IA de reuniones completamente local: qué lanzamos en Hedy 3.2 y qué cuesta en velocidad

Un análisis técnico de la IA en el dispositivo de Hedy 3.2: qué modelos elegimos, cómo encajan en Mac, Windows e iPhone, y qué coste de velocidad tiene la inferencia local.

Publicado por Julian Pscheid · 22 de abril de 2026 · Actualizado 20 de agosto de 2026
Leer este artículo en línea: https://www.hedy.ai/es/post/local-ai-engineering-deep-dive-hedy-3-2/

Un ingeniero de software en un escritorio de madera estudia pensativo un MacBook inclinado en dirección opuesta a la cámara, con un delicado holograma cian y violeta que se eleva del teclado y sugiere cómputo de IA en el dispositivo

Hedy 3.2 puede procesar una reunión completa en su portátil sin que nada salga del dispositivo: audio, transcripción, resúmenes, notas, sugerencias, todo local. La nube sigue siendo nuestro valor por defecto, y para la mayoría de usuarios sigue siendo la opción correcta. Lo local es para quienes prefieren mantener sus conversaciones en su propio hardware, incluso cuando eso implica aceptar cierto coste de velocidad (y en hardware de gama alta, a veces no lo implica). Por primera vez creemos que ese intercambio es lo bastante honesto como para lanzarlo.

Hedy es un coach de reuniones usado por unas 30.000 personas en Mac, Windows, iOS, Android y la web. El reconocimiento de voz se ejecuta localmente en todas las plataformas desde el primer día. La capa de análisis de IA (resúmenes, notas detalladas, el chat dentro de la sesión, las sugerencias en vivo de "qué debería decir ahora") siempre fue en la nube. Eso es lo que cambió en 3.2.

Este artículo trata de qué hizo eso posible en 2026, qué modelos elegimos y qué decidimos deliberadamente no construir. Para la perspectiva orientada al usuario sobre lo que la IA local significa para su flujo de reuniones — privacidad, casos de uso para abogados, profesionales de la salud y periodistas, y cómo activarla — vea nuestra visión general de IA local para reuniones (/es/post/local-ai-meetings-hedy-3-2/) .

Las dos curvas

Había dos líneas de tendencia que llevábamos un par de años observando.

La primera era la calidad de los modelos de pesos abiertos. Llama 2 a mediados de 2023 era una curiosidad para cualquier cosa más allá de una demo de chatbot. A finales de 2024, Llama 3 y Qwen 2.5 ya eran genuinamente útiles en tamaños pequeños, pero todavía se notaba la brecha en tareas como el resumen de reuniones de varios turnos, donde el modelo tiene que seguir quién dijo qué a lo largo de treinta minutos de diálogo. Durante 2025 y a comienzos de 2026, dos familias se adelantaron para nuestro caso de uso: Gemma 4 de Google y Qwen 3.5/3.6 de Alibaba. Ambos equipos han invertido mucho esfuerzo en lo que podría llamarse densidad por parámetro, y entre ambas cubren desde modelos en el dispositivo de clase 2B hasta un MoE de 35B, con varios tamaños que rinden por encima de su peso en seguimiento de instrucciones y salida estructurada.

La segunda curva era el hardware de consumo. La arquitectura de memoria unificada de Apple Silicon resultó tener exactamente la forma adecuada para la inferencia con transformers: los pesos del modelo viven junto al cómputo de la GPU, sin ida y vuelta por PCIe ni presupuesto separado de VRAM. Para la generación M3, un MacBook Air básico de 16GB podía ejecutar un modelo cuantizado de 8-9B a velocidades utilizables. Windows tomó otro camino. Las GPUs discretas con 8-12GB de VRAM son comunes en cualquier máquina vendida para gaming o creación de contenido, y los frameworks de inferencia basados en Vulkan ahora extraen la mayor parte de ese rendimiento del hardware de NVIDIA, AMD e (cada vez más) Intel.

