Vollständig lokale Meeting-KI: Was in Hedy 3.2 ausgeliefert wurde und was es an Geschwindigkeit kostet

Ein technischer Deep Dive zu Hedys KI auf dem Gerät in Hedy 3.2: welche Modelle wir gewählt haben, wie sie auf Mac, Windows und iPhone passen und was lokale Inferenz an Geschwindigkeit kostet.

Veröffentlicht von Julian Pscheid · 22. April 2026
Diesen Artikel online lesen: https://www.hedy.ai/de/post/local-ai-engineering-deep-dive-hedy-3-2/

Ein Softwareentwickler an einem Holztisch betrachtet nachdenklich ein von der Kamera weggedrehtes MacBook, während ein filigranes cyan- und violettfarbenes Hologramm aus der Tastatur aufsteigt und KI-Berechnung auf dem Gerät andeutet

Hedy 3.2 kann ein vollständiges Meeting auf Ihrem Laptop verarbeiten, ohne dass etwas das Gerät verlässt: Audio, Transkription, Zusammenfassungen, Notizen, Vorschläge, alles lokal. Die Cloud bleibt unser Standard, und für die meisten Nutzer ist sie weiterhin die richtige Wahl. Lokal ist für die Menschen, die ihre Gespräche lieber auf ihrer eigenen Hardware behalten, selbst wenn das bedeutet, Geschwindigkeit zu opfern (und auf Top-Hardware manchmal auch nicht). Zum ersten Mal halten wir diesen Tradeoff für ehrlich genug, um ihn auszuliefern.

Hedy ist ein Meeting-Coach, der von rund 30.000 Menschen auf Mac, Windows, iOS, Android und im Web genutzt wird. Spracherkennung läuft seit dem ersten Tag auf jeder Plattform lokal. Die KI-Analyseebene (Zusammenfassungen, detaillierte Notizen, der Chat während der Sitzung, die Live-Vorschläge dazu, was Sie als Nächstes sagen sollten) war immer Cloud. Genau das hat sich in 3.2 geändert.

In diesem Beitrag geht es darum, was das 2026 möglich gemacht hat, welche Modelle wir gewählt haben und was wir bewusst nicht gebaut haben. Die nutzerseitige Perspektive darauf, was lokale KI für Ihren Meeting-Workflow bedeutet – Datenschutz, Anwendungsfälle für Anwälte, Medizin und Journalismus sowie die Aktivierung – finden Sie in unserem Überblick zu lokaler KI für Meetings (/de/post/local-ai-meetings-hedy-3-2/) .

Die zwei Kurven

Es gab zwei Trendlinien, die wir seit ein paar Jahren beobachtet haben.

Die erste war die Qualität von Open-Weight-Modellen. Llama 2 war Mitte 2023 für alles jenseits einer Chatbot-Demo eine Kuriosität. Ende 2024 waren Llama 3 und Qwen 2.5 in kleinen Größen wirklich nützlich, aber bei Aufgaben wie mehrstufiger Meeting-Zusammenfassung merkte man die Lücke noch, wenn das Modell über dreißig Minuten Dialog hinweg verfolgen muss, wer was gesagt hat. Im Laufe von 2025 und bis Anfang 2026 haben sich für unseren Anwendungsfall zwei Familien abgesetzt: Gemma 4 von Google und Qwen 3.5/3.6 von Alibaba. Beide Teams haben viel Arbeit in das gesteckt, was man Dichte pro Parameter nennen könnte, und zusammen decken sie alles ab: von 2B-Klasse-Modellen auf dem Gerät bis zu einem 35B MoE, mit mehreren Größen, die bei Instruction Following und strukturierter Ausgabe über ihrer Gewichtsklasse spielen.

Die zweite Kurve war Consumer-Hardware. Die Unified-Memory-Architektur von Apple Silicon hat sich als genau die richtige Form für Transformer-Inferenz erwiesen: Modellgewichte liegen direkt neben der GPU-Rechenleistung, kein PCIe-Roundtrip, kein separates VRAM-Budget. Ab der M3-Generation konnte ein einfaches 16GB MacBook Air ein quantisiertes 8-9B-Modell mit brauchbarer Geschwindigkeit ausführen. Windows nahm einen anderen Weg. Diskrete GPUs mit 8-12GB VRAM sind in fast jedem Rechner üblich, der für Gaming oder Content-Erstellung verkauft wird, und Vulkan-basierte Inferenz-Frameworks holen inzwischen den Großteil dieser Leistung aus NVIDIA-, AMD- und zunehmend Intel-Hardware heraus.

