# IA de reuniões totalmente local: o que chegou no Hedy 3.2 e o custo em velocidade

Um mergulho técnico na IA no dispositivo do Hedy 3.2: quais modelos escolhemos, como eles cabem no Mac, Windows e iPhone, e o custo da inferência local em velocidade.

Publicado por Julian Pscheid · 22 de abril de 2026
Ler este artigo online: https://www.hedy.ai/pt/post/local-ai-engineering-deep-dive-hedy-3-2/

Um engenheiro de software em uma mesa de madeira estuda atentamente um MacBook inclinado para longe da câmera, com um holograma delicado em ciano e violeta surgindo do teclado para sugerir computação de IA no dispositivo

O Hedy 3.2 pode executar uma reunião inteira pelo seu laptop sem que nada saia do dispositivo: áudio, transcrição, resumos, notas, sugestões, tudo local. A nuvem ainda é nosso padrão e, para a maioria dos usuários, ainda é a escolha certa. O local é para as pessoas que preferem manter suas conversas no próprio hardware, mesmo quando isso significa aceitar algum custo em velocidade (e, em hardware de ponta, às vezes nem significa). Pela primeira vez, achamos que essa troca é honesta o suficiente para lançar.

O Hedy é um coach de reuniões usado por cerca de 30.000 pessoas em Mac, Windows, iOS, Android e Web. O reconhecimento de fala roda localmente em todas as plataformas desde o primeiro dia. A camada de análise de IA (resumos, notas detalhadas, respostas de chat, as sugestões ao vivo de "o que devo dizer em seguida") sempre foi na nuvem. Foi isso que mudou na versão 3.2.

Este post é sobre o que tornou isso possível em 2026, quais modelos escolhemos e o que deliberadamente decidimos não construir. Para a perspectiva voltada ao usuário sobre o que a IA local significa para seu fluxo de reuniões — privacidade, casos de uso jurídicos, médicos e jornalísticos, e como ativá-la — veja nossa visão geral de IA local para reuniões (/pt/post/local-ai-meetings-hedy-3-2/) .

## As duas curvas

Havia duas linhas de tendência que vínhamos observando havia alguns anos.

A primeira era a qualidade dos modelos com pesos abertos. Llama 2, em meados de 2023, era uma curiosidade para qualquer coisa além de uma demonstração de chatbot. No fim de 2024, Llama 3 e Qwen 2.5 já eram genuinamente úteis em tamanhos pequenos, mas você ainda sentia a diferença em tarefas como resumo de reuniões com múltiplos turnos, nas quais o modelo precisa acompanhar quem disse o quê ao longo de trinta minutos de diálogo. Durante 2025 e no início de 2026, duas famílias avançaram à frente para o nosso caso de uso: Gemma 4, da Google, e Qwen 3.5/3.6, da Alibaba. As duas equipes investiram esforço sério no que se poderia chamar de densidade por parâmetro e, juntas, cobrem tudo desde modelos no dispositivo da classe 2B até um MoE de 35B, com vários tamanhos que entregam mais do que o esperado em seguimento de instruções e saída estruturada.

A segunda curva era o hardware de consumo. A arquitetura de memória unificada do Apple Silicon acabou sendo exatamente o formato certo para inferência com transformers: os pesos do modelo ficam ao lado da computação da GPU, sem ida e volta pelo PCIe, sem orçamento separado de VRAM. Na geração M3, um MacBook Air básico de 16GB já conseguia rodar um modelo quantizado de 8-9B em velocidades usáveis. O Windows seguiu outro caminho. GPUs discretas com 8-12GB de VRAM são comuns em qualquer máquina vendida para jogos ou criação de conteúdo, e frameworks de inferência baseados em Vulkan agora extraem a maior parte desse desempenho de hardware NVIDIA, AMD e, cada vez mais, Intel.

As duas curvas se cruzaram no início de 2026. Podíamos escolher um modelo que fazia bem o nosso trabalho, e havia uma base instalada de hardware de consumo capaz de rodá-lo. Então lançamos.

