Technical Research Report

ローカルAI 技術調査レポート

オンデバイス／ローカル推論の現実解と推奨スタック

LLM　VLM　ASR　TTS　RAG　Apple Silicon　Windows GPU

Report Purpose

Apple SiliconおよびWindows環境におけるローカルAIの実運用ラインと推奨スタックを提示。
「意思決定のための要点」と「根拠となる完全な技術詳細」を二層構造で網羅。

2026年2月15日 | Generated by Genspark AI Slides

このスライドの読み方

概要→詳細の二層構造ガイド

構造：二層設計

1. 概要スライド（Executive Summary）

意思決定に必要な「結論」と「要点」を最初に提示します。時間がない場合はここだけ読めば全体像が掴めます。

2. 完全版スライド（Full Detail）

レポート内の表、数値、グラフ、注釈を省略せずに掲載します。エンジニアや実装担当者が参照するための詳細情報です。

相互リンクと参照

概要から詳細へ、詳細から参考文献へ、論理的に接続されています。

設計原則と表記

省略しない

元の技術レポートに含まれる情報は、脚注やURLに至るまで全てスライド内に保持します。

根拠の明示

「推測」と「事実（計算値/公式発表）」を明確に区別し、一次ソースに基づきます。

推奨構成　注意・失敗モード　補足情報

Key Takeaways: 各章の冒頭で「持ち帰るべき要点」を3〜5点で提示します。

目次（Agenda）

Total 13 Chapters

各章は「概要（要点）」→「詳細（完全版）」の順で構成されています

エグゼクティブサマリ（実務要点）

ローカルAI導入における意思決定の重要ポイント

結論：ローカルAIは「4bit量子化＋KV管理」を前提に、
GGUF/llama.cpp系とOllama/LM Studio/MLXで実務化可能です。

定義とスコープ

推論が端末orローカルLAN内で完結

入力データは外部へ送信されない

LANサーブ含む（LM Studio, Ollama）

中核技術

量子化 Weight-only 4bit (AWQ等)

メモリ KVキャッシュ管理最適化

Paged Attention / Quantized KV

主要ランタイム

llama.cpp (GGUF): 汎用・軽量

Ollama / LM Studio: UI + API

MLX: Apple Silicon特化

ハードウェア実用ライン (4bit量子化前提)

Apple Silicon (Unified Memory)

16GB: 3B〜7/8B級

24-32GB: 7B〜14B級 (業務最小ライン)

Windows (NVIDIA GPU VRAM)

8GB: 7B級 / **12GB**: 14B級

24GB: 27B〜34B級 (実務域)

推奨スタック

低コスト: GGUF + llama.cpp

高品質: GPU + TensorRT-LLM

ハード制約: 3B級 + RAG工夫

補足：長文コンテキストはKVキャッシュがメモリを支配します。最終判断は再現ベンチマーク（後述の手順）で確定してください。

エグゼクティブサマリ（詳細1）

結論：ローカルAIは「推論がユーザー管理下で完結する構成」と定義され、重み量子化（4bit）とKVキャッシュ最適化が実運用の技術的基盤です。

ローカルAIの定義と射程

- ユーザー端末（オンデバイス）、PC、またはローカルLAN内サーバで完結。
- 入力データ（文書・音声・画像）がデフォルトでクラウドへ送出されない構成。
- LM StudioやOllamaの「localhost/network公開」機能を含みます。 [1][2]

Localhost | On-Premise LAN | Privacy First

中核技術（Enabling Technologies）

4bit前後のweight-only量子化（AWQ/GGUF等）が主流。メモリ消費を劇的に削減し、コンシューマ機での実行を可能に。 [4]

長文・多同時接続のメモリ管理が鍵。

- **Paged KV Cache:** メモリ断片化を防ぐ（vLLM等） [5]
- **Quantized KV:** KVをFP8/INT4化して容量削減 [6]
- **KV Reuse / Prompt Cache:** 計算再利用で高速化 [7]

エグゼクティブサマリ（詳細2）

結論：ランタイムはllama.cpp/GGUF系・MLXが第一選択となり、ハードウェアはメモリ容量でTier化されます（長文はKV支配）。

実運用ランタイム（第一選択）

llama.cpp クロスプラットフォーム標準：

GGUF形式必須。最小セットアップでCPU/GPU推論が可能。

軽量・汎用で多くのフロントエンドの基盤。[8]

Ollama / LM Studio UI＋ローカルAPI：

OpenAI互換APIサーバとして動作（LAN公開可）。

Windows/macOS/Linux対応で導入が容易。[23]

MLX Apple Silicon特化：

Unified Memory前提で設計された配列フレームワーク。

MLX-LM/MLX-VLMによりMetal最適化推論を実現。[22]

ハードウェア実用ライン（4bit中心概算）

Apple Silicon (Unified Memory):

- **16GB** 3B〜7/8B級（インタラクティブ）
- **24-32GB** 7B〜14B級（業務実用・RAG最小ライン）
- **64GB+** 14B〜32B級以上（高品質・長文要約可）

Windows (Discrete GPU):

- **VRAM 8GB** 7B級（最小ライン）
- **VRAM 12-16GB** 14B級〜27B級（品質寄り）
- **VRAM 24GB** 32-34B級・重い画像生成が実務域

Windows CPU-only:

- **32GB RAM+** 7B級が「使える」境界（AVX活用）。

注意：長文コンテキストとメモリ

上記は重み（weight-only）の概算です。長文（Long Context）を扱う場合、KVキャッシュがメモリを線形に圧迫するため、別途KV量子化やコンテキスト長制限（4k/8k等）の設計が必要です。

エグゼクティブサマリ（詳細3）

結論：用途とリソースに応じて「低コスト／高品質／ハード制約」の3構成を選択します。
※各構成の具体的な失敗モードと回避策は、後述の「ユースケース別推奨スタック」章で詳細に展開します。

低コスト構成

Target HW

CPUのみ / エントリーGPU / Apple 16GB

LLM Runtime

llama.cpp (GGUF Q4/K) Ollama / LM Studio

ASR / TTS

faster-whisper (INT8) Piper / Kokoro (軽量)

RAG / Embedding

BGE-M3 (多言語・多用途) bge-reranker-base

高品質構成

Target HW

GPU搭載機 (VRAM 16-24GB+) Apple 64GB+

LLM Runtime

GPU優先 (TensorRT-LLM / vLLM) 32B〜70B級モデル

ASR / TTS

faster-whisper (GPU) XTTS / StyleTTS2 (高品質)

RAG / VLM

bge-reranker-large Qwen2-VL 7B / InternVL2

ハード制約構成

Target HW

低スペックPC / 古いMac メモリ8GB以下等

LLM Runtime

Phi-3 mini (3.8B) 等 GGUF Q4 (極小モデル)

Strategy

短いコンテキストで完結させる RAGはEmbedding検索を重視

Compromise

LLM自体を軽くし、検索精度で補う 生成タスクを限定する

ローカルAIの定義とスコープ

DEFINITION

本資料の「ローカルAI」とは、推論がユーザー管理下（端末／ローカルPC／ローカルLAN）で完結し、入力データが外部へ送信されない構成を指します。

スコープの射程（範囲）

- **ローカルAPIサーバを含む:**
  localhostだけでなく、LAN公開（network）された推論サーバも対象。 LM Studio / Ollama
- **実装例:**
  Windows/macOS/Linux上で動作するOllama、LM Studio等のOpenAI互換APIサーバ。
- **非対象:**
  推論リクエストがインターネット経由で外部クラウドAPI（OpenAI, Anthropic等）へ飛ぶ構成。

設計含意（Design Implication）

- **プライバシー最優先:**
  機密データ（個人情報・社外秘）を入力しても外部流出しない安全性を担保。
- **コスト固定化:**
  トークン課金を回避し、ハードウェア初期投資のみでランニングコストを抑制。
- **耐障害性:**
  ネットワーク切断時でも推論機能が継続動作する自律性。

未指定事項の扱い（前提条件）

**結論：「完全オフライン」は推論実行時のみを必須要件とし、
導入・更新時のネットワーク利用は許容する現実的な設計を前提とします。**

オフライン要件と対象OS

“完全オフライン”の定義：
「モデル推論自体はオフラインで成立する」レベルを基本とします。モデルの初回ダウンロードやRAG文書の取り込みプロセスにはネットワーク接続が必要になり得る点を前提とします。 [19]

対象OSのスコープ：
macOS Ventura以降およびWindows 10/11を主要ターゲットとします。Linuxは「Windows上でWSL2を利用」または「別筐体サーバ」の選択肢としてのみ扱います。

Offline Inference　macOS / Windows　WSL2 Option

実装環境とライセンス管理

Linux前提の技術（TensorRT-LLM等）：
NVIDIA TensorRT-LLMなどはLinux環境でのドキュメントが中心です。Windowsでこれらを利用する場合、WSL2上でのDocker運用が実務的な解決策となります。 [20]

ライセンスの分離管理：
「推論コード（OSSライセンス）」と「モデル重み（商用利用制限など）」は別物です。特に画像生成や音声系モデルでは重みの利用条件が厳格な場合があるため、個別のリスク確認を必須とします。

04

ローカルAI ランドスケープ

オンデバイス実行環境の
全体像と技術スタック

KEY TAKEAWAYS

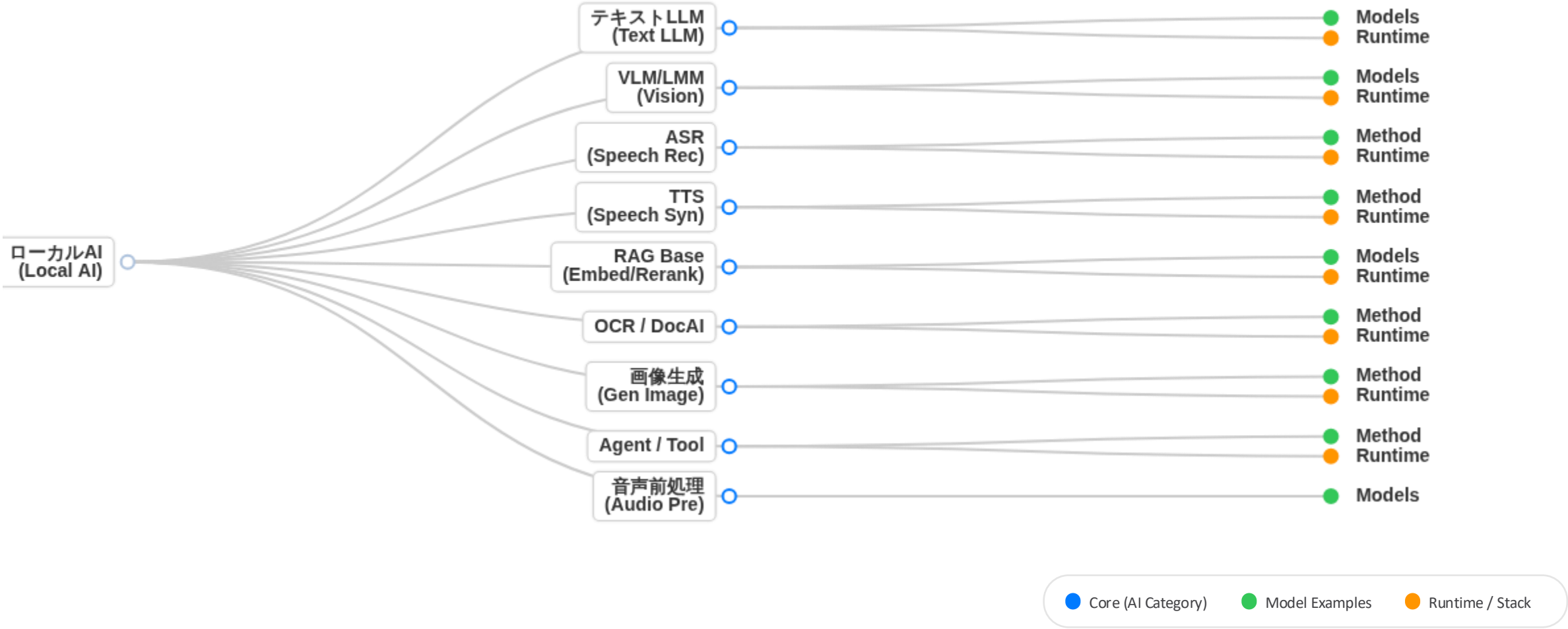
本章の要点

- **テキストLLMの二大潮流：**
  GGUF（llama.cpp系）とGPU量子化形式（GPTQ/AWQ/EXL2等）に大別されます。
- **プラットフォームの最適解：**
  macOSはMetal/Unified Memory、WindowsはOllama/LM Studioが実務的選択肢です。
- **構成要素の多様性：**
  LLMだけでなく、ASR/TTS/VLM/RAG/Agentまで網羅的にマップ化します。

カテゴリ網羅図　代表モデル　ランタイム　主用途

ローカルAIランドスケープ

結論：ローカルAIは9カテゴリ（LLM/VLM/ASR/TTS/Embedding/OCR/画像生成/Agent/音声前処理）で構成され、モデル系とランタイム系が実務上の結節点となります。



ローカルAIランドスケープ（詳細）

カテゴリ別代表モデル・ランタイム・主用途

結論：実務上の収束点は「LLMはGGUF (llama.cpp)かGPU量子化系」「macOSはMLX/Unified Memory最適化」「WindowsはOllama/LM Studio + 必要に応じWSL2」。

テキストLLM

形式: GGUF (llama.cpp必須), G [21] AWQ/EXL2 (GPU向け)

