ChatGPT

ローカルAI（オンデバイス／ローカル推論）技術調査レポート

エグゼクティブサマリ（実務要点）
本レポートは、ローカルAI（端末内・ローカルPC・ローカルLAN内推論）を対象に、カテゴリ網羅・特徴軸評価・Apple Silicon（M1/M2/M3/M4）とWindows（CPU/GPU）前提の実用ライン・ユースケース別推奨スタック（低コスト／高品質／ハード制約）を、一次ソース中心の根拠付きで整理するものです。

ローカルAIの定義は「推論がユーザー端末またはユーザー管理下のローカルLAN内計算資源で完結し、入力データ（文書・音声・画像等）がデフォルトで外部クラウドへ送出されない構成」を指します（ローカルAPIサーバをLAN公開して別端末から使う形も含む）。LANサーブは LM Studio [1] が「localhostまたはnetworkで提供」できること、また Ollama [2] がWindows/macOS/Linuxで提供され、REST APIでローカル利用される設計であることから、実務的に「ローカルLAN内推論」をローカルAIの範囲に含めるのが妥当です。 [3]

技術的に“今”ローカルAIを成立させている中核は、(a) 重み量子化（4bit前後のweight-onlyが主流）と、(b) KVキャッシュ／長文コンテキストのメモリ管理（paged KV、KV量子化、再利用）です。AWQは「on-device LLMが重要（コスト削減・プライバシー）」という問題設定のもと、低bit量子化による実行を狙う代表例です。 [4] さらに、vLLMはpaged KV cache設計を前提にし、KVキャッシュをFP8等に量子化してメモリフットプリントを下げる機能を明示しています。 [5] NVIDIA [6] のTensorRT-LLMもpaged/quantized KV cacheやKV reuseを最適化として公式に強調しています。 [7]

実運用の第一選択になりやすいランタイムは、クロスプラットフォーム汎用の「llama.cpp/GGUF系」と、UI＋ローカルAPIサーバをセットで提供するOllama・LM Studio、Apple Silicon特化のMLX系です。llama.cppはGGUF必須を明記し、複数ハードを対象に最小セットアップでのローカル推論を目的にしています。 [8]
LM Studioは「llama.cpp（GGUF）エンジン＋Apple MLXエンジン」を同梱し、OpenAI互換APIサーバとしても動作します。 [9] Apple Silicon側は、Apple GPUがCPU/GPU同一メモリを共有する“unified memory model”であることがMetal公式ドキュメントとして明示され、MLXもUnified Memory前提で設計されていることが公式に説明されています。 [10]

ハード“実用ライン”の結論（LLM中心、4bit中心、概算）：
- Apple Silicon（Unified Memory）16GB：現実的には3B〜7/8B級（4bit）をインタラクティブに回すライン。M1世代はUMAを強調し、M4世代でも統合メモリ前提で構成される。 [11]
- Apple Silicon 24〜32GB：14B級（4bit）が「RAG含め業務用途の最小実用ライン」になりやすい（同時にASR/TTSを載せても破綻しにくい）。
- Windows（離散GPU）VRAM 8/12/16/24GB：8GBは7B級の最小ライン、12GBで14B級が視野、16GBで“品質寄り”が現実化、24GBで32〜34B級（4bit）や重い画像生成が実務域に入る（ただし長文コンテキストはKVが支配するため別管理）。このVRAMラインはコミュニティ知見が混ざるため、本レポートでは「重みメモリの理論値（計算）」＋「再現ベンチ（後述）」で最終判断する設計を採ります。 [12]
- Windows CPU-only：AVX2以上の多コアCPU＋32GB以上RAMで、7B級4bitが「使える」ライン。Intel CPU最適化はAVX-512/VNNI/AMXを活用し得ることがIntel公式に明示されています。 [13] Intel [14]

推奨スタック（3パターン、総論）：
- 低コスト：GGUF（Q4/K系）＋llama.cpp系（Ollama/LM Studio含む）＋軽量ASR（whisper.cppやfaster-whisperのCPU/INT8）＋軽量TTS（Piper/Kokoro）＋BGE系Embedding（bge-m3）で“最低限の業務自動化”を組む。 [15]

- 高品質：大きいLLM（32B〜70B級）＋reranker（bge-reranker）＋VLM（Qwen2-VL 7B級等）＋GPU優先（NVIDIAならTensorRT-LLM/vLLM/ExLlama等の適材）＋ASRはfaster-whisper GPU、TTSはXTTS/StyleTTS2等を用途に応じ採用。[16]
- ハード制約：3B〜4B級（Phi-3 mini等）＋“短いコンテキストで完結する業務”に寄せ、RAGはEmbeddingを重視してLLMを軽くする。[17] Microsoft [18]

ローカルAIの定義

（0）本レポートにおけるローカルAIとは、「推論（inference）が、ユーザー端末（オンデバイス）・ユーザーのローカルPC・ユーザー管理下のローカルLAN内サーバのいずれかで完結し、入力データがデフォルトで外部クラウドへ送出されないAI実行形態」を指します。LM Studioが「localhostまたはnetworkでローカルLLM APIサーバとして提供できる」こと、OllamaがWindows/macOS/Linuxでローカル APIとして動作することから、ローカルLAN内サーブを明確に包含します。[3]

未指定事項の扱い（明示）：
- “完全オフライン”の要件はユーザー指定がないため、「モデル推論自体はオフラインで成立する」レベルを基本とし、RAGの文書取り込みやモデル取得（初回ダウンロード）は別途ネットワークが必要になり得る点を前提とします。LM Studioは「文書添付（RAG）をオフラインで実行可能」と説明していますが、これは“実行時の外部送信をしない”という意味合いです。[19]
- 対象OSはユーザー指定の仮定どおり macOS Ventura以降／Windows 10/11 とし、Linuxサーバは「Windows上でWSL2や別筐体で併用」の選択肢としてのみ触れます（特にTensorRT-LLMはLinux前提の資料が中心）。[20]

ローカルAIランドスケープ

（1）カテゴリ → 代表モデル／方式 → 代表ランタイム → 主用途（網羅優先）

```
flowchart LR
  A[ローカルAI] --> B[テキストLLM]
  A --> C[VLM/LMM]
  A --> D[ASR]
  A --> E[TTS]
  A --> F[Embedding/Reranker]
  A --> G[OCR/Document AI]
  A --> H[画像生成]
  A --> I[Agent/Tool-use]
  A --> J[音声前処理: VAD/ウェイクワード/ノイズ/話者分離]

  B --> B1[モデル例: Llama 3.1 / Qwen2.5 / Mixtral / Phi]
  B --> B2[ランタイム例: llama.cpp(GGUF) / Ollama / LM Studio / MLX-LM / vLLM / TensorRT-
LLM / ExLlamaV2]
  C --> C1[モデル例: Qwen2-VL / Phi-3-Vision / InternVL2 / LLaVA]
  C --> C2[ランタイム例: MLX-VLM / transformers系 / vLLM(マルチモーダル) 等]
  D --> D1[方式: Whisper系]
  D --> D2[ランタイム例: whisper.cpp / faster-whisper(CTranslate2)]
  E --> E1[方式: 軽量TTS〜音声クローン]
  E --> E2[ランタイム例: Piper / Kokoro / Coqui TTS(XTTS) / StyleTTS2]
```

```
F --> F1[モデル例: bge-m3 / multilingual-e5 / bge-reranker]
F --> F2[ランタイム例: transformers / ONNX / 各種推論サーバ]
G --> G1[方式: 古典OCR + DocAI / OCR-free Doc理解]
G --> G2[ランタイム例: Tesseract / PaddleOCR / Donut / LayoutLMv3]
H --> H1[方式: Diffusion]
H --> H2[ランタイム例: ComfyUI / AUTOMATIC1111 / 各種推論実装]
I --> I1[方式: LLM+ツール(関数呼び出し) + 実行環境]
I --> I2[例: OpenAI互換API + Agentフレームワーク]
J --> J1[モデル例: Silero VAD / openWakeWord / RNNoise / pyannote]
```

