OSS DeepResearch 比較

OSS	LLM利用方法	Web検索機構	技術スタック・コード構造	社内データ統合の拡張性・改修ポイント
HuggingFaceSmolAgents (OpenDeepResearch)	デフォルトはOpenAI o1を利用。CodeAgent方式でLLMがPythonコードでツール呼出しを生成し計算効率と精度向上	SerpAPIを利用してGoogle検索を実行。HTMLをMarkdownに変換してLLMへ供給	HuggingFaceのsmolagentsフレームワーク上で構築。検索・スクレイピング処理は主に `text_web_browser.py` で実装。プラグイン型ツール設計でツール追加が容易	新規ツールクラス(`internal_search`)を追加し内部資料のベクトル検索を実装。エージェント初期化時にツールリストへ登録するだけでWeb検索とハイブリッド運用が可能
JinaAINode-DeepResearch	デフォルトはGemini2.0(Flash)を利用。環境変数でOpenAI等に切替可能	JinaReaderサービスを利用してBrave/DuckDuckGo等から検索結果取得。API呼出しでスクレイピングと要約を一体実現	Node.js/TypeScriptで実装。非同期処理(Promise/async-await)を用いたシンプルなループ(Question→Search→Read→Reason)構造。検索関連はAPIコールに依存	検索ステップに内部向けAPI(Pinecone等のJSSDK利用)を追記しWeb検索結果とマージ可能。検索処理部分の関数改修で内部データを加えたハイブリッド運用が実現
dzhng/deep-research	デフォルトはOpenAI o3-miniを利用	FirecrawlAPIを利用しSERP検索とページスクレイピングを一括実施。取得結果はMarkdown化されLLMへ供給	Node.js/TypeScriptで約500行のシンプル実装。ユーザ入力に基づくループ(breadth/depth)で検索クエリ生成と並列実行(Promise.all)を採用。中心はFirecrawlAPI呼出し	Firecrawl検索呼出し部分をラップし内部資料のベクトル検索結果をマージ。シンプルなコード構造のため、最小限の変更でWeb検索結果に内部情報を付加可能
LangChainOpenDeepResearch	PlannerとWriterの2段階構成。デフォルトはPlannerでOpenAI(o3-mini)、WriterでAnthropic(Claude3.5)を利用。環境変数/設定ファイルで柔軟に変更可能	TavilyAPI(またはPerplexityAPI)を用いたリアルタイムWeb検索。検索結果はLangChainのDocumentオブジェクトとして取得しLLMへ使用	Python＋LangChainフレームワーク上で構築。各コンポーネント(Planner,Searcher,Writer)が明確に分離。チェーン/エージェント仕組みを活用しプロンプトテンプレートやツールとしての検索機能を実装	LangChainのDocumentLoaderやVectorStore(Chroma/FAISS/Pinecone等)を利用したRetrieverを容易に追加可能。カスタムRetrieverでTavily検索結果と内部ベクトル検索結果を統合し、設定変更/少量のコード修正で内部データ統合が実現可能

Hugging Face SmolAgents

LLM利用方法

- OpenAI APIやHugging Face Hub経由でGPT-3.5/4等を利用
- CodeAgent方式で、LLMが直接Pythonコードでツール呼び出しを実現

Web検索機構

- SerpAPIを使用してGoogle検索を実行
- HTTP GETで対象ページを取得、HTMLをMarkdownへ変換

技術スタック・コード構造

- Python（smolagentsフレームワーク）
- 主に `text_web_browser.py` に検索・スクレイピング処理を実装

内部データ統合の拡張性

- 新規ツールクラス（例: `internal_search`）を追加
- エージェント初期化時にツールリストへ登録することでハイブリッド運用が可能

Jina AI Node-DeepResearch

LLM利用方法

- デフォルトでGemini 2.0 (Flash)を利用（OpenRouter経由）
- 環境変数によりOpenAI等への切替が可能

Web検索機構

- Jina ReaderサービスでBrave/DuckDuckGoなどから検索結果を取得
- API呼び出しでスクレイピングと要約処理を実施

技術スタック・コード構造

- Node.js/TypeScript実装
- 非同期処理（Promise, async/await）によるシンプルなループ構造

内部データ統合の拡張性

- 検索ステップに内部向けAPI（例：PineconeのJS SDK）を追記
- 検索処理関数の改修のみでWeb検索結果と内部情報をマージ可能

dzhng/deep-research

LLM利用方法

- 主にOpenAI GPT-3.5系（o3-mini）を利用
- APIキーまたはローカルのOpenAI互換サーバ経由で切替可能

Web検索機構

- Firecrawl APIを利用してSERP検索とスクレイピングを一括実施
- 結果はMarkdown形式に変換され、LLMへ供給

技術スタック・コード構造

- Node.js/TypeScript、約500行のシンプル実装
- ユーザ入力に基づくループ（breadth/depth）で並列実行

内部データ統合の拡張性

- Firecrawl検索部分をラップし、内部資料のベクトル検索結果をマージ
- 最小限の変更で内部情報をWeb検索結果に付加可能

LangChain Open Deep Research

LLM利用方法

- 2段階構成：Planner (OpenAI o3-mini) と Writer (Anthropic Claude 3.5)
- 環境変数や設定ファイルで柔軟に変更可能

Web検索機構

- Tavily API（またはPerplexity API）を用いたリアルタイム検索
- 検索結果はLangChainのDocumentオブジェクトとして取得

技術スタック・コード構造

- Python＋LangChainフレームワーク
- 各コンポーネント（Planner, Searcher, Writer）がモジュール分離されたチェーン/エージェント構造

内部データ統合の拡張性

- Document LoaderやVectorStore（Chroma, FAISS, Pinecone等）を利用したRetriever追加が容易
- カスタムRetrieverでWeb検索結果と内部ベクトル検索結果を統合可能

まとめ

- 各OSSはLLM利用方法、Web検索手法、技術スタック、内部データ統合の拡張性で特徴が異なる
- **SmolAgents**と**dzhng/deep-research**はシンプルかつ柔軟、**Jina AI**はAPI依存、**LangChain**は拡張性に優れる
- 社内データとのハイブリッド運用は、最小限のコード改修で各実装に追加可能
- 要件に合わせたOSS選定と内部データ統合の工夫で、最適なDeepResearchシステムが構築できる

（参考）Deep Research ベンチマーク比較結果

- 各種 Deep Research 系OSSのベンチマーク結果
- 主な比較対象：**Humanity’s Last Exam** と **GAIA Benchmark**
- 主要モデルの正答率・スコアを整理

Humanity’s Last Exam

- **OpenAI Deep Research:** 26.6%
- **Perplexity Deep Research:** 約20.5〜21.1%
- 【参考】他モデル：
  - GPT-4o: 3.3%
  - Grok-2: 3.8%
  - Claude 3.5 Sonnet: 4.3%
  - DeepSeek-R1: 9.4%
  - OpenAI o3-mini: 13.0%

GAIA Benchmark

- **OpenAI Deep Research:** 67.4% (pass@1) ～ 72.6% (cons@64)
- **Hugging Face Open Deep Research (smolagents):** 約54%
- **Microsoft Research Magentic-One:** 約46%

まとめ

- **OpenAI Deep Research** が最も高い成果を示す
- **Perplexity Deep Research** は Humanity’s Last Exam で約21%
- **Hugging Face Open Deep Research**（smolagents）は GAIA で約54%
- 他の実装（Jina、dzhng、LangChain）は具体的な数値未公表
- 今後のアップデートにより評価が変動する可能性あり