Las dos curvas se cruzaron a comienzos de 2026. Podíamos elegir un modelo que hiciera bien nuestro trabajo, y existía una base instalada de hardware de consumo capaz de ejecutarlo. Así que lo lanzamos.

Qué elegimos y por qué

El runtime subyacente es llama.cpp (https://github.com/ggml-org/llama.cpp) , vendorizado como submódulo y compilado por plataforma con el backend adecuado. Usamos pesos cuantizados GGUF de las builds de HuggingFace de Unsloth (https://huggingface.co/unsloth) , que ahora mismo son la fuente más limpia de modelos de pesos abiertos bien cuantizados en este rango de tamaño.

El catálogo activo en el lanzamiento es:

Modelo	Cuant.	Tamaño de archivo	RAM al cargar	Calidad
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	& & &

Además de un puñado de SKUs archivados de versiones anteriores que siguen siendo seleccionables para usuarios que ya los descargaron, pero desaparecen del selector una vez que se elimina el archivo local. El selector reduce todo a una calificación de calidad de tres estrellas: una estrella para modelos de menos de 7B, dos para 7-12B, tres para más de 12B. La cuantización (Q4 frente a Q8) no cambia la calificación; las estrellas tratan de capacidad bruta, no de precisión de despliegue.

El prefijo "E" en Gemma 4 E2B y E4B es la notación de "parámetros efectivos" de Google: E2B son aproximadamente 2.3B parámetros efectivos, con unos 5.1B si se cuentan embeddings, y E4B son aproximadamente 4.5B efectivos, con unos 8B si se cuentan embeddings. Las filas de Qwen usan recuentos directos de parámetros totales. Ordenamos el selector por lo que realmente importa a los usuarios (RAM al cargar y estrellas de calidad), lo que evita que las dos convenciones de nombres se confundan entre sí.

Por qué estas dos familias. Tanto Gemma 4 como Qwen 3.5/3.6 se lanzan en varios tamaños dentro del rango que nos importa, ambas son fuertes en seguimiento de instrucciones en tamaños pequeños (lo que importa porque resumir reuniones es sobre todo "siga este esquema de salida y no alucine"), ambas tienen licencias permisivas para distribución comercial y ambas tienen tokenizadores que se comportan bien en los 51 idiomas de interfaz en los que Hedy se distribuye actualmente. La variante Qwen de 27B usa la cuantización UD-Q4_K_XL ("Unsloth Dynamic") de Unsloth, que conserva más precisión de pesos en las capas que más importan. Es un salto perceptible frente a Q4 ingenuo con el mismo tamaño de archivo.

Lo que usted ve en la app es un selector de modelos que lista los modelos compatibles con su máquina, con tamaño en disco, RAM aproximada al cargar y calificación de estrellas. Si un modelo técnicamente podría ejecutarse pero tendría que derramar algunas capas a la CPU en su hardware, la entrada recibe el sufijo "+ Slow". Preferimos ser honestos en el selector a que alguien descargue 5GB y luego se pregunte por qué los resúmenes tardan cuatro minutos.

Traiga su propio modelo. El selector también tiene una sección "Custom models" donde puede apuntar Hedy a cualquier GGUF compatible en disco y usarlo junto al catálogo curado. No copiamos el archivo; permanece donde usted lo puso, y Hedy conserva una referencia persistente para que el enlace sobreviva entre lanzamientos. No hay garantías de compatibilidad: tiene que ser un GGUF que llama.cpp pueda cargar, la plantilla de chat tiene que ser lo bastante sensata como para que nuestros prompts devuelvan salida estructurada útil, y el presupuesto de memoria corre por su cuenta. Pero si usted es el tipo de persona que lee este párrafo y piensa "podría probar DeepSeek V4 en esto", adelante.

Apple Silicon: el caso fácil

Apple Silicon es la plataforma donde la IA en el dispositivo ya se siente como el futuro que llegó. Tres cosas hacen que funcione.

