# **Pre-Coding Essentials (Component: crates/vm\_algo/src/tabulation/score.rs, Version/FormulaID: VM-ENGINE v0) — 39/89**

## **1) Goal & Success**

Goal: Deterministically compute **UnitScores** for **score** ballots from **per-option score sums** and turnout, honoring scale and normalization knobs.

Success: Returns exact integer **score sums** per OptionId, preserves canonical option order, and carries turnout. Enforces scale/domain sanity (no floats, no RNG). Normalization policy is respected (see §7).

## **2) Scope**

In scope: Per-unit *aggregation* path using **already-summed** scores; caps and consistency checks against Params (VM-VAR-002..004); canonical ordering.

Out of scope: Per-ballot normalization math from raw ballots (that path belongs in a higher layer if raw ballots are present), allocation/gates, I/O/schema.

## **3) Inputs → Outputs**

Inputs:

unit\_id: UnitId

score\_sums: &BTreeMap<OptionId, u64> (sum of scores per option for this unit)

turnout: Turnout (ballots\_cast, invalid\_or\_blank, valid\_ballots)

params: &Params (uses score\_scale\_min/max, score\_normalization)

options: &[OptionItem] (to enforce canonical (order\_index, id) ordering)

Output:

UnitScores { unit\_id, turnout, scores: BTreeMap<OptionId, u64> }

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

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

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

rust

CopyEdit

use std::collections::BTreeMap;

use vm\_core::{

ids::{UnitId, OptionId},

entities::{Turnout, OptionItem},

variables::Params,

};

pub fn tabulate\_score(

unit\_id: UnitId,

score\_sums: &BTreeMap<OptionId, u64>,

turnout: Turnout,

params: &Params,

options: &[OptionItem],

) -> UnitScores;

fn canonicalize\_scores(

score\_sums: &BTreeMap<OptionId, u64>,

options: &[OptionItem],

) -> Result<BTreeMap<OptionId, u64>, TabError>;

fn check\_scale\_and\_caps(

scores: &BTreeMap<OptionId, u64>,

turnout: &Turnout,

params: &Params,

) -> Result<(), TabError>;

## **7) Algorithm Outline (implementation plan)**

**Canonical order**

Iterate options in (order\_index, OptionId) order, take score\_sums.get(&opt.id).copied().unwrap\_or(0), build fresh BTreeMap<OptionId,u64>.

Unknown options present in score\_sums ⇒ **error** (TabError::UnknownOption).

**Scale sanity**

Read min = VM-VAR-002, max = VM-VAR-003; ensure min < max. These are **inclusive** bounds per ballot.

The function **does not** reconstruct per-ballot scores; it only enforces **aggregate plausibility** given valid\_ballots.

**Caps / plausibility checks**

Let V = turnout.valid\_ballots.

If VM-VAR-004 = off: each option’s sum ≤ V \* max (since each counted ballot contributes at most max).

If VM-VAR-004 = linear (per-ballot normalization to span): aggregate sums are already normalized; the same cap ≤ V \* max still applies.

If V == 0: all option sums **must be 0**.

Negative values impossible (u64).

**Assemble**

Return UnitScores { unit\_id, turnout, scores }. No division or percentages here.

Note: If the data source is **raw ballots**, a separate helper (outside this file) must first compute score\_sums from per-ballot vectors respecting min/max and VM-VAR-004. This file’s function assumes we already have per-option sums.

## **8) State Flow**

Pipeline: **TABULATE (score)** → UnitScores → **ALLOCATE** (PR/WTA) → **AGGREGATE** → **GATES**.

Gates that need a binary “support %” for **score** ballots compute it elsewhere via  
 score\_sum\_for\_change / (max\_per\_ballot \* valid\_ballots) using integers.

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

Determinism via canonical option iteration and BTreeMap storage.

Integer math only; no rounding; no RNG.

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

Unknown OptionId in score\_sums ⇒ TabError::UnknownOption.

V=0 with any non-zero sum ⇒ TabError::InconsistentTurnout.

Any option sum > V \* max ⇒ TabError::OptionExceedsCap.

Overflow guard: compute V \* max in u128 then compare after cast to avoid u64 overflow on extreme inputs.

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

Happy path: scale [0,5], V=100, sums within caps → returns identical sums in canonical order.

Caps: with V=50, max=5, any option sum > 250 ⇒ **fail**.

V=0: all sums must be 0; any non-zero ⇒ **fail**.

Unknown option key present ⇒ **fail**.

Determinism: shuffle insertion order of score\_sums/options → identical UnitScores.scores and canonical bytes.