Atomic-Function Repair and Model-Diversity Recovery for Referential Scenario Mining

Technical Report — Argoverse 2 Scenario Mining Challenge,

CVPR 2026 Workshop on Autonomous Driving

Heejae Kim (Team Lilly)
Independent Researcher
happyhj@gmail.com

Abstract

We describe a solo submission to the Argoverse 2 Scenario Mining Challenge that ranks among the top submitted entries on both the Spatiotemporal (HOTA-Temporal) and Temporal (Timestamp Balanced Accuracy, TS-BA) tracks. Building on the RefAV referential-programming baseline—in which a large language model (LLM) compiles each natural-language scenario description into a program over a library of atomic functions*—we find that the single highest-return lever is not prompt engineering or model choice, but* ***repairing latent bugs in the baseline's own atomic-function substrate****: six high-impact implementation fixes (of nine bugs found) alone yield* $+0.8$ *percentage points (pp) on both metrics. On a corrected substrate we add (i) global-context code generation, (ii) a deterministic, ground-truth-free* ***model-diversity empty-recovery*** *scheme that composes three LLMs to fill complementary blind spots, (iii) vision-language-model (VLM)-conditioned score* reweighting *rather than hard filtering, and (iv) a precision-safe per-class mask for the negative-dominated TS-BA metric. The final pipeline reaches* ***HOTA-Temporal 32.51*** *and* ***TS-BA 72.70*** *on the test set, competitive with the strongest public entries on both tracks. We highlight a clean track-asymmetry result: identical recovery is* $+0.39$ *HOTA-Temporal but* -0.003 *TS-BA, showing that the two metrics reward opposite recovery aggressiveness. Orthogonally, accuracy-neutral execution-engine fixes run the* same *programs at byte-identical output while keeping the execution engine tractable within a 32 GB laptop-class budget (Sec. 8).*

1. Introduction

The RefAV benchmark [2], built on Argoverse 2 [6], frames scenario mining as *referential programming*: given a free-text description (*e.g.*, "vehicle changing lanes while at an intersection") and a log of tracked objects on an HD map, a system must identify the referred object(s) and the timestamps at which the scenario holds. The official baseline prompts a large language model (LLM) to emit Python that composes a fixed library of *atomic functions* (`changing_lanes`, `at_pedestrian_crossing`, `being_crossed_by`, `accelerating`, ...) over the perception tracks.

Two metrics are scored. **HOTA-Temporal** (Spatiotemporal track) is a compositional detection/association/temporal-IoU score, extending HOTA [4], that rewards recall and correct temporal extent. **Timestamp Balanced Accuracy** (TS-BA, Temporal track) is $(\mathrm{TPR} + \mathrm{TNR})/2$ over per-timestamp referred/not-referred labels; it is strongly negative-dominated, so false positives are punishing.

Our submission ranks among the top entries on both tracks. The contributions are:

1. **Atomic-function repair as a first-class lever** (Sec. 4). The baseline ships subtly-buggy building blocks; fixing them is higher-ROI than any prompt or model change and benefits *every* downstream generation.
2. **Model-diversity empty-recovery** (Sec. 5.2)—a deterministic, transferable rule that recovers descriptions in one model's blind spot using another model's program, grounded in a measured intersection blind-spot analysis.
3. **VLM-conditioned post-processing as reweighting** (Sec. 5.3)—using a vision-language model's per-pair presence judgement and grounded frames to *rescale* track and frame scores, avoiding the recall collapse of hard filtering.
4. **Track-aware tuning** (Sec. 6)—a single pipeline tuned in opposite directions for the two metrics, with an ablation showing why.
5. **An accuracy-neutral execution engine** (Sec. 8)—removing a super-linear per-call lane recomputation runs the *same* generated programs up to $96\times$ faster wall

/ $\sim$236$\times$ CPU on the worst-case query, with byte-identical output (7/7 queries), turning an intractable 2 h 42 m dense-log query into $\sim$1 min within a 32 GB laptop-class budget.

Hardware, stated precisely. The LLM code generation and the execution engine run on consumer-class hardware—the engine is verified within a 32 GB laptop-class RAM budget (Sec. 8; peak RSS 0.93 GB). The one-time Qwen3-VL-32B-Instruct inference does not fit the 8 GB VRAM of our development laptop's RTX 3070 Ti; it ran *once* on a rented RTX PRO 6000 and is excluded from the efficiency budget.

2. Related Work

The RefAV benchmark and baseline. RefAV [2] casts scenario mining as *referential programming* over Argoverse 2 [6]: $\sim$10k natural-language queries over 1k driving logs, scored by HOTA-Temporal (extending HOTA [4]) and Timestamp Balanced Accuracy. Its baseline prompts an LLM to compose a fixed library of atomic functions; that baseline—substrate and all—is precisely what we build on, and our first contribution is the observation that the substrate itself is the highest-return place to intervene (Sec. 4).