Primero, el modelo de memoria unificada. La GPU y la CPU comparten el mismo pool de RAM. Cargar en memoria un modelo cuantizado de 8B parámetros significa que la GPU puede leer esos pesos a ancho de banda de memoria, sin ninguna de las sobrecargas de copia a VRAM que son estándar en una configuración de GPU discreta en Windows. Para inferencia, donde se transmiten pesos a través del cómputo miles de veces por token, eso es una ventaja estructural significativa.

Segundo, Metal. La API de cómputo de GPU de Apple es madura, está bien documentada y el backend Metal de llama.cpp es uno de los más pulidos. Compilamos con GGML_METAL=ON y no hacemos nada exótico encima.

Tercero, el piso de los chips M-series. Incluso un M1 básico tiene suficiente GPU y ancho de banda de memoria para ejecutar modelos cuantizados pequeños a velocidades interactivas.

En la práctica, esto se asigna a los niveles del catálogo:

• Compacto (Macs de 8GB): Gemma 4 E2B, Qwen 3.5 4B, Qwen 3.5 2B. Calidad de una estrella. Lo bastante bueno para resúmenes cortos y salida estructurada, más débil en reuniones largas.

• Estándar (Macs de 16GB): Gemma 4 E4B, Qwen 3.5 9B. Calidad de dos estrellas. El punto ideal para la mayoría de usuarios, incluidos MacBook Air de especificación base. La calidad se está acercando a un modelo frontier pequeño alojado para nuestra tarea.

• Estándar, mayor precisión (Macs de 16GB+): Qwen 3.5 9B HQ. El mismo modelo que el Qwen 9B de Estándar, en Q8 en lugar de Q4. Mismo recuento de parámetros, más precisión de pesos por capa. Un salto de calidad menor que subir una clase de parámetros, pero real si tiene margen.

• Pro (Macs de 24-32GB): Qwen 3.6 27B, Gemma 4 31B. Calidad de tres estrellas. Ambos archivos rondan los 17GB en disco y quieren 22-23GB de RAM al cargar con margen de KV cache para una reunión larga.

• Max (Macs de 32GB+): Qwen 3.6 35B-A3B. Calidad de tres estrellas. Arquitectura MoE (35B parámetros totales, ~3B activos por token), lo que la hace notablemente más rápida en inferencia de lo que su recuento de parámetros sugiere. Quiere unos 25GB al cargar.

La mayor sorpresa agradable fue lo bien que se comporta el nivel Estándar en un MacBook Air de especificación base. Estábamos preparados para decir a los usuarios que la IA local era para usuarios avanzados. En Apple Silicon, no lo es.

Windows: VRAM y el problema del desbordamiento

Windows es más complicado. La variación de hardware es enorme: un desarrollador con un desktop gaming reciente y una 4080 tiene más potencia de inferencia que cualquier Mac que distribuimos, mientras que un trabajador del conocimiento en un portátil empresarial con gráficos integrados prácticamente no tiene ninguna.

Tomamos una postura marcada en la build de Windows: Vulkan es obligatorio para la IA local . La build de CMake para Windows establece GGML_VULKAN=ON cuando el SDK de Vulkan está presente en tiempo de compilación, y omite por completo el target llama.dll si no lo está. No hay fallback solo-CPU para la IA local en Windows. CUDA sería más rápido en NVIDIA, pero bloquear la experiencia a un único proveedor de GPU habría significado excluir a todos los usuarios de AMD e Intel. Vulkan nos deja a distancia razonable de CUDA en NVIDIA, y es el único camino que funciona entre los proveedores de GPU que nuestra base de usuarios realmente posee.

La forma del problema en Windows es distinta a la del Mac. Las GPUs discretas tienen su propia VRAM, separada de la RAM del sistema, y el modelo tiene que caber en esa VRAM para obtener aceleración completa por GPU. Si el modelo es ligeramente demasiado grande, llama.cpp puede dividirlo: la mayoría de capas van a la GPU, el resto se ejecuta en la CPU. Funciona, pero cada token ahora espera a la capa más lenta de la cadena. Vimos casos en los que pasar de "cabe en VRAM" a "dos capas en CPU" redujo el rendimiento aproximadamente a la mitad. Una vez que desborda más de un puñado de capas, casi daría igual ejecutarlo en CPU.

