Pre-Coding Essentials (Component: crates/vm\_pipeline/src/lib.rs, Version/FormulaID: VM-ENGINE v0) — 49/89

1. Goal & Success

Goal: Expose the pipeline entry points that orchestrate the fixed state machine and return canonical artifacts (Result, RunRecord, optional FrontierMap).

Success: Stage order and stop/continue semantics match Doc 5; outputs’ fields/IDs match Doc 1; determinism/ordering/RNG behaviors match Doc 3.

1. Scope

In scope: Public pipeline API; wiring of LOAD→…→BUILD\_RUN\_RECORD; error surface unification; re-exports of stage structs (LoadedContext, UnitScores, etc.).

Out of scope: Low-level I/O/JSON canonicalization (lives in vm\_io), algorithm math (in vm\_algo), report rendering (Doc 7).

1. Inputs → Outputs (with schemas/IDs)

Inputs: IDs/paths for DivisionRegistry, BallotTally, ParameterSet; optional Adjacency; all local (offline).

Outputs:

* **Result** (DB entity, RES:…) and **RunRecord** (RUN:…); optional **FrontierMap** (FR:…).
* Stage artifacts are assembled exactly per **Doc 5 §3**.

1. Entities/Tables (minimal)

(N/A — this module wires stage structs defined elsewhere.)

1. Variables (only ones used here)

Reads the **ParameterSet** snapshot and passes VM-VAR values to stages; lib itself declares no new variables.  
**Note:** tie resolution uses **VM-VAR-032 tie\_policy** and **VM-VAR-033 tie\_seed** in the tie stage (no VM-VAR-051).

1. Functions (signatures only)

/// High-level: run the full pipeline using already-loaded blobs/IDs.

pub fn run\_with\_ctx(ctx: PipelineCtx) -> Result<PipelineOutputs, PipelineError>;

/// Convenience: parse and verify a manifest (vm\_io), then run.

pub fn run\_from\_manifest(manifest: &Manifest) -> Result<PipelineOutputs, PipelineError>;

/// Accessors for versioning/FormulaID echoes used in RunRecord.

pub fn engine\_identifiers() -> (FormulaId, EngineVersion);

(Types mirror Doc 5 artifacts: LoadedContext, UnitScores, UnitAllocation, AggregateResults, LegitimacyReport, FrontierMap, TieLog, Result, RunRecord.)

1. Algorithm Outline (bullet steps)

* **LOAD** → LoadedContext.
* **VALIDATE** *(fail ⇒ mark Invalid, skip 3–8)*.
* **TABULATE** → UnitScores.
* **ALLOCATE** → UnitAllocation.
* **AGGREGATE** → AggregateResults.
* **APPLY\_DECISION\_RULES** → LegitimacyReport *(Fail ⇒ skip frontier)*.
* **MAP\_FRONTIER** (optional) → FrontierMap.
* **RESOLVE\_TIES** *(only if blocking; uses VM-VAR-032/033)* → TieLog.
* **LABEL\_DECISIVENESS** → {label, reason}.
* **BUILD\_RESULT** → Result.
* **BUILD\_RUN\_RECORD** → RunRecord.

1. State Flow (very short)

Exactly Doc 5 order above; stop/continue semantics enforced (Invalid path, “skip frontier” rule, ties only when blocking).

1. Determinism & Numeric Rules

Stable ordering (Units by ID; Options by order\_index then ID).  
Integer/rational math; half-even only at defined decision points.  
RNG only if tie\_policy = random, seeded by **VM-VAR-033 tie\_seed**; seed is recorded.

1. Edge Cases & Failure Policy

Any validation failure ⇒ label **Invalid** but still build Result & RunRecord with reasons.  
Gate failure ⇒ **Invalid**; FrontierMap omitted by design.  
If a blocking tie occurs and tie\_policy/tie\_seed are inconsistent or missing ⇒ surface MethodConfigError / TieUnresolvedError.

1. Test Checklist (must pass)

* Stage order & stop/continue semantics match Doc 5.
* Produced Result contains required fields/flags; RunRecord includes FormulaID/EngineVersion and tie\_seed (if used).
* Determinism: same inputs + same seed ⇒ identical Result/RunRecord bytes (canonical JSON).