Prior challenge entries. Eight teams entered the inaugural challenge at CVPR 2025. The winning system, SM-Agent (Zeekr UMCV) [7], pairs *global context-aware code generation* with a *multi-agent refinement* loop on a Gemini-2.5-Pro baseline; Chen and Greer [1] combine iterative error correction with spatially-aware prompting on the same baseline. We adopt the former's two ideas as front-end components (Sec. 5.1)—they are prior work, not our contribution—and our novel levers (atomic-function repair, model-diversity recovery, VLM-conditioned reweighting, and track-aware tuning) are orthogonal to and stack on top of them. We note that the 2025 and 2026 challenge editions use different data/perception inputs and leaderboards, so absolute scores are *not* comparable across years; all comparisons in this report are within the 2026 leaderboard.

3. Pipeline Overview

The two tracks share a substrate and a VLM cache but use *different* node graphs; Fig. 1 gives the exact, nothing-omitted sequence that produced each final submission, and the rest of this section explains each node. The shared front end is: repair the atomic-function library (Sec. 4); generate one program per description from a single global-context LLM call at temperature 0 (Sec. 5.1); execute each program against per-log perception tracks to get referred-object tracks with per-timestamp labels; and obtain a per-(log, description) presence judgement plus grounded frames from a Qwen3-VL-32B-Instruct model [5].

The two tracks then *diverge*. The Spatiotemporal (HOTA-Temporal) pipeline recovers *first*—an ungated 3-tier empty-recovery (Sec. 5.2)—then applies VLM post-processing (`track_reassign` then `score_recalibration`, Sec. 5.3) to the merged set, and submits with *no* mask. The Temporal (TS-BA) pipeline instead post-processes each model independently, then performs a *VLM-gated* 1-tier recovery, then applies the per-class GT_NONE mask (Sec. 5.4)—the order and the gating both follow from the track asymmetry (Sec. 6).

4. Layer 1: Atomic-Function Repair

We triaged the lowest-scoring (log, query) pairs by replaying their predictions in EGOLENS [3], a browser-based autonomous-driving dataset viewer, inspecting each scene directly (object density, agent counts, per-scene difficulty) to decide where to focus. This visual review, together with a code review of the baseline's atomic-function library, surfaced nine implementation bugs and nine documentation/specification misalignments; the same inspection flagged the unusually object-dense logs whose per-call cost later motivated the execution-engine profiling of Sec. 8. These were not edge cases: **89 of 403 queries (22%) invoked an affected function, and their mean HOTA-Temporal was 0.146 versus 0.229 overall, with 38 queries scoring exactly zero.** Below we detail the **six highest-impact implementation fixes** (three critical, three significant); these six alone produce the headline $+0.8$ pp gain. The remaining three implementation bugs and the nine documentation/specification fixes are folded into the "full bug-fix set" (Tab. 3), which adds a further $+0.1$ pp.

Three critical bugs. Each is a one- or two-line defect in the baseline that silently corrupts a whole query family; the before/after is shown in Fig. 2.

(a) `being_crossed_by`. The function's `direction` argument selects which half-midplane axis defines a crossing. Inside the candidate loop the baseline computed the local crossing sign into a variable *also* named `direction`, overwriting the argument; from the second candidate onward every test used ± 1 in place of the requested axis, so all but the first candidate were scored in the wrong frame. *Fix:* rename the local to `crossing_sign` ($\sim$20 crossing/overtaking/passing queries).

(b) `at_pedestrian_crossing`. The set of crossings being iterated, `ped_crossings`, was reassigned *inside* its own `for`-loop from a stale per-track query, mutating the iterable mid-loop so each timestamp reused the previous timestamp's crossings. *Fix:* delete the in-loop reassignment ($\sim$15 crosswalk queries).

(c) `changing_lanes`. The left-lane-change branch indexed the boundary tangent with `min(0, idx-1)`,