上図の“実務上の収束点”は以下です。
- テキストLLMの配布・実行は、(a) GGUF（llama.cpp系）か、(b) GPU向け量子化形式（GPTQ/AWQ/EXL2など）＋CUDA系ランタイムに大別されます。llama.cppはGGUF必須を明記し、GGUF自体もGGML系推論向けバイナリ形式として仕様化されています。 [21]
- macOSは「Metal/Unified Memory」前提の最適化が効く領域があり、MLXはApple Silicon最適化の配列フレームワークとして提示され、MLX-LM/MLX-VLMによりLLM/VLMの実運用が成立しています。 [22]
- Windowsは“GUIで回すなら”Ollama/LM Studioが現実的で、GPUをフルに使うServingは（実務上）Linux/WSL2を選ぶ局面が多いです。OllamaはWindowsインストールを公式READMEに明記しています。 [23]

特徴軸の定義

（2）以降の評価は、同一カテゴリ内でも「モデル」「量子化」「ランタイム」「ハード」で結果が変わる前提で、次の特徴軸を共通物差しとして扱います（未指定だったため本レポートで定義）。

指示追従（Instruction following）
プロンプト制約・禁止事項・形式指定（例：箇条書き禁止、JSONのみ等）を守れる度合い。モデルのSFT/DPO設計差と、ランタイムの“制約付きデコード”有無で大きく変動します（例：JSONモード相当）。

幻覚耐性（Hallucination resistance）
根拠のない具体名・数値・手順を捏造しにくい性質。RAGがあっても「引用外の断定」が混ざるため、出力ポリシー設計（根拠提示・不確実性明記）が必要です。

JSON堅牢性（Structured output robustness）
スキーマ遵守・構文破壊率の低さ。ツール呼び出し（function calling）や業務システム連携の成否を直接左右します。OllamaのOpenAI互換APIはResponses API互換（非stateful）を含み、LM StudioもOpenAI互換エンドポイントを提供します。 [24]
（※llama-cpp-python等ではresponse_formatでJSON限定に寄せる設計が存在しますが、ランタイム依存です。 [25] ）

長文耐性（Long-context stability）
長い入力に対する要約崩れ、重要情報の欠落、自己矛盾の増加。根本制約はKVキャッシュがメモリを線形消費する点で、vLLM/TensorRT-LLMはpaged/quantized KVやKV reuse等の最適化を前面に出しています。 [26]

推論計画（Planning）
タスク分解、複数ステップの自己検証、外部ツール呼び出し順序の妥当性。Agent用途で重要。

ツール呼び出し適性（Tool-use suitability）
関数呼び出しの意思決定、必要引数の抽出、アンビギュイティの解消。Ollamaのローカル実行ガイドではChat Completions経由のtool calling例が提示されています。 27

日本語品質
文法・敬語・語彙、業務文書スタイル、固有名詞の扱い。モデルの多言語設計差が出ます（例：多言語Embeddingや多言語LLMなど）。

音声品質（TTS）
自然さ（抑揚・間・破綻）、日本語アクセント、ノイズ、声質再現（クローン）など。用途により「軽量TTS」か「音声クローン寄り」かを分けます。

画像理解精度（VLM）
OCR的読み取り、図表理解、空間関係、UIスクショ解釈。入力解像度・視覚トークン制御で変動し、MLX-VLMはQwen2-VLなどの例をREADMEで示しています。 28

速度（tok/s, RTF）
- tok/s：LLMの生成スループット。対話UXを決める（目安として“体感”は10 tok/s前後で差が出るが、これは推測でありベンチで確定すべき）。
- RTF（Real-Time Factor）：音声処理の実時間比（処理時間 ÷ 音声長）。ASR/TTSでRTF<1なら実時間より速い。

メモリ／VRAM要求
重み＋KVキャッシュ＋ランタイムオーバヘッド。量子化で重みは縮むが、KVが支配する局面が残るため、長文は別設計。

量子化耐性
4bit等に落としたときの品質劣化の小ささ。モデルと量子方式の相性が大きい。

ライセンス／コンプライアンス
「コードのOSSライセンス」と「モデル重みの利用条件」が別である点が重要（例：画像生成やTTSで重みが非商用制限等）。後述でモデルごとにリスク記載。

カテゴリ別モデルカタログ

（3）各カテゴリで3〜10の代表モデルを「サイズ／推奨量子化／得意不得意／ランタイム相性／リスク／採用判断基準」で整理します。互換性・形式は一次ソースで確認できた範囲を“確認済”、確認できないものは“未確認”と明記します。

テキストLLM（汎用）

代表モデル	規模	推奨量子化・実行形式	得意/不得意（要点）	ランタイム相性（macOS/Windows）	主なリスク・採用判断
Llama 3.1 Instruct（8B/70B/405B系）[29]	8B/70B/405B	ローカル実用は8B/70Bを4bit（GGUF等）中心（“方式は環境依存”）	多言語対話用途最適化を明記。大規模は品質が出るがローカルはハード制約。[30]	mac: GGUF/MLX変換、Win: GGUF/各GPUランタイム（形式依存）	ライセンスがコミュニティライセンス系で用途制約の精査が必要。[31]
Qwen2.5 Instruct（0.5〜72B）[32]	0.5〜72B	小〜中は4bit（GGUF/他）でローカル向き。72Bは上位ハード前提。	幅広いサイズ展開を明記。ライセンスはサイズにより差がある旨を公式ブログで明記。[33]	mac: GGUF/MLX、Win: GGUF/ExLlama/vLLM等（形式依存）	「3B/72BはApache 2.0例外」といった条件差があるためリポジトリ個別確認必須。[33]
Gemma 2（2B/9B/27B系）[34]	2B/9B/27B	2B/9Bはローカル向き。27BはVRAM/メモリ要求増。	モデルカード更新日が明示されており、責任ある利用を前提に整理されている。[35]	mac: MLX/GGUF変換、Win: GGUF/各GPU	“事実質問用途での誤生成”は一般に起こり得る（モデル一般リスク）。
Mixtral 8x7B Instruct（MoE）[36]	MoE（8×7B）	GPU向けで真価が出やすい（量子化/実装依存）。	「Llama2 70Bを多くのベンチで上回る」等をモデル説明で明記。[37]	Win GPU: 適合ランタイム（vLLM等）前提、macは難易度高め	MoEは実装・VRAM・スループットのブレが大きい。
Phi-3 Mini（3.8B）[38]	3.8B	4bitでローカル適性が高い。長文版（128K）の運用はKVが支配。	128K文脈をうたうが、長文はメモリ設計が前提。[38]	mac/Winとも比較的回しやすい（形式依存）	“小さい＝万能”ではなく、専門領域はRAG前提で補う判断。
Phi-4（14B）[39]	14B	4bitでローカル上位ライン（32GB級以上推奨）	データ品質重視・合成データ活用を技術報告で明示。[40]	Win: GPU/CPU最適化次第、mac: MLX/変換次第	14Bは“重いが現実的”の境界。KV/長文設計が重要。
gpt-oss（20B/120B）[41]	20B/120B	“MXFP4量子化で出荷”を明記（他量子化なし）。[27]	ツール呼び出し・ローカルAPI利用までガイドがある。[27]	LM Studio/Ollamaで手順が提示される。[41]	20Bは少なくとも16GB VRAM/統合メモリ要求を明記。120Bは60GB以上推奨を明記。[41]