Die zwei Kurven kreuzten sich Anfang 2026. Wir konnten ein Modell wählen, das unsere Aufgabe gut erledigt, und es gab eine installierte Basis von Consumer-Hardware, die es ausführen konnte. Also haben wir ausgeliefert.

Was wir gewählt haben und warum

Die Runtime darunter ist llama.cpp (https://github.com/ggml-org/llama.cpp) , als Submodul eingebunden und pro Plattform mit dem passenden Backend gebaut. Wir verwenden GGUF-quantisierte Gewichte aus Unsloths HuggingFace-Builds (https://huggingface.co/unsloth) , die derzeit die sauberste Quelle für gut quantisierte Open-Weight-Modelle in diesem Größenbereich sind.

Der aktive Katalog zum Launch ist:

Modell	Quant	Dateigröße	RAM beim Laden	Qualität
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	& & &

Dazu kommen einige archivierte SKUs aus früheren Versionen, die für Nutzer auswählbar bleiben, die sie bereits heruntergeladen haben, aber aus dem Picker verschwinden, sobald die lokale Datei gelöscht wird. Der Picker fasst alles in einer Drei-Sterne-Qualitätsbewertung zusammen: ein Stern für Modelle unter 7B, zwei für 7-12B, drei für über 12B. Quantisierung (Q4 vs. Q8) ändert die Bewertung nicht; bei den Sternen geht es um rohe Leistungsfähigkeit, nicht um Deployment-Präzision.

Das Präfix "E" in Gemma 4 E2B und E4B ist Googles Notation für "effective parameters": E2B entspricht ungefähr 2,3B effektiven Parametern und etwa 5,1B inklusive Embeddings, E4B ungefähr 4,5B effektiv und etwa 8B inklusive Embeddings. Die Qwen-Zeilen verwenden einfache Gesamtparameterzahlen. Wir sortieren den Picker nach dem, was Nutzern tatsächlich wichtig ist (RAM beim Laden und Qualitätssterne), sodass die zwei Namenskonventionen sich nicht gegenseitig verwirren.

Warum diese zwei Familien. Sowohl Gemma 4 als auch Qwen 3.5/3.6 werden in mehreren Größen innerhalb des Bereichs ausgeliefert, der uns interessiert. Beide sind in kleinen Größen stark bei Instruction Following (was wichtig ist, weil Meeting-Zusammenfassung meistens heißt: "folge diesem Ausgabeschema und halluziniere nicht"), beide sind für kommerzielle Distribution großzügig lizenziert, und beide haben gutmütige Tokenizer über die 51 Oberflächensprachen hinweg, in denen Hedy derzeit ausgeliefert wird. Die 27B-Qwen-Variante nutzt Unsloths UD-Q4_K_XL-Quantisierung ("Unsloth Dynamic"), die in den wichtigsten Schichten mehr Gewichtpräzision bewahrt. Sie ist ein spürbarer Schritt nach oben gegenüber naivem Q4 bei gleicher Dateigröße.

Was Sie in der App tatsächlich sehen, ist ein Modell-Picker, der Modelle auflistet, die mit Ihrem Rechner kompatibel sind, mit Speicherplatzbedarf, ungefährer RAM-Nutzung beim Laden und Sternebewertung. Wenn ein Modell technisch laufen würde, aber auf Ihrer Hardware einige Schichten auf die CPU auslagern müsste, bekommt der Eintrag ein "+ Slow"-Suffix. Wir sind lieber im Picker ehrlich, als jemanden 5GB herunterladen zu lassen und ihn danach rätseln zu lassen, warum Zusammenfassungen vier Minuten dauern.