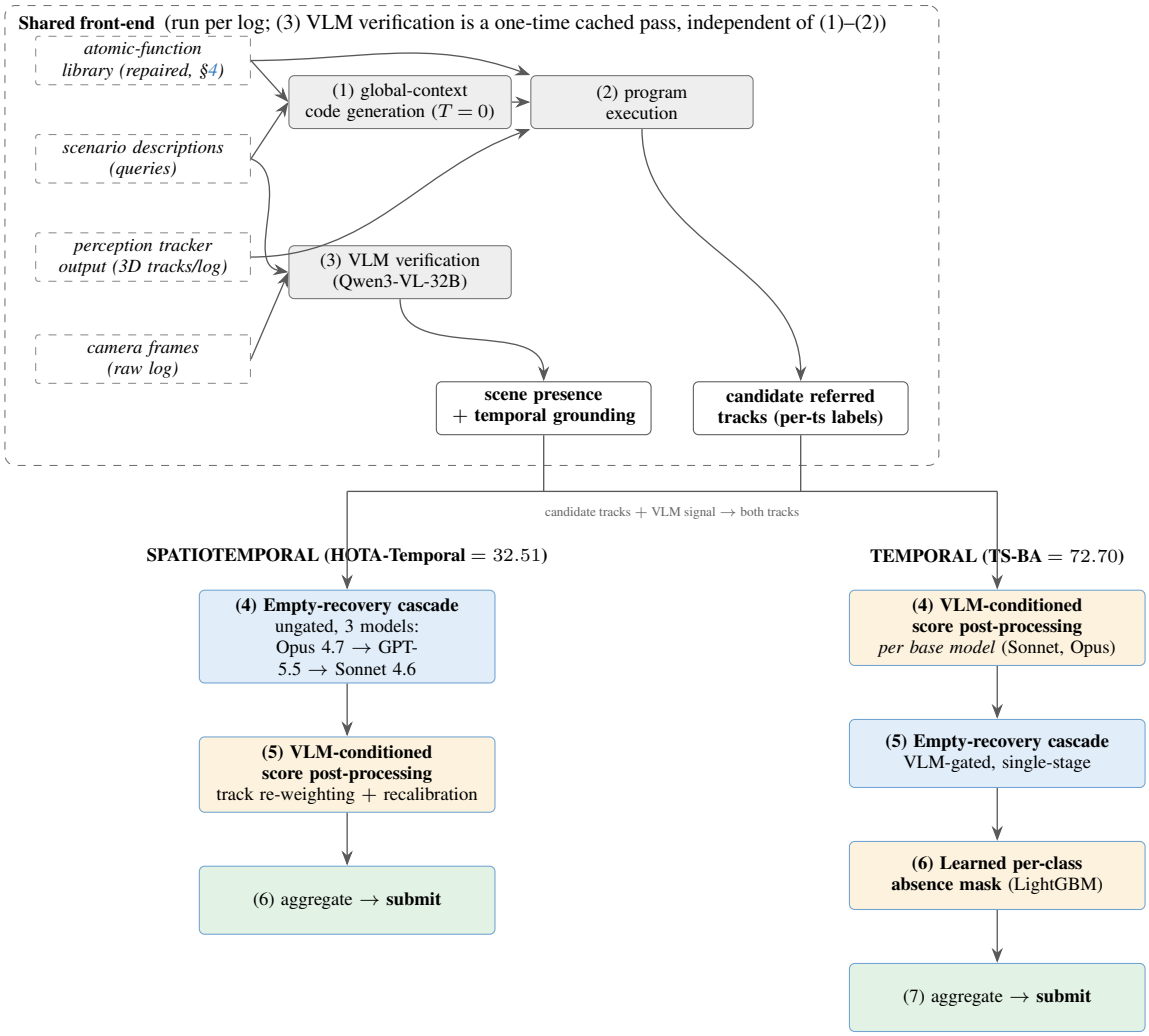

Figure 1. End-to-end computation graph. The shared front-end (dashed) makes the inputs explicit: scenario descriptions feed both code generation (1) and VLM verification (3); the repaired atomic-function library feeds both generation (1) and execution (2); the perception tracker output feeds execution (2); and raw camera frames feed VLM verification (3). The front-end yields two artifacts—candidate referred tracks and the VLM presence/temporal-grounding signal—both of which feed each per-track stack. The two tracks then compose the *same* modules (empty-recovery cascade, VLM-conditioned score post-processing, learned per-class absence mask) in a *different order, placement, and tuning*; hyperparameters are in Tab. 1 and Sec. 5.3.

which is identically 0 for all non-negative `idx`; the lane direction was therefore always taken from the lane *start* rather than the local tangent, and every left lane-change was misoriented. *Fix:* `min`→`max` (the right branch already used `max`).

Three significant bugs. An epsilon added to a *vector* rather than its norm (`norm(vec + 1e-8)`, five sites, perturbing directions); a missing divide-by-zero guard in velocity normalization; and left-turn detection left commented-out as dead code in `turning`.

