ShinkaEvolve チュートリアル

LLM × 進化的アルゴリズムで「コードを進化」させる

---

0. 想定読者とゴール

想定読者

- Python は普段から使っている
- 教師あり学習など ML の基礎はわかる
- 進化的アルゴリズム（遺伝的アルゴリズムなど）は軽く触ったことがある
- 強化学習は「状態・行動・報酬」くらいは知っているが実装経験はほぼない

ゴール

1. **EA（Evolutionary Algorithm = 進化的アルゴリズム）** と ShinkaEvolve の関係がわかる
2. ShinkaEvolve がやっていること（アルゴリズムの自動改良）の流れを理解する
3. 自分の手元で ShinkaEvolve をインストールして、サンプルと自作タスクを動かせる
4. 最後に、日本のエンプラ企業での活用アイデアを 30 個チェックできる

---

1. ShinkaEvolve とは何か？関連プロジェクトとの関係

1.1 EA（Evolutionary Algorithm）とは

ここで使う **EA** は **Evolutionary Algorithm（進化的アルゴリズム）** の略です。

- 個体 = 解候補（ここでは「プログラム」）
- 集団 = 個体の集合
- 適応度（fitness）= 解の良さ（スコア）
- 選択 + 交叉 + 突然変異を世代ごとに繰り返し、
  集団全体の適応度を上げていく最適化手法

強化学習（RL）は 1 つのポリシーを勾配などで更新していくのに対して、
EA は「たくさんの解」を並行して扱い、「選択と変異」で進化させていくイメージです。

1.2 AI Scientist → AI Scientist-v2 → ShinkaEvolve

Sakana AI は「**AI が AI を作る／研究する**」という文脈でいくつかのプロジェクトを出しています：

- **The AI Scientist（v1）**
  - 論文：*The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery*
  - LLM エージェントが
    - 研究アイデア生成
    - コードを書く
    - 実験実行

    - 結果を解析・可視化
    - 論文を書く
    - 疑似査読する
      までを自動で回すフレームワーク（GitHub: `SakanaAI/AI-Scientist`）

- **The AI Scientist-v2**
  - 論文：*The AI Scientist-v2: Workshop-Level Automated Scientific Discovery via Agentic Tree Search*
  - v1 の問題だった「人間が書いたテンプレコードへの依存」を減らし、
    agentic tree search（エージェント木探索）で柔軟な実験探索ができるようになった
  - 完全自動生成の論文が ICLR Workshop の査読で、人間論文と同等以上のスコアを獲得

- **ShinkaEvolve**（本チュートリアルの主役）
  - 論文：*ShinkaEvolve: Towards Open-Ended And Sample-Efficient Program Evolution*
  - GitHub：`SakanaAI/ShinkaEvolve`
  - 内容：
    - LLM を **知的な突然変異オペレータ** として利用する EA フレームワーク
    - Circle Packing / 数学推論エージェント / 競プロ / MoE 損失設計などで
      既存手法より **圧倒的に少ない試行回数** で SOTA 級のコードを発見

ざっくりいうと：

> AI Scientist：研究プロセス全体の自動化
> ShinkaEvolve：その中の「コード（アルゴリズム）をどう進化させるか」に特化した部分

---

2. ShinkaEvolve の考え方：EA × LLM

2.1 進化ループの全体像

ShinkaEvolve の基本ループは、教科書どおりの EA に LLM を組み合わせたものです。

1. **初期プログラムの用意**
   - ユーザが `initial.py` に「とりあえず動く」コードを書く
   - ここが世代 0 の個体になる

2. **評価（fitness 計算）**
   - `evaluate.py` が `initial.py` のコードを実行し、スコアを返す
   - 例：目的関数の値、精度、コスト、制約違反ペナルティなど

3. **親個体の選択（親サンプリング）**
   - アーカイブ（これまで評価されたプログラム群）から、
     スコアに応じた確率で「親プログラム」を選ぶ

4. **LLM による変異生成**
   - 親プログラム + 評価結果 + 過去の有望なプログラムをコンテキストに、
     LLM に「このコードをもっと良くなるようにパッチを当てて」とプロンプト

- LLM が diff や関数の書き換えなど「子プログラム」を生成

5. **新奇性フィルタ**
   - 生成されたパッチをコード埋め込みでベクトル化し、
     既存アーカイブのコードと類似度が高すぎれば **評価前に棄却**
   - （一部設定では LLM に「これは新しいアイデアか？」と判定させる）
6. **子プログラムの評価 → アーカイブ更新**
   - 新しいプログラムを実際に実行・評価し、スコアが良ければアーカイブに追加
   - 必要に応じて劣った個体をアーカイブから押し出しながら、良い個体を蓄積
7. **LLM アンサンブル選択**
   - 複数 LLM（例：安いモデル／高精度モデル）を用意し、
     「どの LLM が良いパッチを出しやすいか」をバンディット的に学習して重み付け
8. 2〜7 を繰り返し、**プログラム群が世代を重ねて進化**していく

2.2 ShinkaEvolve が強い理由（3 つの工夫）

論文・ブログで強調されているポイントは次の 3 つです：

1. **賢い親サンプリング（exploration / exploitation バランス）**
   - 上位個体だけに偏りすぎず、多様性も維持した選択戦略を取る（例：power-law 的な分布）
2. **コード新奇性によるリジェクション**
   - 埋め込み類似度や LLM 判定で、ほぼ同じアイデアのパッチは評価前に弾き、評価コストを節約
3. **LLM アンサンブルのバンディット選択**
   - 複数モデルを用意し、パッチの品質に応じてどのモデルを何回呼ぶかを動的に調整

この結果、Circle Packing や AIME エージェント設計などで、
既存の AlphaEvolve / OpenEvolve / LLM4AD より **1〜2 桁少ない評価回数**で
同等〜それ以上の解に到達できたと報告されています。

---

3. 環境構築：手厚めステップバイステップ

ここからは、あなたのマシンで ShinkaEvolve を動かすところまでを丁寧にまとめます。

3.1 前提環境

- OS
  - Linux / macOS 推奨（Windows でも WSL2 か PowerShell で OK）
- Python
  - **3.11 推奨**（公式 README でも `uv venv --python 3.11` を例示）

- 必要ツール
  - `git`
  - `curl`（`uv` インストールに使用）
  - または、`python -m venv` / `pip` だけでも可
- LLM へのアクセス
  - デフォルト設定では `llm_models=["azure-gpt-4.1-mini"]` になっている（README より）
  - Azure OpenAI 以外を使う場合：
    - 自分が使えるモデル（例：OpenAI の `gpt-4.1-mini`、ローカル LLM など）に変更
    - API キーやエンドポイントを環境変数や設定ファイルで指定

> ここでは「とりあえず LLM が 1 つは呼べる状態になっている」前提で進めます。
> （OpenAI / Azure / local model どれでもよい）

3.2 インストール手順（公式手順ベース）

公式 README の Quick Start をベースにしています。

3.2.1 リポジトリ取得

```
git clone https://github.com/SakanaAI/ShinkaEvolve
cd ShinkaEvolve
```