採用判断基準（テキストLLM）
- “ローカルで回す”最重要制約はメモリです。モデル選定は「必要品質→許容レイテンシ→許容メモリ→形式（GGUF/GPTQ/AWQ/EXL2等）」の順に落とし込み、最後にランタイムを決めるのが破綻しにくい（逆に“ランタイム先行”は形式制約で詰まりやすい）。これは設計上の推奨です。
- 7B〜14Bがローカル実務のボリュームゾーンになりやすく、3B級は“ハード制約モード”で使い分けます（後述Tier表で具体化）。
- ライセンスはモデルごとに条件が異なり、同じファミリーでも例外があるため、モデルカードや公式ブログの個別確認を必須化します（例：Qwen2.5の例外記載）。 33

VLM / LMM（画像理解・マルチモーダル）

代表モデル	規模	推奨量子化・実行形式	得意/不得意（要点）	ランタイム相性	リスク・採用判断
Qwen2-VL（2B/7B/72B） 42	2B/7B/72B	2B/7Bがローカル現実ライン。72Bは上位ハード。	画像＋テキストの汎用。	mac: MLX-VLMがQwen2-VLの利用例を明示。 28	画像トークンが増えるとメモリ/速度が急落。
Phi-3-Vision 128K Instruct 43	軽量MM（Phi-3系）	ローカル適性（形式依存）。	“軽量マルチモーダル＋長文”をうたう。 43	Win: transformers系/推論エンジン、mac: MLX変換次第	長文はKVが支配しやすく、現実には文脈長の設計が必要。
InternVL2（例：4B） 44	4B等	モデル群が複数サイズ。	画像理解の系列として設計され、構成要素もモデルカードに説明。 45	GPU/形式次第	エコシステム差（変換・プロセッサ依存）が運用難所。
LLaVA（例：7B/13B系） 46	7B/13B等	4bitでの運用が示唆され、13Bで12GB VRAM級でも動作可能と記載。 47	画像チャット用途。	Win GPUで構築しやすい。	ベースLLMや実装差が大きく、モデルカード/実装の整合が必要。

採用判断基準（VLM/LMM）
- “画像理解”はテキストLLMより前処理・プロセッサ依存が強い（画像トークン化、解像度、マルチ画像、OCRなど）。従って「ランタイムがそのモデルを明示サポートしているか（README等）」を一次確認するのが安全です（例：MLX-VLMがQwen2-VLの例を具体コマンドで提示）。 28
- 72B級VLMはローカルでは上位機（96GB+統合メモリ、または複数GPU等）が前提になりやすく、現実には2B/7B級で運用設計（不足はRAGやOCR併用で補う）に寄せます。

ASR（音声認識）

代表モデル/方式	規模	推奨量子化・実行形式	得意/不得意（要点）	ランタイム相性	リスク・採用判断
Whisper（研究/公開）[48]	複数サイズ	ローカル実装多数（下記）。	68万時間規模の多言語データで頑健性を示し、翻訳も可能。[48]	直接実装/各派生に分岐	高リスク領域では“誤転記（幻覚）”が実害になる可能性が報道されているため、検証プロセスが必須。[49]
faster-whisper（CTranslate2）[50]	Whisper互換	CPU/GPUで8bit量子化を明記。	同等精度で最大4倍速・低メモリを主張。[50]	Win/mac/Linuxで実装しやすい（Python）	実時間（RTF）要件がある業務はまずこれでベンチし、足りなければGPU化。
whisper.cpp[51]	Whisper互換	ggml系でローカル最適化。	C/C++で軽量運用、各種最適化例が豊富。[51]	mac/Winともに導入可能	バッチ向きに設計し、ストリーミング要件は別設計が必要になるケースがある（一般論）。

採用判断基準（ASR）
- ローカルASRは「RTF」「メモリ」「誤転記」を同時に満たす必要があります。faster-whisperは速度・メモリ面の利点を明示しているため、まず基準実装にする合理性があります。[50]
- 会議用途ではVAD（無音除去）・話者分離（後述）が品質の下限を決めます。

TTS（音声合成）

代表モデル/方式	規模	推奨量子化・実行形式	得意/不得意（要点）	ランタイム相性	リスク・採用判断
Piper（高速ローカルTTS）[52]	軽量	ローカル常駐TTS向き。	“fast, local neural TTS”を明示。リポジトリはアーカイブされ移転先が示されている。[52]	省リソース環境で有利	移転・継続性リスク（運用はフォーク/移転先を確認）。
Kokoro（82M）[53]	82M	軽量TTSとして使い分け。	82Mで高速・コスト効率を主張し、Apacheライセンス重みをうたう。[53]	CPUでも成立しやすい	日本語品質は声・辞書・前処理に依存し、事前確認が必要。
Coqui XTTS-v2（音声クローン寄り）[54]	大きめ	GPU推奨。	数秒の参照音声で多言語クローンをうたう。[54]	GPU環境で実務的	音声クローンは法務・倫理・権利リスクが大きい（業務要件化するなら同意・監査が必要）。

代表モデル/方式	規模	推奨量子化・実行形式	得意/不得意（要点）	ランタイム相性	リスク・採用判断
StyleTTS2 [55]	研究系	高品質TTS志向。	“human-level”を目標とする研究系。 [55]	GPUが望ましい	実運用は依存関係・再現性の確認が必須。

採用判断基準（TTS）
- “読み上げ（通知・要約）”と“クローン（本人声）”は別物として分離設計するのが現実的です。前者はPiper/Kokoroのような軽量系、後者はXTTS等で、リスク管理を別レイヤに置きます。 [56]

Embedding / Reranker（検索・RAG基盤）

代表モデル	規模	推奨量子化・実行形式	得意/不得意（要点）	ランタイム相性	リスク・採用判断
BGE-M3 [57]	（モデルカード参照）	まずはFP16/INT8で安定運用、必要なら量子化。	Multi-Functionality/Multi-Linguality/Multi-Granularityを特徴として明示。 [58]	CPU/GPU/ONNX等で運用可能（環境依存）	RAGはEmbedding品質が上限を決めるため、まずここを堅くする。
multilingual-e5-large(-instruct) [59]	large	多言語検索用途で定番。	技術報告が示され、使用例が明確。 [60]	transformers/ONNX	日本語含む多言語を要するなら候補。
bge-reranker（large/v2等） [61]	278M〜560M等	RAGの“再ランキング”で精度を上げる。	rerankerは「クエリ＋文書を入力しスコアを直接出す」とモデルカードで説明。 [62]	CPU/GPU（遅延要件次第）	rerankerはレイテンシを増やすため、p95要件で採否。

採用判断基準（Embedding/Reranker）
- RAGの失敗の多くは「検索が外れる」「上位が弱い」「文脈が長すぎる」です。Embeddingで recall を確保し、rerankerで precision を上げる二段構えが実務的です（この設計自体は一般的推奨）。
- 同時に、ローカル運用では“Embedding前計算（夜間バッチ）”が効くため、実時間はLLM応答に集中させられます（後述）。