Verification and effect. Each fix was validated *end-to-end* rather than in isolation: because all programs are emitted from one global-context generation (Sec. 5.1), a library edit perturbs more than half of the emitted programs (the "butterfly effect"), so a unit test on one function does not predict the system-level effect. We therefore re-ran the full pipeline on the validation split after each change and read the official metrics. The critical fixes took HOTA-Temporal $22.9 \rightarrow 23.6$ and TS-BA $67.5 \rightarrow 68.2$; the full set reached $23.7/68.3$—a combined $+0.8$ **pp on both metrics** from the substrate alone, the most reliable single gain in the project. As an independent check, the 38 queries that scored exactly zero before the fixes and the affected-function cohort (mean

```
(a) being_crossed_by      # direction = axis arg
-  direction = (y1-y0)/abs(y1-y0)
-  if direction*pos[i,1] < lat_thresh: ...
+  sign = (y1-y0)/abs(y1-y0)
+  if sign*pos[i,1] < lat_thresh: ...

(b) at_pedestrian_crossing
   for pc in ped_crossings:
-      ped_crossings = get_ped_cross(avm, poly)
       if overlaps(track_poly, pc): ...

(c) changing_lanes        # left-change branch
-  start_idx = min(0, idx-1)   # always 0
+  start_idx = max(0, idx-1)   # local tangent
```

Figure 2. The three critical atomic-function bugs (official baseline → ours). The red "−" lines are the released baseline's own code (last maintainer-authored revision of the atomic-function library); each is a 1–2 line defect that mis-scores an entire query family.

HOTA-Temporal 0.146 vs. 0.229 overall) both moved in the expected direction.

New atomic functions. We added `followed_by`, `speed_relative_to`, `reversing`, and `braking`. `speed_relative_to` alone drove **HOTA-Temporal** $22.9 \rightarrow 25.5$ **(+2.6 pp)**, the largest single jump. Notably, `followed_by` *hurt* and was removed—the same butterfly effect means an addition that helps in isolation can still degrade the ensemble of generated programs, so additions must be validated end-to-end. We also switched generation to temperature 0; prior temperature-0.5 runs carried ± 1 pp noise that masked small effects. The resulting deterministic configuration (HOTA-Temporal 25.5, HOTA-Track 35.8, TS-BA 67.9) is the base for all downstream work.

5. Layer 2: Generation, Recovery, Conditioning

5.1. Global-context code generation

Following the 2025 winning system [7], and *not* claimed as a contribution here, we adopt two front-end ideas: rather than prompting per description, we issue a single call containing the full library and every description (encouraging cross-query consistency in how shared concepts—*e.g.*, "active", "approaching"—are compiled), and a multi-agent refinement pass improves validity. Our only addition at this stage is a hardened prompt targeting recurring *Opus-4.7-specific* misuse modes we observed (referred-object extraction, using `accelerating(min_accel=`$-\infty$`)` for braking, constraint completeness, and a `reverse_relationship` operator-precedence pitfall).

Prompt composition (for reproducibility). The single generation call concatenates, in order: (1) a task framing ("find the referred objects; be precise; avoid false positives"); (2) the *full* atomic-function library (signatures + docstrings); (3) the AV2 object categories; (4) a few-shot block of worked example programs; (5) an output protocol—one `### [i] description` header and one `python` block per query, each self-contained, no imports or helper definitions, and each *required* to end in an `output_scenario()` call; (6) a "common-patterns" cheat-sheet of canonical encodings (parked [`stationary`] vs. momentarily stopped [`has_velocity`], reversing, braking via signed `accelerating` bounds, referred-object extraction with `get_`/`has_objects_in_relative_direction`, relative-direction semantics, and a note that `scenario_and` is simultaneous not sequential); and (7) *all* N descriptions enumerated, with a closing instruction to keep similar scenarios consistent. For the Opus primary we append a model-specific hardening addendum that (i) lists the exactly twelve binary-relation functions `reverse_relationship` may wrap and forbids wrapping set/compose/unary operators, (ii) gives the `has_`→`get_` recipe for emitting the subject noun rather than the anchor/ego, (iii) requires both bounds on `accelerating`, and (iv) demands every description constraint be encoded. The exact prompt strings are released with the code.

5.2. Model-diversity empty-recovery

Different frontier models fail on *different* descriptions. A per-query analysis (83 GPT-5.5 wins vs. 99 Opus wins and 224 ties on validation—the models genuinely disagree, not noise) showed that four of the top-five GPT-5.5 wins lie in the **Opus ∩ Sonnet blind spot** (*e.g.*, "vehicle passing ego on a one-way road", which both Opus and Sonnet score 0 but GPT-5.5 composes correctly). We exploit this with a **deterministic, ground-truth-free recovery rule** (a deferral cascade; cf. Fig. 3): for each (log, description), if the primary model's prediction is *empty* (no timestamp has a referred label), fall back to the next model in a fixed order, else emit empty. For the Spatiotemporal track the order is **Opus 4.7** → **GPT-5.5** → **Sonnet 4.6**. Because the rule references only emptiness—never ground truth—it transfers from validation to test. On test it broke a long-standing "HOTA-Temporal 30 wall": 2-tier recovery reached 32.12 and 3-tier reached 32.51.

