TARO v11 Implementation Guide: Offline Learning Integration

Date: 2026-02-07
Companion: `docs/taro_v11_architecture_plan.md`

1. Implementation Goal

Implement a learning-augmented offline builder that improves temporal routing quality while keeping runtime deterministic and contract-safe.

2. Proposed Repository Layout (Incremental)

Add new Python builder packages (suggested):

```
src/main/python/learning/
  datasets/
  encoders/
  candidates/
  selectors/
  calibration/
  export/
  validation/
```

Add tests:

```
src/main/python/tests/learning/
```

No runtime stochastic logic is added under `src/main/java/`.

3. Phase Plan

Phase 0: Foundation and Contracts

Deliverables: - `learning_config` schema - dataset manifest schema - deterministic seeding utility - artifact directory conventions

Acceptance: - same input + seed yields identical candidate ranking - all artifacts include model/data lineage headers

Phase 1: Profile Refinement MVP

Deliverables: - sequence dataset builder from telemetry - temporal context model (RNN/LSTM baseline) - profile multiplier/bucket proposal generator - deterministic selector (top-K with confidence gate)

Acceptance: - offline validation shows ETA metric improvement - no FIFO violations after compile - no runtime API/contract changes needed

Phase 2: Constraint and Calibration Layer

Deliverables: - confidence calibration (e.g., temperature/isotonic) - rejection policy for low-confidence proposals - detailed rejection reason taxonomy

Acceptance: - calibration error improves on validation split - accepted updates remain within bounded risk thresholds

Phase 3: Structural Proposals (Opt-In)

Deliverables: - topology-constrained connector proposal generator - legal-turn, directionality, and geometry checks - impact estimator for route-quality gain

Acceptance: - structure proposals pass graph integrity checks - model size and runtime latency remain within limits

Phase 4: CI and Regression Gates

Deliverables: - parity harness (A* vs Dijkstra) - determinism replay test - model-size and loader smoke checks

Acceptance: - gates block unsafe model export automatically

4. TGSL Adaptation Rules (Practical)

1. Use TGSL sequence/context ideas for offline candidate generation.
2. Keep Gumbel-based exploration only in training experiments.
3. Export deterministic ranked outputs only.
4. Prefer profile refinement over graph mutation by default.

5. Data Preparation Requirements

Minimum columns for sequence construction: - `edge_id` - `timestamp_ticks` - `travel_time` - `speed_factor_observed` (if available) - optional `from_edge_id` for transition context

Derived features: - day-of-week - time-of-day bucket - lagged edge travel-time deltas - corridor/locality embeddings (if available)

6. Training and Selection Workflow

1. Build sequences from historical interactions.
2. Train temporal encoder/context predictor.
3. Generate candidate refinements per edge/profile.
4. Score with supervised objectives.
5. Apply deterministic selection and confidence threshold.

6. Run safety gates and compile only accepted decisions.

7. Validation Matrix

Quality: - ETA MAE/MAPE - route-time regret - calibration quality

Correctness: - FIFO validity - deterministic output reproducibility - A*/Dijkstra parity

Systems: - compile time budget - serialized model size growth cap - runtime p95 latency non-regression

8. Configuration Surface (v11)

Required: - `learning.enabled` - `learning.seed` - `learning.training_window` - `learning.selection.top_k` - `learning.selection.min_confidence` - `learning.mode` (`profile_only`, `profile_plus_structure`)

Recommended defaults: - `learning.enabled = false` initially - `learning.mode = profile_only`

9. Rollout Strategy

1. Shadow mode: generate suggestions, do not export.
2. Controlled export: small region/time-window.
3. Progressive rollout: larger geography after stable gates.
4. Full rollout only after multi-cycle parity and latency checks.

10. Immediate Next Implementation Tasks

1. Create `learning_config` and manifest schemas.
2. Implement dataset builder and deterministic seed helpers.
3. Implement profile-only refinement selector.
4. Add FIFO + parity + determinism gates to compile pipeline.
5. Run offline benchmark and produce first v11 validation report.