OCR / Document AI

代表モデル/方式	方式	推奨実行形態	得意/不得意（要点）	ランタイム相性	リスク・採用判断
Tesseract OCR [63]	古典OCR + LSTM	CPU常駐/バッチに強い	OCRエンジンで、Tesseract 4がLSTMベースOCRを追加したことを明記。 [64]	クロスプラットフォーム	“画像品質が悪いスキャン”は限界が出るため、前処理と併用判断。
PaddleOCR [65]	OCR/文書解析	CPU/GPU	PDF/画像を構造化データ（JSON/Markdown等）にする方向を強調。 [66]	Python中心	依存関係・モデル選択が広く、運用設計が必要。
Donut（OCR-free Doc理解） [67]	OCR-free E2E	GPU推奨	“OCR不要のend-to-end Transformerで文書理解”を明記。 [68]	transformers中心	画像→構造抽出に強いが、学習済みタスク外は崩れやすい。
LayoutLMv3 [69]	DocAI基盤	GPU推奨	DocAI向けの事前学習として提示。 [69]	実装・運用は要設計	“現場で使う”には推論パイプライン化が前提。

採用判断基準（OCR/Document AI）
- 「紙スキャン→検索→要約」の現実解は、(A) OCRでテキスト化してRAG、(B) Donutのように直接構造抽出、の二系統です。Aは堅いがレイアウト情報が落ち、Bはタスク適合で強いが外れると崩れます。 [70]
- 実務でまず必要なのは“再現可能なOCR品質ベンチ”であり、LLMの出来以前にここで詰まるケースが多い（設計上の注意）。

画像生成（Diffusion）

代表モデル/方式	規模	推奨量子化・実行形式	得意/不得意（要点）	ランタイム相性	リスク・採用判断
SDXL base 1.0 [71]	diffusion	基本はFP16中心、環境により最適化	ライセンス（CreativeML Open RAIL++-M）とモデル概要を明示。 [71]	Win: A1111/ComfyUI、mac: 工夫必要	VRAM要求は運用・設定依存で変動が大きい（後述ベンチで確定）。
ComfyUI [72]	ノードベースUI	ワークフロー管理が強い	Windows/Linux/macOS対応を明示。 [72]	Win中心に実務化しやすい	ワークフローが資産になるため、バージョン固定が重要。

代表モデル/方式	規模	推奨量子化・実行形式	得意/不得意（要点）	ランタイム相性	リスク・採用判断
Stable Diffusion WebUI（A1111）[73]	WebUI	拡張が豊富	WebUIとしての位置付け。[73]	Win中心	拡張乱立による再現性低下が起こりやすい。
FLUX.1（open-weight推論実装）[74]	diffusion	形式・量子化は実装依存	“open-weight modelsの推論コード”をGitHubで明示。[75]	ComfyUI等と併用されがち	ライセンス条件（例：devモデルの条件）などを個別精査が必要。[76]

採用判断基準（画像生成）
- 画像生成はLLMよりVRAM消費が尖りやすく、かつ設定（解像度、バッチ、refiner等）で必要VRAMが激変します。そのため「VRAMライン」はコミュニティ知見（例：8GB/12GB/16GB目安）を参照しつつも、必ず自社ワークフローで再現ベンチを取るべきです。[77]

Agent / Tool-use（エージェント）

代表方式/要素	位置付け	得意/不得意（要点）	ローカル実装の要点	一次ソース根拠
OpenAI互換API（ローカル）	“道具になるLLM”の前提	既存ツールチェーンを流用しやすい	LM Studio/Ollamaが互換エンドポイントを提供。[78]	LM Studio server / Ollama OpenAI互換の公式説明[79]
tool calling（関数呼び出し）	Agent中核	JSON堅牢性が失敗点	gpt-oss×Ollamaのガイドでtool calling例が提示。[27]	ガイドに実装例あり[27]
“ローカルLLMに外部ツールを与える”	実務自動化	ファイル操作・RAG・検索などに分解	OpenHandsがLM Studioでローカルを接続するガイドを提示。[80]	OpenHands公式ドキュメント[80]

採用判断基準（Agent/Tool-use）
- ローカルAgentは「LLMの賢さ」より「JSON破壊率」「p95遅延」「ツール失敗時の復旧設計」で決まります。従って、モデル選定と同時に“制約付き生成（JSONモード等）を使えるか”と“ツール再試行の状態管理”を要件化します（設計推奨）。
- 既存業務システム連携では、ローカルLLMを“OpenAI互換の社内API”として扱えるかが開発効率を決めます。LM Studio/Ollamaが互換APIを提供する点は、実務上大きいです。[79]

話者分離／ノイズ除去／ウェイクワード／VAD 等（音声周辺）

代表モデル/方式	役割	得意/不得意（要点）	実行要件	一次ソース根拠
Silero VAD[81]	発話区間検出	30ms+チャンクが単一CPUスレッドで1ms未満等、“高速”を明示。[82]	軽量（モデル小）[82]	GitHub/torch hub

代表モデル/方式	役割	得意/不得意（要点）	実行要件	一次ソース根拠
pyannote.audio [83]	話者分離	話者分離のOSSツールキットとして説明。 [84]	PyTorch依存	GitHub
openWakeWord [85]	ウェイクワード	事前学習済みモデル同梱を明示。 [85]	軽量運用向き	GitHub
RNNoise [86]	ノイズ抑制	RNNベースのノイズ抑制ライブラリと明示。 [86]	C/C++で組込みやすい	GitHub

採用判断基準（音声周辺）
- “会議議事録”や“音声メモ”はASR単体では品質が頭打ちになります。VAD（無音/雑音除去）→ASR→話者分離→LLM整形、の鎖が必要で、特にVADはコストが小さく効果が大きいため最初から入れるべきです。Silero VADは速度・軽量性を一次ソースで明示しています。 [82]

ハード別の実用ライン整理

（4）最重視項目：Apple Silicon（M1/M2/M3/M4）とWindows（CPU/GPU）前提で、速度・メモリ/VRAM・常駐可否をTier化します。ここでは「理論重みメモリ（計算）」「KVキャッシュ概算（計算）」「一次ソースで確認できる最適化要素」を統合し、現場で使える形に落とします。

重みメモリの理論値（4bit中心、+10%オーバヘッド込みの概算）

下表は“重みだけ”の概算です（実際はKVキャッシュ・ランタイムオーバヘッドが追加）。この計算はパラメータ数×bit幅で算出した概算値です（推測ではなく計算）。

モデル規模	4bit(4b)	EXL2 4.5bpw	INT8(8b)	FP16(16b)
0.5B	0.3 GiB	0.3 GiB	0.5 GiB	1.0 GiB
1.5B	0.8 GiB	0.9 GiB	1.5 GiB	3.1 GiB
3B	1.5 GiB	1.7 GiB	3.1 GiB	6.1 GiB
7B	3.6 GiB	4.0 GiB	7.2 GiB	14.3 GiB
8B	4.1 GiB	4.6 GiB	8.2 GiB	16.4 GiB
14B	7.2 GiB	8.1 GiB	14.3 GiB	28.7 GiB
27B	13.8 GiB	15.6 GiB	27.7 GiB	55.3 GiB
32B	16.4 GiB	18.4 GiB	32.8 GiB	65.6 GiB
34B	17.4 GiB	19.6 GiB	34.8 GiB	69.7 GiB
70B	35.9 GiB	40.3 GiB	71.7 GiB	143.4 GiB