5.3. VLM-conditioned post-processing

An early experiment with hard CLIP-based temporal filtering collapsed recall and was abandoned. Instead we use the VLM signal to *rescale* scores.

The problem `track_reassign` solves. The symbolic program labels *every* track that satisfies its predicate as RE-

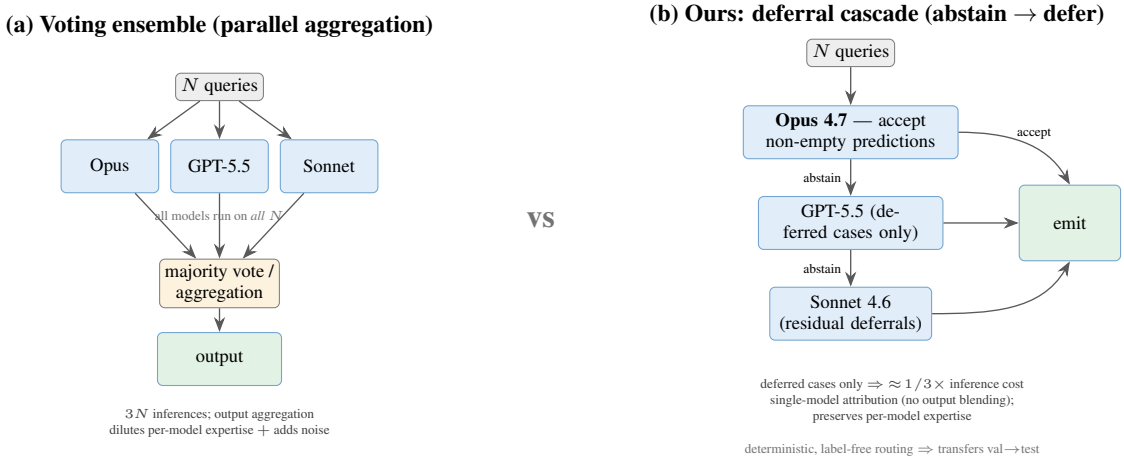

Figure 3. Our recovery is a deferral cascade, not a voting ensemble. **(a)** A majority-vote ensemble runs *every* model on *every* query and aggregates their outputs ($3N$ inferences); aggregating disagreeing models dilutes each model's expertise and injects noise. **(b)** We accept the primary model's non-empty predictions and *defer only the abstained cases* (empty prediction / no grounded frame) to the next model in a fixed order (Opus 4.7 → GPT-5.5 → Sonnet 4.6). Because later models run only on the shrinking set of deferrals, total inference is $\approx 1/3$ of running all three on every query; and because every accepted prediction comes from a single model (no output blending), per-model expertise is preserved. Routing depends on abstention alone—never on ground-truth labels—so it transfers from validation to test, breaking the HOTA-Temporal "30 wall" ($30.03 \rightarrow 32.12 \rightarrow 32.51$).

FERRED. Error analysis showed this fails in two systematic ways. (i) *Whole-log false positives*: in logs where the scenario never occurs, noisy perception still produces tracks that trip the predicate, emitting REFERRED mass that TS-BA punishes and that pollutes HOTA detection. (ii) *Referred-mass dilution*: in logs where the scenario *does* occur, several competing tracks share the REFERRED label, splitting HOTA's association score (AssA) instead of concentrating it on the one correct object. The symbolic layer has no signal to break these ties; the VLM does—it reports per-log presence and grounds *which* frames the scenario appears in.

How `track_reassign` works. For each $(\log, \text{description})$ it branches on the VLM judgement. **VLM No** $\Rightarrow$ multiply all REFERRED scores by `no_mult` $= 0.2$, strongly suppressing whole-log false positives without a hard delete. **VLM Yes** $\Rightarrow$ find the *dominant* track—the REFERRED `track_id` with the highest score inside grounded frames—and boost it ($\times$`boost_mult` $= 2.0$) while demoting the competing REFERRED tracks ($\times$`other_mult` $= 0.3$), collapsing the diluted mass onto a single object and directly raising AssA. **VLM Yes but no REFERRED track lands in the grounded frames** $\Rightarrow$ a milder boost ($\times$`no_track_mult` $= 2.0$) of REFERRED scores at the grounded timestamps, recovering temporal extent without inventing a track. VLM errors/missing pairs are left untouched. Because every branch *reweights* rather than deletes, recall is preserved—the failure mode that sank hard CLIP filtering. `score_recalibration` multiplies each per-timestamp score by α when the timestamp lies inside a VLM-grounded frame and by β otherwise; pairs the VLM marks absent (or that error) are left unchanged, never zeroed, which is what keeps recall from collapsing.

