e descartar valores espúrios (outliers).

E. Interface de Feedback Visual

Dada a natureza ”headless” do assistente, um LED RGB é conectado via GPIO para fornecer feedback de estado:

- **Vermelho:** Estado de espera (Idle), ouvindo a Keyword ”Delta”.
- **Verde:** Ativo, gravando comando de voz do usuário.
- **Ciclo RGB (Rainbow):** Processamento cognitivo (Inferência do SLM).
- **Azul:** Sucesso na execução do comando ou resposta pronta.

F. Conexão de componentes

A conexão dos componentes pode ser visualizada no apêndice (Tabela IV) e também através das figuras 4 e 5.

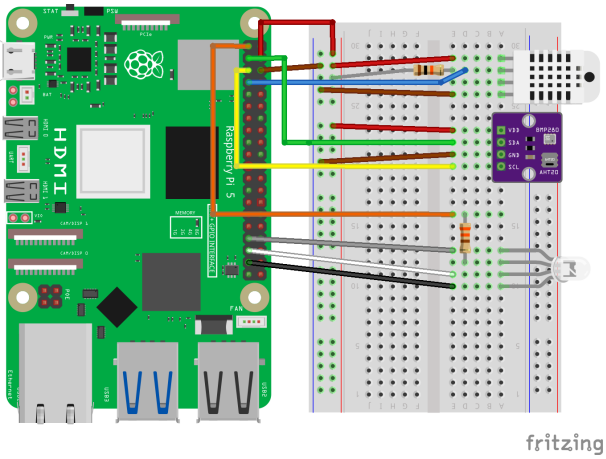

Figura 4: Sketch da conexão dos sensores e LED à Raspberry

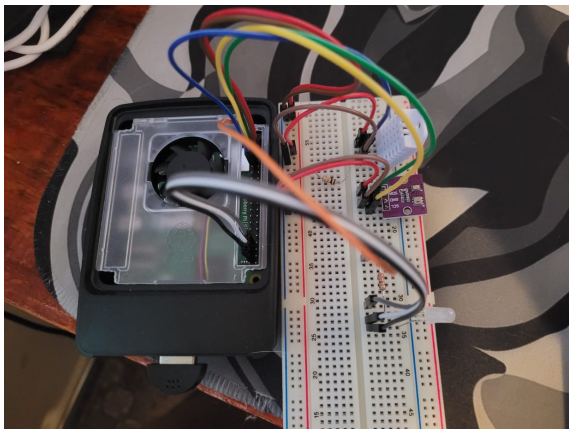

Figura 5: Conexão realizada com base no sketch

IV. Engenharia de Software: O Pipeline Cognitivo

A arquitetura de software é construída em Python, operando em um loop de eventos contínuo.

A. Aquisição de Voz e VAD

A captura é feita via PyAudio (16kHz, mono). O sistema implementa um VAD (Voice Activity Detection) baseado em energia RMS. Se o valor exceder o LIMIAR_RUIDO (300) e o silêncio persistir por mais de 2.0 segundos, o sistema assume o fim do comando.

B. Integração VOSK

1) **Keyword Spotting (KWS):** No estado de repouso, busca apenas a palavra "delta".
2) **Full Transcription:** Após a detecção, transcreve a fala completa até ouvir 2 segundos de silêncio.

C. Camada de Inteligência: Ollama e Prompt Engineering

O sistema constrói um Prompt complexo que inclui:

- **System Prompt:** Define a "persona" do Delta.
- **Contexto Sensorial:** Injeta dados dos sensores (ex: "Temperatura interna: 24.5°C").
- **Definição de Ferramentas:** Fornece o esquema JSON das funções disponíveis (`set_ac_state`, etc.) para habilitar o *Function Calling*.

O *System Prompt* foi desenhado para forçar a saída estrita em JSON, utilizando a técnica de *One-Shot Learning* no preâmbulo: "Você é Delta, uma home assistant... Você deve retornar somente um JSON válido de...".

D. Módulo de Atuação: Function Calling

O Llama 3.2 gera uma estrutura JSON representando a intenção (ex: `{"name": "set_ac_state", "parameters": {...}}`). O script `main.py` intercepta esse JSON e invoca a função em `device_tools.py`, que traduz para comandos Tuya.