注：EXL2はExLlamaV2が「2〜8bit、混合量子化で平均bitrateを調整可能」と説明しています。 [87]

KVキャッシュの支配性（Llama3 8B相当のパラメータ例での概算）

Llama3 8B相当のKVパラメータ例として、llama.cppの議論内に n_layer=32, n_head_kv=8, head_dim=128 等が示されています。 [88]
この条件ではKVキャッシュは“コンテキスト長に比例して”増えます（計算）。

コンテキスト長(tokens)	FP16/BF16	FP8/INT8	INT4
2,048	0.25 GiB	0.12 GiB	0.06 GiB
4,096	0.50 GiB	0.25 GiB	0.12 GiB
8,192	1.00 GiB	0.50 GiB	0.25 GiB
16,384	2.00 GiB	1.00 GiB	0.50 GiB
32,768	4.00 GiB	2.00 GiB	1.00 GiB
131,072	16.00 GiB	8.00 GiB	4.00 GiB

運用上の含意：
- “長文/長時間会議”は、重みよりKVが先にメモリを食い潰します。したがって「文脈長を必要最小にする」「プロンプトキャッシュを使う」「KV量子化/FP8 KVを使う」「要約で圧縮する」といった設計が必須です。vLLMはQuantized KV Cache（FP8）を機能として明示し、TensorRT-LLMもKV cache最適化（paged/quantized/reuse）を公式に強調しています。 [89]
- llama.cppも“プロンプトキャッシュ（cache_prompt）”の存在が明示されており、固定の長いsystem promptを繰り返す用途で効きます。 [90]

Tier定義と“現実ライン”（Apple Silicon/Windows）

以下は「重み（4bit中心）＋KV（4k〜8k想定）＋オーバヘッド」を踏まえた“現実ライン”です。数値は重み理論値（計算）に基づきますが、最終は(6)のベンチで確定させてください（ここでは過度に楽観しない）。

Tier	想定ハード	現実的なLLM規模（目安）	常駐可否（LLM単体）	同時実行（LLM+ASR+TTS）	設計指針
A-16	Apple Silicon 16GB（M1/M3/M4等）	3B〜7/8B（4bit）	“短文中心”なら可	軽量構成なら可（ASR/TTSは小さめ）	文脈長を抑え、RAGはEmbedding先計算。AppleはUMAを公式に強調。 [91]
A-24/32	Apple Silicon 24〜32GB	7B〜14B（4bit）	可（業務最小ライン）	多くのユースケースで成立	14B級を軸に、VLMは2B〜小型を併用。M4世代のメモリ構成例が公式に提示。 [92]
A-64	Apple Silicon 64GB	14B〜32B（4bit）	可	余裕あり	長文会議は要約で圧縮し、KV最適化を活用。M1 Max等で64GB構成が公式に提示。 [93]

Tier	想定ハード	現実的なLLM規模（目安）	常駐可否（LLM単体）	同時実行（LLM+ASR+TTS）	設計指針
A-96/128	Apple Silicon 96〜128GB	32B〜70B（4bit）	可（重い）	構成次第で成立	“高品質ローカル”の現実ライン。M3/M4で最大128GB構成が公式に提示。 94
W-CPU1	Windows CPU-only（8C/16T級 + 32GB）	3B〜7B（4bit）	条件付き（負荷高）	同時は厳しい（オンデマンド推奨）	CPU最適化命令（AVX等）を活かし、常駐は“小型＋短文”に限定。Intel最適化要素は公式に明示。 13
W-CPU2	Windows CPU-only（16C/32T級 + 64GB）	7B〜14B（4bit）	可（ただし速度は要検証）	同時は設計次第	夜間バッチ（Embedding/OCR）で負荷平準化。
G-8	Windows + NVIDIA VRAM 8GB	7B（4bit）中心	可（短文）	同時は軽量なら可	画像生成SDXL等は設定依存で厳しいことがあるため要ベンチ。 77
G-12	Windows + NVIDIA VRAM 12GB	7B〜14B（4bit）	可	同時が現実化	VLM 2B〜7B級やASRもGPUが載りやすい。
G-16	Windows + NVIDIA VRAM 16GB	14B〜27B（4bit）	可	“高品質寄り”が成立	KV（長文）を意識してFP8 KV等の採用余地。 95
G-24	Windows + NVIDIA VRAM 24GB	27B〜34B（4bit）	可	余裕あり	画像生成や大きめVLMを実務的に回せる可能性が高い。

補足（AMD/Intel GPUの現実ライン）
- AMD 96 ROCmはRadeon向けに「ROCm 7.2でRadeon 9000/選択された7000シリーズにWindowsでPyTorch対応」等を明示していますが、対象GPUが限定されるため“購入前にサポート表で確定”が必須です。 97
- Intel系はoneAPI/SYCLやOpenVINO、IPEX-LLMなどが選択肢になります。llama.cppはSYCLバックエンド等を掲げ、Intel自身も「llama.cppのSYCLバックエンドでIntel GPUをサポート」と説明しています。 98

常駐運用 vs オンデマンド運用（差分の要点）
- 常駐：初回ロードは重いが、KVキャッシュ再利用やsystem promptキャッシュで“初動遅延（first token latency）”が改善し得ます。llama.cppではcache_prompt等の概念が示されています。 99
- オンデマンド：RAM/VRAMを節約できるが、毎回ロードで遅い。個人PCやメモリ16GB帯ではオンデマンド設計が現実解になることが多い（設計判断）。
- 夜間バッチ：Embedding生成、OCR、会議音声のASRなどを夜間に回し、日中はLLM応答（対話UX）に資源を寄せるのが「ローカルAIでの生産性最大化」につながります（設計推奨）。

ユースケース別推奨スタック

（5）各ユースケースについて、推奨コンポーネントと「低コスト／高品質／ハード制約」の3構成、失敗モードと回避策を提示します。ここでの推奨は(2)の特徴軸と(4)のTier前提で根拠付けします。

音声メモ → 整文化 → タスク抽出（個人・業務メモ）

低コスト構成（A-16 / W-CPU1 / G-8想定）
- VAD：Silero VAD（先に無音/雑音区間を落とす） [82]
- ASR：faster-whisper（CPUでINT8、GPUがあればGPU） [50]
- LLM整形：7B級（Qwen2.5-7B-Instruct等）を4bitで（GGUF系） [100]
- 出力：JSON（タスク配列）を要求。Ollama/LM StudioのOpenAI互換APIで業務アプリに繋ぐ。 [3]
失敗モード：ASRの誤転記→タスク誤抽出。回避：音声区間分割（VAD）＋重要箇所だけ再確認UI（設計）。

高品質構成（A-64+/G-16+/G-24想定）
- ASR：faster-whisper GPU、必要なら大きめWhisper系列（運用選択） [101]
- LLM：14B〜32B級（Phi-4やQwen2.5上位等）を4bit/GPUで [102]
- タスク抽出：ツール呼び出し（function calling）を前提にschema固定（JSON破壊率を下げる）。 [27]
失敗モード：長文で要点漏れ。回避：逐次要約（rolling summary）＋KV節約。