Eigenes Modell mitbringen. Der Picker hat außerdem einen Abschnitt für eigene Modelle, in dem Sie Hedy auf jede kompatible GGUF-Datei auf dem Laufwerk zeigen und sie neben dem kuratierten Katalog verwenden können. Wir kopieren die Datei nicht; sie bleibt dort, wo Sie sie abgelegt haben, und Hedy hält eine persistente Referenz darauf, damit die Verknüpfung über App-Starts hinweg bestehen bleibt. Es gibt keine Kompatibilitätsgarantien: Es muss ein GGUF sein, das llama.cpp laden kann, das Chat-Template muss vernünftig genug sein, dass unsere Prompts nützliche strukturierte Ausgabe zurückgeben, und für das Speicherbudget sind Sie selbst verantwortlich. Aber wenn Sie zu den Menschen gehören, die diesen Absatz lesen und denken: "Ich könnte darauf DeepSeek V4 ausprobieren", nur zu.

Apple Silicon: der einfache Fall

Apple Silicon ist die Plattform, auf der sich KI auf dem Gerät schon jetzt wie die Zukunft anfühlt. Drei Dinge sorgen dafür.

Erstens das Unified-Memory-Modell. GPU und CPU teilen sich denselben RAM-Pool. Ein quantisiertes 8B-Parameter-Modell in den Speicher zu laden bedeutet, dass die GPU diese Gewichte mit Speicherbandbreite lesen kann, ohne den Copy-to-VRAM-Overhead, der bei einem Windows-Setup mit diskreter GPU Standard ist. Für Inferenz, bei der Gewichte tausende Male pro Token durch Compute gestreamt werden, ist das ein echter struktureller Vorteil.

Zweitens Metal. Apples GPU-Compute-API ist ausgereift, gut dokumentiert, und das Metal-Backend von llama.cpp ist eines der am besten polierten. Wir kompilieren mit GGML_METAL=ON und bauen nichts Exotisches darauf.

Drittens die M-series-Chip-Untergrenze. Selbst ein einfacher M1 hat genug GPU- und Speicherbandbreite, um kleine quantisierte Modelle mit interaktiver Geschwindigkeit auszuführen.

Praktisch bildet sich das auf die Katalogstufen ab:

• Compact (8GB Macs): Gemma 4 E2B, Qwen 3.5 4B, Qwen 3.5 2B. Ein-Stern-Qualität. Gut genug für kurze Zusammenfassungen und strukturierte Ausgabe, schwächer bei langen Meetings.

• Standard (16GB Macs): Gemma 4 E4B, Qwen 3.5 9B. Zwei-Sterne-Qualität. Der Sweet Spot für die meisten Nutzer, einschließlich MacBook Airs in Basisausstattung. Die Qualität kommt bei unserer Aufgabe einem gehosteten kleinen Frontier-Modell näher.

• Standard, höhere Präzision (16GB+ Macs): Qwen 3.5 9B HQ. Dasselbe Modell wie Qwen 9B in Standard, aber mit Q8 statt Q4. Gleiche Parameterzahl, mehr Gewichtpräzision pro Schicht. Ein kleinerer Qualitätssprung als der Wechsel in eine höhere Parameterklasse, aber real, wenn Sie den Spielraum haben.

• Pro (24-32GB Macs): Qwen 3.6 27B, Gemma 4 31B. Drei-Sterne-Qualität. Beide Dateien sind etwa 17GB groß und benötigen beim Laden 22-23GB RAM mit KV cache-Spielraum für ein langes Meeting.

• Max (32GB+ Macs): Qwen 3.6 35B-A3B. Drei-Sterne-Qualität. MoE-Architektur (35B Gesamtparameter, etwa 3B aktiv pro Token), wodurch es bei der Inferenz spürbar schneller ist, als die Parameterzahl vermuten lässt. Benötigt etwa 25GB beim Laden.

Die größte angenehme Überraschung war, wie gut sich die Standard-Stufe auf einem MacBook Air in Basisausstattung verhält. Wir waren darauf vorbereitet, Nutzern zu sagen, dass Local AI für Power-User ist. Auf Apple Silicon ist sie das nicht.

Windows: VRAM und das Spillover-Problem

Windows ist komplizierter. Die Hardware-Varianz ist enorm: Ein Entwickler mit einem aktuellen Gaming-Desktop und einer 4080 hat mehr Inferenzleistung als jeder Mac, den wir ausliefern, während eine Wissensarbeiterin auf einem Business-Laptop mit integrierter Grafik praktisch keine hat.

