



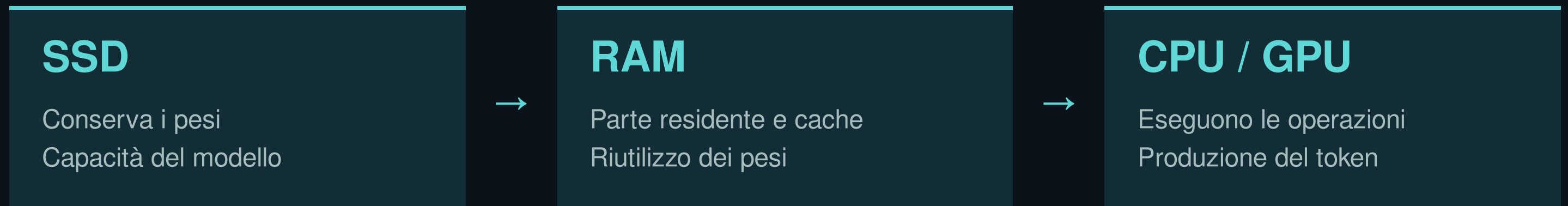
Modelli grandi. *Hardware reale.*

Inferenza in C, pesi distribuiti tra SSD, RAM e GPU.
Il progetto di oggi, la sua architettura e la storia che
lo ha fatto nascere.

Vincenzo Fornaro · Alessandro Tagliati Progetto, architettura e demo

Un motore che organizza *memoria e calcolo*.

Colibrì è un motore di inferenza scritto in C. Esegue modelli locali distribuendo i pesi tra SSD, RAM e memoria della GPU, e porta al calcolo le parti richieste dal modello.



La sparsità diventa una leva di sistema

In un MoE ogni token attiva una parte degli esperti. Colibrì usa questa selezione per caricare, conservare e distribuire il lavoro tra i dispositivi disponibili.

Dal motore alle applicazioni

Chat, API, Colibrì System One e Qwen Image condividono gli strumenti di esecuzione. Brain e Profiling mostrano routing, residenza e tempi.

Un esperimento *condiviso*

≈40.000

stelle

Un segnale di interesse per un motore aperto che chiunque può leggere, provare e migliorare.

La comunità estende il supporto hardware, individua problemi riproducibili e contribuisce nuovi percorsi di esecuzione.

4.352

fork su GitHub · 5 ottobre

13

motori nel ramo di sviluppo



82 su 95

pull request della release 1.12.1 arrivano dai contributor.
Codice, misure e contributi sono pubblici nel repository.

Chat, immagini e *decisioni*

Conversazione e strumenti

CLI, dashboard web, streaming e tool calling nei modelli compatibili. Input visivi sulle famiglie con visione.

Colibrì System One

Domande a scelta chiusa con punteggi delle opzioni. Modelli di chat oppure modelli dedicati alle decisioni.

Generazione immagini

Qwen-Image-2.1 produce PNG con anteprime progressive, dal web e direttamente nel terminale.

UN GATEWAY, PIÙ INTERFACCE

`/v1/chat/completions`

Chat compatibile OpenAI

`/v1/messages`

Interfaccia compatibile Anthropic

`/v1/systemone`

Decisioni e domande tipizzate

`/v1/images/generations`

Generazione immagini

Brain e Profiling rendono visibili routing, memoria e tempi.

`coli mcp` espone gli strumenti di configurazione agli assistenti.

Dieci motori per il linguaggio

Una famiglia può eseguire più checkpoint compatibili. Dimensioni, formati e funzionalità dipendono dal modello.

MOTORE / FAMIGLIA	MODELLI E DIMENSIONI	CARATTERISTICA DEL PERCORSO
GLM	GLM-5.2 / 5.3 · 744B	INT4 gs64, MTP su 5.2
GLM-5.3-Flash	321B	Visione e tool calling
Inkling	975B	Esperti INT4, dense convertibili
Kimi K3	2,8T	Esperti MXFP4 nativi
DeepSeek V4 Flash	284B e variante REAP 150B	Tool calling, esperti FP4
DeepSeek V4.1 Flash	552B	Visione, tool calling, memoria n-gram
MiMo-V2.6	Flash 309B / Pro 1,02T	Visione e tool calling
Qwen3.8-Flash-Next	125B + 51B n-gram	Visione, tool calling, MTP
Qwen3.6 e derivati	35B-A3B, Coder 30B, Qwen3.8-27B	MoE e percorso dense, Clef
OLMoE	7B	Modello compatto, container INT8

In più: Qwen-Image-2.1 per le immagini, Laya e GLiNER2.5-Decide per le decisioni. Totale: 13 motori.

GENERAZIONE IMMAGINI · DEV

Qwen-Image-2.1

in puro C

Un prompt diventa un'immagine. Il checkpoint ufficiale si carica senza conversione preventiva, con pesi quantizzati durante il caricamento.

33 GB

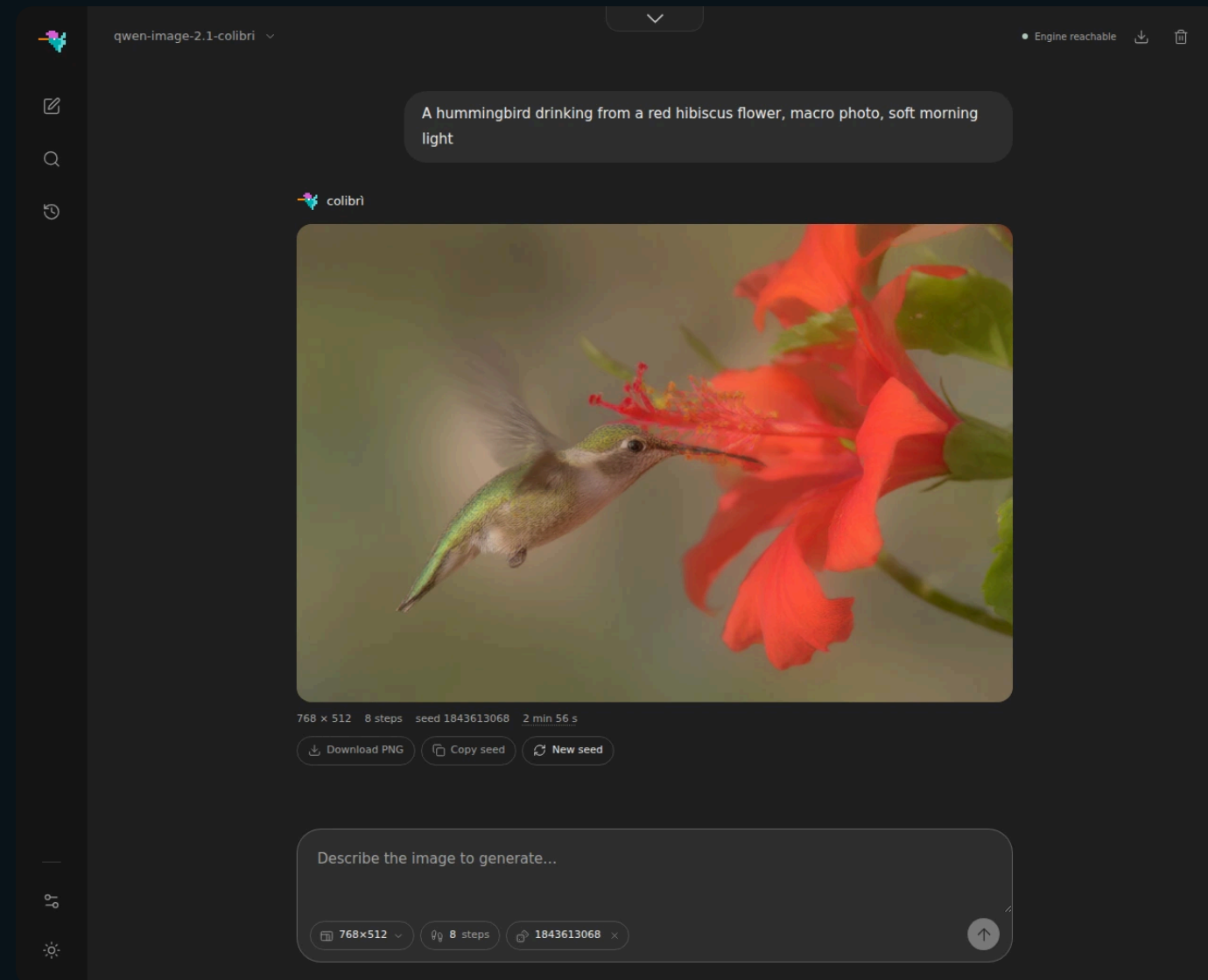
checkpoint su disco

768 × 512

esempio mostrato, 8 step

Web, terminale e API condividono seed, risoluzione e numero di step. Le anteprime mostrano l'immagine mentre prende forma.