## O que escolhemos e por quê

O runtime por baixo é o llama.cpp (https://github.com/ggml-org/llama.cpp) , incluído como submódulo e compilado por plataforma com o backend apropriado. Usamos pesos quantizados GGUF das builds do Unsloth no HuggingFace (https://huggingface.co/unsloth) , que atualmente são a fonte mais limpa de modelos com pesos abertos bem quantizados nessa faixa de tamanho.

O catálogo ativo no lançamento é:

**Modelo | Quant | Tamanho do arquivo | RAM ao carregar | Qualidade**

Qwen 3.5 2B | Q4_K_M | 1.2 GB | 3 GB | &
Qwen 3.5 4B | Q4_K_M | 2.6 GB | 4.5 GB | &
Gemma 4 E2B | Q4_K_M | 2.9 GB | 5 GB | &
Gemma 4 E4B | Q4_K_M | 4.6 GB | 7 GB | & &
Qwen 3.5 9B | Q4_K_M | 5.3 GB | 8 GB | & &
Qwen 3.5 9B HQ | Q8_0 | 8.9 GB | 12 GB | & &
Qwen 3.6 27B | UD-Q4_K_XL | 17.6 GB | 22 GB | & & &
Gemma 4 31B | Q4_K_M | 17.1 GB | 23 GB | & & &
Qwen 3.6 35B-A3B | UD-Q4_K_M | 20.6 GB | 25 GB | & & &

Além de alguns SKUs arquivados de versões anteriores que continuam selecionáveis para usuários que já os baixaram, mas desaparecem do seletor quando o arquivo local é excluído. O seletor reduz tudo a uma avaliação de qualidade em três estrelas: uma estrela para modelos abaixo de 7B, duas para 7-12B, três para acima de 12B. A quantização (Q4 versus Q8) não muda a avaliação; as estrelas tratam da capacidade bruta, não da precisão de implantação.

O prefixo "E" em Gemma 4 E2B e E4B é a notação de "parâmetros efetivos" da Google: E2B tem cerca de 2.3B parâmetros efetivos, com aproximadamente 5.1B contando embeddings, e E4B tem cerca de 4.5B efetivos, com aproximadamente 8B contando embeddings. As linhas Qwen usam contagens diretas de parâmetros totais. Ordenamos o seletor pelo que os usuários realmente se importam (RAM ao carregar e estrelas de qualidade), o que evita que as duas convenções de nomenclatura se confundam.

Por que essas duas famílias. Tanto Gemma 4 quanto Qwen 3.5/3.6 chegam em vários tamanhos dentro da faixa que nos interessa, ambas são fortes em seguir instruções em tamanhos pequenos (o que importa porque resumir reuniões é, em grande parte, "siga este esquema de saída e não alucine"), ambas têm licenças permissivas para distribuição comercial, e ambas têm tokenizers bem comportados nos 51 idiomas de interface em que o Hedy é oferecido hoje. A variante Qwen 27B usa a quantização UD-Q4_K_XL ("Unsloth Dynamic") da Unsloth, que preserva mais precisão de pesos nas camadas que mais importam. É uma melhora perceptível em relação a um Q4 ingênuo no mesmo tamanho de arquivo.

O que você vê de fato no app é um seletor de modelos que lista os modelos compatíveis com a sua máquina, com tamanho em disco, RAM aproximada ao carregar e a avaliação em estrelas. Se um modelo tecnicamente rodaria, mas derramaria algumas camadas para a CPU no seu hardware, a entrada recebe o sufixo "+ Slow". Preferimos ser honestos no seletor a fazer alguém baixar 5GB e depois se perguntar por que os resumos levam quatro minutos.

Traga seu próprio modelo. O seletor também tem uma seção "Custom models", onde você pode apontar o Hedy para qualquer GGUF compatível no disco e usá-lo junto ao catálogo curado. Nós não copiamos o arquivo; ele fica onde você o colocou, e o Hedy mantém uma referência persistente para que o vínculo sobreviva entre aberturas do app. Não há garantias de compatibilidade: precisa ser um GGUF que o llama.cpp consiga carregar, o template de chat precisa ser sensato o suficiente para que nossos prompts retornem uma saída estruturada útil, e o orçamento de memória fica por sua conta. Mas se você é o tipo de pessoa que lê este parágrafo e pensa "eu poderia testar DeepSeek V4 nisso", vá em frente.