Wir haben für den Windows-Build eine klare Position bezogen: Vulkan ist für Local AI erforderlich . Der Windows-CMake-Build setzt GGML_VULKAN=ON , wenn das Vulkan SDK zur Build-Zeit vorhanden ist, und überspringt das llama.dll -Target vollständig, wenn es nicht vorhanden ist. Es gibt keinen CPU-only-Fallback für Local AI unter Windows. CUDA wäre auf NVIDIA schneller, aber die Erfahrung auf einen einzigen GPU-Anbieter festzulegen hätte bedeutet, jeden AMD- und Intel-Nutzer auszuschließen. Vulkan bringt uns auf NVIDIA in Schlagdistanz zu CUDA, und es ist der einzige Weg, der über die GPU-Anbieter hinweg funktioniert, die unsere Nutzerbasis tatsächlich besitzt.

Die Form des Problems ist unter Windows anders als auf dem Mac. Diskrete GPUs haben eigenes VRAM, getrennt vom System-RAM, und das Modell muss in dieses VRAM passen, um vollständige GPU-Beschleunigung zu bekommen. Wenn das Modell etwas zu groß ist, kann llama.cpp es aufteilen: Die meisten Schichten gehen auf die GPU, der Rest läuft auf der CPU. Das funktioniert, aber jedes Token wartet jetzt auf die langsamste Schicht in der Kette. Wir haben Fälle gesehen, in denen der Wechsel von "passt in VRAM" zu "zwei Schichten auf CPU" den Durchsatz ungefähr halbiert hat. Sobald Sie mehr als eine Handvoll Schichten auslagern, könnten Sie genauso gut auf CPU laufen.

Den Split korrekt zu berechnen, ist wichtig. Für jedes Modell im Katalog ist numLayers hinterlegt (24 für Qwen 2B, 32 für Qwen 4B und so weiter, jeweils aus dem HuggingFace- config.json des Modells nachgeschlagen). Einen unrealistisch hohen Wert an n_gpu_layers von llama.cpp zu übergeben bringt

nichts: llama.cpp begrenzt Offload auf die echte Schichtzahl des Modells, und jede Fit-Prognose oder Spill-Berechnung gegen einen aufgeblähten Wert bekommt still die falsche Antwort. Deshalb halten wir diese Zahlen im Katalog ehrlich.

Unser Ansatz im Modell-Picker ist, beim Ergebnis konkret zu sein. Wir prüfen verfügbares VRAM, berechnen den Modell-Footprint (Gewichte plus KV cache plus Arbeitsbuffer) und:

• Wenn das Modell passt, listen wir es normal.

• Wenn es moderat auf die CPU auslagert, fügen wir das "+ Slow"-Suffix hinzu, damit Nutzer mit spürbarer Latenz rechnen.

• Wenn es katastrophal auslagern würde, listen wir es auf dieser Hardware gar nicht.

Wenn der Rechner keine erkannte GPU hat oder nicht genug Speicher selbst für das kleinste Modell, zeigt der Local AI-Bereich in den Einstellungen statt eines Pickers einen Blockierungsgrund.

Ein bestimmtes Windows-Detail ist erwähnenswert: NVIDIA-Treiberversion 595.xx hatte auf bestimmten Windows-Builds ein bekanntes Problem mit Vulkan Compute auf RTX 40-series-Karten, das beim Laden des Modells FAST_FAIL-Abstürze verursachte. Wir haben das untersucht, weil wir dachten, wir hätten einen Bug ausgeliefert, und dann festgestellt, dass dieselbe Crash-Signatur in unabhängigen Vulkan-Anwendungen mit derselben Treiberversion auftauchte. Die Lösung lag upstream, in NVIDIAs 596.21-Treiber. Wir zeigen jetzt eine klarere Fehlermeldung, aber die tatsächliche Lösung lautet: Treiber aktualisieren.

Mobile: kleinste Stufe, ehrlich über die Lücke

Wir haben Local AI nur auf iPhone 15 Pro und neuer (A17 Pro-Chip und höher) sowie auf M-series iPads aktiviert. Die Begründung ist einfach: Alles Ältere hat weder den Unified Memory noch die Neural-Engine-Generation, um selbst die kleinste Stufe mit akzeptabler Geschwindigkeit auszuführen.

