TARO v11 SSOT: Stage-Wise Breakdown (28 Stages)

Status: Proposed Structured Baseline
Date: 2026-02-07
Companion Documents: - `docs/taro_v11_single_source_of_truth.md` - `docs/taro_v11_architecture_plan.md` - `ResearchData/taro_architecture_refined.pdf`

1. Purpose

This document provides the stage-wise breakup of TARO v11 in the same operational structure as the refined architecture blueprint.

Each stage contains: - functional requirements - non-functional requirements - test equivalence classes - dependencies

2. Stage Grouping Overview

- Foundation: Stages 1-3
- Core Data Structures: Stages 4-9
- Cost and Heuristics: Stages 10-12
- Routing Algorithms: Stages 13-14
- Trait System: Stages 15-18
- Python Builder Pipeline: Stages 19-24
- Java Runtime and API: Stages 25-26
- Production Loop: Stages 27-28

Stage 1: ID Translation Layer

Functional Requirements

- Maintain external-to-internal and internal-to-external ID mapping.
- Support deterministic missing-ID behavior.
- Provide immutable read-optimized snapshots.

Non-Functional Requirements

- O(1)-style lookup for both directions.
- Thread-safe concurrent reads.
- Zero allocation in steady-state lookups.

Test Equivalence Classes

- Known ID, unknown ID, boundary ID.
- Unicode and long-string identifiers.
- Concurrent read stress.

Dependencies

- None.

Stage 2: Temporal Utilities and Tick Normalization

Functional Requirements

- Normalize request timestamps to model engine ticks.
- Resolve day-of-week and bucket indices.
- Provide FIFO validation helpers.

Non-Functional Requirements

- O(1) arithmetic conversion.
- Correct negative timestamp behavior.
- No object allocation in hot path.

Test Equivalence Classes

- Sec-to-ms, ms-to-sec conversions.
- Midnight/week boundary transitions.
- FIFO valid and FIFO violating sequences.

Dependencies

- None.

Stage 3: FlatBuffers Schema Contract

Functional Requirements

- Define model schema for topology, profiles, turns, spatial index, metadata.
- Preserve file identifier and compatibility assumptions.
- Support optional extension fields for lineage.

Non-Functional Requirements

- Zero-copy deserialization compatibility.
- Backward/forward-safe schema evolution.
- Alignment and binary stability.

Test Equivalence Classes

- Minimal valid model.
- Missing/invalid identifier.
- Evolved schema with optional fields.

Dependencies

- None.

Stage 4: Edge Graph Runtime Structure

Functional Requirements

- Load CSR-style directed edge graph from model.
- Provide O(1)-style edge property access.
- Support optional coordinates and edge origin mapping.

Non-Functional Requirements

- Cache-friendly primitive-array access.
- Immutable, concurrent-read safe.
- Predictable memory footprint on large graphs.

Test Equivalence Classes

- Empty/minimal graph.
- Vector length mismatches.
- Large sparse and dense graphs.

Dependencies

- Stage 3.

Stage 5: Turn Cost Lookup

Functional Requirements

- Lookup transition penalty by `(fromEdge, toEdge)`.
- Handle missing turns with default cost behavior.
- Represent forbidden transitions deterministically.

Non-Functional Requirements

- O(1)-style lookup.
- Read concurrency safety.
- Low memory overhead for sparse turn tables.

Test Equivalence Classes

- Known turn, missing turn, forbidden turn.
- Duplicate entries and overwrite policy.
- Large sparse turn map.

Dependencies

- Stage 3, Stage 4.

Stage 6: Search Infrastructure

Functional Requirements

- Provide priority queue, visited set, search state containers.

- Support deterministic ordering semantics.
- Support clear/reuse lifecycle across requests.

Non-Functional Requirements

- O(log n) queue operations.
- Low GC pressure with reusable state.
- Thread-local query-state safety.

Test Equivalence Classes

- Empty queue behavior.
- Equal-priority tiebreak behavior.
- Large state insert/extract cycles.

Dependencies

- Stage 4, Stage 5.

Stage 7: Sparse Live Overlay

Functional Requirements

- Apply per-edge live updates with TTL.
- Support blocked-edge representation and expiry semantics.
- Provide canonical live penalty multiplier.

Non-Functional Requirements

- O(1)-style lookup for active entries.
- Bounded memory with capacity policy.
- Concurrent read/write safety with deterministic outcomes.

Test Equivalence Classes

- Active, expired, blocked, missing states.
- Capacity policy behaviors.
- Concurrent reads with update batches.

Dependencies

- Stage 2, Stage 4, Stage 6.

Stage 8: Spatial Runtime (KD Query)

Functional Requirements

- Load serialized KD index.
- Resolve nearest node/edge anchor from coordinates.
- Support trait-based enable/disable mode.

Non-Functional Requirements

- Sublinear nearest-neighbor query behavior.
- Deterministic tie-break policy.
- Concurrent-read safe immutable runtime.

Test Equivalence Classes

- Valid KD parity vs brute force.
- Malformed KD contracts.
- Performance and concurrency stress.

Dependencies

- Stage 3, Stage 4.

Stage 9: Profile Store

Functional Requirements

- Load temporal profiles and day masks.
- Resolve day-aware multiplier and interpolation.
- Provide neutral fallback on missing/inactive profile.

Non-Functional Requirements

- O(1)-style profile and bucket access.
- Immutable read-safe store.
- Stable interpolation semantics.

Test Equivalence Classes

- Active day vs inactive day.
- Missing profile fallback.
- Fractional bucket wrap interpolation.

