ENZYME Framework 開発仕様書（Codex入力用）v0.4

作成日: 2026-02-09

対象: 実験プロトコルの形式化・妥当性検証・完全性/再現性評価フレームワーク「ENZYME」

想定実装言語: Python 3.11+（MVP）

想定CLI名: enzyme

推奨ライセンス: Apache-2.0（必要に応じて変更可）

---

0. 本仕様書のゴール（Codexに貼れば実装できる粒度にする）

このv0.4仕様は、次を満たす「動くENZYME（MVP）」をCodexで一括実装できることを目的とする。

1. ENZYME-IR v0.4（JSON）を読み書きできる（HL-IR / Core-IR）

2. ENZYME-Core（少数の基本関数）だけを信頼核（Kernel）で検証できる

3. ENZYME-Lib（macro）→ Core へのLowering（展開/コンパイル）が動く

4. Kernelが二値（PASS/FAIL）で 形式的妥当性 を検証し、機械可読Issueを返す

5. Scoringが連続/段階で 完全性・曖昧性・語彙カバレッジ・環境適合・同定性 を評価し、説明を返す

6. protocols.io等の入力を 構造（手順の順序、素材リスト）を保って 取り込み、最低限 annotate 中心のIR化ができる（意味解析はMVPでは必須にしない）

7. CLIで import / compile / validate / score / report がE2Eで動く

8. v0.4の意図: 装置が行う区間は run_device、人手の区間は transfer/manipulate、観察は observe を中心に表現する

9. 「何でも箱」を防ぐため、manipulate.action_kind と observe.features は Registry（統制語彙 + 必須キー） で拘束する（未宣言はwarn/error）

0.1 バイオセーフティ上の扱い（仕様に組み込む）

ENZYMEはソフトウェア（形式化/検証/評価）であり、危険な生物実験の実施手順や条件の提供を目的としない。

- **例示擬似コード/例示IRには、具体的な試薬量・温度・時間・濃度・菌株/ウイルス株などを記載しない**
- **必要な値は X, T, PROGRAM など 抽象変数 で表す**
- **LLM連携（任意）では「原文にない情報を推測して補完しない」をデフォルト動作にする（仕様で強制）**

---

1. 設計思想（LEAN/QUEENに近づけるポイント）

1.1 小さな信頼核（Small Trusted Kernel）

- Kernelが信頼するのは ENZYME-Core のみ。
- すべての高級操作（macro）は、コンパイラ（Lowering）によりCoreへ還元される。
- 拡張（macro追加、ドメイン増加）をしても、Kernelの複雑性は増やさない。

1.2 二層評価（二値 + 連続）

- 実験は物理世界/技能/環境依存であるため、正しさを二値で閉じにくい。
- ENZYMEは次の二層構造を採る。

- Kernel Validator: 二値（PASS/FAIL）で「形式的妥当性」を保証
- Scoring Engine: 0-1の連続/段階スコアで「完全性・曖昧性・語彙・環境適合」を評価

1.3 人手・装置・観察の境界（v0.4の中心）

- 装置が行う処理区間は run_device のみ（=装置プログラムを走らせる）。
- 人の手の動作は transfer（移送）と manipulate（手技）に分解する。
- 目視/顕微鏡/装置読出し等の観察は observe として第一級に記録し、条件分岐の根拠にできる。

---

2. 用語

- ENZYME-HL-IR: 高級IR。macro（高級操作）を含みうる。人間/LLMが生成しやすい。
- ENZYME-Core-IR: 正規形IR。Core操作のみで表現される。Kernelの入力。
- Lowering（展開/コンパイル）: HL-IRのmacroをCore操作列へ変換し、Core-IRを生成する処理。
- Registry（レジストリ）: action_kind（手技）、device_kind（装置）、observation_feature（観察項目）などの統制語彙と、必須キー/型制約を保持する設定ファイル。
- Issue: 検証で見つかった問題（error/warn/info）。機械可読な code と path と suggested_fix を含む。
- Detail Level: 記述粒度（0-3）。粒度が上がるほど必須項目が増える（例: 0=骨格、1=主要パラメータ、2=機器設定、3=品質/校正/技能など）。