Selbst auf einem 15 Pro laufen Sie mit Modellen aus der Compact-Stufe: Qwen 3.5 2B, Qwen 3.5 4B, Gemma 4 E2B. Ein quantisiertes 2-5B-Parameter-Modell ist eine andere Klasse von Autor als ein 9B-Modell. Gut bei strukturierter Ausgabe, gut darin, klaren Anweisungen zu folgen, brauchbar bei kurzen Zusammenfassungen. Bei langen Meetings verliert es den Faden, besonders bei vielen Sprecherinnen und Sprechern oder technischem Jargon. Wir sagen das in der mobilen UI offen: Das lokale Modell ist als solches gekennzeichnet, und wir empfehlen Cloud AI als Standard für ernsthafte Meeting-Analyse auf dem Telefon.

Android und Web haben in 3.2 keinen nutzerseitigen Local AI-Modus. Die Android-FFI-Verrohrung existiert im Code (libllama.so wird gebaut), aber wir haben dafür noch keine getestete, begrenzte UI ausgeliefert. Hardware-Varianz auf Android und die Runtime-Einschränkungen (kein Äquivalent zu Metals enger Integration) machen es heute schwer, eine konsistente Erfahrung zu versprechen. Web ist vorerst ausgeschlossen: Browser-Einschränkungen rund um persistenten Speicher, GPU-Compute-Zugriff und Speicherlimits sind weiterhin bewegliche Ziele. Beides kommt wahrscheinlich später. Beides ist nicht zugesagt.

Was lokal bleibt und was nicht

Der Sinn von Local AI ist, dass jemand ein Meeting durch Hedy laufen lassen kann und nichts über dieses Meeting den eigenen Rechner verlässt. Wir haben dieses Ziel ernst genommen, und das bedeutet, präzise zu sagen, was wohin fließt.

Mit Local AI an und Cloud Sync aus:

• Audioerfassung: lokal, nie hochgeladen.

• Transkription: lokal, nie hochgeladen.

• Zusammenfassungen, detaillierte Notizen, Chat während der Sitzung, Live-Vorschläge: lokales Modell, keine API-Aufrufe für KI-Arbeit.

• Sitzungsspeicher: auf dem Gerät.

• Nichts über das Meeting berührt unsere Server.

Mit Local AI an und Cloud Sync an:

• Alles oben Genannte, plus:

• Sitzungsdaten (Transkripte, Zusammenfassungen, Metadaten) synchronisieren mit unserem Backend, verschlüsselt während der Übertragung und im Ruhezustand.

• Das ermöglicht geräteübergreifenden Zugriff. Ihr Telefon kann Notizen aus einem Meeting anzeigen, das Ihr Laptop gerade zusammengefasst hat.

• Die KI-Arbeit selbst bleibt lokal. Cloud Sync ist eine Synchronisierungsschicht, kein Fallback-Inferenzpfad.

Immer, unabhängig von den Einstellungen:

• Kontoinformationen (E-Mail, Abo-Status) laufen über unsere Server. Es gibt keinen Weg, wie ein Kontoprodukt ohne Konto existieren kann.

• Anonyme Nutzungstelemetrie und Crash-Reports gehen an unser Monitoring. Nichts davon enthält Gesprächsinhalte.

• App-Update-Prüfungen, Feature Flags, Modelldownloads (die von einem CDN kommen). HTTPS-GETs von mehr oder weniger öffentlichen Inhalten, ohne nutzerspezifische Payload.

Was wir bewusst nicht gebaut haben, ist ein stiller Cloud-Fallback. Wenn Local AI aktiviert ist und die lokale Pipeline fehlschlägt (Modell stürzt ab, läuft mitten in der Generierung aus dem Speicher, wirft einen Inferenzfehler), erreicht der Fehler den Aufrufer. Es gibt keinen stillen Wiederholungsversuch gegen unsere Server. Wer Local AI aktiviert hat, hat das aus einem Grund getan. Ohne Hinweis auf die Cloud zurückzufallen würde die Annahme verraten, die diese Person beim Opt-in getroffen hat.