E. Injeção Contextual Dinâmica e Fusão de Dados

Para mitigar a falta de "consciência física" típica de modelos de linguagem pré-treinados, o sistema emprega uma técnica de Injeção de Contexto em Tempo Real (*Real-Time Context Injection*). Antes de cada inferência, a função `ler_sensores()` agrega dados dos transdutores DHT22, AHT20 e BMP280, calculando médias ponderadas e formatando um bloco de texto estruturado contendo temperatura, umidade e pressão atmosférica. Este bloco é inserido dinamicamente no *System Prompt* sob a tag `[DADOS REAIS]`, permitindo que o modelo Llama 3.2 utilize raciocínio dedutivo sobre o estado atual do ambiente. Isso transforma a interação de um simples comando-resposta para uma análise contextual, onde a IA pode recusar um comando de "esfriar o quarto" se os sensores indicarem que a temperatura interna já se encontra abaixo do limiar de conforto (ex: $< 18°C$), demonstrando um comportamento pseudo-cognitivo de preservação de energia.

F. Middleware de Validação e Segurança Determinística

Uma camada crítica de abstração foi implementada entre a saída estocástica do Modelo de Linguagem (SLM) e os atuadores físicos. O módulo `device_tools.py` atua como um *middleware* de validação rigorosa. Dado que modelos generativos podem produzir alucinações ou parâmetros fora dos limites operacionais seguros (ex: temperatura alvo de 5°C ou 40°C), as funções de controle implementam lógica de *clamping* e verificação de tipos. No controle do condicionador de ar, por exemplo, qualquer solicitação de temperatura é matematicamente restringida ao intervalo operacional seguro $[16°C \ a \ 30°C]$ antes da conversão para o protocolo DPS. Adicionalmente, modos de operação inválidos gerados pelo modelo são interceptados e descartados ou substituídos por valores padrão (*safe defaults*). Esta arquitetura garante que, independentemente da criatividade ou erro da IA, o hardware físico jamais receba instruções danosas ou não suportadas pelo protocolo Tuya.

V. Lógica de Controle e Automação Contextual

A. Gerenciamento Climático Inteligente

O sistema implementa lógicas como a decisão entre AC e Ventilador baseada no delta de temperatura ambiente e um "Modo Automático" que ativa o AC em 23°C se a temperatura ambiente superar 26°C ou o ventilador se estiver abaixo de 26°C.

B. Automação de Iluminação

Além do controle de temperatura, também é implementado o controle de uma lâmpada inteligente, definindo cor com base no periodo diurno ou noturno, para conforto ocular.

- **Modo Diurno (06:00 - 18:00):** Branco frio, 100% intensidade.
- **Modo Noite (18:00 - 06:00):** Branco quente (Amarelo), média intensidade.

C. Mapeamento de Intenções (DPS Map)

A tabela abaixo mostra o mapeamento entre funções lógicas e IDs do protocolo Tuya:

Tabela II: Mapeamento de Data Points (DPS)

Disp.	Função	ID	Lógica/Valores
Ar Cond.	Switch	”1”	True/False
	Temp	”2”	16-30 (Valor × 10)
	Modo	”4”	”cold”, ”wet”, ”auto”
	Vento	”5”	”auto”, ”low”, ”mid”, ”high”
Interruptor	Ventilador	”1”	True/False
	Velocidade	”3”	1-5 (5 níveis)
	Lâmpada	”5”	True/False
Lâmpada	Brilho	”22”	10-1000 (1%-100%)
	Cor	”23”	0 (Quente)-1000 (Frio)

D. Fluxograma de operação

O fluxograma de operação pode ser visualizado nos Apêndices ou figura 12

VI. Avaliação de Desempenho

A. Latência

A latência total varia de prompt para prompt, dependendo do tamanho da frase a ser enviada e principalmente do tamanho final do prompt com a adição das intruções fixas, que impacta no tempo total de processamento da **SLM**. Prompts menores favorecem o tempo de processamento e invocam resultados melhores, com a latência total variando entre 5 a 10 segundos, sem invocar function callings. Prompts maiores aumentam significativamente o tempo de processamento e tendem a causar alucinação na **SLM** podendo ter latência total de mais de 60 segundos, principalmente se a resposta for complexa ou executar diversos *function callings*. O detalhamento por etapa pode ser visto na Tabela III.