## Apple Silicon: o caso fácil

Apple Silicon é a plataforma em que a IA no dispositivo já parece o futuro que chegou. Três coisas fazem isso funcionar.

Primeiro, o modelo de memória unificada. A GPU e a CPU compartilham o mesmo pool de RAM. Carregar um modelo quantizado de 8B parâmetros na memória significa que a GPU consegue ler esses pesos na largura de banda da memória sem qualquer overhead de cópia para VRAM que é padrão em uma configuração Windows com GPU discreta. Para inferência, em que você transmite pesos pela computação milhares de vezes por token, isso é uma vantagem estrutural significativa.

Segundo, Metal. A API de computação em GPU da Apple é madura, bem documentada, e o backend Metal do llama.cpp é um dos mais polidos. Compilamos com GGML_METAL=ON e não fazemos nada exótico por cima disso.

Terceiro, o piso dos chips M-series. Mesmo um M1 básico tem GPU e largura de banda de memória suficientes para rodar modelos quantizados pequenos em velocidades interativas.

Na prática, isso se mapeia para os níveis do catálogo:

• Compacto (Macs de 8GB): Gemma 4 E2B, Qwen 3.5 4B, Qwen 3.5 2B. Qualidade de uma estrela. Bom o suficiente para resumos curtos e saída estruturada, mais fraco em reuniões longas.

• Padrão (Macs de 16GB): Gemma 4 E4B, Qwen 3.5 9B. Qualidade de duas estrelas. O ponto ideal para a maioria dos usuários, incluindo MacBook Airs de configuração básica. A qualidade está chegando mais perto de um modelo pequeno de ponta hospedado para a nossa tarefa.

• Padrão, maior precisão (Macs de 16GB+): Qwen 3.5 9B HQ. O mesmo modelo Qwen 9B do Padrão, em Q8 em vez de Q4. Mesma contagem de parâmetros, mais precisão de pesos por camada. Um salto de qualidade menor do que subir uma classe de parâmetros, mas real se você tem folga.

• Pro (Macs de 24-32GB): Qwen 3.6 27B, Gemma 4 31B. Qualidade de três estrelas. Os dois arquivos têm cerca de 17GB em disco e querem 22-23GB de RAM ao carregar, com folga de KV cache para uma reunião longa.

• Max (Macs de 32GB+): Qwen 3.6 35B-A3B. Qualidade de três estrelas. Arquitetura MoE (35B parâmetros totais, cerca de 3B ativos por token), o que a torna visivelmente mais rápida em inferência do que sua contagem de parâmetros sugere. Quer cerca de 25GB ao carregar.

A maior surpresa positiva foi o quanto o nível Padrão se comporta bem em um MacBook Air de configuração básica. Estávamos preparados para dizer aos usuários que a IA local era para power users. No Apple Silicon, não é.

## Windows: VRAM e o problema de spillover

O Windows é mais complicado. A variação de hardware é enorme: um desenvolvedor com um desktop gamer recente e uma RTX 4080 tem mais potência de inferência do que qualquer Mac para o qual distribuímos, enquanto um knowledge worker em um laptop corporativo com gráficos integrados não tem praticamente nenhuma.

Tomamos uma posição deliberada na build para Windows: Vulkan é obrigatório para IA local . A build Windows em CMake define GGML_VULKAN=ON quando o SDK Vulkan está presente em tempo de build, e pula o target llama.dll por completo se ele não estiver. Não há fallback somente CPU para IA local no Windows. CUDA seria mais rápido na NVIDIA, mas amarrar a experiência a um único fornecedor de GPU teria significado excluir todos os usuários de AMD e Intel. Vulkan nos deixa a uma distância competitiva de CUDA na NVIDIA, e é o único caminho que funciona entre os fornecedores de GPU que nossa base de usuários realmente possui.