Was es gekostet hat

Cloud AI ist für die meisten Nutzer weiterhin besser. Wir müssen das klar sagen, weil das technische Publikum dieses Beitrags jedes Marketing an dieser Stelle durchschauen würde.

Cloud ist für die meisten Nutzer schneller, oft um eine Größenordnung. Eine Zusammenfassung, die gegen unsere Cloud-Pipeline in ein paar Sekunden fertig ist, kann lokal auf einem Standard-Stufen-Modell 30 Sekunden dauern, und mehrere Minuten, wenn jemand auf einem grenzwertigen Rechner ein Pro-Stufen-Modell gewählt hat. Live-Vorschläge, die während des laufenden Meetings feuern, reagieren im Cloud-Modus spürbar schneller.

Es gibt eine ehrliche Ausnahme, die erwähnenswert ist. Auf einem MacBook Pro mit Top-Ausstattung (zum Beispiel einem M5 Max mit 128GB Unified Memory), das ein starkes lokales Modell ausführt, kann der Vergleich kippen. Cloud-Inferenz ist nicht frei von eigenen Latenzproblemen: Gehostete Modelle teilen GPU-Kapazität über viele Nutzer hinweg, und Time-to-first-token kann stark anwachsen, wenn die Last hoch ist oder ein beliebtes Modell Rückstau hat. Ein lokales Modell, das bereits im Speicher geladen ist, hat keine dieser Schwankungen; es läuft deterministisch jedes Mal mit der Geschwindigkeit, die Ihre Hardware hergibt. Für Nutzer mit leistungsfähiger Hardware, die kurze Abfragen ausführen, während die Cloud beschäftigt ist, kann lokal bei der Wanduhrzeit tatsächlich gewinnen. Wir wollen das nicht

überziehen (die meisten Nutzer auf der meisten Hardware werden Cloud im Durchschnitt weiterhin als schneller erleben), aber es sollte ausgesprochen werden: Der Geschwindigkeitsvergleich ist nicht einseitig.

Cloud funktioniert auf jeder Plattform. Lokal nicht. Wenn Sie Android, Web oder einen älteren Mac oder ein älteres iPhone nutzen, ist der lokale Modus für Sie noch keine Option.

Der Grund, warum die meisten Nutzer lokal wählen werden, ist Datenschutz, nicht Performance. Wir haben versucht, lokal gut genug zu machen, dass der Datenschutz-Tradeoff real ist und keine Geste, aber auf der meisten Hardware bleibt es ein Tradeoff. Lokal zu wählen, weil Sie Ihre Meetings auf Ihrem Rechner haben möchten, ist rational. Es wegen Geschwindigkeit zu wählen ergibt nur Sinn, wenn Ihre Hardware das hergibt.

Eine konkrete Anmerkung: Automatic Suggestions, die Funktion, die das LLM während eines Meetings kontinuierlich laufen lässt, um vorauszuahnen, was Sie als Nächstes sagen sollten, ist schwergewichtig. Es ist die teuerste KI-Workload, die Hedy ausführt. In der Cloud merken Sie das nicht. Lokal, besonders auf einem kleineren Rechner, kann sie die Inferenzschleife sättigen und den Rest Ihres Computers verlangsamen. Die App zeigt einen einmaligen Hinweisdialog, wenn Sie Local AI Processing aktivieren: "Automatic Suggestions lassen die KI während einer Sitzung kontinuierlich laufen. Mit Local AI Processing kann das Ihre CPU und GPU beschäftigt halten und den Rest Ihres Computers verlangsamen. Sie können sie jederzeit in den Einstellungen wieder aktivieren." Wir haben überlegt, Automatic Suggestions im lokalen Modus hart zu deaktivieren, und uns dagegen entschieden: Wer eine High-End-Maschine hat, kann es ausführen, und wir möchten nicht bevormunden.

Architekturentscheidungen, über die es sich zu sprechen lohnt

Ein paar Dinge haben wir bewusst so gemacht, die technische Leserinnen und Leser interessant finden könnten.

Kein stiller Fallback. Schon erwähnt. Die nutzerseitige Steuer ist gelegentlich verwirrende Fehler. Der Nutzen ist eine Funktion, die bedeutet, was sie sagt.