代表: Llama 3.1, Qwen2.5, Gemma 2, Phi-3/4

用途: 汎用対話, 要約, RAG, 翻訳

VLM / LMM

代表: Qwen2-VL, Phi-3-Vision, LLaVA

実行: MLX-VLM (Mac最適化), [28] formers, vLLM

用途: 画像理解, OCR代替, UI解析

ASR（音声認識）

方式: Whisper系 (faster-whisper, w [50] r.cpp) [51]

特徴: CTranslate2で高速・省メモリ, C++軽量実装

用途: 議事録, 音声コマンド, 文字起こし

TTS（音声合成）

軽量: Piper / Kokoro (高速・CPU向け) [52][53]

高品質: XTTS / StyleTTS2 (クローン・GPU推奨) [54][55]

用途: 読み上げ, 通知, ボイスボット

Embedding/Reranker

代表: bge-m3, multilingual-e5, bge-reranker [58-62]

重要性: RAGの検索精度（Recall/Precision）を決定

用途: ベクトル検索, 再ランキング

OCR / Document AI

古典: Tesseract / PaddleOCR (構造化) [64][66]

AI型: Donut / LayoutLMv3 (文書理解) [68][69]

用途: PDF解析, 請求書読取, スキャン処理

画像生成

代表: SDXL, FLUX.1 (要VRAM確認) [71][75]

実行: ComfyUI, S [72] ebUI [73]

用途: 素材生成, UIモック, コンテンツ制作

Agent / Tool-use

方式: OpenAI互換APIでのFunction Calling [24]

実行: Ollama / LM Studioが実務的 [23][41]

用途: タスク自動化, 外部ツール連携

音声前処理

代表: Silero VAD, pyannote, RNNoise [82][84]

役割: 品質の下限担保（無音除去・話者分離）

用途: 会議録音の前処理, ノイズ除去

05

特徴軸（評価軸）の定義

評価の共通物差しと測定基準

KEY TAKEAWAYS

本章の要点

- **13の評価軸**：
  指示追従、JSON堅牢性、長文耐性、速度、メモリなど、多角的な視点でモデルを比較評価します。
- **KVキャッシュ最適化**：
  paged/quantized KVやreuse機能が、長文コンテキストや多同時リクエスト処理の鍵となります。
- **JSON制約の重要性**：
  ツール利用においてJSON出力の安定性は必須ですが、機能はランタイムに強く依存します。

13の評価軸定義　測定手順　設計含意　KV最適化

特徴軸（評価軸）の定義

Total 13 Metrics

01 指示追従 (Instruction Following)
プロンプトの制約・禁止事項・形式指定（箇条書き禁止等）を遵守する能力。

02 幻覚耐性 (Hallucination Resistance)
根拠のない固有名詞や数値を捏造せず、不確実性を提示できる性質。

03 JSON堅牢性 (Structured Output)
スキーマ遵守・構文破壊率の低さ。ツール呼び出しやシステム連携の要。

04 長文耐性 (Long-context Stability)
長い入力に対する要約崩れや重要情報の欠落、自己矛盾の抑制。

05 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。

06 ツール呼び出し適性 (Tool-use)
関数呼び出しの意思決定、必要引数の抽出、曖昧性の解消能力。

07 日本語品質 (Japanese Quality)
文法・敬語・語彙の自然さと、業務文書スタイルの再現性。

08 音声品質 (TTS Quality)
自然さ（抑揚・間）、アクセント、ノイズの少なさ、声質再現性。

09 画像理解精度 (VLM Accuracy)
OCR的読み取り、図表理解、空間関係、UIスクリーンショットの解釈。

10 速度 (Speed / Throughput)
LLMの生成速度(tok/s)およびASR/TTSの実時間比(RTF)。

11 メモリ/VRAM (Memory Footprint)
重み＋KVキャッシュ＋ランタイムオーバーヘッドの総消費量。

12 量子化耐性 (Quantization Robustness)
4bit等に圧縮した際の品質劣化の小ささ（モデルと手法の相性）。

13 ライセンス (License & Compliance)
コードのOSSライセンスと、モデル重みの利用条件（商用制限等）の区別。

各評価軸の測定手法と設計含意は、次スライド以降で詳細に解説します

特徴軸（詳細）：測定と設計含意

13の評価軸における計測手順と技術的背景

機能・性能評価のコア

JSON堅牢性と指示追従

ツール呼び出し等の業務連携ではスキーマ破壊が致命的。Ollama/LM StudioのOpenAI互換APIを用いて同一条件で比較検証を行う。llama-cpp-python等のランタイム依存機能（response_format）も活用。[24] [25]

長文耐性とメモリ管理

コンテキスト長に応じてKVキャッシュメモリは線形に増加し、支配的要因となる。単純なトークン数だけでなく、KVキャッシュの最適化技術（Paged Attention等）の有無が安定性を左右する。[5][6]

速度指標 (tok/s & RTF)

LLMは生成スループット(tok/s)だけでなく、初動遅延(TTFT)を分離して測定。ASR/TTSは実時間比(RTF)で評価し、実務要件（RTF<1など）との適合を見る。

画像理解 (VLM)

視覚トークン数が増加すると速度・メモリ効率が急落するため、解像度やmax-pixels制御が重要。[28]

技術的詳細と設計含意

KVキャッシュ最適化技術

vLLMはPaged KV Cache設計でメモリ断片化を防ぎ、KVキャッシュをFP8等に量子化してフットプリントを削減。NVIDIA TensorRT-LLMもKV reuse（再利用）を最適化として強調。長文・多同時接続時の鍵となる。[5][6] [7]

量子化耐性と相性

4bit化による品質劣化はモデルと量子化方式（AWQ, GPTQ, EXL2等）の相性に依存。実機ベンチマークでの確認が必須。

ライセンスとコンプライアンス

「コード(OSS)」と「モデル重み」のライセンスは別物。特に画像生成やTTSでは重みに非商用制限が含まれる場合が多く、個別精査が必要。

Key Takeaways: 評価は「モデル×量子化×ランタイム×ハード」の組み合わせで決まるため、標準化された測定手順（条件固定・ウォームアップ・ログ保存）が不可欠です。

06

カテゴリ別モデルカタログ

主要9カテゴリの
詳細データと採用判断

SECTION OVERVIEW

提示方針：三層構造

- **概要（Overview）：**
  カテゴリごとの主要トレンドと代表モデルの要点を概説。
- **完全版表（Full Data）：**
  規模・量子化・得意不得意・ランタイム相性・リスク・参考文献を列落ちなく全掲載。
- **採用判断（Decision）：**
  「どう選ぶか」の意思決定基準と失敗回避策を提示。

テキストLLM | VLM/LMM | ASR | TTS | Embedding | OCR | 画像生成 | Agent

テキストLLM（汎用）概要

結論：ローカル実務のボリュームゾーンは7B〜14B（4bit）。
形式はGGUF（llama.cpp系）かGPU量子化（GPTQ/AWQ/EXL2）の二択が現実解です。

代表モデル群（要点）

- **Llama 3.1 Instruct:** 8B/70B/405B系。多言語対話最適化を明記。 [29-31]
- **Qwen2.5 Instruct:** 0.5〜72B。幅広いサイズ展開と高性能。 [32-33]
- **Gemma 2:** 2B/9B/27B。9Bがローカル向き。責任ある利用を明示。 [35]
- **Mixtral 8x7B:** MoE構成。Llama2 70B超の性能を標榜。 [36-37]
- **Phi-3 Mini:** 3.8B (128k)。軽量かつ長文対応（KV設計必須）。 [38]
- **Phi-4:** 14B。合成データ活用で高品質。ローカル上位ライン。 [39-40]
- **gpt-oss:** 20B/120B。ツール呼び出しガイド充実。 [41]

Weights 4bit　GGUF　AWQ

実行環境と形式（Runtime & Format）

クロスプラットフォーム標準。CPU/GPUハイブリッド実行が可能。GGUF形式が必須。 [21][116]

MLX / MLX-LMによる変換・実行、またはGGUF (llama.cpp) が最適。Unified Memoryを活用。 [22][120]

vLLM / ExLlamaV2 / AutoGPTQ等。GPU量子化形式（AWQ/GPTQ/EXL2）で高速推論。 [87][121]

ライセンス注意（License Caution）
モデルファミリー内でもサイズやバージョンによりライセンス条件が異なる場合があります（例：Qwen2.5の一部例外など）。採用時は必ず各モデルカードおよび公式ブログの最新条件を確認してください。 [33]

テキストLLM（完全版表1）

主要モデル詳細比較：Llama 3.1 / Qwen2.5 / Gemma 2 / Mixtral

代表モデル	規模	推奨量子化/形式	得意/不得意（要点）	ランタイム相性	リスク/採用判断
Llama 3.1 Instruct Meta[29-31]	8B 70B 405B	**4bit中心** (GGUF等) ※方式は環境依存	得意多言語対話最適化を明記 得意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）	Risk 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 Alibaba[32-33]	0.5B 〜 72B	**小〜中は4bit** (GGUF/他)でローカル向き 72Bは上位ハード前提	得意幅広いサイズ展開を明記 得意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）	Risk「3B/72BはApache 2.0例外例外」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 Google[35]	2B 9B 27B	**2B/9Bはローカル向き** 27BはVRAM/メモリ要求増	得意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	Risk“事実質問用途での誤生成成”は一般に起こり得る（モモデル一般リスク）
		（量子化/実装依存）		**Mac:** 難易度高め	

⚠ **採用判断の要点**：ローカル実務のボリュームゾーンは7B〜14B級（4bit）です。3B級はハード制約時、32B以上は高品質要件時（メモリ増設前提）に選択します。ライセンスはファミリー内でも異なる場合があるため、必ず最新のモデルカードを確認してください。

Data Source: Technical Survey Report [29-37]

テキストLLM（完全版表2）

主要モデル詳細比較：Phi-3 Mini / Phi-4 / gpt-oss

代表モデル	規模	推奨量子化/形式	得意/不得意（要点）	ランタイム相性	リスク/採用判断
Phi-3 Mini Microsoft[38]	3.8B	**4bit**でローカル適性高い 長文版(128K)運用はKVが支配	得意「128K文脈」をうたうが長文はメモリ設計前提[38]	**Mac/Win:** 比較的回しやすい（形式依存）	Risk“小さい＝万能”ではなく、専門領域はRAG前提で補う判断
Phi-4 Microsoft[39-40]	14B	**4bit**でローカル上位ライン （32GB級以上推奨）	得意データ品質重視・合成データ活用を明示[40]	**Win GPU:** 最適化次第 **Mac:** MLX / 変換次第	Risk14Bは“重いが現実的”の境界。KV/長文設計が重要
gpt-oss OpenAI[41]	20B 120B	**MXFP4量子化**で出荷を明記 （他量子化なし）	得意ツール呼び出し・ローカルAPI利用までガイドあり[27]	**Mac/Win:** LM Studio / Ollamaで手順が提示される[41]	Risk20Bは16GB以上、120Bは60GB以上推奨を明記[41]

⚠ **採用判断の要点**：14Bクラス（Phi-4等）はローカル運用の「上位実用ライン」であり、32GB以上のメモリ環境が推奨されます。長文コンテキスト（128K等）を活用する場合は、モデルサイズだけでなくKVキャッシュのメモリ消費が支配的になるため、メモリ設計（KV量子化やコンテキスト長制限）が不可欠です。

Data Source: Technical Survey Report [38-41]

採用判断基準（テキストLLM）

結論：モデル選定は「必要品質→許容レイテンシ→許容メモリ→形式→ランタイム」の順に行うのが、実務上最も破綻しにくいフローです。

1. **必要品質**
   タスク難易度でサイズ決定
   (例: 70B vs 8B)
2. **許容レイテンシ**
   リアルタイム性かバッチ処理か
3. **許容メモリ**
   4bit量子化前提でVRAM/RAMに収まるか
4. **形式選定**
   GGUF or GPTQ/AWQ要件先行で決定
5. **ランタイム**
   形式に合うものを選択
   (llama.cpp / vLLM等)

実務の現実ライン

- **7B〜14Bがボリュームゾーン**
  一般業務やRAGにおいて、品質と速度のバランスが良いスイートスポット。
- **3B級はハード制約モード**
  メモリ不足時のフォールバックや、抽出特化タスクで使い分ける。

形式とライセンス

- **形式先行は詰まりやすい**
  ランタイム（例: LM Studio使いたい）から入ると形式制約で選択肢が狭まるため逆順推奨。
- **ライセンスの個別確認**
  Qwen2.5等、同ファミリー内でもサイズにより条件が異なる例外あり。 [33]

メモリ・最適化

- **長文はKVキャッシュが支配**
  コンテキスト長が増えるとKVメモリが急増する。
- **設計上の必須要件**
  コンテキスト設計に加え、KV量子化(FP8/INT4)やKV再利用(Reuse)の併用を前提に見積もる。

VLM/LMM 概要（画像理解・マルチモーダル）

結論：ローカル運用の現実ラインは2B/7B級が中心。
画像トークン化の前処理依存が強く、ランタイムの明示サポートが重要です。

代表モデルと特徴

Qwen2-VL (2B/7B/72B) [42]

ローカル現実ラインの主力。2B/7Bが実用的。72Bは上位ハード（96GB+）前提。

Phi-3-Vision (128K) [43]

軽量マルチモーダル＋長文対応を標榜。テキスト重視の画像理解に適正。

InternVL2 (4B等) [44-45]

画像理解の系列として設計。モデル群が複数サイズ展開。

LLaVA (7B/13B系) [46-47]