---

3. ENZYME-Core（v0.4）: 基本関数セット

3.1 コア関数一覧（trusted primitive set）

ENZYME-Coreは次の7操作のみを「信頼できるプリミティブ」として定義する。

1. allocate
2. transfer
3. manipulate
4. run_device
5. observe
6. annotate

7. dispose

> 重要: run_device は「装置に入力を入れ、装置プログラム/設定を走らせる」区間のみ。サンプル調整（混合、反応液作成、分注、ラベル貼り等）は含めない。

3.2 Step共通フィールド（Core-IR/HL-IR共通）

各ステップは次の共通形を持つ（HL-IRでも同じ。macroも同形）。

- **id : string（必須。stable UUIDまたは安定ID）**
- **op : string（必須。Core opまたはmacro名）**
- **label : string（任意。人間可読）**
- **inputs : array of Ref（任意）**
- **outputs : array of Ref（任意）**
- **params : object（任意。ただしopごとの必須キーはここに入れる）**
- **provenance : object（任意。原文トレース。URL/行/ハッシュ等）**
- **annotations : object（任意。自由だが構造はJSON）**
- **tacit_refs : array（任意。勘所エントリ参照）**

3.3 Ref（参照）: 統一形式

Refは「何を指しているか」を最低限機械可読にするための統一形式。

```
{
  "kind": "material|container|equipment|sample|data|waste|location",
  "id": "res:...",
  "location": "A1",
  "role": "source|destination|template|..."
}
```

- **kind="data" は観察結果（observation）や装置出力などを指す。**
- **location はwell/slot/positionなど任意文字列。**
- **role は任意だが、Kernel/Scoringでの解析精度を上げる（例: template, destination）。**

---

4. 値表現（quantity/symbolic/range）の標準化（MVP必須）

プロトコルには「値が書かれていない/曖昧」なケースが必ずあるため、v0.4は 数値 と 記号値（X） を両方許容する。

4.1 Quantity

```
{"value": 10, "unit": "uL"}
```

4.2 Symbolic

```
{"symbol": "X_seed", "unit": "uL"}
```

- **unit は任意だが、ある場合はpintで解釈できる単位文字列にする。**

4.3 Range（任意: v0.4で扱う）

```
{"range": {"min": 1, "max": 3}, "unit": "min"}
```

4.4 Enum/Category

```
{"enum": "gentle"}
```

---

5. Core各操作の仕様（厳密）

5.1 allocate

目的:
サンプル/容器/データ等の論理的エンティティにIDを付与し、後続ステップで参照可能にする。

- **必須:**

- params.allocate_kind: "sample" ¦ "container" ¦ "data" ¦ "label" ¦ "other"

- outputs: 少なくとも1つ

- **任意:**

- params.name: string

- params.initial_state: object

Kernel規則

- **outputsのRef.kindが allocate_kind と整合する（例: allocate_kind=sampleなら outputs[].kind=sample）**

5.2 transfer

目的: 人が行う「移送」を表す最小単位。分注/回収/上清除去/廃液回収などもここで表す。

- **推奨:**

- inputs に source（material/sample/container）を含む

- outputs に destination（container/sample/waste）を含む

- **必須（Detail Level>=1）:**

- params.amount: Quantity¦Symbolic¦Range

- params.transfer_kind: "pipette" ¦ "pour" ¦ "aspirate" ¦ "decant" ¦ "other"

- **任意:**

- params.tip: "new" ¦ "reuse" ¦ "unknown"（例）

- params.notes: string

Kernel規則

- **Detail Level>=1で amount が無い場合は error**
- **amountがQuantity/Rangeでunitありの場合、pintで解釈できること（できなければ error）**

5.3 manipulate

目的: transfer以外の「人手手技」を表す核。巧技/勘所の受け皿。

- **必須:**

- params.action_kind: string（統制語彙。Registry）
- 必須（Detail Level>=1）:
- params.settings: object（action_kindに応じた必須キーをRegistryで定義）
- 任意:
- params.skill: object（勘所/熟練度/注意点など。Kernelは中身を強制しないが保持する）
- params.duration: Quantity¦Symbolic¦Range（任意）

Kernel規則（重要）
- action_kind は次のいずれか:
1) 公式Registryに存在
2) IR内 registries.custom.action_kinds に宣言されている
- 1)でも2)でもない場合:
- Detail Level 0: warn（スコアで減点）
- Detail Level>=1: error
- settings はRegistryが定義する必須キーを満たすこと（満たさない場合は error/warn）