O formato do problema no Windows é diferente do Mac. GPUs discretas têm sua própria VRAM, separada da RAM do sistema, e o modelo precisa caber nessa VRAM para obter aceleração total por GPU. Se o modelo for ligeiramente grande demais, o llama.cpp consegue dividi-lo: a maioria das camadas vai para a GPU, o restante roda na CPU. Funciona, mas cada token agora espera pela camada mais lenta da cadeia. Vimos casos em que passar de "cabe na VRAM" para "duas camadas na CPU" reduziu o throughput aproximadamente pela metade. Depois que você está derramando mais do que algumas poucas camadas, é quase como rodar na CPU.

Calcular a divisão corretamente importa. Cada modelo no catálogo tem seu numLayers registrado (24 para Qwen 2B, 32 para Qwen 4B, e assim por diante, consultado no config.json de cada modelo no HuggingFace). Passar um valor irrealisticamente alto para o n_gpu_layers do llama.cpp não compra nada: o llama.cpp limita o offload à contagem real de camadas do modelo, e qualquer previsão de encaixe ou cálculo de spill contra um valor inflado silenciosamente dá a resposta errada. Então

mantemos esses números honestos no catálogo.

Nossa abordagem no seletor de modelos é ser específico sobre o resultado. Verificamos a VRAM disponível, calculamos a pegada do modelo (pesos mais KV cache mais um buffer de trabalho) e:

• Se o modelo cabe, nós o listamos normalmente.

• Se ele derrama moderadamente para a CPU, adicionamos o sufixo "+ Slow", para que os usuários saibam que devem esperar latência perceptível.

• Se ele derramaria de forma catastrófica, não o listamos nesse hardware.

Se a máquina não tiver GPU detectada, ou memória insuficiente até para o menor modelo, a seção de IA local nas configurações mostra um motivo de bloqueio em vez de um seletor.

Um detalhe específico de Windows que vale mencionar: a versão 595.xx do driver NVIDIA tinha um problema conhecido com computação Vulkan em placas RTX 40-series em certas builds do Windows, que causava crashes FAST_FAIL durante o carregamento do modelo. Investigamos achando que tínhamos lançado um bug, até percebermos que a mesma assinatura de crash aparecia em aplicações Vulkan não relacionadas na mesma versão do driver. A correção veio upstream, no driver 596.21 da NVIDIA. Agora exibimos uma mensagem de erro mais clara, mas a correção real é "atualize seu driver".

## Mobile: menor nível, honestos sobre a diferença

Só habilitamos IA local no iPhone 15 Pro e mais recentes (chip A17 Pro ou superior), e em iPads M-series. A lógica é direta: qualquer coisa mais antiga não tem a memória unificada nem a geração de Neural Engine para rodar até o menor nível em velocidades aceitáveis.

Mesmo em um 15 Pro, você está rodando modelos do nível Compacto: Qwen 3.5 2B, Qwen 3.5 4B, Gemma 4 E2B. Um modelo quantizado de 2-5B parâmetros é uma classe diferente de escritor em comparação com um de 9B. Bom em saída estruturada, bom em seguir instruções claras, decente em resumos curtos. Ele perde o fio em reuniões longas, especialmente aquelas com muitos participantes ou jargão técnico. Somos claros sobre isso na UI mobile: o modelo local é identificado como tal, e recomendamos Cloud AI como padrão para análise séria de reuniões no celular.

Android e Web não têm um modo de IA local voltado ao usuário na versão 3.2. O encanamento FFI do Android existe no código ( libllama.so é compilado), mas ainda não lançamos uma UI testada e controlada para ele. A variação de hardware no Android e as restrições de runtime (sem um equivalente da integração estreita do Metal) tornam difícil prometer uma experiência consistente hoje. Web está fora de questão por enquanto: restrições de navegador em torno de armazenamento persistente, acesso à computação por GPU e limites de memória ainda são alvos móveis. Os dois provavelmente virão mais tarde. Nenhum dos dois está comprometido.