Qwen Research License: uso non commerciale, pesi non redistribuibili. La licenza del modello è distinta da quella del motore.



Tre reti.

Fasi diverse.

01 Text encoder

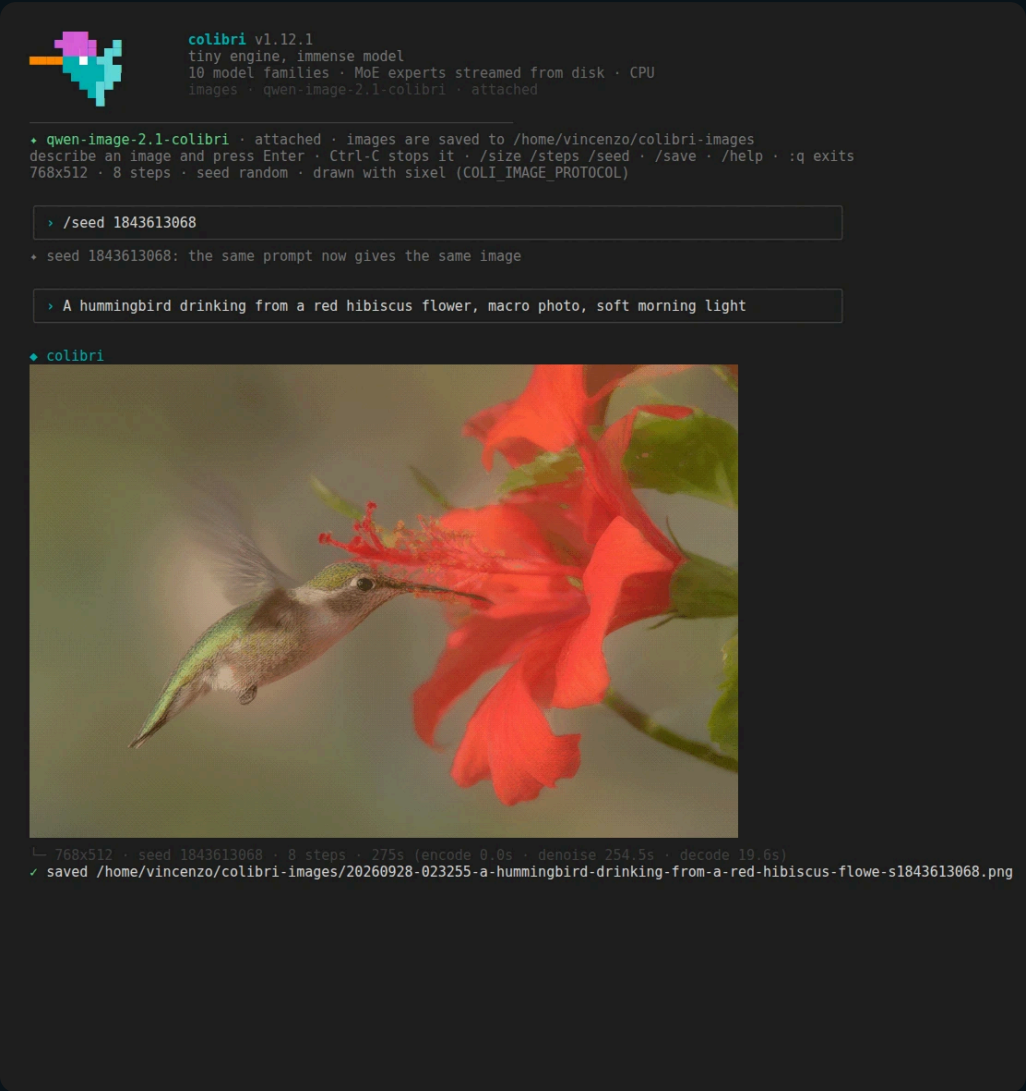
Calcola la rappresentazione del prompt una volta. Può essere liberato prima del denoising.

02 Diffusion Transformer

32 blocchi, iterati per gli step richiesti. Il prefisso del prompt viene riutilizzato.

03 VAE decoder

Trasforma i latenti finali in pixel e salva il PNG.



Le tre reti lavorano in sequenza. Colibrì riutilizza il prefisso del prompt e gestisce la residenza dei pesi tra una fase e la successiva.

La macchina decide *il percorso*

Colibrì esegue i modelli su CPU e integra backend GPU per accelerare le operazioni supportate e conservare pesi sul dispositivo.

PERCORSO	HARDWARE	CHE COSA OFFRE
CPU	Macchine senza GPU dedicata	Inferenza, streaming e cache degli esperti
CUDA	NVIDIA	Esperti in VRAM e catena dense sui motori supportati
Vulkan	AMD, Intel, NVIDIA · driver 1.2	Tier degli esperti, catena dense e percorso DiT per Qwen Image
Metal	Apple Silicon	Calcolo degli esperti su memoria unificata, per più motori
HIP / ROCm	AMD, configurazioni documentate	Backend condiviso con il percorso CUDA, build specifiche

Il backend assegna il lavoro in base a residenza dei pesi, memoria condivisa e costo dei trasferimenti.

CPU e GPU cooperano secondo le operazioni supportate dal backend del modello.

La roadmap di *Colibrì*

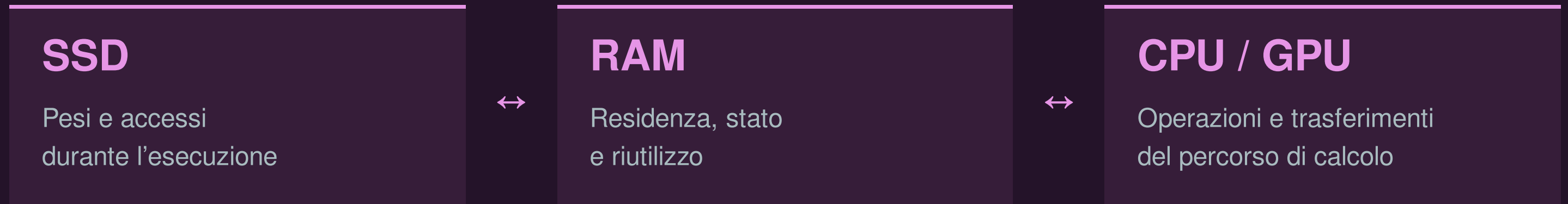
Il punto di partenza era far entrare il modello. Il lavoro ora riguarda tutto il percorso tra pesi, hardware e applicazioni.

L'INIZIO	RELEASE 1.12.1	DEV · 5 OTTOBRE	PROSSIME VERIFICHE
GLM sul laptop Esperti su SSD, cache in RAM e inferenza CPU. La prima prova rende concreto il problema.	Il motore condiviso Nove famiglie, dashboard e osservabilità. Correzioni dalla comunità. Changelog del 24 settembre 2026.	13 motori MiMo, Qwen Image e Colibrì System One. Modelli decisionali e nuovi percorsi Vulkan. Catalogo e backend del ramo dev.	Prestazioni riproducibili Planner, routing e più SSD. Test di sovrapposizione CPU / GPU su carichi diversi. Un ciclo di sviluppo guidato da misure su hardware reale.