ハード制約構成（W-CPU1/A-16でギリギリ）
- ASR：短音声に限定、または夜間バッチ。
- LLM：3B〜4B（Phi-3 mini等）で“抽出のみ”に寄せる。 [103]
失敗モード：LLMの推論不足。回避：抽出テンプレを固定し、自由生成を減らす（設計）。

会議議事録（話者分離＋要約＋アクション）

低コスト構成
- VAD：Silero VAD [82]
- ASR：faster-whisper [50]
- 話者分離：pyannote.audio（精度重視だが依存重い） [84]
- LLM：7B級で要約・論点・決定事項・ToDo抽出
失敗モード：話者誤割当。回避：発話ターン統合ルール（短い切れ目はマージ）＋参加者名の手動マッピング（設計）。

高品質構成
- LLM：32B〜70B級（メモリ余裕必須）＋長文はKV最適化（FP8 KV等）を検討 [104]
- “議事録の事実性”を担保するため、要点ごとに「元発話時刻/区間」を付与する（設計。ASR側でタイムスタンプ活用）。Whisperはタイムスタンプ等の特殊トークンによるマルチタスクを説明しています。 [105]
失敗モード：ASR幻覚（無音部分に文が入る等）。回避：VAD＋“無音区間は破棄”＋要審査フラグ。

ハード制約構成
- 話者分離は諦め、「話者なし→要約→タスク」へ縮退。
- 夜間バッチで長時間音声を処理し、朝に要約だけ見る（設計）。

文書RAG（PDF/スキャン → OCR/解析 → 検索 → 回答）

低コスト構成
- OCR：Tesseractでテキスト化（スキャン品質が高い場合） [63]
- Embedding：BGE-M3（多言語・多用途をうたう） [58]

- Reranker：bge-reranker-base/large（遅延許容に応じ） [61]
- LLM：7B級（4bit）
失敗モード：OCR誤りで検索が外れる。回避：OCR前処理（傾き補正等）＋“原文スニペット引用”をUX要件化。

高品質構成
- OCR：PaddleOCR（構造化出力方向を強調） [66]
- OCR-free：Donutで情報抽出（フォーム/請求書等が適合する場合） [68]
- LLM：14B〜32B級＋reranker大型
失敗モード：文書の図表/レイアウト由来の情報欠落。回避：LayoutLMv3系やDonut系でレイアウト情報を保持。 [106]

ハード制約構成
- OCRは夜間バッチ、Embeddingも夜間生成。日中は検索と短い回答だけ。

画像理解（スクショ/写真の説明、情報抽出）

低コスト構成
- VLM：Qwen2-VL 2B級（ローカル現実ライン） [107]
- ランタイム：macならMLX-VLM、WindowsならGPU推論（形式依存）
失敗モード：UI文字が読めず誤解。回避：必要ならOCR（PaddleOCR等）を併用して文字情報を別経路で供給。 [66]

高品質構成
- VLM：Qwen2-VL 7BやInternVL2等、必要に応じ上位 [108]
- “画像＋文書RAG”融合：画像から抽出→Embedding→RAG
失敗モード：視覚トークン肥大で遅い。回避：解像度/max-pixels制御（MLX-VLMはvideo_generate例でmax-pixels等を示す）。 [28]

ハード制約構成
- 画像理解は“質問を限定”し、OCR＋テキストLLMに逃がす（設計）。

コーディング補助（ローカルIDE支援・リポジトリ理解）

低コスト構成
- コードLLM：Qwen2.5-Coder 7B（サイズ展開を明示） [109]
- RAG：リポジトリをEmbedding化（BGE/E5）＋関連ファイル抽出→LLM回答
- 実行：Ollama/LM StudioをローカルAPI化してIDE拡張と接続（OpenAI互換） [110]
失敗モード：生成コードの幻覚API。回避：コンパイル/テスト実行を“ツール”として組み込み、LLM出力を検証ループに入れる（Agent設計）。

高品質構成
- コードLLM：StarCoder2（3B/7B/15B） [111]
- LLM：大きめ（32B級）を併用し、設計レビューや長距離依存を担当。
失敗モード：長コンテキストで遅い。回避：prefix caching（vLLMのprefix caching概念など）を活用。 [112]

ハード制約構成
- 3B級で「補完＋リファクタ提案」など短い支援に限定。

画像生成（資料・UIモック・社内コンテンツ）

低コスト構成
- モデル：SDXL base 1.0（ライセンス確認） [71]
- UI：ComfyUI（Win/Linux/macOS対応を明示） [72]
失敗モード：VRAM不足。回避：解像度/バッチ/最適化（attention slicing等）を固定プロファイル化（設計）。VRAM目安はモデルカード等に断片的記載があるが環境依存。 [113]

高品質構成
- FLUX.1等の上位モデルを検討する場合、ライセンス条件（devの条件等）を必ず精査。 [76]
- ワークフローはComfyUIで資産化し、seed固定で再現性を担保（設計）。

ハード制約構成
- 画像生成を夜間バッチに回し、日中はLLM中心に資源配分。
- どうしても厳しければ“画像生成は別筐体（ローカルLAN内）”に逃がす（ローカルAI定義内で許容）。

測定と比較の方法

（6）ローカルAIは「モデル＋量子化＋ランタイム＋ハード」の組合せ最適化問題です。再現可能なベンチを定義し、p50/p95・tok/s・RTF・RAM/VRAMピーク・JSON破壊率などを定量化します。

指標（定義）

- p50/p95レイテンシ：
- LLM：TTFT（Time To First Token）と生成完了時間を分離し、p50/p95を取る。KV reuse（TensorRT-LLM）やprompt caching（llama.cpp）でTTFTが変わるため分離が必要。 [114]
- tok/s：生成トークン数 ÷ 生成時間。
- RTF（ASR/TTS）：処理時間 ÷ 音声長。
- RAM/VRAMピーク：OS監視＋プロセス単位（可能ならnvidia-smi等、Appleは統合メモリで要注意）。
- JSON破壊率：要求JSON出力に対し「パース失敗／スキーマ不一致」の比率。Ollama/LM StudioのOpenAI互換APIを経由して同一テストを流すと比較が容易。 [79]
- 意味改変率：要約・整形で“意味が変わった”割合。自動評価はEmbedding類似度＋人手監査が現実的（設計提案）。

手順（再現性のコア）

1. 条件固定：同一プロンプト、最大トークン、温度、top_p、同一量子化、同一コンテキスト長で統一。
2. ウォームアップ：初回ロードと2回目以降を分離（常駐とオンデマンドの差が出る）。
3. KV影響の切り分け：短文（<512）と長文（4k/8k/32k）を分ける。vLLMのpaged KV／quantized KVの効果を見る。 [5]
4. ASRは同一音声・同一前処理：VAD有無で結果が変わるため、Silero VAD等の同一設定を固定。 [82]
5. TTSは同一テキスト・同一話者：速度と品質を分離。
6. ログ保存：モデルハッシュ、量子化、ランタイムバージョン、ハード情報（CPU命令、VRAM等）を記録。

最小テストセット（例）

LLM（テキスト）3プロンプト
- P1（JSON）：
「次の文章からタスクを抽出し、`[{title,due_date,priority,owner}]`だけを返せ。欠損はnull。」＋日本語入力
- P2（長文耐性）：
2,000〜8,000トークン相当の議事録を与え、「決定事項/未決事項/リスク」を抽出
- P3（ツール適性）：
“関数仕様”を渡し、tool呼び出しの引数を生成させる（誤引数率を見る）