## O que fica local, o que não fica

O objetivo de lançar IA local é que alguém possa processar uma reunião pelo Hedy sem que nada sobre essa reunião saia da própria máquina. Levamos esse objetivo a sério, o que significa ser preciso sobre o que flui para onde.

Com Local AI ativado e Cloud Sync desativado:

• Captura de áudio: local, nunca enviada.

• Transcrição: local, nunca enviada.

• Resumos, notas detalhadas, respostas de chat, sugestões ao vivo: modelo local, sem chamadas de API para trabalho de IA.

• Armazenamento da sessão: no dispositivo.

• Nada sobre a reunião toca nossos servidores.

Com Local AI ativado e Cloud Sync ativado:

• Tudo acima, mais:

• Dados da sessão (transcrições, resumos, metadados) são sincronizados com nosso backend, criptografados em trânsito e em repouso.

• É isso que permite acesso entre dispositivos. Seu celular pode mostrar notas de uma reunião que seu laptop acabou de resumir.

• O trabalho de IA em si ainda é local. Cloud Sync é uma camada de sincronização, não um caminho de fallback de inferência.

Sempre, independentemente das configurações:

• Informações da conta (e-mail, status da assinatura) passam pelos nossos servidores. Não há como um produto com conta existir sem uma conta.

• Telemetria anônima de uso e relatórios de crash fluem para nosso monitoramento. Nada disso contém conteúdo de conversas.

• Verificações de atualização do app, feature flags, downloads de modelos (que vêm de uma CDN). HTTPS GETs de conteúdo mais ou menos público, sem payload por usuário.

O que deliberadamente não construímos é um fallback silencioso para a nuvem. Se você tem IA local ativada e o pipeline de IA local falha (o modelo crasha, fica sem memória no meio da geração, lança um erro de inferência), o erro sobe para o chamador. Não há nova tentativa discreta contra nossos servidores. Quem ativou a IA local fez isso por um motivo. Voltar para a nuvem sem avisar trairia a suposição feita quando a pessoa optou por isso.

## O custo

Cloud AI ainda é melhor para a maioria dos usuários. Precisamos dizer isso claramente porque o público técnico deste post perceberia qualquer marketing nesse ponto.

A nuvem é mais rápida para a maioria dos usuários, muitas vezes por uma ordem de grandeza. Um resumo que fica pronto em alguns segundos no nosso pipeline de IA na nuvem pode levar 30 segundos localmente em um modelo do nível Padrão, e vários minutos se o usuário escolheu um modelo do nível Pro em uma máquina no limite. Sugestões ao vivo, que disparam enquanto a reunião está em andamento, são visivelmente mais responsivas no modo em nuvem.

Há uma exceção honesta que vale explicitar. Em um MacBook Pro de configuração máxima (por exemplo, um M5 Max com 128GB de memória unificada) rodando um modelo local forte, a comparação pode se inverter. Inferência na nuvem tem seus próprios problemas de latência: modelos hospedados compartilham capacidade de GPU entre muitos usuários, e o tempo até o primeiro token pode inflar quando a carga está alta ou um modelo popular está com fila. Um modelo local já carregado na memória não tem nenhuma dessa variância; ele roda na velocidade que seu hardware oferece, de forma determinística, sempre. Para usuários com hardware capaz rodando consultas curtas enquanto a nuvem está ocupada, o local pode de fato vencer no tempo de relógio. Não queremos exagerar (a maioria dos usuários, na maioria dos hardwares, ainda verá a nuvem como mais rápida em média), mas vale dizer em voz alta: a comparação de velocidade não é unidirecional.

A nuvem funciona em todas as plataformas. Local não. Se você está no Android, na Web, ou em um Mac ou iPhone mais antigo, o modo local ainda não é uma opção para você.

O motivo pelo qual a maioria dos usuários escolherá local é privacidade, não desempenho. Tentamos tornar o local bom o suficiente para que a troca por privacidade seja real, não um gesto, mas na maioria dos hardwares ela é uma troca. Escolher local porque você quer suas reuniões na sua máquina é racional. Escolher por velocidade só faz sentido se seu hardware conseguir sustentar isso.