Curie · in training Il nostro LLM, progettato per la gerarchia SSD / RAM / CPU / GPU. Modello e motore evolvono insieme.

Stiamo allenando un modello *per questa gerarchia.*

Curie è il nostro LLM: lo sviluppiamo e lo alleniamo pensando fin dall'inizio al percorso SSD → RAM → CPU / GPU. L'architettura del modello e il motore di esecuzione diventano parti dello stesso lavoro.



La domanda di ricerca

Come progettare un LLM tenendo conto dei dati da muovere, della memoria disponibile e del lavoro necessario per ogni token?

Un ciclo ancora in evoluzione

Architettura → training → valutazione → esecuzione in Colibrì. Qualità del modello e comportamento sull'hardware guidano le iterazioni successive.

Prefill: trasformare il prompt *in stato riutilizzabile*.

01 · PREFILL

Spiega

come

funziona

Colibrì

Il testo viene tokenizzato e attraversa i livelli del modello. Il prefill costruisce KV cache e stato ricorrente; i logits dell'ultima posizione producono il primo token.

02 · DECODE

Colibrì

→

usa

→

...

Ogni passo usa il token appena scelto e lo stato già calcolato per produrre quello successivo.

Chunk ed esperti condivisi

Qwen3.8 e Kimi raggruppano i token in chunk. Il motore raccoglie gli esperti richiesti dalle righe e riusa i pesi caricati; proiezioni a blocchi aumentano il lavoro per lettura.

Due misure diverse

TTFT: tempo dall'invio al primo token.

Prompt tok/s: velocità di elaborazione dell'ingresso. Riutilizzare un prefisso già valido riduce il prefill necessario.

Ogni token percorre *il sistema*.

01

Legge il contesto

Il token entra nei livelli del modello. L'attenzione consulta la KV cache; nei motori ibridi avanzano anche gli stati ricorrenti.

02

Attiva gli esperti

Il router seleziona gli esperti del livello MoE. Colibrì riusa quelli residenti e legge da SSD i pesi mancanti.

03

Calcola il seguito

CPU e GPU eseguono il lavoro assegnato. Dopo l'ultimo livello, i logits e la politica di sampling determinano il token successivo.

Il ciclo riparte dal token scelto

Lo stato viene aggiornato e riutilizzato nel passo seguente. La generazione termina con un token di fine, una condizione di stop o il budget richiesto.

La velocità dipende dal percorso critico

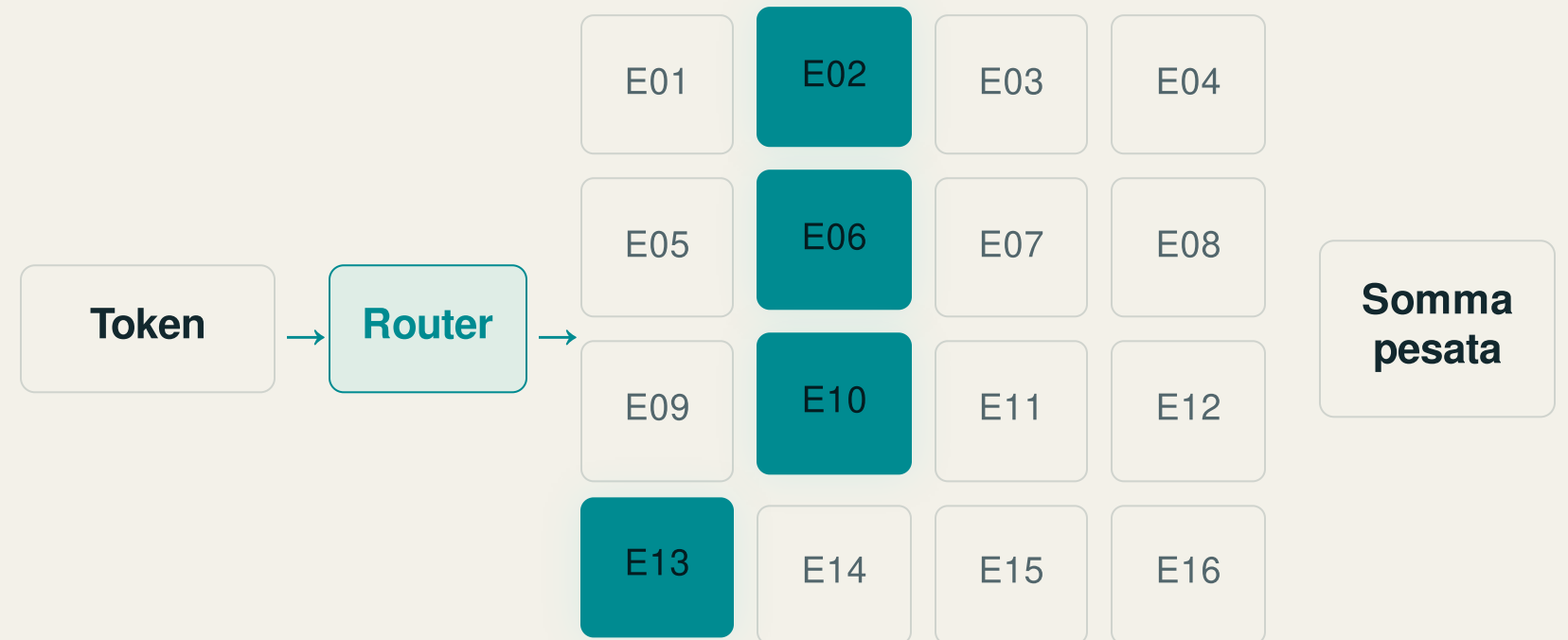
Cache hit, banda SSD, trasferimenti e calcolo determinano il tempo per token. Più riuso dei pesi significa meno dati da muovere a ogni passo.

MIXTURE OF EXPERTS

Molti esperti. Solo alcuni *per token.*

Un router assegna punteggi e seleziona un sottoinsieme di reti, gli “esperti”. I loro risultati si combinano e il calcolo continua.

Ogni esperto è una rete con i propri pesi. Il routing si apprende durante il training e distribuisce il lavoro in funzione della rappresentazione del token.



Schema didattico: 4 su 16.

GLM-5.2 seleziona 8 su 256 per livello MoE.

Token 1: esperti E02, E06, E10, E13.

Dove vivono i pesi. Dove si calcola.

L'SSD conserva il modello. La memoria veloce ospita ciò che conviene tenere vicino al calcolo. La CPU può eseguire gli esperti; una GPU aggiunge un percorso opzionale.

SSD / NVMe

La casa dei pesi completi e degli esperti che mancano nella cache.

Capacità elevata, letture più costose



RAM

Parte residente del modello, cache degli esperti e stato di esecuzione.

Gli esperti frequenti possono restare qui



VRAM

Pesi residenti sul dispositivo per il backend GPU disponibile.

Il budget varia con l'hardware

Vista logica semplificata.
Su alcuni sistemi CPU e GPU condividono la memoria.

CPU

Calcolo dai pesi in RAM

GPU, se presente

Calcolo con il backend supportato

1. L'esperto esiste sul disco. Il router determina se serve al token corrente.

Quali dati stiamo *tenendo in cache*?

PESI · IL MODELLO

Cache degli esperti



Conserva matrici riutilizzabili da token e richieste diverse. Un hit evita di rileggere o trasferire quei pesi.

Il router decide quali esperti servono. Il backend decide dove eseguirli.

ATTIVAZIONI · IL CONTESTO

KV cache e stato ricorrente



Conserva il risultato dell'elaborazione di una specifica sequenza. Riutilizzarlo evita di rifare il lavoro sul prefisso.

