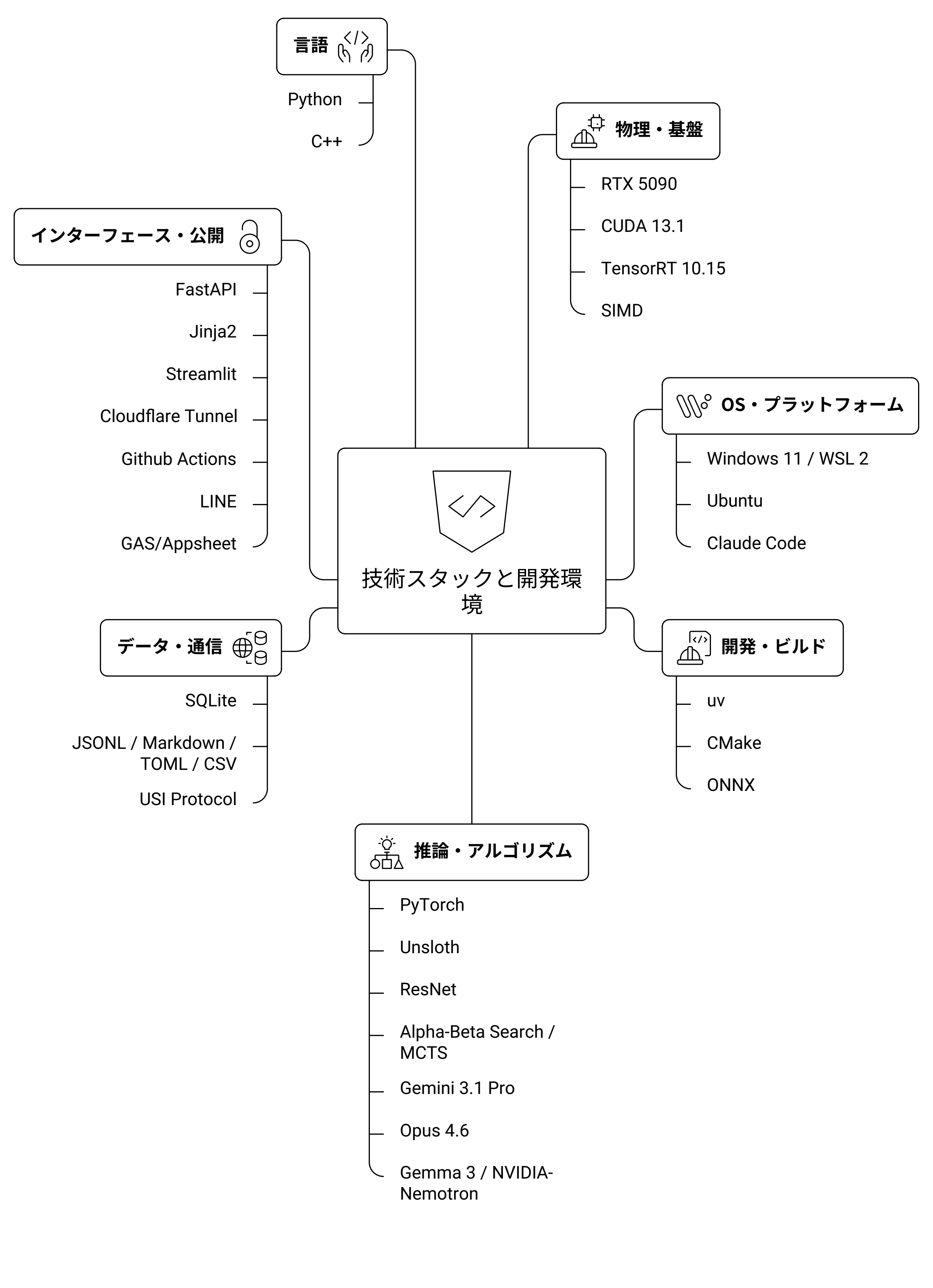
技術スタック・開発環境

技術スタック

- 物理・基盤
  - RTX 5090
  - CUDA 13.1
  - TensorRT 10.15
  - SIMD (AVX-512 / VNNI / AVX2)
- OS・プラットフォーム
  - Windows 11 / WSL 2
  - Ubuntu
  - Claude Code
- 開発・ビルド
  - uv
  - CMake
  - ONNX
- 推論・アルゴリズム
  - PyTorch
  - Unsloth
  - ResNet
  - Alpha-Beta Search / MCTS
  - Gemini 3.1 Pro
  - Opus 4.6
  - Gemma 3 / NVIDIA-Nemotron
- データ・通信
  - SQLite
  - JSONL / Markdown / TOML / CSV
  - USI Protocol
- インターフェース・公開（・デプロイ）
  - FastAPI
  - Jinja2
  - Streamlit
  - Cloudflare Tunnel
  - Github Actions
  - LINE
  - GAS/Appsheet
- 言語
  - Python / C++ 等