画像チャット用途の先駆者。4bit運用で12GB VRAMでも動作可能。

ランタイム相性と技術課題

macOS / Apple Silicon

MLX-VLM: Qwen2-VL等の利用例を具体コマンドで提示。[28]
Unified Memoryを活用し、システムメモリで大型モデルを実行可能。

Windows / GPU

transformers / vLLM: GPU推論が主流。モデル形式（Safetensors/AWQ等）とランタイムの対応に依存。

技術的ボトルネック

画像トークンが増えるとメモリ/速度が急落。解像度制御やmax-pixels設定が運用の肝となる。

Visual Tokens　Multi-modal RAG　Resolution Control

設計含意（Design Implication）

VLM/LMM（完全版表）

Qwen2-VL / Phi-3-Vision / InternVL2 / LLaVA 詳細比較

代表モデル	規模	推奨量子化/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 Qwen[42]	2B 7B 72B	**2B/7B**がローカル現実ライン 72Bは上位ハード前提	得意画像＋テキストの汎用 画像トークン増でメモリ/速度急落→解像度制御が肝	**Mac:** MLX-VLMが利用例を明明示[28] **Win:** transformers / vLLM	Risk視覚トークン肥大によるリソース枯渇
Phi-3-Vision Microsoft[43]	4.2B (128K)	形式依存でローカル適性あり 128K長文対応	得意軽量マルチモーダル＋長文を標榜 “実務的なOCR読取”に強み	**Mac:** MLX変換次第 **Win:** GPU推論	Risk長文はKVが支配しやすく、メモリ管理が必須[43]
InternVL2 OpenGVLab[44-45]	1B/2B 4B/8B等	**4B等**がローカル向き モデル群が複数サイズ展開	得意画像理解の系列として設計 構成要素をモデルカードで詳細説明[45]	**Win:** GPU / 形式次第 **Mac:** 実装依存	Risk エコシステム差（変換・プロセッサ依存）が運用の難所

⚠ **採用判断の要点**：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ため、「ランタイムがそのモデルを明示サポートしているか」を一次確認してください（例：MLX-VLMがQwen2-VLのコマンド提示）。72B級はローカルでは上位機（96GB+等）が必要で、現実は2B/7B級で設計します。

Data Source: Technical Survey Report [28, 42-47]

採用判断基準（VLM/LMM）

結論：画像理解は前処理・プロセッサ依存が強いため、「ランタイムの明示サポート確認」が最優先です。2B/7B級を基本とし、不足分をRAG/OCRで補うのが安全策です。

サポート確認
README等でランタイム(MLX/vLLM)の対応を確認 [28]

規模選定
2B/7B級で設計開始(72B級は上位HW要件)

トークン制御
解像度・max-pixels制御でメモリ爆発を防ぐ

ハイブリッド
OCRやRAGを併用して精度と速度を補完

ランタイム適合性

一次情報の確認
ランタイムがモデルを明示サポートしているか確認が安全（例: MLX-VLMのQwen2-VL対応コマンド提示）。

プロセッサ依存性
テキストLLMより前処理（画像トークン化）の依存が強く、変換ミスが起きやすい。

運用設計（2B/7B級）

現実的なライン
72B級は96GB+メモリや複数GPUが必要なため、実務では2B/7B級での運用設計に寄せる。

視覚トークン制御
長文・多画像時はKV支配とトークン肥大に注意。max-pixels/マルチ画像数を制御する。

補完戦略（OCR/RAG）

不得意のカバー
微細な文字認識や構造化はOCR（PaddleOCR等）に任せ、VLMは「意味理解」に集中させる。

ハイブリッド構成
画像から情報をテキスト抽出し、Embedding化してRAGで検索可能にする設計が堅牢。

ASR（音声認識）概要

結論：ローカルASRは「RTF・メモリ・誤転記」の同時最適化が必須。
faster-whisperを基準実装とし、速度と精度（幻覚抑制）のバランスを図ります。

Whisper系実装の比較

実装名	特徴・最適化	推奨用途
Whisper (Original)[48]	研究/公開ベース、PyTorch依存	精度検証・基準
faster-whisper[50]	CTranslate2、8bit量子化、最大4倍速	実務の第一選択
whisper.cpp[51]	C/C++軽量、ggml、Apple Silicon最	組込み・省メモリ

主な技術要素:

CTranslate2 | VAD Integration | INT8 Quantization | Beam Search

最適化戦略とリスク管理

faster-whisperは同等精度でメモリ消費を大幅削減。 RTF[50]（実時間より速い）を目標にベンチマークを実施します。

高リスク領域では誤転記が実害になる可能性が指摘されています。 V[49]無音除去）や信頼度スコアによるフィルタリングが不可欠です。

文字起こし精度だけでなく、VAD・話者分離（Diarization）の品質が全体のUXを決定します。

設計のポイント（Key Takeaways）

会議議事録などの長尺音声では、単にASRモデルを回すだけでなく、前処理（VADによる無音カット）と後処理（LLMによる整文）を組み合わせたパイプライン設計が品質を左右します。まずはfaster-whisperでベースラインを構築することを推奨します。

ASR（完全版表）

主要モデル・方式比較：Whisper / faster-whisper / whisper.cpp

代表モデル/方式	規模	推奨量子化/形式	得意/不得意（要点）	ランタイム相性	リスク/採用判断
Whisper OpenAI[48-49]	Tiny 〜 Large-v3	**実装多数** （下記派生に分岐）	得意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]	**汎用:** 直接実装 / 各派生	Risk高リスク領域では“誤転記記（幻覚）”が実害になる可可能性が報道されており、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	得意同等精度で最大4倍速・低メモリを主張[50]	**Win/Mac/Linux:** Pythonで実装しやすい	Rec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 ggml[51]	Whisper 互換	**ggml系**で ローカル最適化	得意C/C++で軽量運用、各種最適化例例が豊富	**Mac/Win:** 共に導入可能	Info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）[51]

⚠ **採用判断の要点**：ローカルASRは「RTF（実時間比）」「メモリ」「誤転記リスク」の3要素を同時に満たす必要があります。faster-whisperは速度・メモリ面での利点を明示しているため、まずはこれを基準実装として検証し、要件に応じてGPU化やモデルサイズの調整を行うのが合理的です。

Data Source: Technical Survey Report [48-51]

採用判断基準（ASR）

RTF要件と誤転記リスクに基づく選定フロー

結論：faster-whisperを基準実装とし、RTF要件・誤転記リスクに応じてGPU化やモデル拡大を検討する段階的最適化が合理的です。

1. **基準実装**
   faster-whisper (CPU/INT8)で検証
2. **RTF評価**
   実時間比(RTF)<1 を満たすか確認
3. **品質評価**
   誤転記(幻覚)の許容範囲内か
4. **最適化**
   不足ならGPU化 またはモデル拡大

前処理の固定

- **VAD設定の統一**
  無音区間の除去は認識精度と速度に直結するため、Silero VAD等の設定を固定し再現性を担保する。
- **ベンチマーク条件**
  同一音声・同一前処理での比較が必須（後章のベンチ設計参照）。

会議用途の要点

- **話者分離が品質の土台**
  pyannote.audio等の話者分離（Diarization）精度が議事録の質を左右する。
- **無音除去の設計**
  幻覚（無音区間にテキストが入る現象）を防ぐため、VADによる無音区間の破棄を徹底する。

リスク管理

- **誤転記（幻覚）への対処**
  医療や重要業務では誤転記が実害となる可能性があるため、検証プロセスを必須化。[49]
- **リソース監視**
  ストリーミング用途ではRTFだけでなく、長時間稼働時のメモリリーク等も監視対象とする。

TTS（音声合成）概要

読み上げと音声クローンの分離設計

結論：「読み上げ」は軽量モデル（Piper/Kokoro）で常駐化し、「音声クローン」は高品質モデル（XTTS/StyleTTS2）で分離設計します。

代表システムと特徴

Piper: "fast, local neural TTS" を標榜。省リソース・常駐向き。 [52]

Kokoro: 82Mパラメータで高品質・高効率。 [53]

XTTS-v2: 数秒の参照音声で多言語クローンが可能。 [54]

StyleTTS2: 研究系で人間レベルの自然さを追求。 [55]

Text-to-Speech　Voice Cloning

運用の分離とリスク管理

軽量TTSを採用し、システム負荷を最小化。バックグラウンドでの安定動作を優先。

GPUリソースを割り当て品質を最大化。ただし、法務・権利リスク管理（許諾・ログ監査）を必須要件とします。 [56]

Risk Management　Resource Optimization

日本語品質への注意（Quality Assurance）

日本語の品質（アクセント、抑揚）は、モデルだけでなく「辞書」や「テキスト前処理」に強く依存します。業務投入前には、特定のドメイン用語を用いた事前検証が不可欠です。

TTS（完全版表）

主要モデル詳細比較：Piper / Kokoro / XTTS-v2 / StyleTTS2

代表モデル/方式	規模	推奨量子化/形式	得意/不得意（要点）	ランタイム相性	リスク/採用判断
Piper Fast/Local[52]	軽量	**ローカル常駐** TTS向き	得意"fast, local neural TTS"を明示 得意省リソース環境で有利	**Win/Mac:** 導入容易	Risk リポジトリ移転あり（運用はフォーク/移転先を確認）[52]
Kokoro Open Weight[53]	82M	**軽量TTS**として 使い分け	得意82Mで高速・コスト効率を主張 得意Apacheライセンス重みをうたう[53]	**CPU:** 成立しやすい	Note 日本語品質は声・辞書・前処理に依存（事前確認必須）
Coqui XTTS-v2 Clone[54]	大きめ	**GPU推奨** 音声クローン寄り	得意数秒の参照音声で多言語クローンをうたう[54]	**GPU環境:** 実務的	Risk 法務・倫理・権利リスク大（同意・監査が必須）[54,56]
StyleTTS2 Research[55]	研究系	**高品質TTS**志向	得意"human-level"を目標とする研究系[55]	**GPU:** 望ましい	Risk 実運用は依存関係・再現性の確認が必須

⚠ **採用判断の要点**：「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します。前者はPiper/Kokoroのような軽量系、後者はXTTS等で、法務・倫理リスク管理を別レイヤに置きます。[56]

Data Source: Technical Survey Report [52-56]

採用判断基準（TTS）

結論：「読み上げ（通知・要約）」と「クローン（本人声）」は別レイヤで運用すべきです。前者は軽量化し、後者は法務・倫理リスクを管理します。

1. **用途定義**
   単なる読み上げか
   本人性の再現か
2. **モデル選定**
   軽量(Piper) vs
   高品質(XTTS)
3. **リスク判定**
   声の権利許諾と
   ログ監査設計
4. **実装・運用**
   常駐(CPU) or
   オンデマンド(GPU)

分離運用（Layered）

- **軽量TTSの常駐**
  通知・要約読み上げはPiper/Kokoro等で省リソース化し、システムに常駐させる。
- **クローンは別系統**
  XTTS等はGPUリソースを消費するため、必要な時のみ呼び出す別サービスとして切り出す。

リスク・コンプライアンス

- **権利同意の必須化**
  音声クローンは法務・倫理リスクが高い。業務利用時は対象者の書面同意をプロセス化する。
- **ログ監査**
  「誰が・いつ・何を」生成したかのログを保存し、不正利用を追跡可能にする。[56]

品質保証（QA）

- **日本語品質の事前試験**
  モデルによりアクセントや読み間違い（数値・固有名詞）の差が大きい。
- **辞書・前処理の標準化**
  業務特有の用語はユーザー辞書や前処理ルール（正規化）でカバーする義務を課す。

Embedding / Reranker 概要

結論：RAGシステムの品質上限はEmbeddingの検索精度で決まります。
まずBGE-M3等でRecallを確保し、RerankerでPrecisionを向上させる二段構えが定石です。

Embedding (ベクトル検索)

Retrieval　Vector DB

- Multi-Functionality / Multi-Linguality / Multi-Granularityを特徴とし、多言語・多用途で強力なベースラインとなります。 [58]
- 多言語検索の定番モデル。日本語を含む多言語環境での実績が豊富です。 [59][60]

Reranker (再順位付け)

Precision　Cross-Encoder

- クエリと文書をペアで入力し、関連度スコアを直接出力します。計算コストは高いですが、検索精度を大幅に引き上げます。 [61][62]
- Embedding検索で広めに取得した候補（Top-K）を絞り込み、LLMへのコンテキスト汚染を防ぎます。

運用上の設計含意（Operational Design）

ローカル運用では、リソース競合を避けるため「Embedding化（前計算）」を夜間バッチ等で済ませておくことが推奨されます。これにより、日中の推論時には実時間負荷をLLMの応答生成に集中させることが可能になります。

Embedding/Reranker（完全版表）

検索・RAG基盤：BGE-M3 / multilingual-e5 / bge-reranker

代表モデル	規模	推奨量子化/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 BAAI[58]	Model Card 参照	**FP16/INT8**で安定運用 必要なら量子化	特徴Multi-Functionality Multi-Linguality Multi-Granularityを明示[58]	**CPU/GPU/ONNX** 等で運用可能 （環境依存）	推奨RAGはEmbedding品質が が上限を決めるため、まず ここを堅くする
multilingual-e5 Intfloat[59-60]	large large-instruct	**large** 多言語検索用途で定番	実績技術報告が示され、使用例が 明確[60]	**transformers / ONNX**	候補日本語含む多言語を要す るなら有力候補
bge-reranker BAAI[61-62]	278M ～ 560M等	**large/v2等** RAGの“再ランキング”で 精度向上	機能「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）	Riskrerankerはレイテンシを増 増やすため、p95要件で採否 を判断