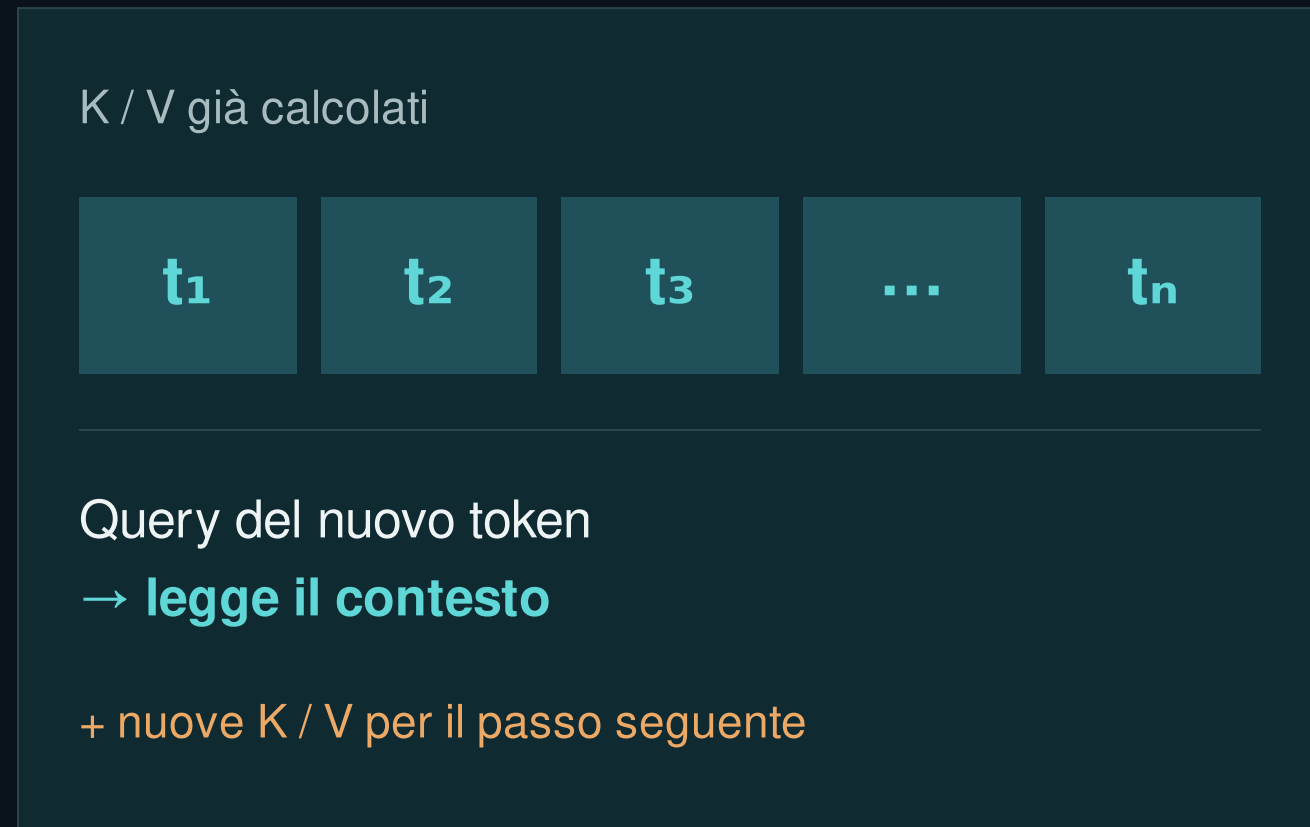
Il contenuto dipende dai token, dal modello e dal percorso di esecuzione.

BUDGET DI MEMORIA

Pesi residenti + cache esperti + stato del contesto + workspace

Aumentare il contesto o gli slot può lasciare meno RAM agli esperti e aumentare le letture dal disco.

Il contesto ha una *rappresentazione*.



Schema dell'attenzione: i K / V passati vengono riutati; il nuovo token aggiunge il proprio stato.

MLA compatta lo stato dell'attenzione. INT4 gs64 riduce i byte dei pesi: due leve complementari sul budget di memoria.

GLM: MLA compressa

Colibrì conserva la rappresentazione latente dell'attenzione: **576 float per token** invece di 32.768 nel confronto riportato dal progetto.

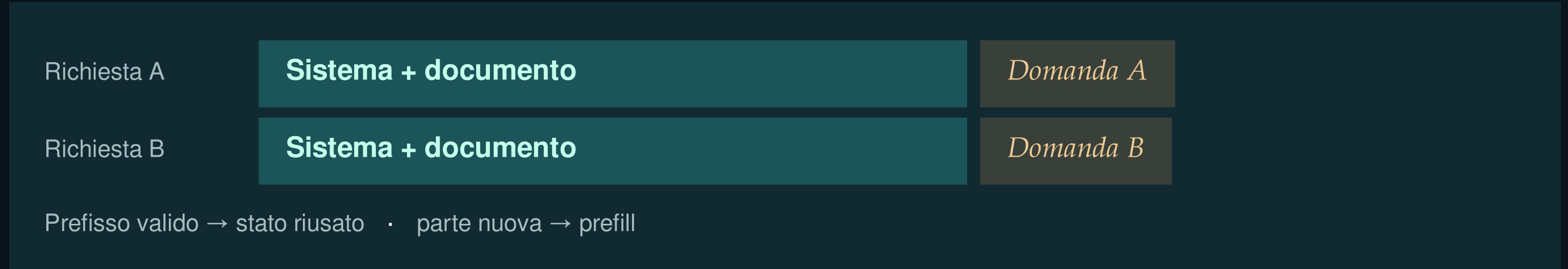
≈57× meno stato in quel confronto per livello

Qwen3.8 e Kimi: stato ibrido

Alle cache dell'attenzione si aggiungono componenti ricorrenti. La ripresa conserva insieme attenzione e stato ricorrente, così la sequenza continua dal punto già elaborato.

UNA NUOVA RICHIESTA NON RIPARTE SEMPRE DA ZERO

Riutilizziamo il *lavoro già fatto*.



GLM · slot e disco

Confronto del prefisso tokenizzato. Fino a 16 contesti isolati con `--kv-slots; file.coli_kv` per riaprire la chat.

Qwen3.8 · snapshot

Riusa l'intero prefisso salvato quando il nuovo prompt lo estende. Prompt più corto o diverso: reset dello stato ibrido.

Kimi · checkpoint

Checkpoint ricorrenti opzionali, in RAM o su disco: ripristino del punto valido più avanzato e prefill della coda.

Ogni slot conserva la propria sequenza. Una coda FIFO con capacità e timeout configurabili regola l'ammissione delle richieste HTTP.

La memoria decide *quando è pronto.*

Il router stabilisce quali esperti servono. Colibrì trasforma quella selezione in letture, accessi alla cache e lavoro per CPU e GPU.

01

Il router seleziona

Per ogni livello MoE identifica gli esperti da eseguire. Su posizioni elaborate insieme, il motore unisce le liste per evitare letture duplicate.

02

La cache risponde

Un esperto già residente può partire subito. Per un miss, il motore avvia la lettura dal disco e prepara i pesi richiesti.

03

Il calcolo prosegue

CPU o GPU eseguono le operazioni supportate. I contributi si combinano e la storia di utilizzo aggiorna le politiche di collocazione.

Routing, cache e collocazione lavorano insieme: lo stesso peso può servire più token prima di essere sostituito.