VLM 3プロンプト
- I1：UIスクショ（設定項目を列挙）
- I2：図表（グラフの傾向要約）
- I3：写真（物体検出＋属性抽出）

ASR 3音声（各1〜3分）
- A1：静音クリア音声
- A2：雑音あり（会議室）
- A3：複数話者（被りあり）

TTS 3テキスト
- T1：短文通知（30字）
- T2：ビジネス要約（200字）
- T3：固有名詞多め（製品/人名）

OCR/DocAI 3文書
- D1：テキストPDF
- D2：スキャンPDF（傾きあり）
- D3：表（請求書/見積）

画像生成 3プロンプト
- G1：社内資料向けアイコン調
- G2：UIモック背景
- G3：人物なし（権利/安全性の運用都合で簡易化）

参考リンク一覧

（7）一次ソース優先（可能なものは日付・更新情報も併記）。リンクは各引用から辿れる形式（citation）で提示します。

ランタイム／配布形態
- llama.cpp（GGUF必須、各種バックエンド、ローカル推論目的） [115]
- GGUF仕様（GGML向けモデル格納形式） [116]
- Ollama（Windows/macOS/Linux対応、llama.cppバックエンド、REST API） [23]
- Ollama OpenAI互換（Responses API互換の説明等） [117]
- LM Studio（Windows/macOS/Linux、ローカルAPIサーバ、互換エンドポイント） [118]
- MLX（Apple Silicon向け、Apple ML研究） [119]
- MLX-LM（Apple SiliconでLLM推論・量子化・Hub連携） [120]

- MLX-VLM（MacでVLM/Omni、Qwen2-VL例） 28
- vLLM（platform: cpu/cuda/rocm/xpu、paged attention、quantized KV cache） 121
- TensorRT-LLM（最適化、paged KV、quantization、KV reuse） 122
- ExLlamaV2（EXL2量子化説明） 87

量子化
- GPTQ論文 123
- AWQ論文／実装 124
- AutoAWQ（AWQ実装） 125
- AutoGPTQ（unmaintained/archivedの注意） 126
- SmoothQuant（INT8最適化の研究） 127

モデル（LLM／VLM）
- Llama 3.1（8B/70B/405Bの説明） 29
- Qwen2.5（0.5〜72Bの説明、ライセンス例外の説明） 128
- Gemma 2 model card（更新日あり） 35
- Mixtral 8x7B Instruct（MoE説明） 37
- Phi-3 mini 128k（3.8B） 38
- Phi-4（14B、技術報告） 40
- Qwen2-VL（2/7/72B） 42
- Phi-3-Vision 128K 43
- InternVL2（4B等） 44
- LLaVA（GitHub、4bit/12GB VRAM言及） 47
- gpt-oss ローカル実行（LM Studio/Ollama、必要VRAM等） 41

音声（ASR/TTS/周辺）
- Whisper紹介（OpenAI公式、68万時間等） 48
- faster-whisper（最大4倍速・低メモリ・8bit量子化） 50
- whisper.cpp（ローカル実装） 51
- Piper（ローカルTTS、アーカイブと移転先） 52
- Kokoro（82M、open-weight、Apache） 53
- XTTS-v2（音声クローン） 54
- Silero VAD（高速の説明） 82
- pyannote.audio（話者分離） 84
- openWakeWord（ウェイクワード） 85
- RNNoise（ノイズ抑制） 86
- Whisperの誤転記リスク（報道調査） 49

OCR/Document AI
- Tesseract（LSTM OCR） 63
- PaddleOCR（構造化出力方向） 65
- Donut（OCR-free） 68
- LayoutLMv3（DocAI事前学習） 69

画像生成
- SDXL base 1.0（モデルカード・ライセンス） 71
- ComfyUI（OS対応） 72
- AUTOMATIC1111 webui 73
- FLUX推論実装／重み（open-weightの説明、ライセンス文書） 129

ハード／Apple Silicon・Unified Memory
- Apple GPUのunified memory model（Metal公式） 130
- Apple M1のUMA言及（Apple公式ニュースルーム） 131
- MLX Unified Memory説明（公式ドキュメント） 132
- M1/M2/M3/M4のユニファイドメモリ構成例（Appleサポート/仕様） 133
- PyTorch MPS（Metal backend） 134

Windows向けAMD/Intelスタック
- ROCm Radeon/Ryzen（Windows対応GPUの明記、WSL互換表） 97
- Intel Extension for PyTorch（AVX-512/VNNI/AMX/XMX最適化明記） 13
- Intel: llama.cpp SYCLバックエンド紹介 135
- OpenVINO（CPU/GPU/NPU対応、GenAI API言及） 136

---

1 11 91 131 https://www.apple.com/newsroom/2020/11/apple-unleashes-m1/
https://www.apple.com/newsroom/2020/11/apple-unleashes-m1/

2 18 80 https://docs.openhands.dev/openhands/usage/llms/local-llms
https://docs.openhands.dev/openhands/usage/llms/local-llms

3 9 78 79 https://lmstudio.ai/docs/developer/core/server
https://lmstudio.ai/docs/developer/core/server

4 124 https://arxiv.org/abs/2306.00978
https://arxiv.org/abs/2306.00978

5 26 https://docs.vllm.ai/en/latest/design/paged_attention/
https://docs.vllm.ai/en/latest/design/paged_attention/

6 89 95 104 https://docs.vllm.ai/en/latest/features/quantization/quantized_kvcache/
https://docs.vllm.ai/en/latest/features/quantization/quantized_kvcache/

7 122 https://github.com/NVIDIA/TensorRT-LLM
https://github.com/NVIDIA/TensorRT-LLM

8 15 21 98 115 https://github.com/ggml-org/llama.cpp
https://github.com/ggml-org/llama.cpp

10 130 https://developer.apple.com/documentation/metal/choosing-a-resource-storage-mode-for-apple-gpus
https://developer.apple.com/documentation/metal/choosing-a-resource-storage-mode-for-apple-gpus

12 77 113 https://huggingface.co/wangkanai/sdxl-fp16
https://huggingface.co/wangkanai/sdxl-fp16

13 https://github.com/intel/intel-extension-for-pytorch
https://github.com/intel/intel-extension-for-pytorch

14 86 https://github.com/xiph/rnnoise
https://github.com/xiph/rnnoise

16 29 30 https://huggingface.co/meta-llama/Llama-3.1-70B
https://huggingface.co/meta-llama/Llama-3.1-70B

17 38 103 https://huggingface.co/microsoft/Phi-3-mini-128k-instruct
https://huggingface.co/microsoft/Phi-3-mini-128k-instruct

19 118 https://lmstudio.ai/docs/app
https://lmstudio.ai/docs/app

20 https://docs.nvidia.com/tensorrt-llm/index.html
https://docs.nvidia.com/tensorrt-llm/index.html

22 119 https://github.com/ml-explore/mlx
https://github.com/ml-explore/mlx

23 110 https://github.com/ollama/ollama
https://github.com/ollama/ollama

24 117 https://docs.ollama.com/api/openai-compatibility
https://docs.ollama.com/api/openai-compatibility

25 https://llama-cpp-python.readthedocs.io/en/latest/api-reference/
https://llama-cpp-python.readthedocs.io/en/latest/api-reference/

27 https://developers.openai.com/cookbook/articles/gpt-oss/run-locally-ollama/
https://developers.openai.com/cookbook/articles/gpt-oss/run-locally-ollama/