Tabela III: Perfil de Latência (RPi 5 - 8GB)

Etapa	Tempo	Fatores
Wake Word	$<$ 300ms	Ruído de fundo.
Transcrição	≈0.5x Áudio	Tamanho da frase.
Inferência	5s - 100s	Contexto/Prompt.
Execução	$<$ 50ms	Rede Wi-Fi.
Total	**10.0s - 120.0s**	Complexidade.

B. Exemplos de comandos executados

Para comandos que dependem somente da **SLM**, sem executar function calling, a latência se prova muito menor, como na figura 6 que apresenta latência total de 13,68s.

```
[STATUS] Aguardando palavra-chave: 'delta'
[STATUS] Palavra-chave detectada. Aguardando comando...
[USER] quem é você o que você faz
[DELTA] Sou Delta, uma IA residencial brasileira. Minha função é control
ar e gerenciar o ambiente em casa, garantindo conforto e segurança para
os ocupantes.

======================================================================
METRICAS DE LATENCIA
======================================================================
Captura de voz (keyword -> silencio):   5139.3 ms
Preparacao de prompt:                      0.0 ms
Processamento SLM:                      8536.9 ms
Geracao de resposta:                       0.0 ms
----------------------------------------------------------------------
LATENCIA TOTAL (keyword -> resposta):  13676.2 ms
Tempo total: 13.68 segundos
======================================================================
```

Figura 6: Prompt que invoca somente a **SLM**.

Para comandos de luz, o sistema de mostra robusto, designando funções de dia e noite como descritos nas funções. A figura 7 mostra um exemplo de definir a luz para modo ”dia” com latência total de 101,30s.

```
[STATUS] Aguardando palavra-chave: 'delta'
[STATUS] Palavra-chave detectada. Aguardando comando...
[USER] agora meio dia ajuste a luz
[DELTA][LAMP_RGB] {'power': True, 'mode': 'dia', 'brightness': 1000, '
temperature': 1000}
[DELTA] Lampada RGB ligada modo dia.

======================================================================
METRICAS DE LATENCIA
======================================================================
Captura de voz (keyword -> silencio):   6564.0 ms
Preparacao de prompt:                    555.6 ms
Processamento SLM:                     85121.8 ms
Execucao de ferramentas:                9054.5 ms
Geracao de resposta:                    9059.5 ms
----------------------------------------------------------------------
LATENCIA TOTAL (keyword -> resposta):  101301.0 ms
Tempo total: 101.30 segundos
======================================================================
```

Figura 7: Prompt que invoca a **SLM** e function calling da lâmpada.

Para comandos de ventilador, o sistema possui liberdade para definir a velocidade do ventilador. As figuras 8 e 9 mostram exemplos de comandos para ligar e desligar o ventilador com diferentes latências totais.

```
[STATUS] Aguardando palavra-chave: 'delta'
[STATUS] Palavra-chave detectada. Aguardando comando...
[USER] ligue o ventilador na velocidade dois
[DELTA][FAN] {'power': True, 'speed': 'level_2'}
[DELTA] Ventilador ligado velocidade 2 (ambiente: 27.6C).

======================================================================
METRICAS DE LATENCIA
======================================================================
Captura de voz (keyword -> silencio):   7491.6 ms
Preparacao de prompt:                    556.2 ms
Processamento SLM:                     22261.8 ms
Execucao de ferramentas:                1285.2 ms
Geracao de resposta:                    1289.7 ms
----------------------------------------------------------------------
LATENCIA TOTAL (keyword -> resposta):  31599.4 ms
Tempo total: 31.60 segundos
======================================================================
```

Figura 8: Prompt que invoca a **SLM** e function calling do ventilador.

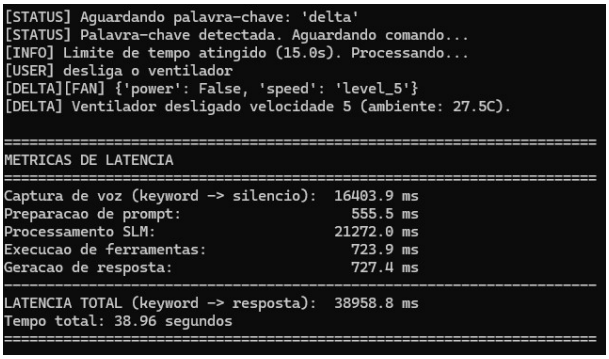

Figura 9: Prompt que invoca a **SLM** e function calling do ventilador.