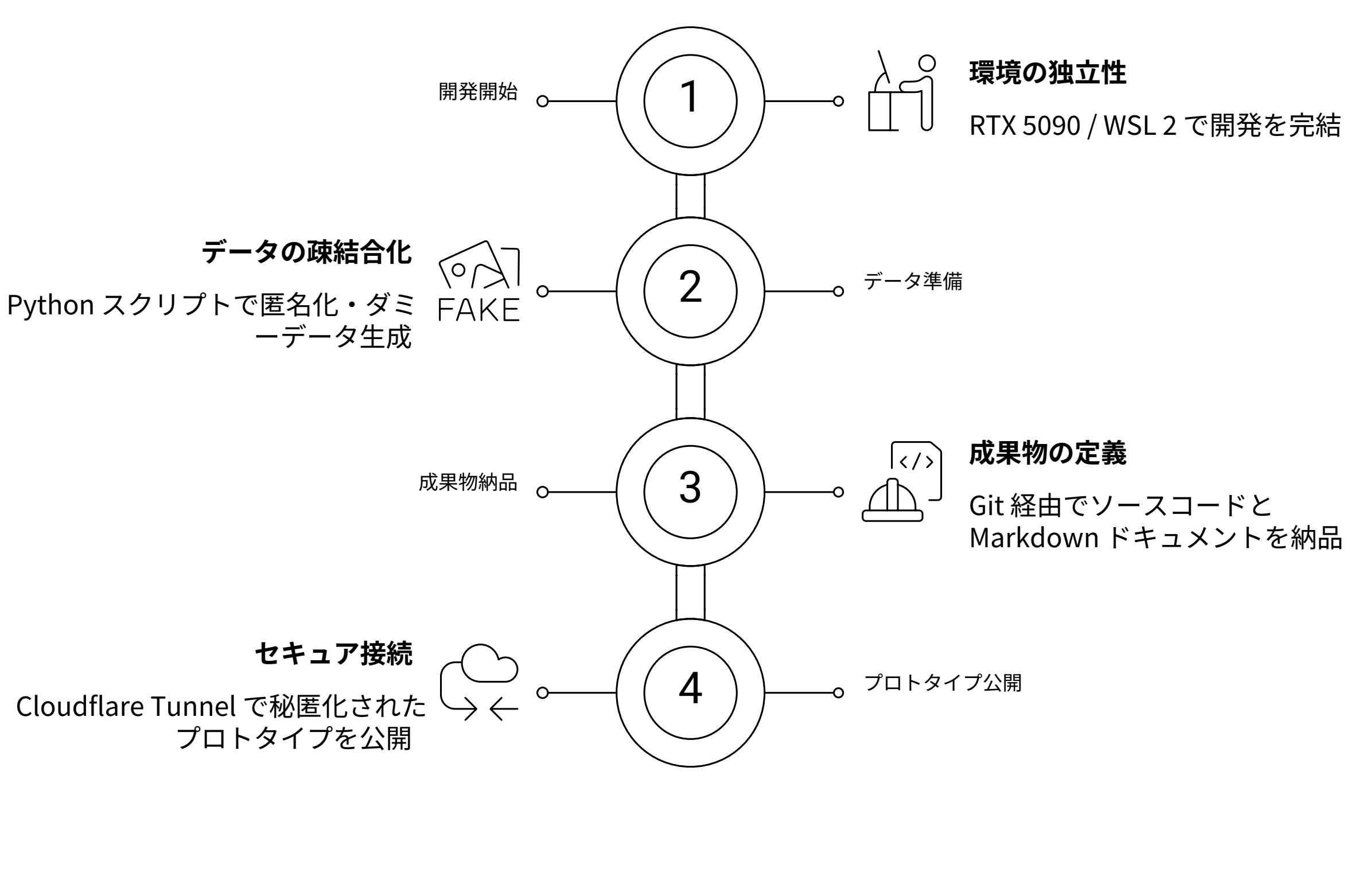
技術スタックと開発環境



開発・運用ルール（疎結合プロトコル）

1. 環境の独立性
   - 開発はすべて私有環境（RTX 5090 / WSL 2）にて完結。
   - 端末管理ソフト（Sentinel等）およびネットワーク制限の干渉を排除。
2. データの疎結合化
   - 顧客保有の生データには直接アクセスしない。
   - Pythonによる匿名化・ダミーデータ生成スクリプトを先んじて提供。
   - 企業側にて生成されたダミーデータのみを開発環境へ受領。
3. 成果物の定義
   - 納品物はGit経由のソースコード（Python, C++）およびMarkdown形式のドキュメントのみ。
   - デプロイ用コードはバイナリを含まず、環境再現性を担保した pyproject.toml (uv) を含む。
4. セキュア接続
   - 外部接続が必要なプロトタイプ公開時は、Cloudflare Tunnelによる秘匿化経路を使用。