Konfiguration pro Gerät. Local AI-Einstellungen (welches Modell ausgewählt ist, welche Funktionen lokal arbeiten) synchronisieren nicht zwischen Geräten. Ihr Mac könnte ein 9B-Modell installiert haben; Ihr Telefon hat ein 2B-Modell; sie werden getrennt konfiguriert. Synchronisierung wäre hier falsch: Das richtige Modell ist eine Funktion der Hardware, auf der Sie gerade sind, nicht dessen, was Sie abstrakt bevorzugen.

Archivierte Modelle bleiben auswählbar, bis sie gelöscht werden. Wenn ein Modell im Katalog abgelöst wird (Qwen 3.5 27B !’ Qwen 3.6 27B zum Beispiel), markieren wir den älteren Eintrag als archiviert, statt ihn zu entfernen. Nutzer, die ihn bereits heruntergeladen haben, können ihn weiter verwenden; sobald sie die lokale Datei löschen, verschwindet der Eintrag vollständig aus der UI. Wir sind nicht im Geschäft, einen 17GB-Download unbrauchbar zu machen, nur weil der Katalog weitergezogen ist.

Quantisierung als kuratierte Entscheidung, nicht als Regler. Unsloth liefert jedes Modell in einem Dutzend Quantisierungsvarianten aus: Q2 through Q8, mit K-, K_M-, K_L-, XL-Flavors und dynamischen Versionen. Alle offenzulegen wäre eine Fußfalle. Die meisten Nutzer können Q4_K_M nicht von UD-Q4_K_XL unterscheiden, und der Qualitätsabbruch am unteren Ende ist steil. Wir wählen spezifische Quantisierungen pro Modell auf Basis des Größen-gegen-Qualitäts-Tradeoffs für genau dieses Modell. Q4_K_M für die meisten. Unsloths dynamisches UD-Q4_K_XL für das 27B, bei dem schichtbewusste Quantisierung bei gleicher Dateigröße echte Qualität zurückholt. Eine Q8_0-"HQ"-Variante des 9B für Nutzer mit RAM-Spielraum, die die fp16-nächste Erfahrung wollen, die in dieser Parameterklasse

verfügbar ist. Das "HQ"-Label ist bewusst gewählt; "Q8_0" als UI-String anzuzeigen würde bedeuten, jedem Nutzer erklären zu müssen, was GGUF-Quantisierung ist, und diese Steuer möchten wir nicht erheben.

Fit und Geschwindigkeit werden berechnet, nicht geraten. Jeder Katalogeintrag läuft durch einen Scoring-Algorithmus gegen die erkannte Hardware, bevor er im Picker erscheint. Der Scorer schätzt benötigten Speicher (Gewichte + KV cache + Arbeitsbuffer bei der Katalogquantisierung des Modells) gegenüber dem, was tatsächlich verfügbar ist, und schätzt Tokens pro Sekunde anhand der Speicherbandbreite der GPU, wenn bekannt (mit Modus-Abzügen für teilweisen GPU-Offload und CPU-only-Ausführung). Daraus erzeugt er eine Fit-Klassifikation (Great fit / Tight fit / Won't fit) und ein "+ Slow"-Suffix, wenn die Ausführung irgendeinen CPU-Offload umfasst. Unter Windows ist das der offensichtliche Fall: Modellschichten laufen aus VRAM in System-RAM über. Unter macOS feuert dasselbe Suffix, wenn das Modell nur passt, indem es das OS dazu drängt, andere Apps zu verdrängen, um Speicher für die Gewichte zu verdrahten, was technisch lädt, aber spürbar ist. Ein "Recommended"-Badge geht an das bestbewertete Modell, das bei Great fit landet (nicht Tight); wir empfehlen bewusst keine Tight-fit-Modelle, selbst wenn sie bei roher Qualität höher scoren würden, weil es schlechter ist, jemanden zu einem Modell zu schubsen, das spürbar langsamer läuft, als es sollte, als zu einer Stufe darunter, die sauber läuft.