⚠ **採用判断の要点**：RAGの失敗の多くは「検索外れ」「上位が弱い」「文脈過多」です。EmbeddingでRecallを確保し、rerankerでPrecisionを上げる二段構えが実務的推奨です。ローカル運用では、Embeddingの前計算（夜間バッチ）により実時間の負荷をLLM応答に集中させる設計が有効です。

Data Source: Technical Survey Report [58-62]

採用判断基準（Embedding / Reranker）

検索精度と運用安定化のための二段構え設計

結論：Embeddingで網羅性(Recall)を確保し、Rerankerで精度(Precision)を上げる「二段構え」が実務的な最適解です。

1

Embedding

広く候補を取得 (Recall重視)
BGE-M3 / E5

2

Reranker

上位を厳選 (Precision重視)
bge-reranker

3

LLM生成

精度の高い文脈で回答
(幻覚抑制)

運用の安定化

前計算（夜間バッチ）
検索対象文書のEmbedding化やOCR処理は夜間にバッチ実行し、日中の計算資源をLLMの応答生成に集中させる。

インデックス更新
頻繁な更新が必要ない場合は、静的インデックスとして管理し、運用負荷を下げる。

失敗パターンと回避策

検索が外れる（Recall不足）
回避策：文書の前処理（チャンク分割）を見直す、またはHybrid検索（キーワード検索併用）を導入する。

上位文書の関連度が弱い
回避策：Rerankerを導入して上位の並び順を補正し、LLMに渡すノイズを減らす。

文脈長対策

文脈が長すぎる失敗
検索結果を詰め込みすぎるとLLMの「Lost in the Middle」現象やメモリ枯渇を招く。

Rerankerでの絞り込み
Rerankerでスコアの低い文書を大胆にカットし、LLMには「確度の高い少数精鋭」のコンテキストを渡す。

OCR / Document AI 概要

スキャン品質と構造化要件に基づく手法選択

結論：スキャン品質やレイアウトの複雑度に応じて、古典的OCRとOCR-free（Doc理解）を使い分けるハイブリッド戦略が推奨されます。

代表的なシステム・モデル

Tesseract OCR [64]

古典的OCR＋LSTM。単純なテキスト化のデファクトスタンダード。

PaddleOCR [66]

高精度かつ多機能。表構造や縦書きなど複雑なレイアウトに強い。

Donut (OCR-free) [68]

画像を直接Transformerで処理し、構造化データを抽出。

LayoutLMv3 [69]

テキスト、画像、レイアウト情報を統合して文書理解を行う。

運用指針と使い分け

高品質スキャン・単純文書：

Tesseractを採用。高速かつ軽量で、単純なテキスト化に最適。

低品質・複雑レイアウト・構造化：

PaddleOCR、Donut、LayoutLMv3を採用。傾きやノイズに強く、フォームや表の理解が可能。

Hybrid Strategy　Layout Aware　Structure Extraction

RAG精度への影響（Key Takeaway）

OCR / Document AI（完全版表）

主要モデル詳細比較：Tesseract / PaddleOCR / Donut / LayoutLMv3

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
Tesseract OCR Google[64]	古典OCR + LSTM	**テキスト化基盤** スキャン品質高なら CPUで十分	得意 長年の実績と安定性 不得意 複雑レイアウトや手書き文字は苦手	**Cross-Platform:** Win/Mac/Linux問わず導入容易
PaddleOCR Baidu[66]	Deep Learning OCR	**高精度・多機能** 構造化出力向け	得意 軽量かつ高精度な認識 特徴 表認識やレイアウト解析機能も充実	**Python / ONNX:** Python環境で容易に実装可能
Donut Clova AI[68]	OCR-free Transformer	**Doc理解・抽出** 画像から直接 JSON等へ	得意 OCR誤りを経由せず直接情報抽出 特徴 フォームや請求書等の定型文書に強み	**Transformers:** Hugging Face Transformers対応
LayoutLMv3 Microsoft[69]	Multimodal Transformer	**レイアウト保持** 文脈理解統合	得意 視覚情報とテキスト情報を統合して理解 特徴 表/図/レイアウト由来の欠落を補完[69]	**Transformers / ONNX:** 推論エンジンでの最適化が可能

⚠ **レイアウト保持の重要性**：RAGにおいて、単なるテキスト化では図表やレイアウト構造に含まれる情報が欠落しがちです。PaddleOCRの構造化出力や、Donut/LayoutLMv3のようなDocument AIモデルを活用し、レイアウト情報を保持することが検索精度の向上に直結します。

Data Source: Technical Survey Report [64, 66, 68, 69]

OCR/Document AI採用判断基準

結論：誤読リスクをゼロにはできない前提で、前処理の標準化と「原文引用」による人間系確認のUXを要件化します。

1. **品質評価**
   スキャン品質と
   レイアウト複雑度を確認
2. **前処理**
   傾き補正・二値化の
   標準パイプライン化
3. **モデル選択**
   PaddleOCR(構造化)
   +Donut(レイアウト)
4. **UX要件化**
   原文スニペット表示で
   誤読リスクを緩和
5. **運用設計**
   OCR/Embeddingは
   夜間バッチ処理へ

前処理の標準化

品質の下限を担保
傾き補正、ノイズ除去、二値化をOCR前段に固定的に組み込むことで、認識精度のベースラインを確保。

画像の正規化
解像度やコントラストのバラつきを抑え、モデルの得意な入力形式に合わせる。

高品質方針：ハイブリッド

構造化とレイアウト保持
PaddleOCRでテキスト構造化を行い、Donut/LayoutLMv3でレイアウト情報を保持してRAG連携精度を高める。

図表・UIへの対応
複雑な図表やUIスクショはVLMとOCRを併用し、視覚情報と文字情報を相互補完させて堅牢化する。

UX要件と運用

原文スニペット引用
AI回答の根拠となった原文箇所（画像切り出し等）を提示し、ユーザーが誤読を検証できるUXを必須化。

夜間バッチへのオフロード
重いOCR/Embedding処理は夜間に回し、日中のリソースは検索と短い回答生成に集中させる。

画像生成（Diffusion）概要

モデル・ランタイム・実務ポイント

結論：画像生成はSDXLを軸に、ComfyUI/SD WebUIで運用。
FLUX等はライセンス精査が必須であり、VRAM要件はワークフローに依存します。

主要モデルとUI（Runtime）

- 高品質生成の標準モデル。ライセンス確認の上、低コスト構成の軸に。 [71]
- ノードベースUI。Windows/Linux/macOS対応を明示し、ワークフローの再利用性が高い。 [72]
- 豊富なプラグインエコシステムを持つ定番UI。 [73]
- 最新の上位モデル。高い品質ポテンシャルを持つが、商用利用等のライセンス条件（dev版等）に注意。 [75][76]

Diffusion　ComfyUI　VRAM Optimization

実務運用ポイントとリスク

- 解像度・ステップ数・バッチサイズによりVRAM消費が激増します。attention slicing等の最適化適用が必須。
- 特にFLUX.1-dev等はライセンス条件が異なる場合があるため、採用前に必ず確認してください。 [76]
- 業務利用では「seed固定」と「ワークフロー固定（ComfyUI json）」で出力を安定させます。

画像生成（完全版表）

主要モデル・UI詳細比較：SDXL / ComfyUI / SD WebUI / FLUX

モデル / UI	方式 / 特徴	VRAM要件	得意/不得意（要点）	リスク/採用判断
SDXL base 1.0 Stability AI[71]	Diffusion	**ワークフロー依存** （解像度・バッチ・ステップ数に大きく左右される）	得意 高品質生成（ベースモデルとして標準的） 注意 解像度・バッチ設定で負荷が大幅変動	判断 attention slicing等の最適化を前提にプロファイルを固定化
ComfyUI UI[72]	ノード型UI Win/Linux/macOS対応	**構成次第** （効率的なメモリ管理が可能）	得意 再利用性・資産化（ワークフロー保存） 得意 seed固定による再現性担保	判断 ワークフローのブラックボックス化を防ぐため管理が必要
Stable Diffusion WebUI UI[73]	プラグイン豊富	**設定・拡張依存**	得意 導入が比較的容易 注意 設定差による品質ブレが発生しやすい	判断 簡易利用には適するが、厳密な再現性には注意
Black Forest Labs[75]		（高品質化のリソース要求）		商用利用可否を必ず確認

⚠ **採用判断の要点**：VRAM容量が最大の制約となります。解像度、バッチサイズ、ステップ数を固定したプロファイルを作成し、VRAM不足を防ぐ運用設計が重要です。特にFLUX等の最新・上位モデルを採用する場合は、ライセンス条件（商用利用制限など）を事前に入念に確認してください。

Data Source: Technical Survey Report [71-76]

\\n\\n\\n\\n\\n\\n\\n\\n\\n\\n\\n\\n
\\n
\\n
\\n\\n**採用判断基準（画像生成）**\\n
\\n
VRAM制約と再現性の担保
\\n
\\n
\\n\\n
\\n

\\n 結論：VRAM制約に合わせて解像度/バッチ/ステップを固定し、ワークフローを資産化して再現性を担保します。\\n

\\n
\\n\\n
\\n
\\n
1
\\n
VRAM制約
\\n
ハードウェア上限を
まず確認
\\n
\\n
\\n
\\n
2
\\n
解像度固定
\\n
VRAMに収まる
最大サイズを決定
\\n
\\n
\\n
\\n
3

Agent / Tool-use 概要

結論：Agentは「LLM＋ツール呼び出し（関数）＋実行環境」で構成され、Ollama/LM StudioのOpenAI互換APIを用いることで、ローカルでの実務的な構築が現実化しています。

構成要素とAPI基盤

エージェントの基本構成。LLMが判断し、定義されたツール（関数）を呼び出し、ローカル環境で実行して結果を返すループ構造。

OllamaやLM Studioは、ローカルで動作しながらOpenAI互換のエンドポイントを提供。これにより、既存のAgentフレームワークやクライアントツールをそのまま利用可能。[23][24][41]

Function Calling (Tool Use)

ローカルAPIでも tool_choice や functions パラメータを利用可能。モデルがJSON形式で引数を生成し、システム側で実行します。[24][27]

ローカルモデル（特に小型）ではJSONスキーマの破壊が発生しやすいため、型定義の厳格化と、パース失敗時の再試行ロジックが不可欠です。

設計含意（Design Implication）

JSONスキーマ遵守と検証ループ設計（失敗時のリトライ/フォールバック）がローカルAgentの成否を分けます。特に「幻覚API（存在しない関数呼び出し）」への対策として、実行可能な関数リストのホワイトリスト化が必要です。

OpenAI API Compatible　Local Execution　JSON Schema　Verification Loop

Agent/Tool-use（完全版表）

OpenAI互換API / Function Calling / 実装基盤の詳細比較

項目・方式	詳細仕様・提供形態	利点・特徴	注意点・リスク
OpenAI互換API インターフェース[24,41]	**提供**：Ollama / LM Studio ローカルLLMを標準的なREST APIとして公開 Chat Completions API互換	互換性既存のエージェントフレームワークやクライアントツールをそのまま接続可能	仕様差一部機能（Stateful等）が完全互換ではない場合があるため検証が必要
Function Calling Tool Use[24,27]	**出力形式**：構造化JSON LLMがツール実行を判断し、引数をJSONで生成	構造化自然言語ではなく実行可能なデータ構造で出力 実務的業務システム連携の核	JSON破壊スキーマ遵守率（破壊率）はモデル性能と量子化に依存
実装基盤: Ollama / LM Studio Runtime[3,19,23]	**Ollama**：CLI/API中心、Modelfile管理 **LM Studio**：GUI中心、サーバー機能内蔵	導入容易複雑な環境構築なしでAPIサーバー化	設定コンテキスト長やGPUオフロード設定の最適化が必要

実装のポイント：安定運用の鍵は「JSONスキーマの固定」と「パラメータ（温度/top_p）の固定」です。幻覚（Hallucination）による不正な関数呼び出しを防ぐため、実行前に許可された関数リストと照合するホワイトリスト方式を推奨します。また、トラブルシューティング用にAPIの要求・応答ログを必ず保存してください。

Data Source: Technical Survey Report [2,3,19,23,24,27,41]

採用判断基準（Agent / Tool-use）

結論：JSONスキーマ固定＋検証ループ（自動チェック）による安全運用を基本とし、Ollama/LM StudioのOpenAI互換APIで実装を統一します。

スキーマ定義

必須引数・型・制約を厳格化して固定

ツール実行

Ollama/LM Studio経由 OpenAI互換API

検証ループ

コンパイル/テスト実行 失敗時は再生成

完了/リトライ

結果確認または フォールバック

スキーマと検証

- **JSONスキーマの厳格化** 必須引数、型定義、Enum制約を厳密に記述し、モデルの構造化出力能力を最大限に活用。破壊時は即時再生成へ。
- **検証ループの実装** 「コード生成→コンパイル/テスト実行」自体をツールとして組み込み、LLM自身に結果をフィードバックするループを構築。

実装基盤の統一

- **OpenAI互換APIで統一** O[24,41]M Studio等の互換APIを採用し、クライアントコードを標準化。移行性・保守性を確保。
- **ランタイム機能の活用** 必要に応じて、llama-cpp-python等のランタイム固有機能（文法制約機能など）を補助的に活用。[25]

幻覚API対策

- **ホワイトリスト化** 実行可能な関数セットを厳密にホワイトリスト化し、存在しないAPIの呼び出し（幻覚）をシステム側で遮断。
- **タイムアウトとリトライ** 無限ループや応答遅延を防ぐため、標準でタイムアウト設定とリトライ上限を設ける設計を義務化。