疎結合プロトコル開発・運用ルール



> 技術スタック紹介文　2026年2月

■ はじめに

個人開発環境で実際に稼働しているシステムの技術スタックを紹介します。
AI学習・LLM+RAGの研究開発と、将棋AIの開発を通じて構築した環境です。

==

ハードウェア基盤と低レイヤー最適化

==

物理層は RTX 5090（Blackwell / VRAM 32GB GDDR7）と
Intel Core Ultra 9 285K（24C/24T, 5.7GHz）の組み合わせです。

推論加速は CUDA 13.1 + TensorRT 10.15。CUDAコンパイルは
-arch=sm_120（Blackwell）指定。CPU側では AVX-512 / VNNI を活用。

将棋AIの推論エンジンの量子化、TensorRT最適化や、
FP8ビルド時のカーネル選択問題（-DFP16混入によるNPSロス）など、
低レイヤーのチューニング知見を蓄積しています。

==

AI学習・推論パイプラインとLLM活用

==

メインの関心領域はAI学習とLLM+RAGです。

環境管理は uv を採用。pyproject.toml ベースで再現性を確保し、
venv管理の手間を排除しています。ビルドシステムは CMake、
モデルのポータビリティは ONNX で確保。

推論は PyTorch + Unsloth をメインに、用途に応じて
llama.cpp（GGUF量子化モデル）も使用。

LLMは用途別に使い分けています：
・クラウド: Gemini 3.1 Pro（Google）、Claude Opus 4.6（Anthropic）
Claude Codeを開発の中核ツールとして常用
・ローカル: Gemma 3、NVIDIA Nemotron、Qwen3系

稼働中のシステム（一例）：
・判例検索AI: SQLite FTS + Gemini RAGで判例分析
・特許検索AI: 400万件の米国特許DBをRAG検索
・技術ブログ: FastAPI + XML、Gemini Search Groundingで記事更新

==

データ処理とインターフェース設計

==

データ層は JSONL・CSV・SQLite をメインとしています。

UIは Streamlit　FastAPI + Jinja　Github Actions　LINE　GAS/Appsheet　etc.
外部公開は Cloudflare Tunnel でポート開放なしに構成
デプロイはローカルサーバの他、Community Cloud、Github、GAS等
いくつかのサービスを常時稼働中（hanrei / patentllm / blog 等）。

RAG設計の基本パターン：
1 専門ツールの出力を構造化データとして取得
2 PromptBuilder層でLLM用プロンプトに変換
3 LLMには"翻訳"だけさせ、Hallucination検証層でフィルタリング

==

FAQ

==

Q1. 生データに触れずにどうやって精度を出すのか?

統計的性質を維持したダミーデータを顧客側で生成してもらう
パイプラインを先に構築します。ロジックの正当性と
エッジケースの処理を担保すれば、本番環境との乖離は小さい。
ロジックが正しければデータの"量"より"分布の代表性"が重要です。

Q2. C++ と Python の使い分けは?

SIMD命令を直接叩く部分やMCTS探索コアはC++。
具体的には将棋のMCTS探索、TensorRTエンジン呼び出し、
NNUE評価関数の高速演算など。

LLM統合・RAGオーケストレーション・Webインターフェース・
パイプライン制御はPython。

Q3. 将棋AIの知見はどう業務に転用できるか?

実践で検証済みの知見：
・LLMには"考えさせる"のではなく"翻訳させる"設計が安定する
→ 判例AI・特許AIのRAGパイプラインで同じ設計を適用
・複数エンジンの合議（NNUE + DL）アーキテクチャは
複数LLMのアンサンブルと同じ設計原則
・ローカルLLMで一次処理 + クラウドLLMで高度判断の
ハイブリッド構成がコストと精度のバランスに有効

勝敗という明確な評価関数があるため、
LLM研究では曖昧になりがちな設計判断を定量的に検証できます。

Q4. 得意領域は？

・LLM + 専門ドメインのRAG設計・実装
・SaaS依存からのAIエージェント自作（プロンプト設計含む）
・ローカルLLM × クラウドLLMのハイブリッド推論構成
・Streamlit / FastAPIによる高速プロトタイピング
・GPU推論の最適化（TensorRT / CUDA / SIMD）」

FAQ: AI開発環境と技術スタック