Comandos complexos que requerem o valor de sensores indicam bons resultados, como por exemplo ”ajuste o clima”, que primeiro obtém o valores dos sensores e define a partir dele qual ação tomar, se irá ligar o ar condicionado ou o ventilador. A figura 10 exemplifica uma dessas execuções com latência total de 29,57s.

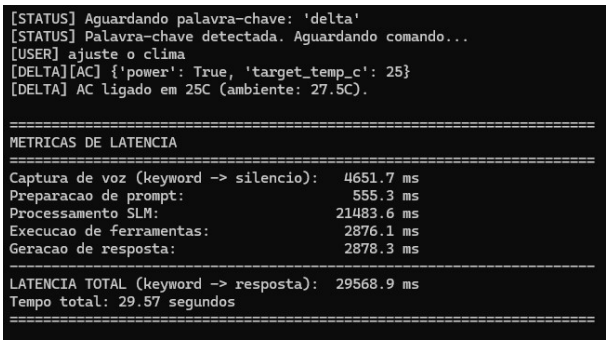

Figura 10: Prompt que invoca a **SLM** e function calling do ar condicionado.

C. Confiabilidade e Robustez

Curiosamente, e de certa forma característimo à **SLMs**, o sistema apresenta uma robustez à entendimentos errados pela **ASR**, como na figura 11, onde a mensagem verdadeira era ”ligue o ar condicionado em vinte e um graus”, que foi entendida como ”acondicionado em vinte e um grãos”, mas, ainda assim, foi processada de maneira correta pela **SLM**.

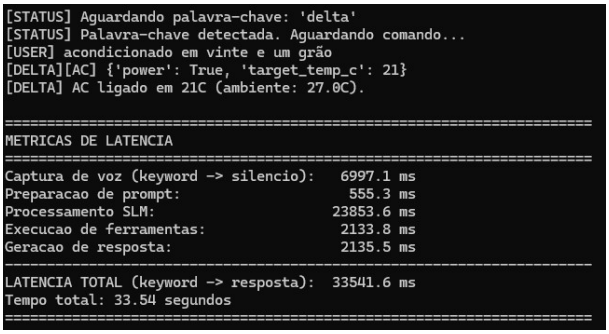

Figura 11: Prompt mostrando a robustez da **SLM**.

VII. Discussão, Desafios e Futuro

- **Mitigação de Alucinação em Function Calling:** Devido a natureza dos Modelos de Linguagem Pequenos, a geração de estruturas JSON podem ser malformadas ou a pode ser gerado parâmetros operacionais inseguros (alucinações), como temperaturas alvo fora da escala física do equipamento. Para solucionar isso, o sistema não aciona os atuadores diretamente via LLM mas através de uma camada intermediária de software (*middleware*) em Python (`device_tools.py`) que intercepta a saída da IA. Este módulo aplica restrição de valores, assegurando que apenas comandos dentro dos limites operacionais seguros (ex: 16°C a 30°C para o AC) sejam transmitidos ao hardware, garantindo segurança independente da ”criatividade”do modelo.
- **Privacidade e Soberania de Dados:** Diferentemente das arquiteturas *cloud-first* comerciais, o Delta valida a tese de operação autônoma na borda. O sistema garante privacidade total ao processar o reconhecimento de voz (VOSK) e a inferência cognitiva (Llama 3.2) inteiramente na CPU do Raspberry Pi 5, sem envio de áudio para servidores externos. Adicionalmente, o controle de IoT utiliza o protocolo Tuya com *Local Keys*, permitindo o envio de payloads criptografados diretamente aos endereços IP dos dispositivos na rede local (LAN). Isso elimina a dependência de conectividade com a internet e impede a mineração de dados de comportamento do usuário por terceiros.

Além disso, cinco melhorias futuras que podem ser implementadas no projeto são:

1) Melhorar a qualidade do microfone: o microfone usado possui baixa qualidade, resultando em captação de áudio em baixo volume e pouco ou nenhum filtro de ruídos.
2) Expansão da lista de palavras para ativação das tools: oferece maior alcance para a operação e suporte a comandos mais variados e naturais.
3) Resposta por voz: integração de um módulo de síntese de voz (TTS), para que seja possível ouvir

a resposta gerada pela SLM através de um alto-falante.