音声周辺（VAD/話者分離/ウェイクワード/ノイズ除去）概要

結論：前処理（VAD/話者分離/ウェイクワード/ノイズ除去）が音声UXの品質下限を決定します。
Silero VADによる無音除去とpyannote.audioによる話者分離が実務上の標準構成です。

主要コンポーネント（Primary）

VAD（Voice Activity Detection）：Silero VAD [82]

無音・雑音区間を事前除去し、ASRの幻覚（Hallucination）を抑制。軽量・高精度でデファクトスタンダード。

話者分離（Diarization）：pyannote.audio [84]

「誰が話したか」を特定。会議議事録の品質に直結するが、依存関係が重く計算コストも高い。

Silero VAD　pyannote.audio

補助コンポーネント（Secondary）

ウェイクワード（Wake Word）：openWakeWord [85]

「Hey Siri」等の起動語検出。オンデバイスで動作し、誤検知（False Positive）とのトレードオフ設計が鍵。

ノイズ除去（Noise Suppression）：RNNoise [14]

RNNベースの軽量ノイズ抑制。実用十分な性能だが、過度な適用は音質劣化を招くためチューニングが必要。

openWakeWord　RNNoise

設計含意（Design Implication）

これら前処理の設定差（閾値、モデルバージョン）で最終的な認識精度が大きく変動します。ベンチマーク時は「同一前処理条件」を固定することが再現性の担保に不可欠です。

音声周辺（完全版表）

Silero VAD / pyannote / openWakeWord / RNNoise 詳細比較

代表モデル/ツール	役割	実行要件/形式	利点/特徴	リスク/注意点
Silero VAD Snakers4[82]	**無音/雑音除去** (Voice Activity Detection)	軽量・CPU動作可 ONNX / PyTorch	標準 前処理のデファクト ASR負荷を大幅に削減	Attention 閾値設定に依存（誤って語頭を切るリスクあり）
pyannote.audio HuggingFace[84]	**話者分離** (Speaker Diarization)	GPU推奨 依存関係が重め	高精度 会議録で「誰が話したか」を特定するのに必須	Risk 処理負荷が高い 環境構築の難易度がやや高い
openWakeWord dscripka[85]	**ウェイクワード検出** (Wake Word Detection)	ローカル動作 CPU / tflite等	実用 カスタムウェイクワード作成が可能 非常に軽量	Attention 誤検知（False Positive）の調整が必要
RNNoise Xiph.Org[14]	**ノイズ除去** (Noise Suppression)	極めて軽量 C/C++ / WebAssembly	高速 リアルタイム処理向き RNNベースで背景雑音を抑制	Risk 音質変化（声がロボットっぽくなる等）のリスク

設計の推奨：会議系ワークフローでは「VAD → ASR → 要約」の直列処理を基本とし、負荷の高い話者分離（pyannote等）は必要時のみ追加する設計が実用的です。すべての音声に一律で話者分離を適用すると、処理時間（RTF）が大幅に悪化する可能性があります。

Data Source: Technical Survey Report [14, 82, 84, 85]

採用判断基準（音声周辺）

結論：前処理パイプラインを標準化し、閾値・モデルバージョン・RTF目標を固定することで、UXの品質下限を担保します。

1. **ユースケース定義**
   会議録音か
   ウェイクワードか
2. **パイプライン構成**
   VAD+ASR+分離など
   プリセット化
3. **閾値固定**
   VAD感度・分離閾値
   を数値で管理
4. **バージョン固定**
   Silero/pyannoteの
   バージョン統一
5. **ログ監視**
   RTF・エラー率の
   継続モニタリング

パイプラインの標準化

- **基準設定のプリセット化**
  VAD閾値、話者分離の有無、ノイズ除去の有無をユースケースごとに固定セットとして定義。
- **再現性の担保**
  モデルバージョンとパラメータをコードで固定し、環境による挙動差を排除。

ログ管理・監査

- **パフォーマンス監視**
  RTF（実時間係数）とエラー率をログに保存し、処理遅延や異常を検知。
- **長時間処理のオフロード**
  長時間の音声処理は夜間バッチ化し、日中の計算資源をASRや要約などの対話的タスクに集中。

誤検知対策

- **UIによる再確認設計**
  誤検知（ウェイクワード）や誤割当（話者分離）が発生した場合、ユーザーが手動修正できるUIを用意。
- **安全側の設計**
  重要な判定では、自動処理の結果を「提案」として提示し、最終決定をユーザーに委ねるフェイルセーフ設計。

07

メモリ設計のコア

重み量子化・KVキャッシュ管理と最適化技術

KEY TAKEAWAYS

本章の要点

- **重みメモリの現実解：**
  4bit量子化（weight-only）が基本。理論値＋10%オーバーヘッドで設計します。
- **KVキャッシュの支配性：**
  長文コンテキストではKVがメモリを圧迫。vLLMのpaged/quantized KV[5,6]やTensorRT-LLMのKV reuse[7]が必須です。
- **最適化技術の活用：**
  llama.cppのprompt cache[88,90]等でTTFTを短縮し、実用性を高めます。

重み理論値計算 | KVキャッシュ詳細 | 数式・数値例 | 最適化技術

メモリ設計のコア（概要）

結論：ローカルAIのメモリ制約は、「重み4bit量子化」と「KVキャッシュ最適化」の2点によって決定される支配的な要因です。

重みメモリ（Weights）

パラメータ数とbit幅で物理的な下限が決まります。

```
メモリ ≈ パラメータ数 × bit幅 ÷ 8 （+約10% ランタイム/CUDA オーバーヘッド）
```

FP16（16bit）と比較して約1/4のサイズで、品質劣化を最小限に抑えつつコンシューマGPUに載せるための必須技術です。

GGUF Q4_K_M　AWQ 4bit　GPTQ

KVキャッシュと最適化

KVキャッシュは「コンテキスト長に比例」して増加します。長文・多同時接続では重み以上にメモリを圧迫します。

例：Llama3 8B (n_layer=32, n_head_kv=8, head_dim=128)

- **Paged KV:** メモリ断片化を防ぐ（vLLM等） [5]
- **Quantized KV:** FP8/INT4化で容量削減 [6]
- **KV Reuse:** 計算再利用（TensorRT-LLM） [7]
- **Prompt Cache:** TTFT短縮（llama.cpp） [88][90]

重みメモリ理論値（完全版表1：0.5B〜14B）

モデル規模別 推奨メモリ容量（+10%オーバーヘッド込み概算）

モデル規模 (Parameters)	4bit (Q4_K_M等) ローカル推奨	INT8 (8bit) 標準的量子化	FP16 (16bit) 元精度/学習時
0.5B (Qwen2.5-0.5B等)	0.3GiB	0.5GiB	1.0GiB
1.5B (Qwen2.5-1.5B等)	0.8GiB	1.5GiB	3.1GiB
3B (Phi-3 Mini等)	1.5GiB	3.1GiB	6.1GiB
7B (Qwen2.5-7B等)	3.6GiB	7.2GiB	14.3GiB
8B (Llama 3.1 8B等)	4.1GiB	8.2GiB	16.4GiB
14B (Qwen2.5-14B/Phi-4等)	7.2GiB	14.3GiB	28.7GiB

計算根拠：上記数値は「パラメータ数 × bit幅 ÷ 8」に、ランタイムオーバヘッドとして約10%を加算した概算理論値です。これらは**重みのみ**のメモリ消費であり、実運用ではこれに加えてKVキャッシュ（コンテキスト長に比例）が必要となります。特に7B〜14Bモデルは、8GB/16GBメモリ環境での動作可否の境界線となるため、4bit量子化の活用が実務上必須となります。

Data Source: Technical Survey Report (Calculated Estimates)

重みメモリ理論値（完全版表2：27B〜70B）

大規模モデル（4bit/INT8/FP16）のメモリ要件

モデル規模	4bit (Q4) 推奨	INT8 (Q8)	FP16 (Half)
27B Gemma 2 27B等	**13.8**GiB	**27.7**GiB	**55.3**GiB
32B Qwen2.5 32B等	**16.4**GiB	**32.8**GiB	**65.6**GiB
34B Yi-34B等	**17.4**GiB	**34.8**GiB	**69.7**GiB
70B Llama 3 70B等	**35.9**GiB	**71.7**GiB	**143.4**GiB

計算前提： パラメータ数 × bit幅 ÷ 8 で基本容量を算出後、実運用におけるランタイムオーバヘッド等を考慮して約+10%を加算した概算値です。

設計含意： 30B級モデルの実運用には、4bit量子化でも約16-18GiBのVRAM/統合メモリが必要です。70B級では4bitでも約36GiBを消費するため、Apple Silicon 64GB/96GBや、VRAM 24GB×2枚構成などの上位ハードウェア構成が現実的なラインとなります。

Source: Technical Survey Report (Calculated Values based on Weight Quantization)

KVキャッシュの支配性（完全版表1）

2k〜32k tokensにおけるメモリ消費量比較（Llama3 8B相当）

コンテキスト長 (Tokens)	FP16 (標準) (16-bit)	FP8 (量子化) (8-bit)	INT4 (量子化) (4-bit)
2,048 tokens 標準的な短文対話	**0.25 GiB**	**0.12 GiB**	**0.06 GiB**
8,192 tokens 長文記事・文書要約	**1.00 GiB**	**0.50 GiB**	**0.25 GiB**
32,768 tokens RAG / 全体像把握	**4.00 GiB**	**2.00 GiB**	**1.00 GiB**

設計含意：KVキャッシュはコンテキスト長に比例して線形増加します。32kトークンなどの長文コンテキストでは、FP16のままでは4GBものVRAMを消費し、モデル重み（8B 4bitで約4.5GB）と合わせると8GB VRAMの限界を超えます。FP8/INT4量子化やPaged KV Cacheの導入が、長文運用成立の鍵となります。 [5-7]

※計算前提: Llama3 8B (n_layer=32, n_head_kv=8, head_dim=128), GQA有効

Data Source: Technical Survey Report [5-7]

KVキャッシュの支配性（完全版表2：超長文域）コンテキスト長131k tokensにおけるメモリ消費量比較（Llama3 8B相当）

コンテキスト長	FP16 (Base)	FP8 (Optimized)	INT4 (Highly Optimized)
131,072 tokens128k context	**16.00 GiB**Critical モデル本体(8B)と合わせると32GB超	**8.00 GiB**Heavy 50%削減（実用域へ）	**4.00 GiB**Manageable 75%削減（余裕あり）

メモリ内訳の逆転現象

超長文（128k等）では、**KVキャッシュのメモリ消費量がモデル本体（重み）のメモリ消費量を上回る**現象が発生します。
例：Llama3 8B（4bit重み≒4.1GiB）に対し、131k tokensのKV（FP16）は16.00GiBに達し、総メモリの約80%をKVが占有します。

設計による回避策（Design Implications）

要約・圧縮：Rolling Summary等でコンテキストを常に一定長以下に保つ。
分割処理：長文を一括入力せず、チャンク分割して処理（Map-Reduce等）。
RAG活用：全文をコンテキストに入れず、Embedding検索で必要箇所のみ抽出。
KV量子化：vLLM等のpaged/quantized KV機能を積極的に利用[5,6]。

設計含意：長文対応の現実解

「128k対応」を謳うモデルでも、ローカル環境（特にGPU VRAM制約下）ではKVキャッシュがボトルネックとなり物理的にロードできない場合があります。KV量子化（FP8/INT4）は品質劣化を抑えつつメモリを劇的に削減できるため、長文タスクにおける必須の最適化技術となります。

Data Source: Technical Survey Report [Calculated based on Llama3 architecture]

メモリ最適化技術

paged KV／quantized KV／KV reuse／prompt cache

KV構造と管理の最適化

Paged KV Cache (vLLM) [5]

メモリを固定サイズのページ単位で管理し、断片化を抑制する技術。OSの仮想メモリと同様の仕組みで、GPUメモリの利用効率を劇的に向上させ、スループットを高めます。

Quantized KV Cache [6]

KVキャッシュを標準のFP16からFP8やINT4へ量子化。精度劣化を最小限に抑えつつメモリフットプリントを50〜75%削減し、より長いコンテキストや大きなバッチサイズでの推論を可能にします。

再利用とレイテンシ短縮

KV Cache Reuse (TensorRT-LLM) [7]

共通のプレフィックス（システムプロンプトやマルチターン対話の履歴）に対するKVキャッシュを計算・保存。次回推論時に再利用することで、計算コストとメモリ転送時間を節約します。

Prompt Cache (llama.cpp) [88,90]

固定の長いシステムプロンプトやドキュメントをキャッシュ（--cache-prompt）。アプリ再起動後やセッション間での初回ロード時TTFT（Time To First Token）を大幅に短縮します。

Key Takeaways: KVキャッシュ最適化とプロンプト再利用を組み合わせることで、メモリ制約下での「長文コンテキスト」と「高速な応答性」を両立します。

Tier定義と現実ライン（概要）

ハードウェア別の実用ライン定義

結論：Tierは「重み（4bit）＋KV（4k〜8k）＋オーバーヘッド」を前提に定義。
理論計算値をベースとしつつ、最終判断は実機再現ベンチで行う設計です。

Apple Silicon (UMA)

- 16GB (A-16) 3B 〜 7/8B (4bit)
- 24-32GB (A-24/32) 7B 〜 14B (4bit)
- 64GB (A-64) 14B 〜 32B (4bit)
- 96-128GB (A-96+) 32B 〜 70B (4bit)