Calcular correctamente la división importa. Cada modelo del catálogo tiene registrado su numLayers (24 para Qwen 2B, 32 para Qwen 4B, y así sucesivamente, consultado desde el config.json de HuggingFace de cada modelo). Pasar un valor irrealmente alto al n_gpu_layers de llama.cpp no le da nada: llama.cpp limita la descarga al recuento real de capas del modelo, y cualquier predicción de encaje o cálculo de desbordamiento contra un valor inflado obtiene silenciosamente una respuesta equivocada. Así que mantenemos esos números honestos en el catálogo.

Nuestro enfoque en el selector de modelos es ser específicos sobre el resultado. Comprobamos la VRAM disponible, calculamos la huella del modelo (pesos más KV cache más un búfer de trabajo) y:

• Si el modelo cabe, lo listamos normalmente.

• Si desborda moderadamente a CPU, añadimos el sufijo "+ Slow", para que los usuarios sepan que deben esperar latencia perceptible.

• Si desbordaría de forma catastrófica, no lo listamos en ese hardware.

Si la máquina no tiene GPU detectada, o memoria insuficiente incluso para el modelo más pequeño, la sección de IA local en la configuración muestra un motivo de bloqueo en lugar de un selector.

Un detalle específico de Windows que vale la pena mencionar: la versión 595.xx del driver de NVIDIA tenía un problema conocido con cómputo Vulkan en tarjetas RTX 40-series en ciertas builds de Windows que causaba cierres FAST_FAIL durante la carga del modelo. Investigamos esto pensando que habíamos lanzado un bug, y luego nos dimos cuenta de que la misma firma de cierre aparecía en aplicaciones Vulkan no relacionadas con la misma versión de driver. La solución estaba upstream, en el driver 596.21 de NVIDIA. Ahora mostramos un mensaje de error más claro, pero la solución real es "actualice su driver".

Móvil: el nivel más pequeño, honestos sobre la brecha

Solo habilitamos IA local en iPhone 15 Pro y posteriores (chip A17 Pro y superior), y en iPads con chip M-series o A17 Pro. El razonamiento es directo: cualquier cosa anterior no tiene la memoria unificada ni la generación de Neural Engine para ejecutar incluso el nivel más pequeño a velocidades aceptables.

Incluso en un 15 Pro, está ejecutando modelos del nivel Compacto: Qwen 3.5 2B, Qwen 3.5 4B, Gemma 4 E2B. Un modelo cuantizado de 2-5B parámetros es una clase de redactor distinta a uno de 9B. Bueno en salida estructurada, bueno siguiendo instrucciones claras, decente en resúmenes cortos. Pierde el hilo en reuniones largas, especialmente aquellas con muchos interlocutores o jerga técnica. Somos claros sobre esto en la UI móvil: el modelo local se etiqueta como tal, y recomendamos Cloud AI como valor por defecto para análisis serio de reuniones en teléfono.

Android y web no tienen un modo de IA local de cara al usuario en 3.2. La infraestructura FFI de Android existe en la base de código (libllama.so se compila), pero todavía no hemos lanzado una UI probada y con gating para ello. La variación de hardware en Android y las restricciones de runtime (sin equivalente a la integración estrecha de Metal) hacen difícil prometer hoy una experiencia consistente. Web queda fuera por ahora: las restricciones del navegador alrededor de almacenamiento persistente, acceso a cómputo de GPU y límites de memoria siguen siendo objetivos móviles. Probablemente ambos lleguen más adelante. Ninguno está comprometido.

Qué permanece local y qué no

El propósito de lanzar IA local es que alguien pueda procesar una reunión en Hedy sin que nada sobre esa reunión salga de su máquina. Nos tomamos ese objetivo en serio, lo que significa ser precisos sobre qué fluye hacia dónde.

Con IA local activada y Cloud Sync desactivado:

• Captura de audio: local, nunca se sube.