Uma observação específica: Sugestões Automáticas, o recurso que executa o LLM continuamente durante uma reunião para antecipar o que dizer em seguida, é pesado. É a carga de trabalho de IA mais cara que o Hedy executa. Na nuvem, você não percebe. Localmente, especialmente em uma máquina menor, ele pode saturar o loop de inferência e deixar o resto do computador mais lento. O app mostra um diálogo de orientação único quando você ativa o Local AI Processing: "Automatic Suggestions run the AI continuously during a session. With Local AI Processing, this can keep your CPU and GPU busy and slow down the rest of your computer. You can turn them back on any time in Settings." Consideramos desabilitar Sugestões Automáticas à força no modo local e decidimos contra isso: alguém com uma máquina de alto desempenho consegue rodá-las, e preferimos não tratar o usuário de forma paternalista.

## Decisões arquiteturais que valem comentar

Algumas coisas que fizemos deliberadamente e que o leitor técnico talvez ache interessantes.

Sem fallback silencioso. Já mencionado. O custo voltado ao usuário são erros ocasionalmente confusos. O benefício é um recurso que significa o que diz.

Configuração por dispositivo. As configurações de IA local (qual modelo está selecionado, quais recursos usam local) não sincronizam entre dispositivos. Seu Mac pode ter um modelo 9B instalado; seu celular tem um modelo 2B; eles são configurados separadamente. Sincronizar seria errado aqui: o modelo certo é função do hardware em que você está, não da preferência abstrata do usuário.

Modelos arquivados continuam selecionáveis até serem excluídos. Quando um modelo no catálogo é substituído (Qwen 3.5 27B !’ Qwen 3.6 27B, por exemplo), marcamos a entrada mais antiga como arquivada em vez de removê-la. Usuários que já a baixaram podem continuar usando; quando excluem o arquivo local, a entrada desaparece completamente da UI. Não estamos no negócio de inutilizar um download de 17GB porque o catálogo seguiu em frente.

Quantização como escolha curada, não como knob. A Unsloth distribui cada modelo em uma dúzia de variantes de quantização: Q2, Q3, Q4, Q5, Q6, Q7 e Q8, com sabores K, K_M, K_L, XL e versões dinâmicas. Expor todas elas seria uma armadilha. A maioria dos usuários não consegue distinguir Q4_K_M de UD-Q4_K_XL, e o penhasco de qualidade na ponta baixa é íngreme. Escolhemos quantizações específicas por modelo com base na troca entre tamanho e qualidade para aquele modelo específico. Q4_K_M para a maioria. O UD-Q4_K_XL dinâmico da Unsloth para o 27B, onde a quantização consciente das camadas recupera qualidade real no mesmo tamanho de arquivo. Uma variante "HQ" Q8_0 do 9B para usuários com folga de RAM que querem a experiência mais próxima de fp16 disponível nessa classe de parâmetros. O rótulo "HQ" é deliberado; expor "Q8_0" como string de UI significaria ensinar todo usuário o que é quantização GGUF, e esse não é um custo que queremos cobrar.

Encaixe e velocidade são computados, não chutados. Cada entrada do catálogo passa por um algoritmo de pontuação contra o hardware detectado antes de aparecer no seletor. O scorer estima a memória necessária (pesos + KV cache + buffer de trabalho na quantização do catálogo do modelo) versus o que

realmente está disponível, e estima tokens por segundo usando a largura de banda de memória da GPU quando conhecida (com penalidades de modo para offload parcial em GPU e execução somente CPU). A partir disso, ele produz uma classificação de encaixe ("Great fit" / "Tight fit" / "Won't fit") e um sufixo "+ Slow" quando a execução envolver qualquer offload para CPU. No Windows, esse é o caso óbvio: camadas do modelo derramam da VRAM para a RAM do sistema. No macOS, o mesmo sufixo aparece quando o modelo só cabe pressionando o OS a expulsar outros apps para fixar memória para os pesos, o que tecnicamente carrega, mas você vai sentir. Um selo "Recommended" vai para o modelo de maior pontuação que cai em "Great fit" (não "Tight fit"); deliberadamente não recomendamos modelos "Tight fit" mesmo quando pontuariam mais alto em qualidade bruta, porque empurrar alguém para um modelo que roda visivelmente mais lento do que deveria é pior do que empurrar para um nível abaixo que roda limpo.