Abbruch, der wirklich funktioniert. llama.cpp-Generierung läuft in einem Worker-Isolate, damit der UI-Thread reaktionsfähig bleibt, mit Token-Streaming zurück ins Main-Isolate über einen NativeCallable.listener . Abbruch läuft über llama_set_abort_callback , das ein atomares Flag prüft, gegen den Upstream-Quellcode verifiziert statt geraten, sodass das Schließen einer Sitzung mitten in einer langen Generierung die Arbeit tatsächlich stoppt, statt sie im Hintergrund fertiglaufen zu lassen und Akku zu verschwenden.

Was wir noch nicht wissen

Die ehrliche Liste:

• Wir haben noch keine großartigen echten Latenzzahlen für eine breite Hardware-Spanne. Wir haben ausführlich auf den Maschinen getestet, die wir besitzen, Beta-Tester befragt und Crash-Reports gelesen, aber der Long Tail von "Windows-Laptop mit integrierter Grafik von 2018" ist schwer zu charakterisieren. Frühe Produktionsdaten werden uns zeigen, wo die rauen Kanten tatsächlich liegen. In unseren eigenen Tests haben wir lokale Inferenz je nach Länge des Meetings, Kontext und gewähltem Modell zwischen fünf Sekunden und fünf Minuten laufen sehen.

• Wir kennen noch nicht den richtigen Rhythmus für neue Modellversionen. Katalog-Churn ist bereits real (Qwen 3.5 !’ 3.6 in einem Zyklus). Bisher haben wir das durch Archivieren statt Entfernen gelöst, aber wenn die Speichernutzung insgesamt aus dem Ruder läuft, brauchen wir vielleicht eine andere Policy.

• Stromverbrauch auf Laptops wird manche Nutzer bei Continuous-AI-Workloads überraschen. Ein Meeting mit lokal laufenden Automatic Suggestions ist deutlich anspruchsvoller als dasselbe Meeting auf Cloud AI. Wir haben noch kein batterieabhängiges Throttling gebaut, und sollten es wahrscheinlich. Der Fall, in dem jemand am Ende des Meetings nur eine Zusammenfassung möchte, ist eine andere Geschichte: Das ist eine einzelne One-shot-Generierung, und auf jedem Rechner, der ein Pro- oder Max-Stufen-Modell bequem ausführen kann, liegt sie gut innerhalb typischer Akku- und Thermal-Spielräume.

Wohin das führt

Die interessante Geschichte ist nicht Hedy 3.2 im Speziellen. Es ist, dass die Kurven weiter wandern.

Open-Weight-Modelle werden in kleinen Größen immer besser. Die 2B-Klasse ist heute ungefähr dort, wo die 7B-Klasse vor achtzehn Monaten war. Wie auch immer die nächsten zwölf Monate von Gemma, Qwen und den anderen ernsthaften Open-Weight-Familien aussehen, die Untergrenze dessen, "was auf einem normalen Laptop läuft", steigt weiter.

Consumer-Hardware wird leistungsfähiger. Apples Chip-Generationen haben Inferenz leise in die Silicon-Roadmap eingebaut. Windows beginnt, NPUs auszuliefern, die irgendwann für Inferenz relevant werden, auch wenn die heutige erste Generation vor allem eine Marketinggeschichte ist.

Die Kombination bedeutet, dass eine leise Verschiebung im Gange ist. KI bedeutete früher: "Einige wenige Unternehmen betreiben große Modelle in Ihrem Auftrag, und Sie senden ihnen Ihre Daten." Daraus wird: "Einige wenige Unternehmen trainieren große Modelle, aber Sie können sie auf Ihrem eigenen Gerät mit Ihren eigenen Daten end to end ausführen." Die gehostete Version ist heute noch besser, und wird es wahrscheinlich noch eine Weile bleiben. Aber die Lücke schließt sich, und für eine relevante Teilmenge von Nutzern schlägt "gut genug und lokal" "am besten und remote".

Hedy 3.2 ist unsere erste konkrete Wette auf diese Verschiebung. Die Architektur ist so gebaut, dass wir den Hebel weiter ziehen können, sobald Modelle und Hardware es erlauben. Es wird mehr geben.

Hedy AI · Live-KI-Coaching für wichtige Gespräche

Hedy kostenlos testen: https://www.hedy.ai/de/downloads/

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