Um modelo de linguagem rodando no ESP32-S3

Inferência de um transformer dentro do microcontrolador: por que a parede é a memória, onde o tempo é gasto e como reproduzir.

Publicado em	iiot.esp32.tec.br/tutoriais/llama2-c-no-esp32-s3
Data	27 de agosto de 2026
Nível	Avançado
Framework	ESP-IDF (não usa Arduino nem PlatformIO)

Aviso importante: os números de desempenho citados neste manual são dos autores dos projetos de origem, e não foram medidos pelo portal. O que você medir na sua placa é o número que vale para você.

1. O que este material é, e o que não é

Aqui um transformer de verdade roda inferência dentro de um ESP32-S3: sem nuvem, sem Wi-Fi, sem servidor do outro lado. Antes de qualquer comando, o aviso que evita frustração: o modelo que cabe no chip não é um assistente.

Os pesos usados vêm do conjunto TinyStories, histórias curtas em inglês com vocabulário reduzido, criadas para pesquisar quão pequeno um modelo pode ser e ainda produzir texto coerente. Ele não responde sobre a sua planta, não lê manual e não substitui o modelo que você usaria na camada de supervisório.

O que ele prova é outra coisa, e essa vale: o silício que já está no seu painel aguenta a matemática de um transformer.

Dois pontos de referência publicados

Projeto	Modelo	Placa	Desempenho
esp32-llm (2024) Dave Bennett	260 mil parâmetros TinyStories	ESP32-S3FH4R2 2 MB PSRAM	19,13 tokens/s
esp32-ai (2026) slvDev	28,9 milhões 14,9 MB em 4 bits	ESP32-S3 8 MB PSRAM, 16 MB flash	9,88 tokens/s 94,9 ms por token

2. A parede é a memória, não o processador

A armadilha está na compra da placa, e pega quase todo mundo na primeira tentativa. PSRAM não é opcional: mesmo o modelo de 260 mil parâmetros precisa de cerca de 1 MB de RAM, e o ESP32-S3 tem 512 KB de SRAM interna.

Boa parte dos anúncios genéricos de ESP32-S3 DevKit não informa quanto de PSRAM traz, ou não traz nenhuma. **Se o vendedor não informa, considere que não tem.** Essa é a causa mais comum de o projeto não subir.

O ESP32 clássico e o ESP32-C3 ficam de fora deste caminho. O C3 tem um único núcleo, e as rotinas vetorizadas que dão a velocidade são específicas do S3.

3. A hierarquia que faz 28,9 milhões de parâmetros caberem

O salto de 260 mil para 28,9 milhões não veio de um chip melhor. Veio de decidir onde cada parte do modelo mora.

Memória	Tamanho	O que guarda	Frequência de leitura
SRAM	512 KB	ativações e pesos de normalização	muitas vezes por token
PSRAM	8 MB	núcleo do modelo e cabeça de saída	uma vez por posição
FLASH	16 MB	tabela de 25 milhões de parâmetros	450 bytes por token

O truque se chama **per-layer embeddings**, emprestado da família Gemma. A tabela de embeddings costuma ser o maior bloco único de um modelo, e é justamente ela que vai para a flash. A cada token, apenas cerca de 450 bytes são buscados de lá. A flash é lenta, mas 450 bytes por vez não pesam.

Vale registrar a inversão, porque ela ensina algo além deste caso: a peça maior foi para a memória mais lenta e funcionou, porque o que importa não é o tamanho do bloco, e sim quantas vezes ele é lido.

4. Onde o tempo é gasto

Num transformer, a maior parte do tempo é multiplicação de matriz. O llama2.c original faz isso com um laço escalar, que é portátil e lento.

No ESP32-S3 existe um caminho mais rápido: a biblioteca ESP-DSP traz rotinas de produto escalar escritas para as instruções SIMD do núcleo LX7. A função usada é **dsps_dotprod_f32_aes3**, e o sufixo aes3 indica exatamente a variante do S3. É ela que dá a velocidade, e é por isso que o C3 não chega ao mesmo resultado.