The ± 300 ms tolerance band. The VLM grounds presence at *camera-frame* indices, which we map back to timestamps; these do not line up exactly with the 10 Hz LiDAR-track timestamps, and the VLM's frame localization is itself coarse. A timestamp is therefore counted as "inside" a grounded frame if it falls within 300 ms ($|t - t_{\text{vlm}}| < 3 \times 10^8$ ns) of any grounded frame; the band was set to roughly one camera-keyframe spacing and is the smallest value that did not erode recall in the symmetric-erosion ablation (Sec. 7).

Tuning α, β. (α, β) were chosen by a grid sweep on validation, independently per track, optimizing that track's own metric. The two optima are *opposite*: HOTA-Temporal prefers a mild reweight ($\alpha = 1.0, \beta = 0.7$) that trims out-of-frame mass while preserving recall, whereas TS-BA—which punishes false-positive timestamps—prefers an aggressive ($\alpha = 3.0, \beta = 0.5$) that sharply concentrates score inside grounded frames (Sec. 6).

5.4. Per-class GT_NONE masking (TS-BA)

For the negative-dominated TS-BA metric we train a gradient-boosted classifier (LightGBM, 300 trees, learning rate 0.05, 31 leaves) that predicts, per (log, description) pair, whether the ground truth contains *any* REFERRED frame (target 1) or is entirely GT_NONE (target 0). The $\sim$24 features are cheap and label-free: pipeline outputs (max referred score, referred/other/related counts, active-frame ratio), the VLM signal (presence, grounded-frame count and ratio, error flag), prompt cues (length, negation, "at least"/multiplicity, proximity, maneuver keywords), and class-availability features that cross the classes named in the prompt against the tracker's per-log class counts (how many named classes are absent from the log).

Calibration and operating point. The classifier is trained and calibrated *entirely on validation* (no test labels): we obtain out-of-fold (OOF) probabilities via 5-fold stratified cross-validation, then sweep the demotion threshold on those OOF predictions. Rather than one global cutoff we set a *per-class* threshold equal to the loosest value that still gave 100% OOF precision for that class—SCHOOL_BUS 0.50, WHEELCHAIR 0.70, CONSTRUCTION 0.20, WEATHER 0.10, BUS 0.05—so a REFERRED→OTHER demotion fires only where validation showed zero true-positive loss (16 demoted scenarios total). The final model is retrained on the full validation set and applied unchanged to test, where it added +0.29 TS-BA over the unmasked stack; loosening the thresholds further over-pruned and regressed, confirming the precision-first operating point.

6. Track-Aware Tuning and Recovery Asymmetry

The two metrics reward opposite behaviour, and our single pipeline is tuned in opposite directions (Tab. 1). The cleanest evidence is one ablation: the **ungated 3-tier recovery that gains** +0.39 **HOTA-Temporal loses** −0.003 **TS-BA** on validation. Because TS-BA is $(\mathrm{TPR} + \mathrm{TNR})/2$ and dominated by negatives, ungated over-recovery introduces false-positive timestamps whose TNR cost outweighs the TPR gain—whereas HOTA-Temporal tolerates false positives for recall. The TS-BA track therefore uses VLM-*gated* recovery (recover only when the VLM confirms presence) plus the precision-safe mask.

7. Results

Final test scores (EvalAI challenge 2662) are shown in Tab. 2; the stage-by-stage progression is in Tab. 3.

	HOTA-Temporal	TS-BA
Recovery	broad (ungated)	VLM-gated
`score_recalibration`	$\alpha=1.0, \beta=0.7$	$\alpha=3.0, \beta=0.5$
Per-class mask	—	yes

Table 1. The same pipeline, tuned oppositely per track.

Track	Metric	Baseline	Ours	Δ
Spatiotemporal	HOTA-Temporal	26.27	**32.51**	+6.24
Temporal	TS-BA	68.07	**72.70**	+4.63

Table 2. Final test results against the official RefAV (RefProg) baseline. The full pipeline improves HOTA-Temporal by +6.24 and TS-BA by +4.63 over the released baseline. (We compare against the fixed official baseline rather than the public leaderboard, whose standings change up to the deadline.)

Stage	HOTA-T	TS-BA
LLM baseline (single-call)	22.1	67.2
+ global context	22.9	67.5
+ critical bug fixes	**23.6**	**68.2**
+ full bug-fix set	23.7	68.3
+ `speed_relative_to`	**25.5**	67.9
+ reassign + recal (*test*)	30.03	72.33
+ mask (*test*, TS-BA)	—	72.62
+ 2-tier recovery (*test*)	32.12	—
+ 3-tier (*test*)	**32.51**	—
+ gated recovery+mask (*test*)	—	**72.70**

Table 3. Stage-by-stage progression (validation unless tagged *test*).