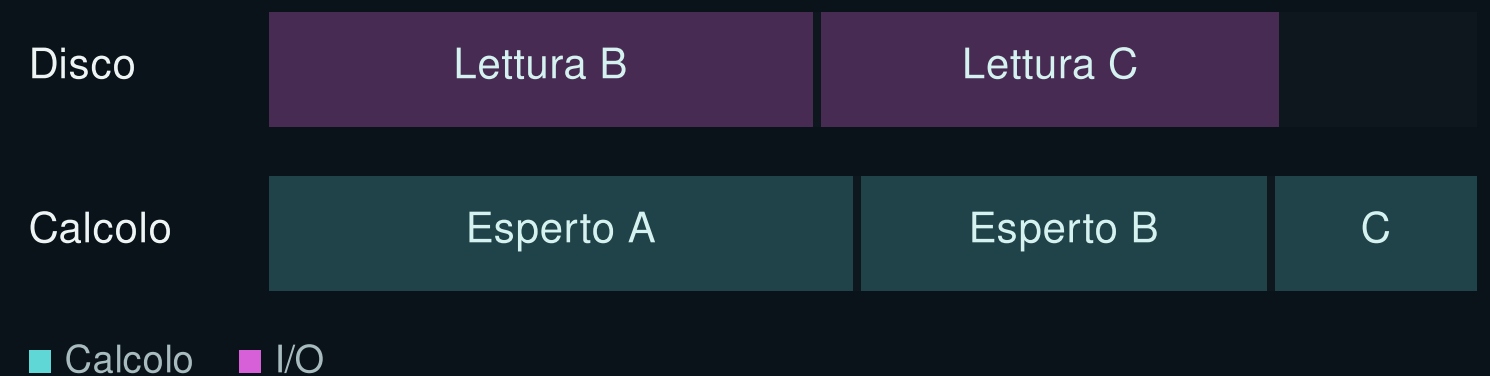
EVITARE L'ATTESA

Il disco è lento. La strategia è *aspettarlo meno.*

La cache riutilizza gli esperti caricati. Il pinning mantiene quelli più frequenti. L'I/O asincrono prova a leggere i mancanti mentre altri esperti calcolano.

Il prefetch anticipa gli esperti probabilmente necessari. Cache, pinning e letture asincrone trasformano la storia di utilizzo in meno attese nel passo successivo.

Un esempio di sovrapposizione



Una parte del tempo di lettura può sovrapporsi al tempo di calcolo.

Schema concettuale, durate illustrative. La sovrapposizione reale dipende da hardware, carico e disponibilità dei pesi.

Una scala *ogni 64 pesi*.

I valori quantizzati hanno bisogno di una scala per approssimare quelli originali. Una scala per piccoli gruppi si adatta meglio alle differenze locali rispetto a una sola scala per tutta la riga.



$64 \text{ pesi} \times 4 \text{ bit} + 1 \text{ scala FP32} = \mathbf{36 \text{ byte per gruppo}}$

32 byte di pesi e 4 byte di scala: **4,5 bit per peso** nel caso illustrato, prima di padding e altri dettagli del container.

Gruppi piccoli seguono meglio la distribuzione locale dei pesi. Il kernel ricostruisce i valori usando la scala del gruppo durante il prodotto con le attivazioni.

Vulkan: gli esperti residenti *lavorano sulla GPU.*

Gli shader compute eseguono le operazioni sulle GPU AMD, Intel e NVIDIA compatibili. Un tier adattivo conserva gli esperti nella memoria del dispositivo e li riutilizza durante prefill e decode.

GPU · ESPERTI RESIDENTI

Un batch asincrono

Gli esperti già sul dispositivo vengono eseguiti insieme. Un hit nel tier GPU evita sia la lettura da SSD sia l'uso di uno slot della cache RAM.

CPU · ESPERTI RESTANTI

Lettura e calcolo in parallelo

Mentre la GPU lavora, la CPU carica ed esegue gli altri esperti. Il motore riunisce i contributi secondo i pesi assegnati dal router.

La residenza segue l'utilizzo

Il tier può partire dalla storia `.coli_usage`, poi si adatta alle richieste entro il budget di VRAM configurato.

Oltre gli esperti

I percorsi implementati includono proiezioni dense e, su GLM, attenzione MLA con KV sul dispositivo. Il prefill usa anche prodotti matrice-matrice a blocchi.

Proporre. Verificare. *Accettare.*

01 · DRAFT

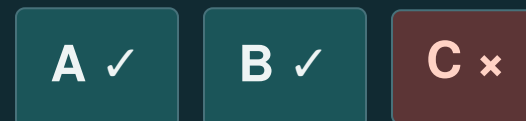
Una proposta economica



La testa MTP propone token futuri usando lo stato del modello.

02 · VERIFY

Il modello controlla



Un forward verifica la proposta. Il risparmio dipende da quanta viene accettata.

03 · COMMIT / ROLLBACK

Lo stato resta coerente



Si conserva il tratto valido e si ripristina lo stato della parte rifiutata.

Come lo gestiamo

GLM-5.2 usa la testa MTP in INT8. Qwen3.8 verifica due righe e ripristina anche DeltaNet, PLE e attenzione quando il draft non coincide.

Più token per forward

L'accettazione permette di riusare il lavoro della verifica. Il profiler misura token per forward e tok/s per scegliere il percorso più efficace sulla macchina.

Schema generale illustrativo. Il numero di token proposti e le garanzie numeriche dipendono da motore e backend.

Tre motori. Priorità diverse.

Colibrì mette al centro lo streaming MoE. llama.cpp offre un ecosistema per l'inferenza locale. vLLM organizza il serving di molte richieste concorrenti.

Colibrì	Streaming degli esperti MoE e collocazione esplicita fra livelli di memoria.	SSD ↔ RAM / CPU ↔ VRAM / GPU
llama.cpp	Inferenza locale versatile, mmap e configurazioni ibride CPU / GPU.	File / mmap ↔ CPU + GPU
vLLM	Serving concorrente, batching e gestione della KV cache. Include opzioni di offload.	RAM / offload ↔ GPU / serving

La scelta del motore parte dal carico: capacità del modello, memoria disponibile, latenza e numero di richieste da servire.