Núcleo da função, no arquivo main/llm.c de DaveBben/esp32-llm:

```
void matmul(v4sf *xout, v4sf *x, v4sf *w, int n, int d)
{
    *matmul_params = (MatMulTaskParams){xout, x, w,
                        d / 2, d, n, d, TASK_1_BIT};
    xSemaphoreGive(semaDataReady);
    for (int i = 0; i < d / 2; i++)
    {
        v4sf val = 0.0f;
        v4sf *row = &w[i * n];
        dsps_dotprod_f32_aes3(row, x, &val, n);
        xout[i] = val;
    }
```

Os dois núcleos, na prática

Repare no que a função faz antes de começar a contar: preenche **matmul_params** e libera um semáforo, e só então entra no laço. Esse semáforo acorda uma tarefa fixada no outro núcleo, que processa a metade de baixo das linhas enquanto o núcleo chamador processa a metade de cima. O corte é literal: **d** é o número de linhas, e cada lado fica com **d / 2**.

Meça, não confie na impressão

O próprio port traz a conta pronta ao fim da geração:

```
float tks = (pos - 1) / (double)(end - start) * 1000;
fprintf(stderr, "achieved tok/s: %f", tks);
```

Anote o número da sua placa antes de mexer em qualquer configuração: sem a medida de antes, não há como saber se a mudança seguinte ajudou ou atrapalhou.

5. Passo a passo

O caminho abaixo é o dos repositórios originais. Os dois usam ESP-IDF, e não Arduino nem PlatformIO — a estrutura de projeto é outra.

Passo 1 — Confirme a placa antes de comprar

Procure a especificação de PSRAM no anúncio, não apenas o nome do chip. Para o modelo de 260 mil parâmetros, o mínimo é 2 MB de PSRAM. Para o de 28,9 milhões, são 8 MB de PSRAM e 16 MB de flash.

Passo 2 — Instale o ESP-IDF

Siga o guia oficial da Espressif para o seu sistema e confirme que o ambiente está ativo antes de seguir. Todo o restante depende disso funcionando.

Passo 3 — Compile

Com o repositório clonado e o ambiente carregado, a compilação é a padrão da ferramenta: idf.py build. A primeira demora, porque baixa e monta as dependências, incluindo a ESP-DSP.

Passo 4 — Grave e acompanhe

Troque a porta pela da sua placa: idf.py -p /dev/ttyUSB0 flash monitor. No Windows a porta é uma COM. O que você deve ver: o monitor abre, a placa reinicia e o texto começa a sair token a token, em inglês, no estilo de história infantil. Ao final aparece a linha com o valor de tokens por segundo.

Passo 5 — Quando não couber

Falha de alocação no boot, ou erro de memória ao carregar o modelo, aponta quase sempre para o mesmo lugar: PSRAM desabilitada no menuconfig, ou placa sem PSRAM. Confira também o tamanho da partição: para o modelo de 14,9 MB, a tabela padrão não dá conta.

6. Solução de problemas

Sintoma	Causa provável	O que fazer
Erro de alocação no boot	PSRAM desabilitada ou ausente	Verifique o menuconfig e a especificação da placa
Texto sai muito devagar	CPU ou PSRAM fora da frequência	CPU em 240 MHz, PSRAM em 80 MHz, cache de instruções ampliado
Não cabe na flash	Tabela de partições padrão	Amplie a partição do firmware para o modelo de 14,9 MB
Compilação falha na ESP-DSP	Dependência não baixada	Refaça o idf.py build com a rede disponível

7. Créditos e origem do código

O código apresentado neste manual não é do portal. Ele vem dos projetos abaixo, e o mérito é de seus autores:

Projeto	Autor	Endereço
esp32-llm	Dave Bennett	github.com/DaveBben/esp32-llm
esp32-ai	slvDev	github.com/slvDev/esp32-ai
llama2.c	Andrej Karpathy	github.com/karpathy/llama2.c

Consulte a licença de cada repositório antes de reutilizar o código em produto. Este manual é material didático do portal iiot.esp32.tec.br e descreve o trabalho publicado por terceiros, com atribuição.