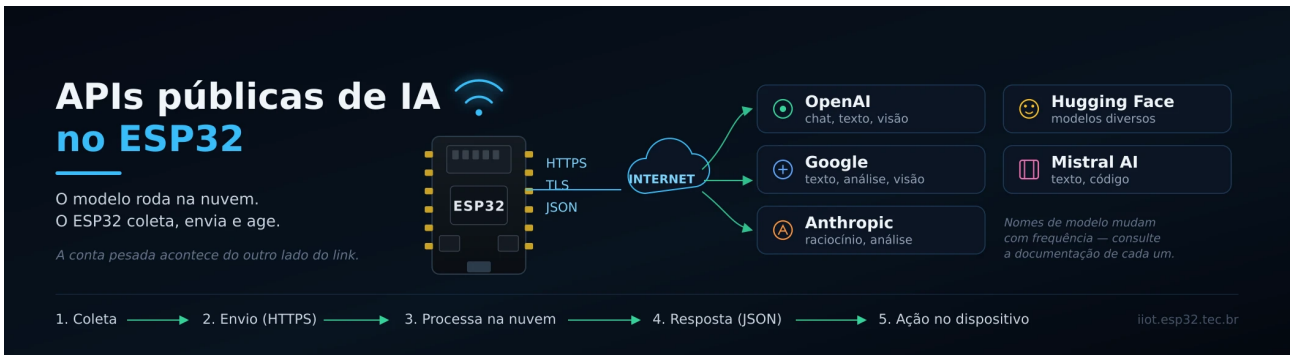
ESP32 e modelos de IA

Nuvem ou gateway na sua rede - manual prático



Placa	ESP32 ou ESP32-S3; para HTTPS, prefira uma com PSRAM
Framework	Arduino Core, via PlatformIO
Biblioteca	ArduinoJson (única dependência externa)
Caminhos	LOCAL por HTTP simples, ou NUVEM por HTTPS
Nível	Intermediário
Artigo online	iiot.esp32.tec.br/tutoriais/usar-ia-da-internet-no-esp32-como-funciona

Firmware escrito para o artigo. **Não foi validado em bancada** contra nenhum fornecedor real nem contra uma instalação do Ollama: os formatos de requisição e resposta seguem o que está documentado, mas variam entre fornecedores e entre versões. Confira a documentação da API que você for usar.

1. O modelo não roda no ESP32

Este é o ponto de partida, e a confusão mais comum. O modelo roda do outro lado - num servidor da internet ou numa máquina da sua rede. Ferramentas como o Ollama baixam o modelo para uma máquina sua e executam localmente: num PC, num mini-servidor ou num gateway. **Nunca no ESP32.**

O papel da placa não muda

O que o ESP32 faz bem	Por que o servidor não substitui
Ler sensor com temporização confiável	latência de rede destroi a periodicidade
Reagir na hora ao que é crítico	esperar resposta de rede seria inaceitável
Publicar o dado para quem analisa	alguém precisa estar na ponta, no campo
Ficar de pe por meses com consumo baixo	servidor não vai dentro do painel

Por isso a consulta neste firmware é **não bloqueante por construção**: a leitura do sensor e a reação local continuam rodando enquanto a resposta não chega, e continuam funcionando se o link cair.

2. Os dois caminhos

	LOCAL	NUVEM
Protocolo	HTTP simples	HTTPS com TLS
Custo de RAM	baixo	dezenas de KB no handshake
Latência	poucos milissegundos	dezenas a centenas de ms
Custo por consulta	nenhum	cobrado pelo fornecedor
Depende de	sua rede	link externo de pe
Privacidade	o dado não sai	o dado vai para terceiro

Comece pelo LOCAL. Sem custo, sem TLS, e você valida a lógica antes de gastar.

O custo escondido do HTTPS

O handshake TLS consome dezenas de KB de RAM durante a negociação. Se o firmware já está apertado, é aí que ele quebra - e o sintoma é **reinício por falta de memória**, não erro de rede. Muita gente perde horas procurando problema de conexão.

O firmware imprime o heap livre depois de cada consulta justamente para você acompanhar isso.

> Sobre o setInsecure: ele aceita QUALQUER certificado, o que serve para bancada e não para campo, porque abre a porta para interceptação. Em produção, use setCACert com o certificado raiz do fornecedor.

3. Passo a passo

Passo 1 - Criar o arquivo de configuração

O firmware espera um include/config.h que **não vem no pacote**, porque carrega a chave de acesso da API.

```
copy include\config.example.h include\config.h
```

No Linux ou macOS use cp. Depois preencha rede, endereços e chave. O .gitignore do projeto já ignora o config.h: **chave em repositório público e varrida por robôs em minutos.**

Passo 2 - Escolher o destino

No topo do main.cpp, a constante DESTINO decide o caminho. Comece com LOCAL.

Passo 3 - Compilar e gravar

```
pio run -t upload
pio device monitor
```

Para a S3 com PSRAM, que da folga para o caminho da nuvem, use -e esp32-s3.

Passo 4 - Ler o que aparece

```
Destino: gateway na rede local
[wifi] conectado, IP 192.168.1.55
[local] valor critico 29.34 - acao imediata, sem rede
[ia] consultando (local)...
[ia] O comportamento parece normal para a faixa observada.
[ia] heap livre apos a consulta: 198432 bytes
```

A linha [local] aparecendo **sem** depender da consulta é o comportamento correto: a reação crítica não espera modelo nenhum.

Passo 5 - Escolher o intervalo

O padrão é uma consulta por minuto, conservador de propósito. Consultar a cada leitura é desperdício: o sensor lê a cada segundo, mas a conclusão sobre o comportamento não muda nessa velocidade.

> Em API paga, o intervalo é diretamente o seu custo mensal.

4. Filtro de JSON

A resposta pode trazer alguns KB de metadados que você não vai usar, e a RAM do ESP32 não sobra. O filtro do ArduinoJson extrai só o campo que interessa, sem alocar o documento inteiro:

```
JsonDocument filtro;
filtro["response"] = true;
deserializeJson(doc, json,
DeserializationOption::Filter(filtro));
```

Sem isso, uma resposta grande pode simplesmente não caber na memória. O firmware traz um filtro para cada destino, porque os formatos são diferentes.

5. Solução de problemas

Sintoma	Causa provável	O que fazer
Não compila, falta config.h	arquivo não criado	copie o config.example.h conforme o passo 1
Resposta 401 ou 403	chave invalida ou sem permissão	confira a chave e o plano contratado
Resposta 429	limite de requisições	aumente INTERVALO_CONSULTA_MS
Reinicia ao usar HTTPS	RAM insuficiente no handshake	use o caminho local, ou uma placa com PSRAM
JSON invalido	formato diferente do esperado	ajuste o filtro ao formato do seu fornecedor
Resposta sem o campo esperado	caminho do JSON mudou	confira a documentação da API
Falha de conexão no local	modelo não acessível	teste a URL pelo navegador de um PC da mesma rede

6. Checklist final

	Item
[ ]	config.h criado a partir do config.example.h
[ ]	config.h fora do controle de versão
[ ]	Destino definido como LOCAL para os primeiros testes
[ ]	URL testada pelo navegador de um PC da mesma rede
[ ]	Leitura do sensor acontecendo independente da rede
[ ]	Reação crítica NÃO esperando resposta do modelo
[ ]	Heap livre acompanhado após cada consulta
[ ]	Filtro de JSON conferido contra a resposta real
[ ]	Intervalo de consulta dimensionado pelo custo, se for API paga
[ ]	setCACert no lugar de setInsecure, se for para campo