• Transcripción: local, nunca se sube.

• Resúmenes, notas detalladas, chat dentro de la sesión, sugerencias en vivo: modelo local, sin llamadas API para trabajo de IA.

• Almacenamiento de sesión: en el dispositivo.

• Nada sobre la reunión toca nuestros servidores a menos que comparta explícitamente la sesión. El audio nunca forma parte de lo compartido.

Con IA local activada y Cloud Sync activado:

• Todo lo anterior, además:

• Los datos de sesión (transcripciones, resúmenes, metadatos) se sincronizan con nuestro backend, cifrados en tránsito y en reposo.

• Esto es lo que permite el acceso entre dispositivos. Su teléfono puede mostrar notas de una reunión que su portátil acaba de resumir.

• El trabajo de IA en sí sigue siendo local. Cloud Sync es una capa de sincronización, no una ruta de inferencia de fallback.

Siempre, independientemente de la configuración:

• La información de cuenta (correo electrónico, estado de suscripción) pasa por nuestros servidores. No hay forma de que un producto con cuentas exista sin una cuenta.

• La telemetría anónima de uso y los reportes de cierres fluyen a nuestra monitorización. Nada de eso contiene contenido de conversaciones.

• Comprobaciones de actualización de la app, feature flags, descargas de modelos (que vienen de un CDN). Solicitudes HTTPS GET de contenido más o menos público, sin payload por usuario.

Lo que deliberadamente no construimos es un fallback silencioso a la nube. Si usted tiene la IA local activada y el flujo local falla (el modelo se cierra, se queda sin memoria a mitad de generación, lanza un error de inferencia), el error se muestra al llamador. No hay reintento silencioso contra nuestros servidores. Quien activó la IA local lo hizo por una razón. Recurrir a la nube sin decírselo traicionaría la suposición que hizo al optar por esa opción.

Qué costó

Cloud AI sigue siendo mejor para la mayoría de usuarios. Tenemos que decirlo claramente porque el público técnico de este artículo verá a través de cualquier marketing en este punto.

La nube es más rápida para la mayoría de usuarios, a menudo por un orden de magnitud. Un resumen que se completa en un par de segundos contra nuestro flujo en la nube puede tardar 30 segundos localmente en un modelo de nivel Estándar, y varios minutos si el usuario eligió un modelo de nivel Pro en una máquina al límite. Las sugerencias en vivo, que se disparan mientras la reunión está en curso, son notablemente más responsivas en modo nube.

Hay una excepción honesta que vale la pena sacar a la superficie. En un MacBook Pro de especificación máxima (por ejemplo un M5 Max con 128GB de memoria unificada) ejecutando un modelo local fuerte, la comparación puede invertirse. La inferencia en la nube no está libre de sus propios problemas de latencia: los modelos alojados comparten capacidad de GPU entre muchos usuarios, y el tiempo hasta el primer token puede dispararse cuando la carga es alta o un modelo popular tiene cola. Un modelo local ya cargado en memoria no tiene nada de esa variación; se ejecuta a la velocidad que le da su hardware, de forma determinista, cada vez. Para usuarios con hardware capaz que ejecutan consultas cortas mientras la nube está ocupada, lo local puede ganar de verdad en tiempo de reloj. No queremos exagerarlo (la mayoría de usuarios en la mayoría del hardware seguirá viendo la nube como más rápida en promedio), pero vale la pena decirlo en voz alta: la comparación de velocidad no es unidireccional.

La nube funciona en todas las plataformas. Lo local no. Si está en Android, la web, o un Mac o iPhone antiguo, el modo local aún no es una opción para usted.

La razón por la que la mayoría de usuarios elegirá lo local es la privacidad, no el rendimiento. Hemos intentado hacer lo local lo bastante bueno como para que el intercambio por privacidad sea real, no un gesto, pero en la mayoría del hardware es un intercambio. Elegir lo local porque quiere sus reuniones en su máquina es racional. Elegirlo por velocidad solo tiene sentido si su hardware puede respaldarlo.