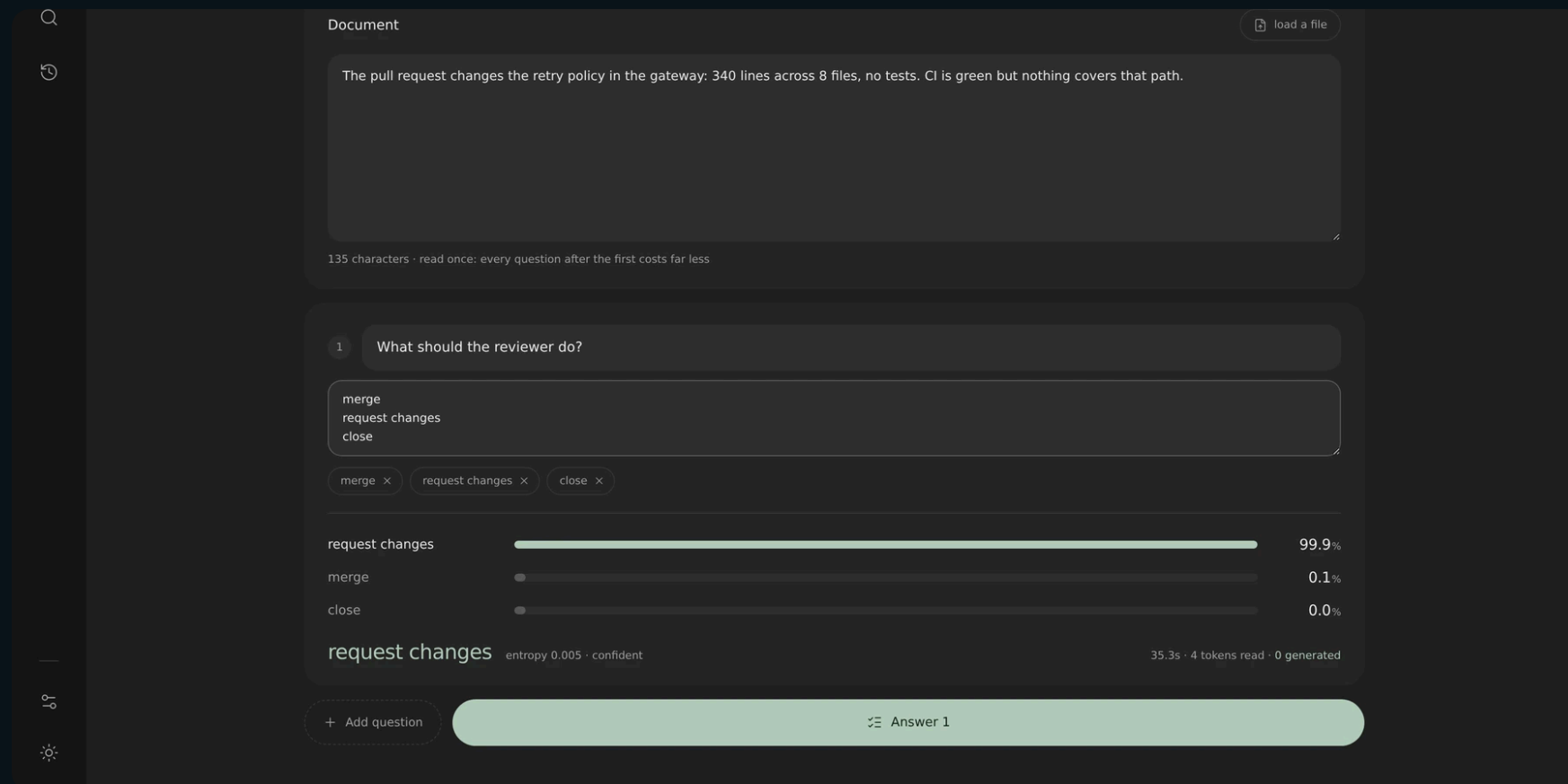
DOMANDE CHIUSE E MODELLI DECISIONALI

Colibrì *System One*

Uno stato, anche JSON, e domande tipizzate: una scelta, sì o no, oppure un punteggio. Le scelte restano nelle opzioni ammesse.

I modelli di chat assegnano punteggi alle opzioni. **Laya**, **GLiNER2.5-Decide** e **Clef** offrono percorsi nativi per le decisioni.

POST /v1/systemone



0 token di risposta generati.

Scelta strutturata e riuso del documento.

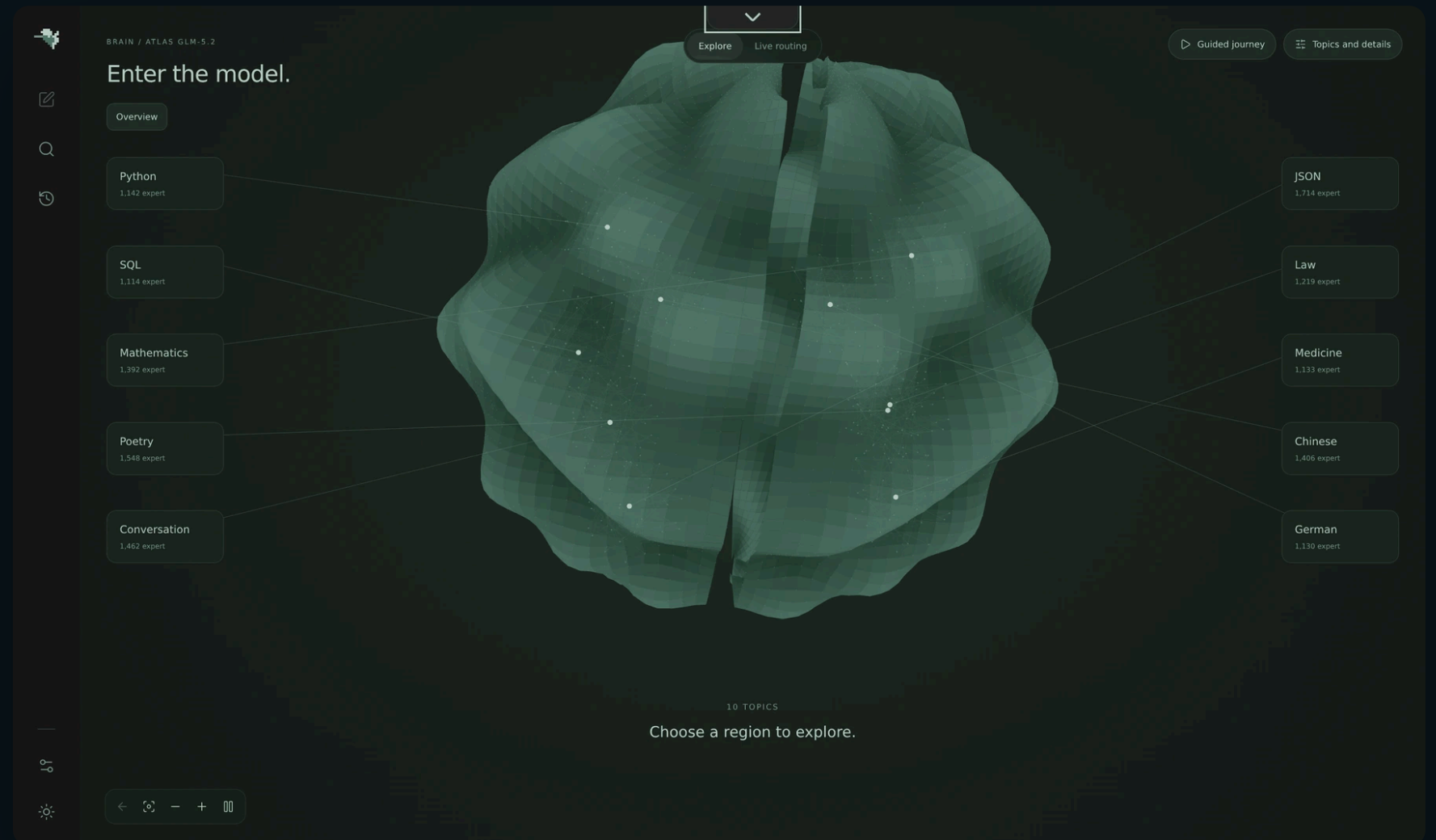
Il documento viene elaborato e riutilizzato per valutare le opzioni. L'applicazione riceve una scelta strutturata. Nei modelli di chat, i punteggi sono relativi alle opzioni proposte.

OSSERVABILITÀ

Un modello che puoi anche *guardare.*

Brain mostra l'atlante delle affinità misurate. La vista Live routing segue invece gli esperti del modello in esecuzione e il loro livello di memoria.