Windows GPU (VRAM)

- VRAM 8GB (G-8) 7B (4bit) ※短文中心
- VRAM 12GB (G-12) 7B 〜 14B (4bit)
- VRAM 16GB (G-16) 14B 〜 27B (4bit)
- VRAM 24GB (G-24) 27B 〜 34B (4bit)

Windows CPU-only

- RAM 32GB (W-CPU1) 3B 〜 7B (4bit)

⚠

7B級が「実用的に動く」境界線。
AVX命令等の最適化が必須。
速度はGPU比で大幅劣後。

Apple Silicon Tier（完全版表1）

実用ライン定義：A-16 / A-24/32 (Unified Memory Architecture)

Tier	想定ハード	現実的なLLM規模	常駐可否	同時実行	設計指針
A-16 Entry	**Apple Silicon 16GB** (M1/M2/M3/M4等) [1,11,92]	**3B〜7/8B** (4bit量子化)	条件付 “短文中心”なら可	軽量構成 ASR/TTSは小さめなら可	**文脈長を抑え、RAGはEmbedding先計算算。** Apple公式のUnified Memory Architecture (UMA) 特性を理解し、システムメモリとの競合を避ける設計が必要。[10,132]
A-24/32 Standard	**Apple Silicon 24〜32GB** (M3/M4 Pro/Max等) [93]	**7B〜14B** (4bit量子化)	可 （業務最小ライン）	成立 多くのユースケースで実用可	**14B級を軸に、VLMは2B〜小型を併用。** 。 「RAG含め業務用途の最小実用ライン」になりやすい。M4世代でも統合メモリ

UMA (Unified Memory Architecture) の特性： CPUとGPUが同一のメモリプールを共有するため、データ転送のオーバーヘッドが極小化されます。Metal Performance Shaders (MPS) バックエンドを使用するMLXやPyTorchの実装では、この統合メモリを効率的に利用できますが、画面表示やOS自身のメモリ消費（数GB）を差し引いた残量が実効VRAMとなる点に注意が必要です。

[1,132]

Data Source: Technical Survey Report [1,10,11,92,93,132]

Apple Silicon Tier（完全版表2）

A-64 / A-96/128：高品質ローカルの到達点と運用指針

Tier	想定ハード	現実的なLLM規模	常駐可否	同時実行	設計指針
A-64 High-End	**Apple 64GB** (M1 Max等) [1, 92]	**14B〜32B** (4bit)	可	余裕あり LLM+VLM等	**長文会議は要約で圧縮し、KV最適化を活用。** M1 Max等で64GB構成が公式に提示されており、32B級モデルの実用的なスイートスポット。 32Bモデル（4bit）で約16-18GB消費、残りでVLM/ASR/OSを十分に賄える。 長文コンテキスト（RAG/会議録）ではKVキャッシュが数GB単位で消費されるため、paged KV等の活用が効果的。
A-96/128 Ultra	**Apple 96〜128GB** (M2/M3/M4 Max/Ultra) [93, 94]	**32B〜70B** (4bit)	可 (重い)	構成次第 高品質RAG等	**“高品質ローカル”の現実ライン。** 70B級（4bitで約40GB）をロードしても、まだ50GB以上の余裕がある圧倒的なメモリ空間。 M3/M4で最大128GB構成が公式に提示され、ローカル[illegible] マルチモーダル（VLM 72B等）や複雑なAgentワークフロー[illegible]

到達点の意味： A-64以上は「妥協のないローカルAI」を実現する領域です。特にUnified Memory Architecture (UMA) の恩恵により、同等VRAMを持つディスクリートGPU構成よりも低コストかつ省電力に大規模モデルを扱えます。ただし、推論速度（スループット）は専用GPUに劣る場合があるため、レイテンシ要件が厳しい場合は量子化レベルの調整やプロンプトキャッシュの活用が重要になります。

Data Source: Technical Survey Report [1, 92, 93, 94]

Windows Tier（完全版表1）

W-CPU1 / W-CPU2 / G-8 / G-12 の現実ライン

Tier	想定ハード	現実的なLLM規模	常駐可否	同時実行	設計指針
W-CPU1	**CPU-only** 8C/16T級 + 32GB RAM	**3B〜7B** (4bit)	条件付き 負荷高	厳しい オンデマンド推奨	CPU最適化命令（AVX-512/VNNI/AMX等）を）を活用[13]。常駐は“小型＋短文”に限定し定し、重い処理は避ける。
W-CPU2	**CPU-only** 16C/32T級 + 64GB RAM	**7B〜14B** (4bit)	可 ただし速度要検証	設計次第	夜間バッチ（Embedding/OCR）で負荷平化。メモリ余裕はあるが計算速度が律速。
G-8	**NVIDIA GPU** VRAM 8GB	**7B中心** (4bit)	可 短文に限る	軽量なら可	画像生成SDXL等は設定依存で厳しいためベンチ。7B級LLM単体なら快適に動作。
G-12	**NVIDIA GPU** VRAM 12GB	**7B〜14B** (4bit)	可	現実化	14B級が視野に入る。VLM 2B〜7B級やGPU版を同時に載せやすく、実用性が高い。

最適化のポイント：WindowsのCPU推論はIntel拡張（IPEX）やOpenVINO等の最適化が効く場合があります。GI [13,135,136] では、VRAM不足時にシステムRAMへ溢れると劇的に遅くなるため、タスクマネージャー等でVRAM使用率を厳密に監視してください。

Data Source: Technical Survey Report [13, 135, 136]

Windows Tier（完全版表2）

高品質・実務用：G-16 (16GB) / G-24 (24GB) 定義

Tier	想定ハード	現実的なLLM規模	常駐可否 / 同時実行	設計指針
G-16	**VRAM 16GB** GeForce 4080 Laptop RTX 4070 Ti Super 等	**14B 〜 27B** (4bit量子化)	常駐: 可 同時: 高品質寄り成立 ASR/TTS/VLMとの マルチモデル運用が可能	**高品質なマルチモデル運用ライン** 長文コンテキストはKVメモリを圧迫するため、FP8 KVキャッシュ等の採用余地があります。 20B級モデルの運用も視野に入ります。
G-24	**VRAM 24GB** GeForce RTX 3090/4090 RTX 6000 Ada 等	**27B 〜 34B** (4bit量子化)	常駐: 可 同時: 余裕あり 重い画像生成や大型VLMも 実務的に動作可能	**実務における高品質ローカルの基準点** 30B級LLMに加え、SDXL/FLUXなどの重い画像生成や大型VLMを同時に回せるポテンシャルがあります。70B級は単体ならギリギリ動作可能です（要4bit/EXL2等）。

KVキャッシュに関する重要な注意点：
VRAMに余裕があっても、長文コンテキスト（32k〜128k）を扱うとKVキャッシュがVRAMを大量に消費し、OOM（Out Of Memory）の原因となります。
G-16/G-24であっても、長文を扱う際は「KV量子化（FP8/INT4）」や「コンテキスト設計（要約・分割）」によるメモリ管理が必須です。

Based on Weight Memory Theory + Reproduction Benchmarks

補足：AMD/Intel GPUの現実ライン

公式最適化とエコシステムの現状

結論：AMD/Intel環境でも公式最適化による選択肢が存在しますが、運用は実装・ドライバ依存となり、個別検証が必要です。

AMD Radeon GPU (ROCm)

ROCm | HIP SDK | ZLUDA (Deprecated)

ROCmのWindowsサポートが進展しており、一部のWSL2環境やネイティブWindowsでの動作が可能になりつつあります。 [97]

llama.cppのHIP BLASバックエンドや、MLC LLMなどがAMD GPUをサポート。VRAM容量あたりのコストパフォーマンスに優れるケースがあります。

NVIDIA CUDAエコシステムと比較すると、ライブラリの対応状況やトラブルシューティング情報が限定的です。

Intel Arc / iGPU (OpenVINO/SYCL)

OpenVINO | IPEX | SYCL / oneAPI

PyTorch拡張としてXPU（Intel GPU）サポートを提供。Arc GPUでの推論加速が可能。 [13]

llama.cppのSYCLバックエンドやOpenVINO最適化により、iGPU（Core Ultra等）を含めた幅広いハードウェアで動作。 [135][136]

ドライババージョンへの依存性が高く、セットアップ手順が頻繁に更新される傾向があります。

常駐運用 vs オンデマンド運用

TTFT（初速）、安定性、リソース効率のトレードオフ

基本運用モデルの比較

常駐（Always-on）

推奨: チャットBot

モデルをメモリに保持し、Prompt Cacheを維持。

利点：TTFTが最短、応答が安定。
欠点：メモリを常時占有（他アプリと競合）。

オンデマンド（On-demand）

推奨: 翻訳/要約

リクエスト時のみロードし、完了後に解放。

利点：アイドル時のメモリ節約、複数モデル併用。
欠点：毎回ロード時間が発生（TTFT増大）。

負荷分散と決定基準

夜間バッチ（Nightly Batch）

重い処理をオフピークに移動し、前計算する。

用途：RAGのEmbedding生成、OCR、議事録再要約。
効果：日中のリソースをLLMの即時応答に集中。

運用方針の決定基準

Tier（ハード制約）とユースケース要件で決定。

p95要件：即応が必要なら「常駐」。
同時実行数：リソース不足なら「オンデマンド」か「バッチ」。

Key Takeaways: 常駐は「即応性」、オンデマンドは「リソース効率」。重い処理（OCR/Embedding）は夜間バッチへ逃がすのが鉄則です。

ユースケース別推奨スタック：音声メモ→整文化→タスク抽出

用途に応じて「低コスト／高品質／ハード制約」の3構成を選択

結論：ASR精度とLLM推論能力のバランスで構成を決定します。 ※誤転記防止にはVAD（無音除去）が必須。タスク抽出にはJSONスキーマ固定が有効です。

低コスト構成

Target HW

A-16 / W-CPU1 / G-8想定 (Apple 16GB / CPU / VRAM 8GB)

VAD & ASR

Silero VAD (無音除去) faster-whisper (CPU/INT8)

LLM 整形・抽出

7B級 (Qwen2.5-7B等) 4bit (GGUF系)

Output Interface

JSON（タスク配列） Ollama/LM Studio (OpenAI互換)

高品質構成

Target HW

A-64+ / G-16+ / G-24想定 (Apple 64GB+ / VRAM 16GB+)

VAD & ASR

faster-whisper (GPU) 必要なら大型Whisper系列

LLM 整形・抽出

14B〜32B級 (Phi-4/Qwen上位) 4bit / GPU実行

Output Interface

Function Calling前提 Schema固定でJSON破壊率低減

ハード制約構成

Target HW

W-CPU1 / A-16 (ギリギリ) (低スペックPC / メモリ不足)

VAD & ASR

短音声限定 or 夜間バッチ処理 リアルタイム性は犠牲にする

LLM 整形・抽出

3B〜4B級 (Phi-3 mini等) 「抽出のみ」に機能限定

Output Interface

抽出テンプレート固定 自由生成を極力減らす

音声メモ→整文化→タスク抽出（失敗モードと回避策）

ユースケース別推奨スタック

結論：誤転記・要点漏れ・推論不足は「設計」で抑え込む。
VADによる区間分割、逐次要約、テンプレ固定が鍵となります。

ASR誤転記→抽出ミス

✕ 失敗モード

ASRが固有名詞や数値を誤認識し、そのままタスクとして抽出されてしまう。

✓ 回避策（Design）

1. **VADで区間分割**：無音除去で認識精度向上。
2. **重要箇所再確認UI**：抽出されたタスクの元音声をワンクリック再生できるUIを提供。

Silero VAD　Playback UI

長文での要点漏れ

✕ 失敗モード

コンテキスト長超過やAttentionの分散により、後半の重要事項が無視される。

✓ 回避策（Design）

1. **逐次要約（Rolling Summary）**：一定区間ごとに要約し、次区間の入力に含める。
2. **KV節約**：量子化KVやウインドウ制限でメモリ枯渇を防ぐ。

Rolling Context　Quantized KV

LLMの推論不足

✕ 失敗モード

小型モデル（3B-7B）が指示に従わず、タスク以外の雑談や幻覚を出力する。

✓ 回避策（Design）

1. **テンプレ固定**：自由生成を禁止し、抽出テンプレート（JSON Schema等）を強制。
2. **Few-shot提示**：プロンプトに抽出成功例を含める。

JSON Schema　Few-shot

システム連携のポイント

Ollama/LM StudioのOpenAI互換APIを活用して業務アプリと接続する場合、受け取ったJSON出力の型チェック（バリデーション）をクライアント側で厳格に行うことで、モデルの不安定さを吸収できます。

ユースケース：会議議事録（話者分離＋要約＋アクション）

推奨スタック3構成（Tier別）

結論：規模と要件に応じて3構成を選択。無音除去（VAD）と発話ターン統合ルールが品質下限を決定します。
※評価はp50/p95、RTF、意味改変率で監視することを推奨します。

低コスト構成

前処理 (VAD)

Silero VAD [82] 無音区間を確実に除去

ASR (音声認識)

faster-whisper [50] (CPU / INT8量子化)

話者分離 (Diarization)

pyannote.audio [84] (必要時のみ適用、負荷高)

LLM (要約・抽出)

7B級 (Qwen2.5等) 要約・論点・ToDo抽出

✓ 個人利用 / 小規模MTG

高品質構成

ASR (GPU加速)

faster-whisper (GPU) 大型Whisperも視野 [48,50]

タイムスタンプ

ASRタイムスタンプ活用 各要点に元発話時刻を付与

LLM (高度な分析)