Una nota específica: Automatic Suggestions, la función que ejecuta el LLM continuamente durante una reunión para anticipar qué decir a continuación, es pesada. Es la carga de trabajo de IA más costosa que Hedy realiza. En la nube, no lo nota. En local, especialmente en una máquina más pequeña, puede saturar el bucle de inferencia y ralentizar el resto de su ordenador. La app muestra un diálogo de orientación una sola vez cuando activa 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 desactivar por completo Automatic Suggestions en modo local y decidimos no hacerlo: alguien con una máquina de gama alta puede ejecutarlo, y preferimos no tratarle con condescendencia.

Decisiones arquitectónicas que vale la pena comentar

Algunas cosas que hicimos deliberadamente y que el lector técnico puede encontrar interesantes.

Sin fallback silencioso. Ya mencionado. El coste de cara al usuario son errores ocasionalmente confusos. El beneficio es una función que significa lo que dice.

Configuración por dispositivo. La configuración de IA local (qué modelo está seleccionado, qué funciones usan local) no se sincroniza entre dispositivos. Su Mac puede tener instalado un modelo de 9B; su teléfono tiene un modelo de 2B; se configuran por separado. Sincronizar sería incorrecto aquí: el modelo adecuado depende del hardware en el que está, no de lo que el usuario prefiere en abstracto.

Los modelos archivados siguen siendo seleccionables hasta que se eliminan. Cuando un modelo del catálogo es reemplazado (Qwen 3.5 27B !’ Qwen 3.6 27B, por ejemplo), marcamos la entrada anterior como archivada en lugar de eliminarla. Los usuarios que ya la descargaron pueden seguir usándola; una vez que eliminan el archivo local, la entrada desaparece por completo de la UI. No estamos en el negocio de romper una descarga de 17GB porque el catálogo avanzó.

La cuantización como elección curada, no como perilla. Unsloth lanza cada modelo en una docena de variantes de cuantización: Q2 through Q8, con sabores K, K_M, K_L, XL y versiones dinámicas. Exponerlas todas sería una trampa. La mayoría de usuarios no puede distinguir Q4_K_M de UD-Q4_K_XL, y el precipicio de calidad en el extremo bajo es pronunciado. Elegimos cuantizaciones específicas por modelo en función del intercambio entre tamaño y calidad para ese modelo concreto. Q4_K_M para la mayoría. La UD-Q4_K_XL dinámica de Unsloth para el 27B, donde la cuantización consciente de capas recupera calidad real con el mismo tamaño de archivo. Una variante "HQ" Q8_0 del 9B para usuarios con margen de RAM que quieren la experiencia más cercana a fp16 disponible en esa clase de parámetros. La etiqueta "HQ" es deliberada; mostrar "Q8_0" como cadena de UI significaría enseñar a cada usuario qué es la cuantización GGUF, y ese no es un coste que queramos cobrar.

El encaje y la velocidad se computan, no se adivinan. Cada entrada del catálogo pasa por un algoritmo de puntuación contra el hardware detectado antes de aparecer en el selector. El evaluador estima la memoria requerida (pesos + KV cache + búfer de trabajo con la cuantización de catálogo del modelo) frente a lo que realmente está disponible, y estima tokens por segundo usando el ancho de banda de memoria de la GPU cuando se conoce (con penalizaciones de modo por descarga parcial a GPU y

ejecución solo-CPU). A partir de eso produce una clasificación de encaje (Great fit / Tight fit / Won't fit) y un sufijo "+ Slow" cuando la ejecución implicará cualquier descarga a CPU. En Windows ese es el caso obvio: las capas del modelo desbordan de VRAM a RAM del sistema. En macOS el mismo sufijo aparece cuando el modelo solo cabe presionando al sistema operativo para expulsar otras apps y reservar memoria cableada para los pesos, lo que técnicamente carga pero usted lo notará. Una insignia "Recommended" va al modelo con mayor puntuación que cae en Great fit (no Tight); deliberadamente no recomendamos modelos Tight-fit incluso cuando puntuarían más alto en calidad bruta, porque empujar a alguien hacia un modelo que corre notablemente más lento de lo que debería es peor que empujarlo hacia un nivel inferior que corre limpiamente.