28 https://github.com/Blaizzy/mlx-vlm
https://github.com/Blaizzy/mlx-vlm

31 https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct

32 100 128 https://huggingface.co/Qwen/Qwen2.5-7B-Instruct
https://huggingface.co/Qwen/Qwen2.5-7B-Instruct

33 https://qwenlm.github.io/blog/qwen2.5/
https://qwenlm.github.io/blog/qwen2.5/

34 35 https://ai.google.dev/gemma/docs/core/model_card_2
https://ai.google.dev/gemma/docs/core/model_card_2

36 37 https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1

39 40 102 https://arxiv.org/abs/2412.08905
https://arxiv.org/abs/2412.08905

41 https://developers.openai.com/cookbook/articles/gpt-oss/run-locally-lmstudio/
https://developers.openai.com/cookbook/articles/gpt-oss/run-locally-lmstudio/

42 https://huggingface.co/Qwen/Qwen2-VL-72B
https://huggingface.co/Qwen/Qwen2-VL-72B

43 https://huggingface.co/microsoft/Phi-3-vision-128k-instruct
https://huggingface.co/microsoft/Phi-3-vision-128k-instruct

44 45 https://huggingface.co/OpenGVLab/InternVL2-4B
https://huggingface.co/OpenGVLab/InternVL2-4B

46 47 https://github.com/haotian-liu/LLaVA
https://github.com/haotian-liu/LLaVA

48 105 https://openai.com/index/whisper/
https://openai.com/index/whisper/

49 https://apnews.com/article/90020cdf5fa16c79ca2e5b6c4c9bbb14
https://apnews.com/article/90020cdf5fa16c79ca2e5b6c4c9bbb14

50 101 https://github.com/SYSTRAN/faster-whisper
https://github.com/SYSTRAN/faster-whisper

51 https://github.com/ggml-org/whisper.cpp
https://github.com/ggml-org/whisper.cpp

52 56 https://github.com/rhasspy/piper
https://github.com/rhasspy/piper

53 https://github.com/hexgrad/kokoro
https://github.com/hexgrad/kokoro

54 https://huggingface.co/coqui/XTTS-v2
https://huggingface.co/coqui/XTTS-v2

55 https://github.com/yl4579/StyleTTS2
https://github.com/yl4579/StyleTTS2

57 58 https://huggingface.co/BAAI/bge-m3
https://huggingface.co/BAAI/bge-m3

59 https://huggingface.co/intfloat/multilingual-e5-large
https://huggingface.co/intfloat/multilingual-e5-large

60 https://huggingface.co/intfloat/multilingual-e5-large-instruct
https://huggingface.co/intfloat/multilingual-e5-large-instruct

61 https://huggingface.co/BAAI/bge-reranker-large
https://huggingface.co/BAAI/bge-reranker-large

62 https://huggingface.co/BAAI/bge-reranker-v2-m3
https://huggingface.co/BAAI/bge-reranker-v2-m3

63 64 70 https://github.com/tesseract-ocr/tesseract
https://github.com/tesseract-ocr/tesseract

65 66 https://github.com/PaddlePaddle/PaddleOCR
https://github.com/PaddlePaddle/PaddleOCR

67 68 https://github.com/clovaai/donut
https://github.com/clovaai/donut

69 106 https://arxiv.org/pdf/2204.08387
https://arxiv.org/pdf/2204.08387

71 https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0
https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0

72 https://github.com/Comfy-Org/ComfyUI
https://github.com/Comfy-Org/ComfyUI

73 https://github.com/AUTOMATIC1111/stable-diffusion-webui
https://github.com/AUTOMATIC1111/stable-diffusion-webui

74 75 129 https://github.com/black-forest-labs/flux
https://github.com/black-forest-labs/flux

76 https://github.com/black-forest-labs/flux/blob/main/model_licenses/LICENSE-FLUX1-dev
https://github.com/black-forest-labs/flux/blob/main/model_licenses/LICENSE-FLUX1-dev

81 82 https://github.com/snakers4/silero-vad
https://github.com/snakers4/silero-vad

83 84 https://github.com/pyannote/pyannote-audio
https://github.com/pyannote/pyannote-audio

85 https://github.com/dscripka/openWakeWord
https://github.com/dscripka/openWakeWord

87 https://github.com/turboderp-org/exllamav2
https://github.com/turboderp-org/exllamav2

88 https://github.com/ggerganov/llama.cpp/discussions/7949
https://github.com/ggerganov/llama.cpp/discussions/7949

90 96 99 https://github.com/ggml-org/llama.cpp/discussions/8860
https://github.com/ggml-org/llama.cpp/discussions/8860

92 https://www.apple.com/macbook-pro/specs/
https://www.apple.com/macbook-pro/specs/

93 133 https://support.apple.com/en-us/111901
https://support.apple.com/en-us/111901

94 https://support.apple.com/ja-jp/117736
https://support.apple.com/ja-jp/117736

97 https://rocm.docs.amd.com/projects/radeon-ryzen/en/latest/index.html
https://rocm.docs.amd.com/projects/radeon-ryzen/en/latest/index.html

107 https://huggingface.co/collections/Qwen/qwen2-vl
https://huggingface.co/collections/Qwen/qwen2-vl

108 https://huggingface.co/Qwen/Qwen2-VL-7B-Instruct
https://huggingface.co/Qwen/Qwen2-VL-7B-Instruct

109 https://huggingface.co/Qwen/Qwen2.5-Coder-7B-Instruct
https://huggingface.co/Qwen/Qwen2.5-Coder-7B-Instruct

111 https://huggingface.co/docs/transformers/en/model_doc/starcoder2
https://huggingface.co/docs/transformers/en/model_doc/starcoder2

112 https://docs.vllm.ai/en/v0.9.2/design/automatic_prefix_caching.html
https://docs.vllm.ai/en/v0.9.2/design/automatic_prefix_caching.html

114 https://nvidia.github.io/TensorRT-LLM/advanced/kv-cache-reuse.html
https://nvidia.github.io/TensorRT-LLM/advanced/kv-cache-reuse.html

116 https://github.com/ggml-org/ggml/blob/master/docs/gguf.md
https://github.com/ggml-org/ggml/blob/master/docs/gguf.md

120 https://github.com/ml-explore/mlx-lm
https://github.com/ml-explore/mlx-lm

121 https://docs.vllm.ai/en/latest/api/vllm/platforms/
https://docs.vllm.ai/en/latest/api/vllm/platforms/

123 https://arxiv.org/pdf/2210.17323
https://arxiv.org/pdf/2210.17323

125 https://github.com/casper-hansen/AutoAWQ
https://github.com/casper-hansen/AutoAWQ

126 https://github.com/AutoGPTQ/AutoGPTQ
https://github.com/AutoGPTQ/AutoGPTQ

127 https://arxiv.org/abs/2211.10438
https://arxiv.org/abs/2211.10438

132 https://ml-explore.github.io/mlx/build/html/usage/unified_memory.html
https://ml-explore.github.io/mlx/build/html/usage/unified_memory.html

134 https://developer.apple.com/metal/pytorch/
https://developer.apple.com/metal/pytorch/

135 https://www.intel.com/content/www/us/en/developer/articles/technical/run-llms-on-gpus-using-llama-cpp.html
https://www.intel.com/content/www/us/en/developer/articles/technical/run-llms-on-gpus-using-llama-cpp.html

136 https://github.com/openvinotoolkit/openvino
https://github.com/openvinotoolkit/openvino