32B〜70B級 KV最適化 (FP8 KV等) [5–7]

検証・品質

発話と要約の突合検証 文脈保持とアクション紐付け

✓ 公式議事録 / 業務記録

ハード制約構成

機能縮退

話者分離を省略 「話者なし要約→タスク」へ縮退

処理タイミング

夜間バッチ処理 長時間の音声ファイルは非同期

軽量モデル

Whisper tiny/base + 3B級 LLM (Phi-3 mini等)

目的限定

「決定事項のみ」抽出に特化 全文書き起こしは放棄

✓ リソース極小 / バッチ処理

会議議事録：失敗モードと回避策

ユースケース詳細分析

結論：話者誤割当・ASR幻覚・長時間処理の負荷を設計で抑止。
VAD（無音除去）の厳格化とタイムスタンプベースの検証が品質の防波堤となります。

主な失敗モード（Failure Modes）

話者誤割当（Diarization Error） 高頻度

発話の切れ目が不明確で、Aさんの発言がBさんとして記録される。特に割り込み発話で多発。

ASR幻覚（Hallucination） 致命的

無音区間やノイズに対して、存在しない文章（「ご視聴ありがとうございました」等）を生成してしまう現象。

長文による要点漏れ

コンテキスト長超過により、議論の後半や中間部分の重要決定事項がLLMの要約から欠落する。

回避策と設計（Mitigation）

短い切れ目のマージ＋手動マッピング 設計

極端に短い発話区間を前後の発話者に統合するルールを適用。参加者名とIDの紐付けUIを用意。

VAD厳格化＋無音区間破棄 前処理

Silero VAD等の閾値を調整し、確実に音声がある区間のみASRへ渡す。「要審査フラグ」で怪しい出力をマーク。

逐次要約（Rolling Summary）＋KV節約 LLM最適化

議論を短めのチャンクに分割して逐次要約し、最終的に統合。KV量子化やコンテキスト分割でメモリ節約。

評価指標（Monitoring Metrics）

運用時は**p50/p95レイテンシ**に加え、**RTF（Real-Time Factor）**と**意味改変率**を監視し、品質劣化の兆候を早期検知します（ベンチ設計参照）。

ユースケース：文書RAG

PDF/スキャン → OCR → 検索 → 回答

結論：RAG品質はEmbeddingで上限が決まるため、「Embeddingでrecall確保→Rerankerでprecision向上」の二段構えが基本戦略です。

低コスト構成

OCR / Pre-process

Tesseract OCR ※スキャン品質が高い場合

Embedding / Reranker

BGE-M3 (多用途) bge-reranker-base (軽量)

LLM Runtime

7B級 4bit (CPU) llama.cpp / Ollama

✓ テキスト中心の標準PDF

高品質構成

OCR / DocAI

PaddleOCR (構造化) Donut (OCR-free情報抽出)
LayoutLMv3 (レイアウト保持)

Embedding / Reranker

BGE-M3 + 大型reranker (GPU推論推奨)

LLM Runtime

14B〜32B級 (GPU) 図表・レイアウト情報を加味

✓ 図表・帳票を含む文書

ハード制約構成

Processing Strategy

夜間バッチ処理 OCR・Embeddingを夜間に生成

Daytime Operation

検索と短い回答のみ実行 LLM推論負荷を最小化

LLM Runtime

3B〜7B級 (4bit/INT8) コンテキスト長を制限

✓ リソース極小環境

文書RAG：失敗モードと回避策

OCR誤読・検索失敗・幻覚への対策設計

結論：文書RAGの失敗は「OCR誤読」「検索精度不足」「幻覚引用」に大別され、前処理の標準化と原文スニペット引用のUX要件化で回避します。

主な失敗モード（Failure Modes）

- スキャン品質低下や傾きにより、固有名詞や数値が誤認識され、正しい文書がヒットしない。 Recall低下
- 表組みや段組みが崩れてテキスト化され、文脈が断絶。LLMが意味を誤解釈する。 Context喪失
- 検索結果に含まれない情報を、さも引用したかのように回答する。 信頼性毀損

回避策と設計（Mitigation）

- 傾き補正・二値化をパイプライン化。構造化にはPaddleOCR、レイアウト保持にはDonut/LayoutLMv3を採用。 PaddleOCR/Donut
- 回答の根拠となったOCRテキストの断片（スニペット）を必ず提示し、ユーザーが原文を確認できるUIにする。 Verify UI
- OCRとEmbeddingは夜間バッチで済ませ、検索時はRerankerで精度向上に集中。LLMには「引用外は回答不可」を指示。 Batch/Rerank

実務上のポイント（Key Takeaway）

PDF/画像RAGでは、Embeddingモデルの性能以前に「テキスト抽出の品質」が上限を決めます。Tesseract等の古典OCRで限界を感じたら、早めに視覚モデル（VLMやDocument AI）への切り替えを検討すべきです。

ユースケース：画像理解（VLM）— 3構成

スクショ/写真の説明・抽出：推奨スタック比較

結論：2B〜7B級VLMがローカルの現実ライン。72B級は上位機前提となります。
※視覚トークン肥大による速度/メモリ急落を防ぐため、解像度・max-pixelsの制御が必須です。

低コスト構成

Target HW

Apple 16GB / VRAM 8GB級

VLM Model

Qwen2-VL 2B 軽量・高速な視覚理解

Runtime

mac: MLX-VLM (公式例あり) Win: GPU推論 (transformers等)

Key Point

解像度を制限しメモリ圧迫を回避

✓ スクショ説明・簡易OCR

高品質構成

Target HW

Apple 32GB+ / VRAM 16-24GB+

VLM Model

Qwen2-VL 7B / InternVL2 (必要に応じて上位モデル)

Strategy

画像→テキスト抽出→Embedding →RAG融合 (ハイブリッド検索)

Key Point

OCR-freeの図表理解能力を活用

✓ 図表理解・情報抽出

ハード制約構成

Target HW

VRAM不足 / メモリ制約大

Substitute Strategy

OCR (PaddleOCR等) + テキストLLM VLMモデルを使用しない

Operation

OCRでテキスト化し、LLMで整形 画像入力自体を避ける

Compromise

空間認識・文脈理解を諦め 文字情報の抽出・整理に限定

✓ 文字主体の処理

画像理解：失敗モードと回避策

結論：UI文字の誤読と視覚トークン肥大による速度低下を、OCR併用と解像度・max-pixels制御で回避します。

主な失敗モード（Failure Modes）

- 2B/7B級VLMでは、スクリーンショット内の小さなフォントや密集した情報を正確に読み取れないケースが頻発。
- 高解像度画像をそのまま入力すると、視覚トークン数が数千に達し、推論速度（TTFT/生成）が急激に悪化。
- 複雑な表組みやグラフの空間関係を誤認し、存在しない数値や関係性を捏造する。

回避策・設計（Mitigation）

- 文字情報はPaddleOCR等で別経路からテキストとして供給し、VLMは「状況説明」に専念させる。
- MLX-VLM等のmax-pixelsパラメータで上限を設定し、トークン数を抑制。実務的には1024px程度にリサイズ。
- 構造化が必要な帳票類は、レイアウト情報を保持できる専用モデル（LayoutLMv3）やDonutを活用。

PaddleOCR　max-pixels制御　LayoutLMv3

ユースケース：コーディング補助

推奨スタック3パターン（IDE支援・リポジトリ理解）

結論：補完は小型モデル、設計レビューや長距離依存解決は中〜大型モデルで役割分担します。
※生成コードは自動コンパイル/テスト実行を“ツール”化し、検証ループに組み込むことが重要です。

低コスト構成

Code LLM

Qwen2.5-Coder 7B サイズ展開が豊富で軽量

RAG / Embedding

リポジトリEmbedding (BGE/E5) 関連ファイル抽出→LLM回答

Connection

Ollama / LM Studio OpenAI互換APIでIDE拡張接続

✓ 一般的なIDE補完・Q&A

高品質構成

Code LLM

StarCoder2 (7B/15B) + 32B級汎用LLM併用

Optimization

prefix caching (vLLM等) 長文コンテキストでの遅延抑制

Capability

設計レビュー / 長距離依存 複雑なリファクタリング提案

✓ 大規模開発・設計支援

ハード制約構成

Target HW

メモリ8GB / エントリー機 古い開発環境

Code LLM

3B級モデル Phi-3 mini / StarCoder2-3B

Strategy

短い支援に限定 「行補完」＋「関数リファクタ提案」

✓ スニペット補完のみ

コーディング支援：失敗モードと回避策

結論：幻覚API生成と長文遅延を「検証ループ」と「キャッシュ活用」で抑止し、コード品質とセキュリティを自動化プロセスで担保します。

主な失敗モード（Failure Modes）

- ライブラリのバージョン不一致や、もっともらしいが実在しない関数（Hallucination）を生成。
- リポジトリ全体を読み込むとKVキャッシュが肥大化し、補完のレスポンスが低下する。
- 古い構文や廃止されたメソッドを使用し、コンパイルエラーやセキュリティリスクを招く。

回避策・設計（Mitigation & Design）

- ビルド・単体テスト実行をfunction callingとしてLLMに提供。失敗時はエラーログをフィードバックして再生成させる。
- vLLMやllama.cppのキャッシュ機能を活用し、共通コンテキスト（ライブラリ定義等）の再計算を回避。[88][90][112]
- Linterや依存関係解決ツールを“ツール”として組み込み、生成コードを静的解析してから提示する。

パラメータ設定の含意

ユースケース：画像生成（Diffusion）— 3構成

推奨スタックとVRAM要件の最適化

結論：SDXLを軸にComfyUIでワークフローを資産化。FLUX等はライセンス精査が必須です。
※VRAM要件は解像度・バッチ・ステップ数に依存するため、プロファイル固定が重要です。

低コスト構成

Target VRAM

VRAM 8GB〜12GB
(Apple Silicon 16GB)

Model / UI

SDXL base 1.0 + Refinerなし
ComfyUI (Win/Linux/macOS)

Optimization

Attention Slicing有効化
FP16 / Tiled VAE活用

Resolution

1024x1024固定
Batch size = 1

高品質構成

Target VRAM

VRAM 24GB以上
(Apple Silicon 64GB+)

Model / UI

FLUX.1 [dev/pro] 等の上位系
ComfyUI (ワークフロー資産化)

Optimization

Seed固定で再現性担保
Refiner / LoRA多重適用

License Check

⚠ FLUX.1-dev等の商用条件
ライセンス精査が必須

ハード制約構成

Target VRAM

VRAM 8GB未満
(メインメモリ共有等)

Model / UI

SD 1.5系 / LCM-LoRA (高速化)
Stable Diffusion WebUI (Forge)

Offloading strategy

夜間バッチ処理
またはLAN内別筐体へオフロード

Limitation

解像度512x512等に制限
日中はLLMにリソース集中

画像生成：失敗モードと回避策

結論：VRAM不足と生成結果のブレを「標準プロファイル」で抑制し、ライセンス違反リスクを「事前精査と監査」で排除します。

主な失敗モード（Failure Modes）

- ×
  高解像度や大バッチ指定時にプロセスがクラッシュ。特にSDXL/FLUX等の大型モデルで頻発。
- ×
  同じプロンプトでもSeedや設定の違いで出力が変動し、UIモック等の修正サイクルが回らない。
- ×
  FLUX.1-dev等の非商用/開発用ライセンスを、誤って商用プロダクトや社内資料に利用してしまう。

回避策と設計（Mitigation Strategies）

- ✓
  VRAM容量に応じた上限テーブルを作成。Attention SlicingやVAE Tiling等の最適化を強制適用。
- ✓
  Seedを固定し、ComfyUIワークフローをJSONとして資産化・バージョン管理する。 [72]
- ✓
  使用モデルのライセンス条項（商用可否）をリスト化し、生成ログにモデルハッシュを紐付ける。 [76]

設計含意（Design Implication）

10

測定と比較の方法

ベンチマーク設計と
再現性の担保

KEY TAKEAWAYS

本章の要点

- **指標の厳密定義：**
  p50/p95レイテンシ、TTFT、tok/s、RTF等を定義し、JSON破壊率や意味改変率も定量化します。
- **測定手順の標準化：**
  条件固定（温度・量子化等）、ウォームアップ分離、KV影響の切り分け（短文/長文）を徹底します。
- **再現性のコア要素：**
  ASR/TTSの前処理固定と、最小テストセット（短文/長文/構造化/音声）による比較を実施します。

指標定義　測定手順　ログ保存　テストセット

測定指標の定義

p50/p95・TTFT・tok/s・RTF・品質指標

結論：LLMの「初速（TTFT）」と「生成速度（tok/s）」を分離して計測し、品質（JSON破壊率等）とリソース消費（RAM/VRAM）を定量化します。

速度・リソース指標（Latency & Resource）

TTFT（Time To First Token）と生成完了時間を分離して計測。KV reuseやprompt cacheの効果はTTFTに現れるため分離が必須。

