

runops + Claude Code

HPC シミュレーション研究のための
AI 支援ワークフロー

対象環境: camphor3 (京都大学) + EMSES

runops v0.3.0 / Claude Code (Anthropic)

2026-04-10

1. はじめに — なぜ runops + Claude Code か

HPC シミュレーション研究では、パラメータファイルの設定、run の準備・投入・管理、解析・可視化、考察のまとめといった定型作業に多くの時間を取られます。これらは研究に不可欠ですが、研究者が本来やりたいこと — 物理の考察、新しい仮説の検証、論文の執筆 — ではありません。

また、パラメータ設定のコピペミスは計算資源の浪費に直結します。手作業が多いほどミスが増え、数日分の計算が無駄になることもあります。

runops は、これらの定型作業を構造化し AI Agent に委譲するために設計された HPC シミュレーション管理ツールです。**Claude Code** (Anthropic 社の AI コーディングエージェント) と組み合わせることで、「やるべきこと」ではなく「やりたいこと」に集中できるワークフローを実現します。

さらに、すべての run が構造化・記録されるため、過去の実験が将来の研究資産として再利用可能になります。

2. runops の概要

runops は Python 製の CLI ツールで、以下の機能を提供します。

主な機能

- **Run 中心の管理:** 各シミュレーション実行を一意的な run_id で識別し、入力・出力・設定・来歴を 1 つのディレクトリに集約
- **パラメータサーベイ:** survey.toml にサーベイ軸を定義し、直積展開で run を自動生成 (手作業でのパラメータ計算が不要)
- **Slurm 統合:** ジョブスクリプトの自動生成、投入 (sbatch)、状態同期 (squeue/sacct)、ログ取得を一元管理
- **Simulator Adapter:** EMSES・BEACH 等のシミュレータごとに入力形式・実行方法・出力検出を抽象化 (新シミュレータの追加が容易)
- **知見管理:** 実験の知見を curated knowledge (insights + facts) と lab notebook (時系列ノート) の 2 層で記録
- **来歴 (Provenance):** manifest.toml に run の全情報を記録し、「いつ・誰が・どの設定で・どの commit から」実行したか追跡可能

基本概念: 4 層の階層構造

runops はシミュレーション研究を 4 つの階層で整理します:

階層	定義ファイル	役割
Campaign	campaign.toml	研究テーマ全体 (仮説・変数・観測量)
Case	case.toml	シミュレーション設定のテンプレート
Survey	survey.toml	パラメータサーベイ定義 (掃引軸)
Run	manifest.toml	個々のシミュレーション実行 (自動生成)

例えば「イオン温度を 1-19 eV で 10 点掃引」という実験は、Campaign の中の 1 つの Survey として定義され、AI Agent が自動的に 10 個の Run を生成します。各 Run は固有の run_id を持ち、入力ファイル・ジョブスクリプト・manifest が自動で配置されます。

3. 初期セットアップ (唯一の手動操作)

ユーザーが直接コマンドを実行するのは、このセットアップだけです。以降の操作はすべて Claude Code 経由で行います。camphor のような HPC 環境では sudo が使えないため、すべてユーザー権限でインストールします。

前提条件

- camphor3 のアカウント (Slurm が使える環境)
- Python 3.10 以上 (module load で利用可能)
- Git / GitHub アカウント

Step 1: uv のインストール

Python パッケージマネージャ。runops のインストールに必要です:

```
curl -LsSf https://astral.sh/uv/install.sh | sh
source ~/.bashrc # パスを反映
```

Step 2: Node.js のインストール (nvm 経由)

Claude Code と gh (GitHub CLI) の実行に Node.js が必要です。sudo 不要な nvm を使います:

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.3/install.sh | bash
source ~/.bashrc # nvm を反映
nvm install --lts # 最新 LTS 版をインストール
```

Step 3: gh (GitHub CLI) のインストールとログイン

AI Agent がフィードバック issue を投稿するのに gh を使います。Node.js 環境があれば npm でインストールできます:

```
npm install -g @anthropic-ai/claude-code # Claude Code 後述()
npm install -g gh # GitHub CLI 非公式(npm)
```

npm に gh パッケージがない場合は、公式バイナリを直接ダウンロードします:

```
# Linux x86_64 の場合
GH_VER=2.74.0 # https://github.com/cli/cli/releases で最新版を確認
curl -L "https://github.com/cli/cli/releases/download/\
v${GH_VER}/gh_${GH_VER}_linux_amd64.tar.gz" | \
tar xz --strip-components=1 -C ~/.local
```

インストール後、GitHub にログインします:

```
gh auth login
```

対話プロンプトに従い、GitHub.com → HTTPS → ブラウザ認証 (または token 入力) を選択します。camphor からブラウザが開けない場合は Paste an authentication token を選び、<https://github.com/settings/tokens> で生成した Personal Access Token を貼り付けてください。

Step 4: runops でプロジェクトを初期化

uvx (uv の一時実行機能) を使い、1 コマンドで初期化できます:

```
mkdir my_simulation && cd my_simulation && git init
uvx --from runops runops init
source .venv/bin/activate
```

これにより runops のインストール、venv 作成、プロジェクト構造 (runops.toml, simulators.toml, campaign.toml, .runops/ 等) の生成がすべて自動で行われます。

Step 5: Claude Code を起動 → 以降は AI に任せる

```
claude
```

起動後は自然言語で指示するだけです:

```
> 環境を確認して (runops doctor を実行)
> EMSES 用の flat_plate ケースを作って
> Ti=1-19eV で 10 点のサーベイを設計して
> smoke test として 3 点を gr20001a に投入して
> 完了した run を解析して
```

ケース作成、サーベイ設計、run 生成・投入、監視、解析、知見記録 — これらはすべて AI Agent が runops コマンドを適切に選択・実行します。

4. Claude Code との連携

Claude Code とは

Claude Code は Anthropic 社が提供する AI コーディングエージェントです。ターミナル上で対話的に動作し、ファイルの読み書き・コマンド実行・Git 操作などを自律的行えます。

- ターミナルで claude コマンドを起動するだけで使える CLI ツール
- コードの読解・編集・デバッグに加え、ファイル操作やシェルコマンドの実行が可能
- プロジェクトの CLAUDE.md を読んで文脈を理解し、プロジェクト固有のルールに従う
- 危険な操作 (ジョブ投入、ファイル削除等) は人間の承認を求める安全設計

runops のハーネス (Agent 連携基盤)

runops は Claude Code 用のハーネスを自動生成します。runops update-harness コマンドにより以下が配置されます:

ファイル	役割
CLAUDE.md	プロジェクト運用ルール・スキル一覧・役割分担
.claude/skills/	定型作業のスキル定義 (15+ 種類)
.claude/rules/	Agent が守るべき制約 (ファイル保護、承認フロー等)
.claude/settings.json	権限設定 (許可/確認/拒否するツール)

自然言語で指示 → AI が自動で実行

runops のハーネスには「スキル」と呼ばれる定型作業の手順書が内蔵されており、AI Agent はユーザーの自然言語の指示に応じて適切なスキルを自動選択・実行します。**ユーザーがスキル名を覚える必要はありません。**

自然言語の指示例	AI が行うこと
「EMSES 用の新しいケースを作って」	ケース作成
「Ti を 10 点掃引するサーベイを設計して」	パラメータサーベイ設計
「全 run を生成して投入して」	run 生成 → 投入 (承認あり)
「今の進捗を確認して」	状態確認・Slurm 同期
「完了した run を解析して」	解析・集計
「失敗した run を調べて」	失敗 run の診断
「結果を知見として記録して」	知見の整理・保存
「今の作業をノートに残して」	lab notebook に記録

安全設計: 人間の承認が必要な操作

以下の操作は AI が自動実行せず、必ずユーザーの確認を求めます:

- **runops runs submit** (ジョブ投入): 対象 run・queue・資源量を提示してから実行
- **runops runs delete / purge-work** (実ファイル削除)
- 大規模 survey の一括投入 (--all)
- 設定ファイル (CLAUDE.md, settings.json 等) の変更

5. 典型的なワークフロー例

実際の研究サイクルを例に、runops + Claude Code のワークフローを示します。

Phase 1: 計画 (人間 → AI に伝える)

研究の方向性を AI Agent に自然言語で伝えます。AI が campaign.toml の作成・編集を行います。