4) Interface Visual: integração de um display para mostrar dados pertinentes, como temperatura da Raspberry, temperatura ambiente, dispositivos disponíveis, histórico de comandos e execuções, mensagem visual e também desenhos, ícones e símbolos para interagir com o usuário, imitando um assistente amigável.
5) Adição de NPU: pode reduzir drasticamente o tempo de inferência em troca de maior preço do projeto.

VIII. Conclusão

O Projeto Delta demonstrou a viabilidade técnica de um assistente doméstico baseado em Edge AI, validando a premissa de que a inteligência artificial generativa pode ser executada em hardware acessível e de baixo consumo. A combinação do Raspberry Pi 5 com o modelo Llama 3.2 e o motor VOSK criou um ecossistema funcional, privado e responsivo. Mais do que apenas executar comandos, o sistema provou ser capaz de realizar raciocínio contextual através da fusão de dados de sensores, permitindo decisões autônomas de climatização e iluminação que priorizam o conforto real do usuário.

A arquitetura descentralizada validou a tese de *Privacy by Design*, assegurando que dados sensíveis de voz e rotina permaneçam confinados à rede local, eliminando a dependência de nuvem característica das soluções comerciais atuais. Embora a latência de inferência da CPU ainda represente um desafio em comparação ao processamento em nuvem, a implementação de *Function Calling* local e controle via protocolo Tuya direto (LAN) garantiu uma execução de comandos físicos ágil e robusta.

Apêndice

A. Pinout de referência

Tabela IV: Pinagem (Pinout) do Sistema

Comp.	Pino	GPIO	Função
VCC SENSORES	4	-	Alimentação
GND	6	-	Alimentação
I2C (SDA)	3	2	AHT20/BMP280 (Dados)
I2C (SCL)	5	3	AHT20/BMP280 (Clock)
DHT22	7	4	Dados One-Wire
VCC LED	1	-	Alimentação
LED RGB (R)	33	13	Vermelho
LED RGB (G)	35	19	Verde
LED RGB (B)	37	26	Azul

B. Repositório

O código-fonte completo com links para o dataset e vídeos está disponível no repositório: GitHub.

C. Créditos

Os códigos utilizados no projeto bem como este texto e a figura 1 foram parcialmente gerados pelas inteligências artificiais Gemini e Claude.

Referências

[1] Rovai, Marcelo. Edge ML Made Easy: A Comprehensive Guide to Machine Learning on Edge Devices. 2025. Disponível em: https://mjrovai.github.io/EdgeML_Made_Ease_ebook/

[2] Hernandez Acosta, et al. *A Survey on Privacy Issues and Solutions for Voice-controlled Digital Assistants*. 2022. Disponível em: https://www.uni-goettingen.de/en/655470.html

[3] *Applications of Small Language Models in Edge Computing and IoT-Based Autonomous Systems*. ResearchGate, 2025. Disponível em: https://www.researchgate.net/publication/397638558_Applications_of_Small_Language_Models_in_Edge_Computing_and_IoT-Based_Autonomous_Systems

[4] IBM. *Meta's Llama 3.2 models now available on watsonx*. 2025. Disponível em: https://www.ibm.com/think/news/meta-llama-3-2-models

[5] *Review of Llama3.2 vs Llama3.1: Performance on LattePanda Mu*. 2025. Disponível em: https://www.lattepanda.com/blog-323232.html

[6] *Optimizing LLMs Using Quantization For Mobile Execution*. arXiv, 2025. Disponível em: https://www.arxiv.org/abs/2512.06490

[7] *Performance Evaluation of Offline Speech Recognition on Edge Devices*. MDPI, 2021. Disponível em: https://pdfs.semanticscholar.org/f06d/0182f779f496416e47625f4546d0002d2040.pdf

[8] Alpha Cephei. *VOSK Offline Speech Recognition API*. Disponível em: https://alphacephei.com/vosk/

[9] Jason A. Cox. *tinytuya: Python API for Tuya WiFi smart devices*. GitHub, 2025. Disponível em: https://github.com/jasonacox/tinytuya

[10] Tuya Help Center. *How does Tuya IoT PaaS ensure data security?*. 2025. Disponível em: https://support.tuya.com/en/help/_detail/K8sdy0leoncqi

[11] *DHT22 sensor on Raspberry Pi with WiringPi*. GitHub, 2025. Disponível em: https://randomnerdtutorials.com/raspberry-pi-dht11-dht22-python/

