

TARO v11 Architecture Plan: Learning-Augmented Offline Builder

Status: Proposed

Date: 2026-02-07

Scope: Offline learning module and compile pipeline only (runtime remains deterministic)

1. Objective

v11 introduces a learnable offline refinement module that uses temporal interaction patterns to improve TARO models before serialization, while preserving v10.1 runtime contracts:

- deterministic query behavior under fixed model + overlay snapshot
- strict FIFO correctness gates
- immutable runtime data structures (`EdgeGraph`, `ProfileStore`, `SpatialRuntime`, `LiveOverlay`)

The design takes ideas from TGSL (Time-aware Graph Structure Learning via Sequence Prediction on Temporal Graphs) and adapts them to routing constraints rather than temporal link prediction alone.

2. Why v11 Is Needed

Current TARO strengths:
- strong runtime contracts and safety checks
- explicit temporal profile handling
- robust spatial and overlay components

Current gap:
- offline pipeline lacks a structured learning loop to repair missing temporal structure and noisy profile curves from real telemetry.

v11 fills this gap without moving stochastic behavior into runtime.

3. Non-Negotiable Contracts (Carried Forward)

1. Runtime does not run neural models.
2. Runtime accepts only compiled model artifacts.
3. Any learned output must pass schema, physics, and FIFO gates.
4. Exported artifacts must be reproducible from pinned data + config + seed.
5. A *admissibility* and A/Dijkstra parity must not regress.

4. Core Design: Two-Plane Architecture

4.1 Plane A: Learning Builder (Offline)

New module family in Python builder stack: - temporal encoder - context predictor - candidate generator - scorer/selector - profile refiner - safety/calibration/export pipeline

This plane can use TGSL-style sequence learning and candidate selection.

4.2 Plane B: Runtime Engine (Online)

Java runtime consumes only deterministic outputs: - refined per-edge profile IDs / bucket curves - optional validated structural additions (feature-flagged) - unchanged query semantics and contracts

No gumbel sampling or gradient logic exists in runtime.

5. TGSL-to-TARO Component Mapping

TGSL concept	TARO v11 equivalent	TARO-specific adaptation
Edge-centric temporal encoder	Edge-time representation extractor over road graph + telemetry	Must encode turn legality and directionality
RNN context predictor	Sequence model over recent traversals by edge and timestamp	Sequence keyed by origin corridor / OD neighborhood
Candidate edge sampling	Candidate refinement proposals (profile shifts, optional connectors)	Constrained by map legality and topology invariants
Gumbel-Top-K selection	Training-time exploration only	Export uses deterministic top-K by calibrated score
Joint training on original + augmented graphs	Multi-objective learning for ETA and route quality	Losses include ETA error, calibration, and constraint penalties

6. v11 Refinement Targets (Priority Order)

1. Profile-first refinement (default path)

Learn better temporal multipliers/buckets for existing edges.

2. Connector-edge proposal (opt-in)

Add candidate edges only when physically/legally valid and beneficial.

3. Edge suppression/removal (research mode)

Allowed only with strict guardrails; default disabled for production.

Default production posture in v11: profile refinement enabled, structural mutation restricted.

7. Offline Pipeline Stages (v11)

1. Ingestion and normalization
Inputs: base map, historical speeds, live telemetry, incident logs.
2. Sequence dataset construction
Build temporal sequences per edge/corridor/time-of-week.
3. Representation learning
Train edge/context encoders with fixed reproducible seeds.
4. Candidate proposal and scoring
Generate profile and optional structural candidates.
5. Constraint-aware selection
Apply deterministic score ranking + hard safety constraints.
6. Calibration and uncertainty handling
Keep only confidence-qualified updates.
7. Compile and serialize
Emit `.taro` with lineage metadata and validation reports.

8. Safety and Constraint Layer

Every selected refinement passes:

- schema integrity checks
- physical feasibility checks
- profile monotonic/FIFO checks
- route-level parity checks (A* vs Dijkstra on pinned seeds)
- regression guards (latency, memory, determinism)

Rejected candidates are logged with reason codes for auditability.

9. Data and Artifact Contracts

Required builder artifacts: - `dataset_manifest.json` (sources, date ranges, filters) - `learning_config.yaml` (model class, seed, thresholds) - `candidate_report.parquet` (proposal-level scores and reasons) - `refinement_decisions.parquet` (accepted/rejected with reasons) - `validation_report.json` (gates, metrics, failures)

Model metadata lineage additions (backward-compatible optional fields): - `learning_module_version` - `learning_seed` - `training_data_window` - `refinement_policy_hash` - `validation_report_hash`

10. Evaluation Criteria

Primary: - ETA MAE/MAPE reduction - route regret reduction vs held-out observed trips - calibration improvement (predicted vs actual travel time)

Safety: - zero FIFO violations after compile - zero determinism regressions - A*/Dijkstra parity unchanged

Systems: - bounded builder runtime - bounded model size growth - runtime p95 latency non-regression

11. Risks and Mitigations

1. Overfitting to sparse telemetry

Mitigation: confidence gating + fallback to base profile.

2. Noisy structural additions

Mitigation: keep structure learning opt-in and heavily constrained.

3. Drift between training and serving distribution

Mitigation: lineage metadata + retraining window controls.

4. Runtime contract erosion

Mitigation: keep learning strictly offline and enforce parity tests.

12. Recommended v11 Rollout

1. Implement profile-only learning path first.

2. Ship behind compile-time feature flags.

3. Add connector-edge path only after stable profile gains.

4. Keep edge drop/removal as research-only until validated.

This gives measurable quality gain while protecting the current production-safe runtime architecture.