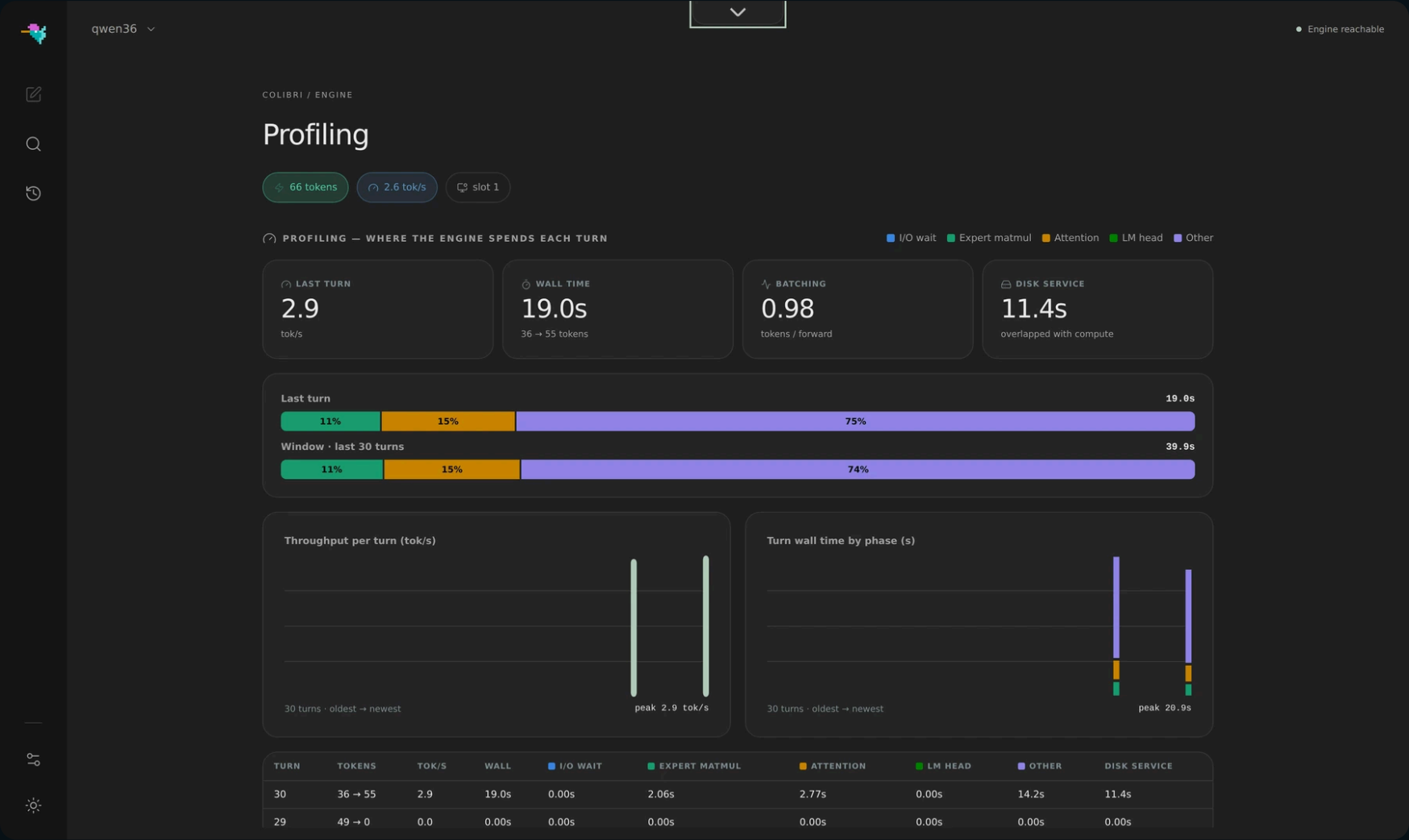
Le affinità del routing rendono esplorabile il comportamento del modello: quali esperti si attivano insieme, quanto vengono riusati e su quale memoria lavorano.



Misurare. Capire. *Migliorare.*

Profiling scompone il tempo in letture, trasferimenti e operazioni. Collega il comportamento del modello alle risorse della macchina.

La comunità porta nuove configurazioni, individua i colli di bottiglia e trasforma le misure in miglioramenti del motore.



Il progetto chiede di misurare correttezza e qualità insieme a latenza, throughput, memoria e costo.

IL PRIMO RICHIAMO

Mi ero dimenticato di *divertirmi*.

Inseguivo la prossima startup da creare, e quella startup non arrivava mai. Intanto avevo perso il gusto di costruire.

Una frase del fondatore di OpenClaw mi ha ricordato l'importanza di divertirmi: scegliere un problema che avessi davvero voglia di capire.

Prima di chiedermi se sarebbe diventata una startup, potevo chiedermi se avevo davvero voglia di costruirla.

Colibrì nasce da questo cambio di prospettiva.



*Ci sono tante cose nel tuo mondo,
perché tu non fai niente
per cambiarlo?*

Don Giancarlo Minotta, un mio amico parroco

Il primo richiamo mi aveva restituito il piacere di costruire.
Questa domanda mi chiedeva cosa volessi farne.

Ho scelto un problema che mi interessava davvero, senza
aspettare di avere la macchina perfetta o tutte le risposte.

IL PRIMO RISULTATO

Era lento.
Ma *rispondeva.*

Nel primo post raccontavo circa **un token ogni dieci secondi**. L'esperimento era lontano da una chat veloce.

Eppure, il modello stava funzionando sul mio computer. Per me contava la gioia di esserci arrivato, e la voglia di continuare.

~0,1

token al secondo, all'inizio

| “I just wanted it to work at all costs, even slowly.”

Dal post originale su Hacker News. Un risultato storico, non la prestazione attuale del progetto.

«Sul mio computer lento»

Ho pubblicato quello che avevo provato, compresi i limiti, e ho chiesto feedback. Un esperimento personale diventava una conversazione aperta.

* Show HN: Getting GLM 5.2 running on my slow computer (github.com/justvugg)

937 points by vforno 87 days ago | [hide](#) | [past](#) | [favorite](#) | [240 comments](#)

A few days ago I found myself trying out GLM 5.2 and was really positively impressed. The capabilities and security I was getting from this LLM are similar to those I've gotten from models like Claude or GPT, and this really surprised me.

But then I thought, "I wonder how it would work on a normal computer like mine," and above all, "I wonder if it would work without going into OOM on a computer like mine." So I started working with the help of agents to test this possibility.

I started converting the model to int4, understanding MTP usage, and if possible implementing DSA for long context. How it responds in int4 and whether the quality is maintained or not. Until I got to the point, on my computer with 32GB of RAM, I was able to communicate with GLM 5.2 with times that, of course, aren't high in cold start, but even then, we're talking about 0.1 tok/s, but that wasn't important to me. The important thing was the journey to reach this goal. I just wanted it to work at all costs, even slowly.

So I created Colibrì, which was born from a very simple idea, to be honest, but tested in every way, where a 744B Mixture-of-Experts model activates only ~40B parameters per token—and only ~11 GB of those change from token to token (the routed experts). So:

The dense part (attention, shared experts, embeddings—~17B params) stays resident in RAM at int4 (~9.9 GB); The 21,504 routed experts (75 MoE layers × 256 experts + the MTP head, ~19 MB each at int4) live on disk (~370 GB) and are streamed on demand, with a per-layer LRU cache, an optional pinned hot-store, and the OS page cache as a free L2.

The engine is a single C file (`c/glm.c`, ~1,300 lines) plus small headers. No BLAS, no Python at runtime, no GPU. No GPU or serious hardware because I don't have that hardware so I can't test it on hardware that is more powerful than my computer. Colibrì is a one-person project, written and tested entirely on a 12-core laptop with 25 GB of RAM — the numbers above are the ceiling of what I can measure at home.

Any feedback is welcome! (and if anyone wanted to participate in the project I would be delighted)

Repo: <https://github.com/JustVugg/colibri>

Io avevo una macchina. Il progetto ne ha incontrate *molte*.

Publicare il codice permetteva ad altre persone di entrare nello stesso problema.

Hardware diversi, nuove prove e risultati che io, da solo, non potevo raccogliere.

Chi aveva più memoria poteva misurare la residenza.

Che cosa cambia quando gli esperti smettono di arrivare dal disco?

Chi aveva una GPU poteva esplorare un altro percorso.

Quanto aiutano il calcolo e quanto costano i trasferimenti?