```
> 研究テーマは「太陽風プラズマ中の吸収平板によるイオン枯渇」。  
  仮説は「枯渇角はイオン温度に依存する」。  
  Ti を 1-19eV で掃引して、イオン密度と電位を観測したい。  
  ベース入力は ~/large1/exp_flat/plasma.inp を使って。
```

Phase 2: 準備 (AI が自動実行)

AI がケース作成、サーベイ設計、run 生成を一貫して行い、設計の経緯を lab notebook に自動記録します。

```
> Ti サーベイを設計して、run を生成して
```

Phase 3: 実行 (AI 提案 → 人間承認)

ジョブ投入は AI が提案し、**ユーザーが承認**してから実行されます。投入後の状態監視も AI に任せられます。

```
> smoke test 3 点を gr20001a に投入して  
> [AI が投入内容を提示 -> ユーザーが承認]  
> 進捗を確認して
```

Phase 4: 解析 (AI が自動実行)

```
> 完了した run を解析して、Ti 依存性をプロットして  
> run 間で比較して
```

Phase 5: 知見整理 (AI が自動実行)

```
> 結果を知見として記録して  
> 次のサーベイの方針を議論しよう
```

AI が curated knowledge (insights + facts) を保存し、次のサーベイの方針を提案します。ユーザーとの議論を経て Phase 1 に戻ります。

6. 運用フィードバックの仕組み

runops は開発中のツールであり、日常運用の中で改善点が見つかることがあります。runops にはこのフィードバックループを組織化するための仕組みが組み込まれています。

upstream-feedback ルール

runops のハーネスには、AI Agent が日常運用中に runops 自体の改善点 (バグ・使いにくさ・不足機能・わかりにくいエラーメッセージなど) を検出したとき、GitHub issue としてフィードバックを提出するルールが組み込まれています。AI が自動で気づくこともありますが、ユーザーから「この挙動は runops に issue を出したほうがいい」と指示することもできます:

1. **AI が改善提案をユーザーに説明** (勝手に issue を切らない)
2. ユーザーが OK したら **AI が gh issue create で issue を作成**
3. 投げた issue の URL を **lab notebook に自動記録**
4. 当面の研究作業は **workaround で継続** (フィードバックで作業を止めない)

/update-runops スキル (ユーザーが手動で呼ぶ)

runops 本体やハーネスファイルの更新は、ユーザーが Claude Code 上で /update-runops と入力して実行します。これは意図的なタイミングで行う操作のため、自動ではなくスキルとして明示的に呼び出します:

```
> /update-runops
AI: tools/runops を pull して update-harness を実行します...
    -> CLAUDE.md, skills/, rules/ が最新テンプレートで更新されました
    -> runops doctor で環境を確認 ... OK
```

実行内容:

- **tools/runops/** リポジトリを git pull で最新化
- **runops update-harness** でハーネスファイル (CLAUDE.md, skills, rules 等) を再生成
- **runops update** でシミュレータパッケージを更新
- 編集済みファイルとの競合は .new ファイルとして出力され、diff 確認後にマージ

チームで runops を使う場合、開発者が runops 本体を更新したら、各プロジェクトで /update-runops を実行するだけでハーネスが最新化されます。

フィードバックが runops を育てる

重要なのは、「この 1 回だけ回避できればよい」ではなく、「次に同じ作業をする人が困らないようにする」という視点です。AI が運用中に自然とフィードバックを収集し、ユーザーの承認を経て upstream に還元する — このサイクルにより runops は**使いながら育つツール**になります。

7. Claude Code の補足

認証

Claude Code の初回起動時に Anthropic の認証が求められます。Max プラン (月額サブスクリプション) の場合は API キーなしでブラウザ認証で直接利用可能です。API プランの場合は <https://console.anthropic.com/> で API キーを取得してください。

使い方のコツ

- **具体的に指示する** — 「解析して」より「完了した run の Ti 依存性をプロットして」のほうが精度が高い
- **承認プロンプトを確認する** — ジョブ投入やファイル削除など危険な操作では AI が確認を求めるので、内容を読んでから承認する
- **会話を続ける** — 1 つのセッション内で計画→準備→解析まで一貫して指示できる

8. 運用のコツとベストプラクティス

Git コミットは AI が自動で行う

runops のハーネスには「意味のある作業単位ごとに Git コミットする」というルールが組み込まれており、AI Agent は case 作成後・run 生成後・解析完了後・知見保存後などのタイミングで自動的にコミットを作成します。もしコミットされていないと感じたら、「コミットして」と指示すれば即座に対応します。

Lab notebook は自然言語で指示する

AI Agent に「今の議論をノートに残して」「サーベイの設計理由を記録して」と伝えれば、lab notebook (notes/YYYY-MM-DD.md) に自動記録します。「なぜこのパラメータ範囲にしたか」「なぜこの case を保留したか」といった情報は再現性に直結するので、積極的にメモを指示してください。/note スキルを直接呼ぶこともできますが、自然言語での指示のほうが柔軟です。

Smoke test を活用する

大規模サーベイの前に、両端と中央の 2-3 点だけ先行投入 (smoke test) して入力 of 妥当性を確認するのが効果的です。「まず 3 点だけ smoke test して」と AI に指示すれば、適切な点を選んで投入します。

プロジェクトを GitHub private repo で管理する

プロジェクトディレクトリは GitHub の private リポジトリとして管理することを強く推奨します。runops はシミュレーションの出力データ (大容量) を .gitignore で除外し、パラメータファイル・設定・解析スクリプト・知見など再現に必要な情報だけを Git 管理するように設計されているため、GitHub での管理が容易です。

これにより:

- HPC 上のデータ消失に対するバックアップ
- 研究の全履歴 (設計意図、パラメータ変更、解析結果) の永続的な記録
- 将来の自分や共同研究者が過去の実験を再現・参照可能

リポジトリの作成も AI Agent に任せられます:

```
> このプロジェクトを GitHub に private repo として上げて
```

AI が gh repo create を実行し、リモート設定と初回 push まで行います。

バグを見つけたら feedback を

runops のバグや改善点を見つけたら、AI Agent に「この挙動は runops に issue を出して」と伝えてください。AI がユーザーの承認を得てから GitHub issue を作成し、当面の研究は workaround で継続します。

9. 導入チェックリスト

runops + Claude Code を始めるためのステップ:

1. **uv** — `curl -LsSf https://astral.sh/uv/install.sh | sh`
2. **nvm + Node.js** — `curl -o- https://.../nvm/install.sh | bash && nvm install --lts`
3. **Claude Code + gh** — `npm install -g @anthropic-ai/claude-code gh`
4. **gh ログイン** — `gh auth login`
5. **プロジェクト初期化** — `mkdir my_sim && cd my_sim && git init`
6. **runops 初期化** — `uvx --from runops runops init && source .venv/bin/activate`
7. **Claude Code 起動** — `claude`
8. **以降はすべて AI に指示** — 「環境を確認して」「ケースを作って」「サーベイを設計して」...

10. 参考リンク

runops:

- GitHub: <https://github.com/Nkzono99/runops>
- ドキュメント: `tools/runops/docs/` (プロジェクト内)

Claude Code:

- 公式: <https://claude.ai/code>
- ドキュメント: <https://docs.anthropic.com/en/docs/claude-code>
- GitHub: <https://github.com/anthropics/claude-code>

camphor3:

- マニュアル: <https://web.kudpc.kyoto-u.ac.jp/manual/ja>