5.4 run_device
目的: 装置（機器）が行う処理区間を表す唯一のコア。装置プログラム/設定を実行する。

- 必須:
- params.device_kind: string（Registry）
- params.program: object（装置プログラム/プロファイル/設定）
- 推奨:
- params.device_ref: "eq:..."（resources.equipment.idを参照）
- 任意:
- params.qc: object（校正情報/ログ参照など）
- params.run_id: string（装置ログと紐づける）

Kernel規則
- device_kind はRegistry/カスタム宣言のいずれか
- program は device_kind ごとの必須キー（Registry）を満たす

5.5 observe
目的: 目視/顕微鏡/装置読出しなどの観察結果を構造化して記録する。条件分岐・判断の根拠として参照される。

- 必須:
- params.modality: string（Registry）
- params.features: object（観察項目の集合）
- outputs: kind="data" のRefを少なくとも1つ（観察結果ID）
- 推奨:
- params.target_refs: array of Ref（観察対象。inputsでも可だが明示推奨）
- params.confidence: number 0-1（任意: 主観観察の自信度など）
- 例 features（抽象）:

- confluency, morphology, detached_enough, colonies_present, approx_count, pass_fail, viability

Kernel規則（重要）

- features のキーは次のいずれか:

1) 公式Registryの observation_features に存在

2) IR内 registries.custom.observation_features に宣言

- 値の型はRegistryに従う（number/bool/string/enum/range/symbolic）
- 未宣言featureは:

- Detail Level 0: warn

- Detail Level>=1: error

5.6 annotate

目的: 出典、注意、勘所、判断根拠、コメント、LLM confidence等の付与。

- 推奨:

- params.note: string

- params.tags: array of string

- params.links: array（URL等）

- Kernelは内容の真偽を評価しないが、JSON構造として妥当であることは検証する。

5.7 dispose

目的: 廃棄を明示化する（バイオ廃棄物、使用済み資材など）。

- 必須（Detail Level>=1）:

- params.disposal_kind: "biohazard" ¦ "chemical" ¦ "general" ¦ "other"

- 推奨:

- inputs に廃棄対象

- outputs に waste（任意）

---

6. 制御構造（if/while相当）: プロトコルグラフで表す

6.1 protocolはWorkflow Graph

- protocol.start_step_id から開始し、protocol.edges に従って遷移する。
- edgeは次を持つ:

- from: step_id

- to: step_id

- condition: Expression（任意。無い場合は無条件）

- priority: number（任意。複数edgeがtrueのときの優先）

6.2 Expression（条件式）: JSON AST

例:

```
{"op":"ge","left":{"ref":"data:obs0.features.confluency"},"right":{"symbol":"THRESH"}}
```

演算（MVP）:

- **比較: eq, ne, gt, ge, lt, le**
- **論理: and, or, not**
- **存在: exists**
- **値: const, ref, symbol**

Kernelが行うこと:

- **AST構造の妥当性**
- **ref が参照するデータIDが存在しうること（最低限: outputsにあるdata id、またはresources.data）**
- **参照パス（features.）のnameがRegistryで定義されていること（可能なら）**

---

7. ENZYME-IR v0.4（JSON）仕様

7.1 トップレベル

- **schema_version: "0.4"（必須）**
- **ir_kind: "hl" ¦ "core"（必須）**
- **metadata（必須）**
- **resources（必須）**
- **protocol（必須）**
- **environment（任意）**
- **registries（任意: custom宣言）**
- **tacit_knowledge（任意）**
- **x-...（拡張用）**

7.2 metadata（必須）

- **title : string**
- **version : string（任意。プロトコル版）**
- **source : object（type, uri, id, retrieved_at）**
- **created_at : ISO8601**
- **authors[] : string（任意）**
- **license : string（任意）**
- **tags[] : string（任意）**

7.3 resources（必須）

- **materials[]: {id, name, vendor?, catalog_number?, identifiers?}**
- **containers[]: {id, type, max_volume?, geometry?, identifiers?}**
- **equipment[]: {id, type, model?, calibration?}**
- **samples[]: {id, name?, container_ref?, notes?}（任意）**
- **data[]: {id, name?, schema?}（任意）**

7.4 protocol（必須）

- **detail_level: 0-3（デフォルト0）**
- **steps[]: Step（必須）**
- **start_step_id: string（必須）**
- **edges[]: Edge（必須）**
- **step_order[]: array of step_id（任意だが推奨。canonicalizationに使用）**