生成トークン数÷生成時間。ユーザーの体感速度やバッチ処理能力の指標。

ASR/TTS用指標。処理時間÷音声長。

RTF < 1.0 なら実時間より高速

OS監視またはプロセス単位（nvidia-smi等）で計測。Apple Silicon等の統合メモリ環境ではシステム全体の圧迫度も注視。

品質・堅牢性指標（Quality & Robustness）

指定スキーマに対する「パース失敗」や「スキーマ不一致」の発生率。ツール呼び出し（Tool Use）の安定性を左右する重要指標。

要約やフォーマット変換において、元の意味が変わってしまった割合。Embedding類似度による自動評価＋人手監査で測定。

同一条件下（温度、シード等）での結果の安定性。推論ごとのブレ幅を確認。

測定上の注意（Apple Silicon）：

測定手順（条件固定・ウォームアップ・KV切り分け）

再現可能なベンチマーク設計の要件

再現条件の固定

生成パラメータの統一

同一プロンプト、最大トークン数、温度（temperature）、top_p を固定し、ランダム性を排除または制御します。

モデル環境の固定

同一の量子化形式（GGUF Q4_K_Mなど）、コンテキスト長設定を使用します。

音声・画像系の前提

ASR/TTSは同一音声・同一前処理（VAD有無）、同一話者設定で品質と速度を分離測定します。

Key Takeaway: 条件のわずかな違い（VAD有無など）が結果を大きく左右するため、メタデータとして全条件を記録します。

測定手順

ウォームアップ分離

初回ロード（モデル読込＋コンパイル）と2回目以降の推論を分離。常駐運用とオンデマンド運用の性能差を明確化します。

KV影響の切り分け

短文（<512 tokens）と長文（4k/8k/32k）を分けて測定し、paged/quantized KVの効果を評価します。

ログ保存要件

モデルハッシュ、量子化方式、ランタイムVer、ハード情報（CPU命令、VRAM）を記録し、再現性を担保します。

条件固定　ウォームアップ　KV分離

リスクとコンプライアンス

結論：「コードのOSSライセンス」と「モデル重みの利用条件」を分離管理し、ローカル完結の利点を活かしつつ、検証ループによる安全性確保が必須です。

ライセンス確認
コードと重みを
分離して精査

商用利用判定
規約・例外条件
(Family内差分)確認

ローカル完結
データ送信遮断
プライバシー保護

検証ループ
幻覚・攻撃の
自動検知

継続監視
ログ監査と
是正サイクル

ライセンス管理

コードと重みの分離
ランタイム(MIT/Apache)とモデル重み(Community/非商用)は別条件。利用範囲を台帳化する。

個別確認の義務化
Qwen2.5のファミリー内例外や、FLUX.1-dev等の派生モデル条件を一次ソースで確認する。[33]

プライバシー・安全

ローカル完結の徹底
入力データがデフォルトで外部送信されない構成を物理/論理的に担保。LAN内APIも管理対象。

入力ガードレール
プロンプトインジェクション対策やPII（個人情報）フィルタリングを前段に実装する。

幻覚対策・監視

検証ループの実装
生成コードの自動コンパイル/テスト実行や、RAG引用元の存在確認を自動化し、幻覚をフィルタする。

ログ監査
入出力、パラメータ、実行環境情報をログ保存し、事後監査や精度改善サイクルに活用する。

まとめ（意思決定フロー）

1 要件定義 Requirements

業務課題から技術要件へ変換する始点。

判断基準 (Criteria)

品質: 日本語、JSON、専門性

速度: p95レイテンシ、tok/s

機能: Tool-use、RAG有無

アクション

許容レイテンシ目標（例: 10 tok/s）とJSONスキーマの厳格さを決定。

2 Tier選定 Hardware Tier

メモリ制約に基づく現実ラインの把握。

判断基準 (Criteria)

Apple: 16〜128GB (UMA)

Windows: VRAM 8〜24GB

CPU: AVX/RAM容量

アクション

「重み4bit + KVキャッシュ」の理論値でTier (A-16〜G-24)を特定。

3 モデル選定 Model Select

Tier内で動く最適モデルの選定。

判断基準 (Criteria)

規模: 7B-14B (汎用), 32B+ (高品質)

形式: GGUF, EXL2, AWQ

権利: ライセンス条件

アクション

Llama/Qwen/Phi等からサイズ適合候補を選び、ライセンスを確認。

4 ランタイム Runtime

ハードウェア性能を引き出す実行環境。

判断基準 (Criteria)

Mac: MLX / llama.cpp

Win: Ollama / LM Studio / vLLM

API: OpenAI互換性

アクション

OSとモデル形式に合ったランタイムを選択。Server機能の有無を確認。

5 検証・最適化 Verify

実機ベンチによる実用性の確定。

判断基準 (Criteria)

指標: TTFT, p95, JSON破壊率

負荷: メモリピーク, KV推移

品質: 幻覚, 誤転記

アクション

条件固定ベンチで測定。KV量子化やプロンプトキャッシュで調整。

意思決定の要点 (Key Takeaways)

「ランタイム先行」ではなく**「要件→メモリ(Tier)→モデル」の順**で決定することで、手戻りを防ぎます。
特に長文コンテキストではKVキャッシュが支配的になるため、**KV最適化（paged/quantized KV）**の活用が成否を分けます。

総括（実務への適用）

結論：ローカルAIは「4bit量子化＋KV最適化」を前提に、失敗モードを設計で抑え込むことで実務運用が可能です。

実装の原則（Principles）

- 重みは4bit（GGUF/AWQ/GPTQ）でメモリ理論値を計算し、Tierに合わせる。 Cost-Efficiency
- 長文・多同時接続時は、paged KV、KV量子化、prompt cacheを活用してメモリ爆発を防ぐ。 Scalability
- 幻覚・誤転記・JSON破壊は「起きるもの」とし、UI確認・検証ループ・再生成でカバーする。 Robustness
- p50/p95レイテンシと品質（tok/s, RTF, JSON破壊率）を定量ベンチマークで測定し確定させる。

次のステップ（Action Items）

1. **Tierの確定：** 手持ちハードウェア（Apple/Win）とTier表を照合し、現実的なモデル規模を把握。
2. **パイロット構築：** 推奨スタック（低コスト/高品質）に基づき、llama.cpp/Ollama/faster-whisper等でプロトタイプ作成。
3. **ベンチマーク実施：** 条件固定（温度/トークン数）でログを取り、p95レイテンシと実用性を計測。
4. **リスク監査：**

References（1/4）

Technical Research Report Sources [1-34]

[1] https://www.apple.com/newsroom/2020/11/apple-unleashes-m1/

[2] https://docs.openhands.dev/openhands/usage/llms/local-llms

[3] https://lmstudio.ai/docs/developer/core/server

[4] https://arxiv.org/abs/2306.00978

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

[6] https://docs.vllm.ai/en/latest/features/quantization/quantized_kvcache/

[7] https://github.com/NVIDIA/TensorRT-LLM

[8] https://github.com/ggml-org/llama.cpp

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

[12] https://huggingface.co/wangkanai/sdxl-fp16

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

[14] https://github.com/xiph/rnnoise

[16] https://huggingface.co/meta-llama/Llama-3.1-70B

[17] https://huggingface.co/microsoft/Phi-3-mini-128k-instruct

[19] https://lmstudio.ai/docs/app

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

[22] https://github.com/ml-explore/mlx

[23] https://github.com/ollama/ollama

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

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

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

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

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

[32] https://huggingface.co/Qwen/Qwen2.5-7B-Instruct

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

References（参考文献リスト 2/4）

[35]〜[68] Llama / Qwen / Gemma / Phi / VLM / Whisper / TTS / OCR

NO.	SOURCE URL / DESCRIPTION
[35]	Gemma 2 Model Card https://ai.google.dev/gemma/docs/core/model_card_2
[37]	Mixtral 8x7B Instruct https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1
[40]	Phi-4 Technical Report (Arxiv) https://arxiv.org/abs/2412.08905
[41]	OpenAI Cookbook (Run locally LM Studio) https://developers.openai.com/cookbook/articles/gpt-oss/run-locally-lmstudio/
[42]	Qwen2-VL 72B https://huggingface.co/Qwen/Qwen2-VL-72B
[43]	Phi-3-vision 128k Instruct https://huggingface.co/microsoft/Phi-3-vision-128k-instruct
[45]	InternVL2 4B https://huggingface.co/OpenGVLab/InternVL2-4B
[47]	LLaVA Repository https://github.com/haotian-liu/LLaVA
[48]	OpenAI Whisper https://openai.com/index/whisper/

NO.	SOURCE URL / DESCRIPTION
[52]	Piper TTS https://github.com/rhasspy/piper
[53]	Kokoro TTS https://github.com/hexgrad/kokoro
[54]	Coqui XTTS-v2 https://huggingface.co/coqui/XTTS-v2
[55]	StyleTTS2 https://github.com/yl4579/StyleTTS2
[58]	BGE-M3 Embedding https://huggingface.co/BAAI/bge-m3
[59]	Multilingual-E5 Large https://huggingface.co/intfloat/multilingual-e5-large
[60]	Multilingual-E5 Large Instruct https://huggingface.co/intfloat/multilingual-e5-large-instruct
[61]	BGE Reranker Large https://huggingface.co/BAAI/bge-reranker-large
[62]	BGE Reranker v2 M3 https://huggingface.co/BAAI/bge-reranker-v2-m3
[64]	Tesseract OCR https://github.com/tesseract-ocr/tesseract

References（3/4）

参考文献リスト [69] - [102]

No.	Reference Details (Layout / Image Gen / Voice / HW)
[69]	**LayoutLMv3: Pre-training for Document AI with Unified Text and Image Masking** https://arxiv.org/pdf/2204.08387
[71]	**Stable Diffusion XL Base 1.0** https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0
[72]	**ComfyUI (A powerful and modular stable diffusion GUI)** https://github.com/Comfy-Org/ComfyUI
[73]	**Stable Diffusion WebUI (AUTOMATIC1111)** https://github.com/AUTOMATIC1111/stable-diffusion-webui
[75]	**FLUX (Black Forest Labs)** https://github.com/black-forest-labs/flux
[76]	**FLUX.1-dev Model License** https://github.com/black-forest-labs/flux/blob/main/model_licenses/LICENSE-FLUX1-dev
[82]	**Silero VAD (Pre-trained enterprise-grade Voice Activity Detector)** https://github.com/snakers4/silero-vad
[84]	**pyannote.audio (Neural building blocks for speaker diarization)** https://github.com/pyannote/pyannote-audio

No.	Reference Details (Voice / Optimization / Hardware)
[85]	**openWakeWord (Open-source wake word detection)** https://github.com/dscripka/openWakeWord
[87]	**ExLlamaV2 (Fast inference library for modern LLMs)** https://github.com/turboderp-org/exllamav2
[88]	**llama.cpp Discussion: KV Cache Management** https://github.com/ggerganov/llama.cpp/discussions/7949
[90]	**llama.cpp Discussion: Prompt Caching** https://github.com/ggml-org/llama.cpp/discussions/8860
[92]	**Apple MacBook Pro Specs** https://www.apple.com/macbook-pro/specs/
[93]	**Apple Support: Mac Memory Configurations** https://support.apple.com/en-us/111901
[94]	**Apple Support: Mac Studio Memory Specs** https://support.apple.com/ja-jp/117736
[97]	**AMD ROCm Documentation for Windows (Radeon/Ryzen)** https://rocm.docs.amd.com/projects/radeon-ryzen/en/latest/index.html

References（参考文献リスト4/4）

音声周辺・ベンチマーク・最適化技術・ハードウェア関連 [103] - [136]

[103] Phi-3-mini-128k-instruct Model Card
https://huggingface.co/microsoft/Phi-3-mini-128k-instruct

[104] vLLM: Quantized KV Cache
https://docs.vllm.ai/en/latest/features/quantization/quantized_kvcache/

[105] OpenAI Whisper
https://openai.com/index/whisper/

[106] LayoutLMv3: Pre-training for Document AI
https://arxiv.org/pdf/2204.08387

[107] Qwen2-VL Collection
https://huggingface.co/collections/Qwen/qwen2-vl

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

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

[110] Ollama Repository
https://github.com/ollama/ollama

[111] StarCoder2 Documentation
https://huggingface.co/docs/transformers/en/model_doc/starcoder2

[112] vLLM: Automatic Prefix Caching
https://docs.vllm.ai/en/v0.9.2/design/automatic_prefix_caching.html

[113] SDXL-FP16 Model Card
https://huggingface.co/wangkanai/sdxl-fp16

[114] TensorRT-LLM: KV Cache Reuse
https://nvidia.github.io/TensorRT-LLM/advanced/kv-cache-reuse.html

[115] llama.cpp Repository
https://github.com/ggml-org/llama.cpp

[120] MLX-LM Repository
https://github.com/ml-explore/mlx-lm

[121] vLLM Supported Platforms
https://docs.vllm.ai/en/latest/api/vllm/platforms/

[122] NVIDIA TensorRT-LLM
https://github.com/NVIDIA/TensorRT-LLM

[123] GPTQ: Accurate Post-Training Quantization
https://arxiv.org/pdf/2210.17323

[124] AWQ: Activation-aware Weight Quantization
https://arxiv.org/abs/2306.00978

[125] AutoAWQ Repository
https://github.com/casper-hansen/AutoAWQ

[126] AutoGPTQ Repository
https://github.com/AutoGPTQ/AutoGPTQ

[127] SmoothQuant: Accurate and Efficient 8-bit LLMs
https://arxiv.org/abs/2211.10438

[128] Qwen2.5-7B-Instruct
https://huggingface.co/Qwen/Qwen2.5-7B-Instruct

[129] FLUX Repository
https://github.com/black-forest-labs/flux

[130] Choosing a Resource Storage Mode for Apple GPUs
https://developer.apple.com/documentation/metal/...

[131] Apple M1 Announcement
https://www.apple.com/newsroom/2020/11/apple-unleashes-m1/

[132] MLX Unified Memory Guide
https://ml-explore.github.io/mlx/build/html/usage/unified_memory.html

Technical Survey Completed.