Cancelación que realmente funciona. La generación de llama.cpp se ejecuta en un isolate worker para que el hilo de UI siga respondiendo, con streaming de tokens de vuelta al isolate principal mediante un NativeCallable.listener . La cancelación pasa por llama_set_abort_callback comprobando una bandera atómica, verificada contra el código fuente upstream en lugar de adivinada, así que cuando un usuario cierra una sesión en medio de una generación larga, el trabajo se detiene de verdad, en vez de dejar que termine en segundo plano y desperdicie batería.

Qué todavía no sabemos

La lista honesta:

• No tenemos grandes números de latencia en el mundo real para una amplia variedad de hardware. Hemos probado extensamente en las máquinas que poseemos, encuestado a beta testers y leído reportes de cierres, pero la cola larga de "portátil Windows con gráficos integrados de 2018" es difícil de caracterizar. Los primeros datos de producción nos dirán dónde están realmente los bordes ásperos. En nuestras propias pruebas, hemos visto la inferencia local tardar desde cinco segundos hasta cinco minutos, dependiendo de la duración de la reunión, el contexto y el modelo seleccionado.

• No sabemos cuál es la cadencia adecuada para lanzar nuevas versiones de modelos. La rotación del catálogo ya es real (Qwen 3.5 !’ 3.6 en un ciclo). Hasta ahora lo hemos manejado archivando en lugar de eliminar, pero si el uso de disco en conjunto se descontrola, quizá necesitemos una política distinta.

• El consumo de energía en portátiles va a sorprender a algunos usuarios con cargas de trabajo de IA continua. Una reunión con Automatic Suggestions ejecutándose localmente es significativamente más exigente que la misma reunión con Cloud AI. Todavía no hemos construido ninguna regulación consciente de batería, y probablemente deberíamos. El caso en el que alguien solo quiere un resumen al final de la reunión es otra historia: es una única generación puntual, y en cualquier máquina que pueda ejecutar cómodamente un modelo de nivel Pro o Max, cae bien dentro del margen típico de batería y térmico.

Hacia dónde va esto

La historia interesante no es Hedy 3.2 específicamente. Es que las curvas siguen moviéndose.

Los modelos de pesos abiertos siguen mejorando en tamaños pequeños. La clase 2B de hoy está aproximadamente donde estaba la clase 7B hace dieciocho meses. Sea cual sea el aspecto de los próximos doce meses de Gemma, Qwen y las otras familias serias de pesos abiertos, el piso de "lo que se ejecuta en un portátil normal" sigue subiendo.

El hardware de consumo sigue ganando capacidad. Las generaciones de chips de Apple han estado incorporando inferencia discretamente en la hoja de ruta del silicio. Windows está empezando a distribuir NPUs que acabarán importando para inferencia, aunque la primera generación actual sea sobre todo una historia de marketing.

La combinación significa que está en marcha un cambio silencioso. IA solía significar "un puñado de empresas operan modelos grandes en su nombre, y usted les envía sus datos". Se está convirtiendo en "un puñado de empresas entrenan modelos grandes, pero usted puede ejecutarlos en su propio dispositivo con sus propios datos, de principio a fin". La versión alojada sigue siendo mejor hoy, y probablemente lo será durante un tiempo. Pero la brecha se está cerrando, y para un subconjunto significativo de usuarios, "lo bastante bueno y local" vence a "lo mejor y remoto".

Hedy 3.2 es nuestra primera apuesta concreta por ese cambio. La arquitectura está construida para permitirnos tirar más de esa palanca a medida que los modelos y el hardware lo permitan. Habrá más.

Hedy AI · Coaching de IA en vivo para conversaciones importantes

Prueba Hedy gratis: https://www.hedy.ai/es/downloads/

https://www.hedy.ai/es/post/local-ai-engineering-deep-dive-hedy-3-2/