7.5 registries（任意: custom宣言）

- **custom.action_kinds[]: {name, required_settings_keys[], settings_schema?}**
- **custom.device_kinds[]: {name, required_program_keys[], program_schema?}**
- **custom.observation_features[]: {name, value_type, constraints?}**
- **custom.modalities[]: {name}**

---

8. Registry v0.4（統制語彙）: ファイル仕様

同梱ファイル: enzyme_registry.v0_4.json

8.1 必須トップレベル構造

- **registry_version: "0.4"**
- **core_ops[]**
- **action_kinds : dict（name -> spec）**
- **device_kinds : dict（name -> spec）**
- **modalities[]**
- **observation_features : dict（name -> spec）**
- **macros : dict（name -> macro spec）**

8.2 action_kinds spec

例:

```
"mix_gently": {
  "required_settings_keys": [],
  "settings_schema": {"type":"object","additionalProperties":true},
  "description": "gentle mixing by human hand"
}
```

8.3 device_kinds spec

例:

```
"thermocycler": {
  "required_program_keys": ["profile_name"],
  "program_schema": {"type":"object","properties":{"profile_name":{"type":"string"}},"required"
    :["profile_name"]},
  "description": "thermocycler run"
}
```

8.4 observation_features spec

例:

```
"confluency": {"value_type":"number", "constraints":{"min":0,"max":100}}
"detached_enough": {"value_type":"bool"}
```

8.5 macros spec（Lowering用）

**macroは「どのCore
op列に展開するか」を宣言する。MVPでは次2方式のどちらでも良いが、実装は(1)推奨。**

1) 実装側にPython関数としてmacro展開を持ち、Registryは宣言（必要キー）だけ持つ（推奨）

2) Registryに展開テンプレ（mini-DSL）を持つ

v0.4では(1)を採用し、Registryには「必要paramsキー」を明示する。

---

9. Compiler / Lowering（HL-IR → Core-IR）

9.1 基本要件

- **入力HL-IRの steps を走査し、macro opをCore op列へ展開する。**
- **出力Core-IRは:**

- ir_kind="core"

- 全step.opがCore opsのみ

- step_order が決定的

- JSONのキー順/配列順が正規化されている（canonical JSON）

9.2 ID生成（決定的）

- **親step S をmacro展開して k 個のCore stepを生成する場合、**

- S.1, S.2, ..., S.k のような安定IDを生成する（衝突時はsuffix）

- **provenanceは親から継承し、生成stepに annotations.lowered_from = S を付与する。**

9.3 必須macro（MVP）

- **thermocycle:**

- 入力: params.profile_name（必須）

- 展開: run_device(device_kind="thermocycler", program={"profile_name": ...})

- **incubate:**

- 入力: params.program_name

- 展開: run_device(device_kind="incubator", program={"program_name": ...})

- **centrifuge:**

- 入力: params.program_name

- 展開: run_device(device_kind="centrifuge", program={"program_name": ...})

- **measure:**

- 入力: params.device_kind, params.program, params.features_template?

- 展開: run_device(...) + observe(modality="instrument_readout", features=...)

---

10. Kernel Validator（検証核）仕様

10.1 入力/出力

- 入力: Core-IR（ir_kind="core"）
- 出力: ValidationResult

- status: "pass" | "fail"
- issues[]: Issue配列
- summary: counts（error/warn/info）

10.2 検証カテゴリ（MVP）

1. Schema検証（enzyme_core_ir.schema.v0_4.json）
2. Core op制約（opがCoreのみ）
3. Registry制約
- manipulate.action_kind
- run_device.device_kind + program required keys
- observe.modality + observation_features
4. 参照整合
- resources参照が解決できる
- edgesのfrom/toが解決できる
5. グラフ整合
- start_step_idが存在
- 1つもedgeが無い場合は error
- 到達不能ステップはwarn
6. 単位/値検証
- Quantity/Rangeのunitはpintで解釈可能
- constraints（min/max）があるfeatureは範囲チェック（数値の場合）
7. 軽量ルール検証（任意）
- container.max_volumeがある場合のtransfer超過（数値の場合）