7.1. Ablations and negative results

We report negatives because several "obvious" levers failed informatively. **Best coding model ≠ best codegen**: GPT-5.5 as the *primary* generator lost to Opus on HOTA-Temporal (−0.0018 val) and badly on TS-BA (−0.07); its heavier programs caught objects slightly better but mistimed them, so it is valuable only as a *diversity* tier (Sec. 5.2). **DPO fine-tuning** of the VLM regressed test HOTA-Temporal (−0.23). A **tracker swap** collapsed to 25.3. **Hard CLIP filtering** and **symmetric window erosion** both cratered recall, as ground-truth-active frames sit at/inside our window ends. Finally, **validation→test transfer is unreliable**: every validation-only gain we chased regressed on test; only the emptiness-based recovery rule transferred. This shaped our methodology of pre-registered validation gates with strict noise thresholds before any test submission.

7.2. Reproducibility

The pipeline is deterministic (temperature 0, fixed recovery order, fixed post-processing parameters) given the cached VLM judgements. The released repository[1] provides exact commands to regenerate both test cards from the code, generated programs, and VLM cache.

8. Execution Efficiency

Efficiency here is not a speed/accuracy trade-off but the removal of an *architectural anti-pattern*: the baseline engine recomputes static lane geometry on *every* predicate call, so its cost scales with query density rather than with the map. We identify this redundant-recomputation pattern and eliminate it—loading the map once and reusing it in-process—a change that is **output-preserving by construction** and generalizes beyond this benchmark to any per-call recomputation of immutable scene structure.

Concretely, an accuracy-neutral set of execution-engine fixes runs the *same* generated programs substantially faster than the organizers' baseline engine, on the same logs and hardware, **without changing the output**. The optimizations are confined to the engine (batched cuboid construction, ego-pose memoization, annotation caching, in-process lane reuse, unscored-track stripping), so we hold **code generation frozen**—replaying identical `claude-sonnet-4-6_global` programs in both arms—and measure only the deterministic engine. Two git checkouts—the pristine challenge-release baseline (`d7f8077`, 2026-02-17) vs. ours—run each (log, query) pair in a fresh process from a cold cache on the same container, **cgroup-capped at 32 GB / 4 cores**, so completion is itself a laptop-class feasibility proof. VLM inference is a separate one-time cached pass and is excluded. We benchmark against the challenge-release baseline; a later organizer commit (`5c5be64`, 2026-05-04) parallelizes the *object-color* SigLIP inference across GPUs but does not touch the lane-geometry recomputation path that our in-process lane reuse addresses, so it is orthogonal to the speedup reported here.

Metric note (host contention). The benchmark box is a shared host under heavy neighbour load, so wall-clock overstates compute time. We therefore report **CPU-seconds** as the contention-robust primary metric, with wall-clock alongside; the A/B ratio is fair because both arms run sequentially on the same box. (`getrusage(CHILDREN)` misses terminated worker-pool processes—reading only 10 s for the baseline—so CPU-seconds are sampled across the whole process tree from `/proc`.)

[1] https://github.com/happyhj/refav-team-lilly, commit `30a9c99`.

Arm	wall (s)	CPU (s)	peak RSS
Baseline (n=2)	9720	5187	934.9 MB
Ours (n=2)	101	22	926.6 MB
Speedup	**96.1**×	∼236×	comparable

Table 4. Headline lane-sharing query (dense log), frozen programs, cold cache. Output is byte-identical between arms.

Headline (real completion, not a timeout). On a dense log, the lane-sharing query `in_same_lane(vehicles, vehicles)`—the worst case, where the baseline recomputes lane geometry per call—takes baseline **2 h 42 m** (median 9720 s wall over n=2, both to completion) versus ours ∼101 **s**, a **96.1**× wall speedup; in CPU-seconds the gap is ∼236× (baseline 5187 s vs. ours ∼22 s), because the baseline is CPU-bound on per-call lane recomputation while ours, after a single cached map load, is I/O-bound (Tab. 4). Both arms run under the *same* 4-core / 32 GB cap, so this CPU-second gap is not a core-count artifact: in-process lane reuse *eliminates* the per-call recomputation rather than parallelizing or deferring it, so the work disappears (hence ours is I/O-bound after a single map load). The baseline's CPU-seconds fall below its wall-clock only because the shared host inflates wall time under neighbour load—precisely why we report CPU-seconds as the primary, contention-robust metric.