Cancelamento que realmente funciona. A geração do llama.cpp roda em um worker isolate para que a thread de UI continue responsiva, com streaming de tokens de volta para o isolate principal via um NativeCallable.listener . O cancelamento passa pelo llama_set_abort_callback , que verifica uma flag atômica, conferido contra o código-fonte upstream em vez de chutado, de modo que um usuário fechando uma sessão no meio de uma geração longa realmente interrompe o trabalho, em vez de deixá-lo terminar em segundo plano e desperdiçar bateria.

## O que ainda não sabemos

A lista honesta:

• Não temos ótimos números de latência em mundo real para uma grande variedade de hardwares. Testamos extensivamente nas máquinas que possuímos, pesquisamos beta testers e lemos relatórios de crash, mas a cauda longa de "laptop Windows com gráficos integrados de 2018" é difícil de caracterizar. Os primeiros dados de produção vão nos dizer onde as arestas ásperas realmente estão. Nos nossos próprios testes, vimos inferência local levar de cinco segundos a cinco minutos, dependendo da duração da reunião, do contexto e do modelo selecionado.

• Não sabemos qual é a cadência certa para lançar novas versões de modelos. A rotatividade do catálogo já é real (Qwen 3.5 !’ 3.6 em um ciclo). Até agora, lidamos com isso arquivando em vez de removendo, mas se o uso de disco no agregado sair do controle, talvez precisemos de outra política.

• O consumo de energia em laptops vai surpreender alguns usuários com cargas de trabalho de IA contínua. Uma reunião com Sugestões Automáticas rodando localmente é significativamente mais exigente do que a mesma reunião em Cloud AI. Ainda não construímos nenhum throttling sensível à bateria, e provavelmente deveríamos. O caso em que alguém só quer um resumo ao fim da reunião é outra história: isso é uma única geração one-shot e, em qualquer máquina capaz de rodar confortavelmente um modelo do nível Pro ou Max, fica bem dentro da folga típica de bateria e térmica.

## Para onde isso está indo

A história interessante não é o Hedy 3.2 especificamente. É que as curvas continuam se movendo.

Modelos com pesos abertos continuam ficando melhores em tamanhos pequenos. A classe 2B hoje está mais ou menos onde a classe 7B estava dezoito meses atrás. Seja qual for a aparência dos próximos doze meses de Gemma, Qwen e outras famílias sérias com pesos abertos, o piso de "o que roda em um laptop normal" continua subindo.

O hardware de consumo continua ficando mais capaz. As gerações de chips da Apple vêm incorporando inferência ao roadmap de silício de forma discreta. O Windows está começando a entregar NPUs que eventualmente importarão para inferência, ainda que a primeira geração de hoje seja principalmente

uma história de marketing.

A combinação significa que uma mudança silenciosa está em andamento. IA costumava significar "um pequeno grupo de empresas opera grandes modelos em seu nome, e você envia seus dados a elas". Está se tornando "um pequeno grupo de empresas treina grandes modelos, mas você pode rodá-los no seu próprio dispositivo com seus próprios dados, de ponta a ponta". A versão hospedada ainda é melhor hoje, e provavelmente continuará sendo por um tempo. Mas a diferença está diminuindo, e para um subconjunto significativo de usuários, "bom o suficiente e local" vence "melhor e remoto".

O Hedy 3.2 é nossa primeira aposta concreta nessa mudança. A arquitetura foi construída para nos permitir puxar essa alavanca mais adiante conforme modelos e hardware permitirem. Haverá mais.

---