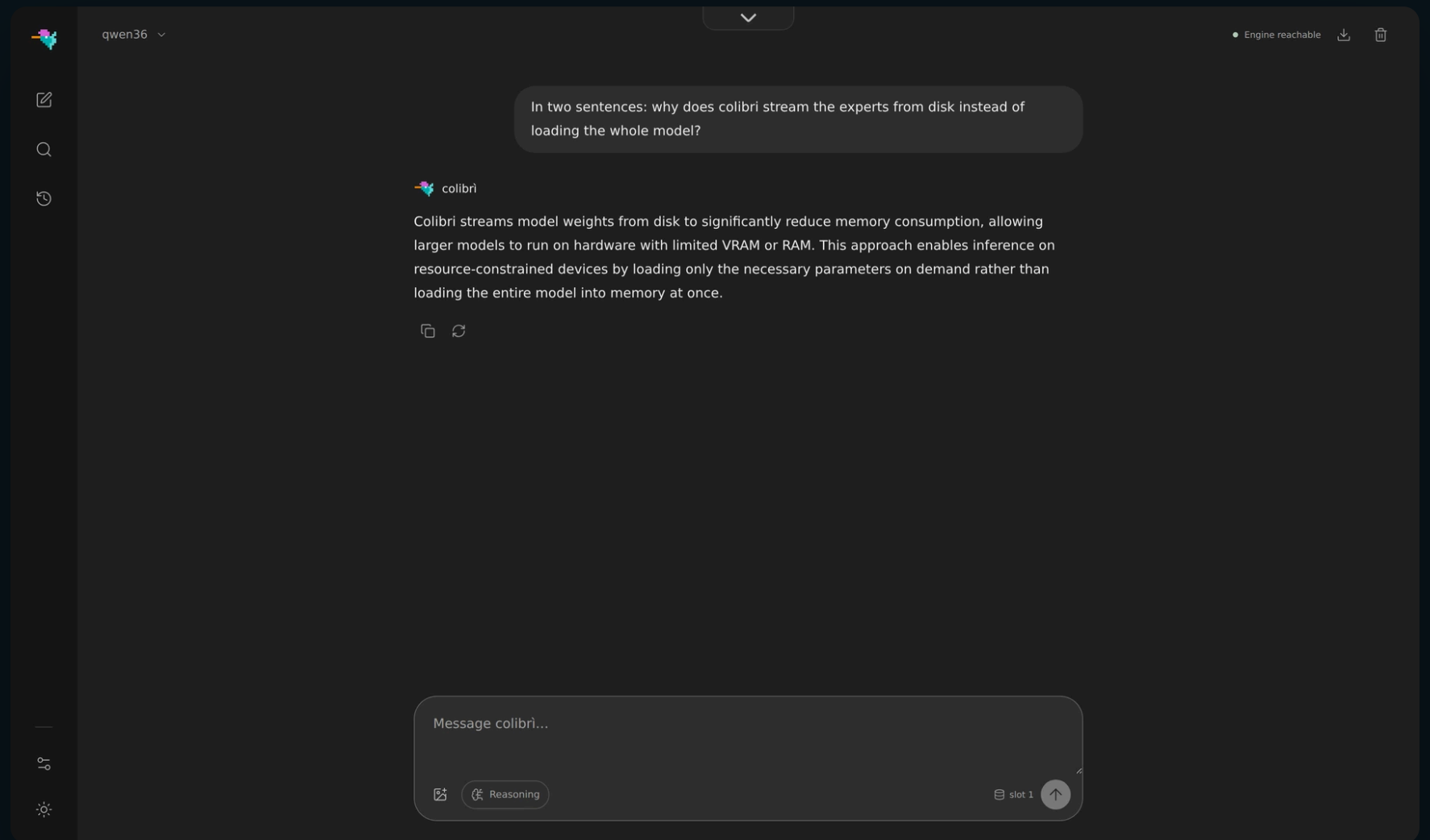
Chi trovava un problema poteva renderlo riproducibile.

Una segnalazione precisa può valere quanto un'ottimizzazione.

DEMO DAL VIVO

Adesso, *lo vediamo.*

Chat, Colibrì System One e generazione immagini. Prima mostriamo hardware e modello, poi seguiamo routing e tempi reali.



01

Chat e Brain

02

System One

03

Qwen Image



Ho ricominciato a divertirmi. E ho iniziato a *cambiare qualcosa.*

Due persone mi hanno rimesso in movimento. La comunità ha portato l'esperimento molto oltre il mio computer. La domanda di don Giancarlo, però, resta la stessa.

github.com/JustVugg/colibri ↗

Il modello è grande. Il lavoro per token è *più piccolo*.

Il README descrive 744 miliardi di parametri complessivi e circa 40 miliardi attivi per token.

Tutti i pesi vanno conservati. Solo una parte partecipa al calcolo di ciascun token. Questa differenza rende interessante lo streaming degli esperti.

744B

parametri complessivi

~40B

attivi per token



Circa **5,4%** dei parametri è attivo per token.

Visualizzazione approssimata del rapporto, non una mappa degli esperti del modello.

QUANTIZZAZIONE

Ogni byte
che non muovi
può aiutare.

INT4 rappresenta un peso quantizzato con 4 bit, invece dei 16 bit di FP16. Il payload scende, in prima approssimazione, a un quarto.

Servono però scale e metadati, e alcuni tensori restano a precisioni diverse. **La quantizzazione approssima i pesi:** la qualità deve essere misurata insieme alla velocità.

16 bit_{FP16}



4 bit_{INT4}



Rapporto del payload rispetto a FP16, prima delle scale e delle eccezioni. Compressione con perdita.

Far partire un modello è il primo passo.

Quanto va veloce?

Quando molti esperti arrivano dal disco, l'I/O può dominare. Con più residenza, il collo di bottiglia può spostarsi sul calcolo.

Questi esempi del README descrivono macchine e condizioni diverse. Non sono una curva di scaling né un confronto con altri motori.

25 GB disponibili

GLM-5.2, streaming a freddo

0,05–0,1

token / secondo

Ryzen AI Max+ 395, 128 GB

GLM-5.2, cache calda

1,83

token / secondo

6 × RTX 5090

GLM-5.2, residenza completa

9,0–9,2

token / secondo

Risultati riportati dal progetto. Configurazioni e fonti nelle note.

Più SSD. Memoria NUMA. *Cluster locale.*

01

Mirror su più SSD

Copie identiche dei pesi su dischi indipendenti. Il motore distribuisce le letture e può usare anche un mirror parziale.

+37,5%

decode in una misura con due NVMe

Risultato specifico, da riprodurre su altri controller e carichi.

02

NUMA e residenza

I pesi residenti possono distribuirsi sui controller di memoria. Se tutti gli esperti entrano in RAM o VRAM, il disco esce dal decode.

`COLI_NUMA=1`

La configurazione migliore dipende dalla macchina e dal modello.

03

Worker nella rete locale

Il coordinatore mantiene token, routing e KV. I worker eseguono gli esperti richiesti, raggruppati per livello.

`coli cluster worker`

Trasferimenti e latenza di rete diventano parte del profilo.

I dettagli che cambiano il risultato

RAM totale

I circa 9,9 GB della parte residente citati per GLM-5.2 non sono il fabbisogno totale: servono anche KV cache, buffer, cache degli esperti e sistema operativo.

MTP

Il modello verifica i token proposti dal draft. La speculazione aiuta soltanto quando l'accettazione ripaga il costo della verifica. Il percorso GLM-5.2 documentato richiede la testa MTP int8.

MLA

Il caso documentato usa 576 anziché 32.768 float per token per lo stato KV. Il rapporto di circa 57× riguarda quello stato, non tutta la memoria della macchina.

Container e licenze

Dimensioni, formati e requisiti cambiano fra modelli. La licenza del motore e quella dei pesi sono distinte e vanno controllate per il modello usato.

Il materiale dietro il racconto

Il progetto

Catalogo e novità: ramo **dev fd145ef**, 5 ottobre 2026. Formato INT4 e riferimenti iniziali: **main ce370e8**.

Logo e schermate originali del repository. Qwen Image: schermate reali della PR #1780.

I confronti

Documentazione ufficiale di llama.cpp e vLLM. Le descrizioni non costituiscono un benchmark comparativo.

La storia

Racconto personale di Vincenzo Fornaro e post originale su Hacker News. La domanda di don Giancarlo Minotta accompagna la nascita del progetto.

Le immagini illustrative

Sfondo della scrivania generato a scopo illustrativo. Le schermate Qwen Image mostrano output reali di Colibrì.