# **Pre-Coding Essentials (Component: crates/vm\_cli/src/main.rs, Version/FormulaID: VM-ENGINE v0) — 68/89**

## **1) Goal & Success**

**Goal:** Orchestrate the **fixed pipeline** (LOAD → … → BUILD\_RUN\_RECORD), producing canonical **Result**, **RunRecord**, and optional **FrontierMap**, then render reports — all **offline** and **deterministic**.

**Success:** Same inputs (+seed if used) ⇒ **byte-identical** artifacts across OS/arch; section-ordered reports with **one-decimal** display only.

## **2) Scope**

**In scope:** Parse Args; dispatch loader/pipeline stages; apply **stop/continue** rules; write canonical JSON artifacts; call JSON/HTML reporters.

**Out of scope:** Core math (tabulation/allocation/gates), schema definitions, UI; those live in other crates/docs.

## **3) Inputs → Outputs (with schemas/IDs)**

**Inputs:** Local files only — DivisionRegistry, BallotTally or Ballots, ParameterSet; optional Adjacency/Frontier, Autonomy; all identified via IDs/ordering conventions.

**Outputs:**

**Result** (RES:<hash>), **RunRecord** (RUN:<timestamp>-<hash>), optional **FrontierMap** (FR:<hash>) written in **canonical JSON** (UTF-8, LF, sorted keys; UTC timestamps).

Reports consume only Result/RunRecord/FrontierMap; **approval-denominator sentence** mandatory for approval ballots.

## **4) Entities/Tables (minimal)**

## **5) Variables (only ones used here)**

## **6) Functions (signatures only)**

rust

CopyEdit

fn main() -> anyhow::Result<()>;

fn run(args: Args) -> anyhow::Result<ExitCode>;

fn load\_inputs(args: &Args) -> anyhow::Result<LoadedContext>; // LOAD

fn validate(ctx: &LoadedContext) -> anyhow::Result<()>; // VALIDATE (fail ⇒ invalid path)

fn tabulate(ctx: &LoadedContext) -> UnitScores; // TABULATE

fn allocate(scores: &UnitScores) -> UnitAllocation; // ALLOCATE

fn aggregate(alloc: &UnitAllocation) -> AggregateResults; // AGGREGATE

fn apply\_rules(aggr: &AggregateResults) -> LegitimacyReport; // APPLY\_DECISION\_RULES

fn map\_frontier(..) -> Option<FrontierMap>; // MAP\_FRONTIER (if enabled)

fn resolve\_ties(..) -> TieLog; // RESOLVE\_TIES (only if blocking)

fn label(..) -> DecisivenessLabel; // LABEL\_DECISIVENESS

fn build\_result(..) -> ResultDb; // BUILD\_RESULT

fn build\_run\_record(..) -> RunRecordDb; // BUILD\_RUN\_RECORD

fn render\_reports(res:&ResultDb, run:&RunRecordDb, fr:&Option<FrontierMap>) -> anyhow::Result<()>;

(Pipeline names/sequence and artifact types align with Doc 5.)

## **7) Algorithm Outline (bullet steps)**

**Parse args** (already validated upstream).

**LOAD** files → **LoadedContext**. **VALIDATE**; if it fails, follow **invalid path** (skip 3–8), still **label & build outputs** with reasons.

**TABULATE → ALLOCATE → AGGREGATE**.

**APPLY\_DECISION\_RULES**. If any **Fail**, mark **Invalid**, **skip MAP\_FRONTIER**, continue to **RESOLVE\_TIES** only if blocking; then **label & build outputs**.

If enabled and applicable, **MAP\_FRONTIER**; this never invalidates the run but can make label **Marginal**.

**RESOLVE\_TIES** only when required; if policy=random, apply **rng\_seed** and log.

**LABEL → BUILD\_RESULT → BUILD\_RUN\_RECORD**.

**Render reports** from artifacts; include **approval-rate** sentence for approval ballots; show **integrity** identifiers & fixed footer.

## **8) State Flow (very short)**

Follows Doc 5 state machine exactly; artifacts/IDs per Annex conventions; main performs **no network I/O**.

## **9) Determinism & Numeric Rules**

**Stable ordering** for reductions; **integer/rational** comparisons; **round-half-even** at defined points (done in core); **one-decimal** only in reports.

**Canonical JSON**: UTF-8, LF, **sorted keys**; timestamps UTC; hashes via **SHA-256** over canonical bytes.

## **10) Edge Cases & Failure Policy**

**Validation failed**: output **Invalid** Result/RunRecord; skip 3–8; reports use fallbacks (omit Frontier).

**Gates failed**: mark **Invalid**; **skip Frontier**; show ❌ in panel; outcome “Invalid (gate failed…)”.

**Frontier present but mediation/protected flags**: never invalidates; may set **Marginal**; include diagnostics in report.

**Seed handling**: if provided, record in **RunRecord**; never pull OS RNG/time.

## **11) Test Checklist (must pass)**

End-to-end over Annex B Part 0 fixtures produces artifacts with **canonical bytes** and stable hashes; **expected\_canonical\_hash** can be filled after certified run.

Stage order/stop-continue semantics match Doc 5; report checklist satisfied (section order, approval sentence, footer IDs).