Distribution. A 6-pair lighter-query A/B shows the speedup is concentrated exactly where the lane-reuse optimization applies: lane-relational queries speed up **1.64–8.43**× on light logs and scale to the ∼236× dense worst case, while pure-kinematic queries sit at **parity** (∼1.0–1.14×)—the reuse is a no-op there, never a regression. The benefit grows with (lane-dependence × log density); we report it as a range, not a single number.

Accuracy preservation. The speedup is not an accuracy regression: on the headline query the two engines' scored predictions are **bit-identical** (track-id sets and per-timestamp labels exact, scores/geometry within atol=10^{-6}; both pkls 952,849 B; parity PASS). Across the headline plus all 6 distribution queries, **7/7 output pkls are byte-identical**. The one optimization that *would* have changed results—a spatial pre-filter that failed 2/10 queries—was rejected; only semantically-preserving speedups shipped.

An iso-accuracy efficiency protocol. Beyond the numbers, the measurement procedure is itself a contribution. Efficiency is only meaningful at *iso-accuracy*, so we gate every speedup behind a byte-identical output-parity check—a lossy spatial pre-filter that failed 2/10 queries was rejected on exactly this ground—report **CPU-seconds sampled**

across the whole process tree rather than `getrusage` (which silently undercounts terminated pool workers), and require real completion rather than a self-chosen timeout. This parity-gated, contention-robust, completion-based protocol is reusable for any future efficiency track and resists the standard failure mode of speed leaderboards: rewarding fast-but-wrong shortcuts.

9. Conclusion

On the RefAV referential-programming baseline, the highest-return lever was not the model or the prompt but the *substrate*: repairing latent bugs in the baseline's own atomic-function library yielded $+0.8$ pp on both metrics and benefited every downstream generation. On the corrected substrate, a deterministic, ground-truth-free model-diversity recovery rule broke a long-standing HOTA-Temporal wall by composing three LLMs' complementary blind spots, and VLM-conditioned *reweighting* (rather than hard filtering) avoided recall collapse while still suppressing false positives. A single pipeline, tuned in opposite directions for the two metrics, ranks among the top entries on both the Spatiotemporal and Temporal tracks—reaching HOTA-Temporal 32.51 and TS-BA 72.70 on test. Finally, an accuracy-neutral execution engine—byte-identical to the baseline on 7/7 benchmarked queries—makes the setup not just competitive but *efficient*, collapsing a 2 h 42 m worst-case query to $\sim$1 min within a 32 GB laptop-class budget.

Two methodological lessons generalize beyond this challenge. First, in LLM-as-compiler systems the library of callable primitives is a first-class optimization target: subtle bugs there are amplified across every generated program. Second, validation$\rightarrow$test transfer is fragile when gains are tuned against a held-out split; only rules grounded in metric-invariant signals (here, prediction emptiness) transferred reliably, which argues for pre-registered validation gates before any test submission.

References

[1] Yifei Chen and Ross Greer. Technical report for Argoverse2 scenario mining challenges on iterative error correction and spatially-aware prompting. *arXiv preprint arXiv:2506.11124*, 2025. 2

[2] Cainan Davidson, Deva Ramanan, and Neehar Peri. RefAV: Towards planning-centric scenario mining. *arXiv preprint arXiv:2505.20981*, 2025. 1, 2

[3] Heejae Kim. EgoLens: A browser-based viewer for autonomous-driving datasets. Zenodo, 2026. doi:10.5281/zenodo.20460189. 2

[4] Jonathon Luiten, Aljoša Ošep, Patrick Dendorfer, Philip Torr, Andreas Geiger, Laura Leal-Taixé, and Bastian Leibe. HOTA: A higher order metric for evaluating multi-object tracking. *IJCV*, 129(2):548–578, 2021. 1, 2

[5] Qwen Team. Qwen3-VL: Vision-language models. https://github.com/QwenLM/Qwen3-VL, 2025. 2

[6] Benjamin Wilson, William Qi, Tanmay Agarwal, John Lambert, Jagjeet Singh, Siddhesh Khandelwal, Bowen Pan, Ratnesh Kumar, Andrew Hartnett, Jhony Kaesemodel Pontes, Deva Ramanan, Peter Carr, and James Hays. Argoverse 2: Next generation datasets for self-driving perception and forecasting. In *Proc. Neural Inf. Process. Syst. (NeurIPS) Datasets and Benchmarks Track*, 2021. 1, 2

[7] Zeekr UMCV Team. SM-Agent solution for the AV2 2025 scenario mining challenge. https://www.neeharperi.com/files/zeekr_umcv_techreport_cvprw25.pdf, 2025. 1st place, CVPR 2025 WAD Argoverse 2 Scenario Mining Challenge (HOTA-Temporal 53.38). 2, 4