10.3 Issueコード一覧（MVPで実装必須）

- SCHEMA_INVALID（error）
- UNKNOWN_CORE_OP（error）
- UNKNOWN_ACTION_KIND（warn/error）
- UNKNOWN_DEVICE_KIND（warn/error）
- MISSING_REQUIRED_PROGRAM_KEY（error）
- UNKNOWN_MODALITY（warn/error）
- UNKNOWN_OBSERVATION_FEATURE（warn/error）

- REF_NOT_FOUND（error）
- STEP_ID_NOT_FOUND（error）
- START_STEP_NOT_FOUND（error）
- EDGE_INVALID（error）
- UNIT_PARSE_ERROR（error）
- VALUE_OUT_OF_RANGE（warn/error）

---

11. Scoring Engine（評価）仕様（MVP）

11.1 スコア軸（6軸）
- S_structural: Kernel issuesから算出（error=0、warnで減点）
- S_param: detail_levelに応じたrequired set充足率（transfer.amount等）
- S_vocab: 標準語彙率（custom/unknownが多いほど減点）
- S_ident: 資源同定性（vendor/catalog/model/identifierの有無）
- S_ambiguity: Symbolic/Range幅/unknownの多さで減点
- S_exec_env: environment情報/設備参照の充足（device_ref等）、容量チェック

11.2 算出の具体（実装要件）
- 各スコアは0-1のfloat
- ScoreBreakdown を返し、上位の減点要因（Issue集計）を含める

---

12. Importer（protocols.io等）仕様（MVP）

12.1 方針
MVPでは「完全な意味解析」をImporterに要求しない。構造保持を最優先。

12.2 protocols.io変換（MVP）
- materials: resources.materials へ
- steps: すべて annotate stepとして格納（原文テキスト/順序/provenanceを保持）
- step順序: previous_guid があれば復元して step_order を作る
- start_step_id: 最初のstep

12.3 出力IR
- ir_kind="hl"（ただし全stepがannotateのみなら core でもよい）
- schema_version="0.4"

---

13. CLI仕様（MVP）

- **enzyme import protocolsio --in --out**
- **enzyme compile --in --out**
- **enzyme validate --in --out**
- **enzyme score --in --validation --out**
- **enzyme report --in --validation --scores --format md --out**

---

14. レポート仕様（MVP）

Markdownレポートは次を含む。

- **メタデータ（title, source）**
- **Kernel結果（PASS/FAIL、Issue件数）**
- **Score（6軸 + total）**
- **改善提案（上位Issueの要約）**
- **用語/注記（detail_level等）**

---

15. 実装ブループリント（Codexが迷わないための具体API）

15.1 推奨リポジトリ構成

```
enzyme/
  __init__.py
  cli.py
  enzyme_ir/
    models.py
    schema_hl.json
    schema_core.json
    canonicalize.py
  registry/
    registry_v0_4.json
    loader.py
  compiler/
    lowering.py
    macros.py
  kernel/
    validate.py
    checks_schema.py
    checks_registry.py
    checks_refs.py
    checks_graph.py
    checks_units.py
  scoring/
    score.py
    profiles.py
  importers/
    protocolsio.py
  reports/
    render_md.py
  fixtures/
    protocolsio_fixture.json
    expected_core.json
```

```
  examples/
    example_transformation_hl.json
    example_transformation_core.json
    example_colony_pcr_hl.json
    example_cell_passage_hl.json
  tests/
    test_schema.py
    test_lowering.py
    test_kernel.py
    test_e2e.py
pyproject.toml
README.md
```

15.2 主要関数シグネチャ（MVP）

- **load_ir(path) -> dict**
- **save_ir(ir: dict, path) -> None**
- **compile_hl_to_core(hl_ir: dict, registry: Registry) -> dict**
- **validate_core_ir(core_ir: dict, registry: Registry) -> ValidationResult**
- **score_core_ir(core_ir: dict, validation: ValidationResult, registry: Registry) -> ScoreReport**
- **render_report_md(core_ir, validation, scores) -> str**

---

16. テスト/受け入れ基準（MVP）

16.1 必須テスト

- **Schema validation**
- **Lowering決定性（同一入力→同一出力）**
- **Registry未知語彙のwarn/error切替（detail_level）**
- **protocols.io fixtureでE2E（import→compile→validate→score→report）**
- **3代表例（抽象例）がCore-IRとしてvalidate PASSになる（ただしスコアは高くなくてよい）**