[12] *Less is More: Optimizing Function Calling for LLM Execution on Edge Devices*. arXiv, 2025. Disponível em: https://arxiv.org/html/2411.15399v1

[13] *Echoes of Privacy: Uncovering the Profiling Practices of Voice Assistants*. PoPETs, 2025. Disponível em: https://petsymposium.org/popets/2025/popets-2025-0050.pdf

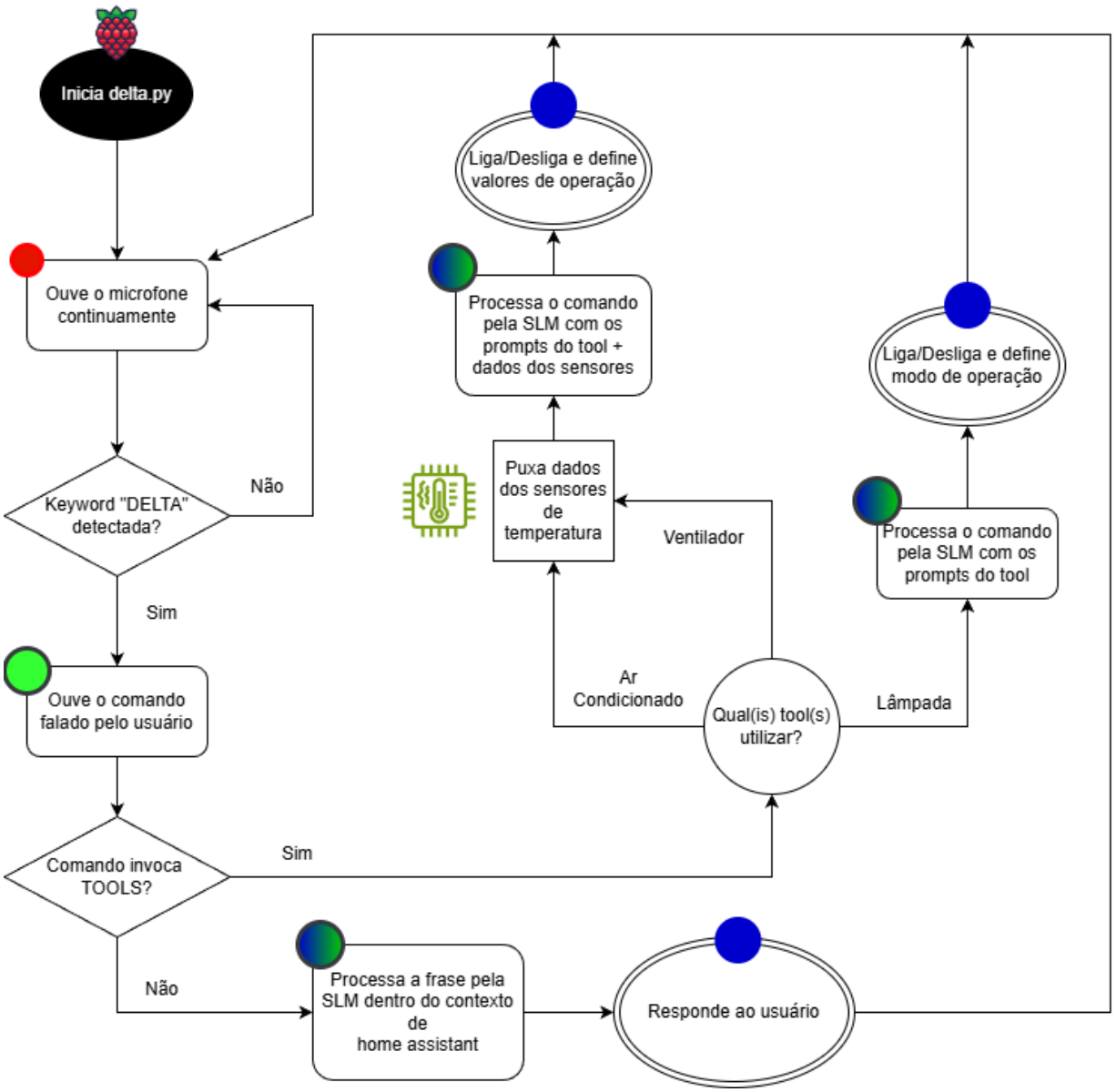

Figura 12: Diagrama de operação do sistema Delta