16.2 受け入れ基準

- **CLIが一通り動き、JSON/MDを出力する**
- **validateがPASS/FAILとIssueを返す**
- **scoreが6軸+totalと根拠を返す**
- **reportが人間可読**

付録A: 3つの一般的実験プロセス（抽象擬似コード）

本付録は「表現力テスト」のための抽象擬似コードであり、具体的な実験条件を含まない。

A-1. プラスミドベクターの形質転換（抽象）

```
plasmid      = allocate(kind="sample")
cells        = allocate(kind="sample")
```

```
mix_tube     = allocate(kind="container")
select_plate = allocate(kind="container")

transfer(plasmid -> mix_tube, amount=X_dna)
transfer(cells   -> mix_tube, amount=X_cells)
manipulate(action_kind="mix_gently", inputs=[mix_tube])

run_device(device_kind="dna_delivery_device", inputs=[mix_tube], program=DNA_DELIVERY_PROGRAM)
run_device(device_kind="incubator", inputs=[mix_tube], program=RECOVERY_PROGRAM)

transfer(mix_tube -> select_plate, amount=X_plate)
manipulate(action_kind="spread_on_plate", inputs=[select_plate])

run_device(device_kind="incubator", inputs=[select_plate], program=COLONY_GROWTH_PROGRAM)
obs = observe(modality="visual", targets=[select_plate], features={...})
annotate(note=f"obs={obs}")
```

A-2. コロニーピッキング→Genotyping PCR（抽象）

```
plate       = allocate(kind="container")
colony_tube = allocate(kind="container")
pcr_tube    = allocate(kind="container")

obs_plate = observe(modality="visual", targets=[plate], features={"candidate_colonies": "..."})
manipulate(action_kind="pick_colony", inputs=[plate], outputs=[colony_tube],
  settings={"pick_target":"..."})

transfer(reagent_mastermix -> pcr_tube, amount=X_mm)
transfer(primers -> pcr_tube, amount=X_primers)
transfer(colony_tube -> pcr_tube, amount=X_template)
manipulate(action_kind="mix", inputs=[pcr_tube])

thermocycle(profile_name=PCR_PROFILE)   # macro
measure(device_kind="amplicon_readout_system", program=READOUT_PROGRAM)  # macro
obs = observe(modality="instrument_readout", targets=[pcr_tube], features={"pass_fail":"..."})
```

A-3. 一般培養細胞の継代（抽象）

```
vessel     = allocate(kind="container")
new_vessel = allocate(kind="container")

obs0 = observe(modality="microscope", targets=[vessel], features={"confluency": "...",
  "morphology":"..."})

if obs0.confluency >= THRESH:
    transfer(vessel -> waste, amount=X_all)
    transfer(wash_buffer -> vessel, amount=X_wash)
    manipulate(action_kind="rinse", inputs=[vessel])
    transfer(vessel -> waste, amount=X_all)

    transfer(detach_reagent -> vessel, amount=X_detach)
    incubate(program_name=DETACHMENT_PROGRAM)   # macro -> run_device(incubator)

    while True:
        obs_detach = observe(modality="microscope", targets=[vessel],
          features={"detached_enough":"..."})
        if obs_detach.detached_enough: break
        incubate(program_name=DETACHMENT_CONTINUE_PROGRAM)

    transfer(medium -> vessel, amount=X_neutralize)
    manipulate(action_kind="dislodge_cells", inputs=[vessel])
    transfer(vessel -> tube, amount=X_all)
```

```
centrifuge(program_name=CENTRIFUGE_PROGRAM)  # macro
observe(modality="instrument_readout", targets=[tube], features={"cell_count":"...",
  "viability":"..."})

transfer(tube -> new_vessel, amount=X_seed)
incubate(program_name=CULTURE_PROGRAM)
```

付録B: ファイル構成（v0.4パッケージ）

- **ENZYME_Spec_v0_4.md**
- **ENZYME_Spec_v0_4.docx**
- **ENZYME_Spec_v0_4.pdf**
- **enzyme_ir.schema.v0_4.json（HL-IR）**
- **enzyme_core_ir.schema.v0_4.json（Core-IR）**
- **enzyme_registry.v0_4.json**
- **examples/（抽象例のHL/Core IR JSON）**
- **fixtures/（protocols.io fixtureと期待出力）**
- **Codex_Prompt_ENZYME_v0_4.txt（Codex貼り付け用プロンプト）**