Dependencies

- Stage 2, Stage 3, Stage 4.

Stage 10: Time-Dependent Cost Engine

Functional Requirements

- Compute effective cost from base, profile, and live layers.
- Enforce blocked-edge infinite cost behavior.
- Expose explainable cost breakdown.

Non-Functional Requirements

- O(1)-style cost computation per expansion.
- No allocation in hot path.
- Deterministic numeric behavior.

Test Equivalence Classes

- Base-only cost path.
- Profile + live slowdown path.
- Blocked-edge and expired-live fallback.

Dependencies

- Stage 2, Stage 4, Stage 7, Stage 9.

Stage 11: Geometry Heuristic Providers

Functional Requirements

- Provide heuristic interface and implementations (`null`, `euclidean`, `spherical`).
- Preserve admissibility contract with cost model assumptions.
- Integrate with routing strategy selection.

Non-Functional Requirements

- Low-latency per-call distance computation.
- Stateless thread-safe functions.
- Numerically stable behavior near coordinate boundaries.

Test Equivalence Classes

- Known geometry distance checks.
- Edge coordinate boundaries.
- Admissibility verification set.

Dependencies

- Stage 4, Stage 8, Stage 10.

Stage 12: Landmark Preprocessing (ALT)

Functional Requirements

- Select landmarks and precompute distance arrays.
- Export landmark artifacts with model.
- Support bidirectional ALT heuristic use.

Non-Functional Requirements

- Offline compute bounded by configured budget.
- Compact serialized landmark footprint.
- Deterministic landmark selection under fixed seed.

Test Equivalence Classes

- Small graph correctness baseline.
- Disconnected graph handling.
- Admissibility and pruning effectiveness checks.

Dependencies

- Stage 4, Stage 10, Stage 11.

Stage 13: Bidirectional Time-Dependent A*

Functional Requirements

- Compute optimal point-to-point route.
- Respect time-dependent costs and transition semantics.
- Return route path, arrival time, and cost.

Non-Functional Requirements

- Deterministic expansion behavior under fixed snapshot.
- Efficient pruning with admissible heuristics.
- Bounded memory under configured query size.

Test Equivalence Classes

- Trivial route, disconnected route, long route.
- Departure-time-sensitive route differences.
- A* vs Dijkstra parity cases.

Dependencies

- Stage 4, Stage 5, Stage 6, Stage 10, Stage 11, Stage 12.

Stage 14: One-to-Many Dijkstra Matrix

Functional Requirements

- Compute one-to-many and matrix travel costs.
- Early terminate when all targets are settled.
- Return reachability and per-target arrival data.

Non-Functional Requirements

- Predictable performance across target counts.
- Reusable state for repeated queries.
- Deterministic outputs under fixed snapshot.

Test Equivalence Classes

- Single destination equivalence with route query.
- Many-destination matrix behavior.
- Unreachable destination handling.

Dependencies

- Stage 4, Stage 5, Stage 6, Stage 10.

Stage 15: Addressing Trait

Functional Requirements

- Support abstract ID and spatial coordinate addressing.
- Resolve typed request input to internal graph anchors.
- Provide reverse mapping for responses.

Non-Functional Requirements

- O(1)-style ID path and sublinear spatial path.
- Deterministic snapping and threshold policy.
- Trait-driven plugin-style extensibility.

Test Equivalence Classes

- Valid and invalid external IDs.
- Coordinates in/out of bounds.
- Mixed addressing mode requests.

Dependencies

- Stage 1, Stage 4, Stage 8.

Stage 16: Temporal Trait

Functional Requirements

- Provide linear and calendar temporal behavior.
- Resolve bucket/day selection per trait policy.
- Enforce model timezone contract where applicable.

Non-Functional Requirements

- Low-latency time translation.
- DST and timezone consistency under configured rules.
- Stateless request-path behavior.

Test Equivalence Classes

- Linear periodic buckets.
- Calendar weekday/weekend behavior.
- DST transition boundary cases.

Dependencies

- Stage 2, Stage 9.

Stage 17: Transition Trait

Functional Requirements

- Support node-based and edge-based transition logic.

- Integrate turn penalties where required.
- Preserve routing correctness across modes.

Non-Functional Requirements

- Configurable without runtime ambiguity.
- Stable performance impact characterization.
- Deterministic transition expansion semantics.

Test Equivalence Classes

- Node-based route comparison.
- Edge-based route with turn penalties.
- Illegal transition handling.

Dependencies

- Stage 4, Stage 5, Stage 10.

Stage 18: Trait Registry and Configuration

Functional Requirements

- Register available trait implementations.
- Validate trait compatibility and required dependencies.
- Expose selected trait bundle to runtime components.

Non-Functional Requirements

- Fast startup-time validation.
- Clear rejection diagnostics for invalid configs.
- Stable trait-hash generation for lineage.

Test Equivalence Classes

- Valid trait bundle resolution.
- Incompatible trait rejection.
- Missing dependency trait rejection.

Dependencies

- Stage 15, Stage 16, Stage 17.

Stage 19: Python Ingestion and Validation

Functional Requirements

- Parse source CSV/data feeds.
- Validate schema and required fields.
- Normalize IDs, times, and units for compile pipeline.

Non-Functional Requirements

- Scalable batch ingestion throughput.
- Deterministic parse/normalize behavior.
- Clear row-level validation diagnostics.

Test Equivalence Classes

- Valid source datasets.
- Missing/invalid columns and row values.
- Large-file ingestion behavior.

Dependencies

- Stage 2, Stage 3.

Stage 20: Graph Connectivity Linter

Functional Requirements

- Detect unreachable components and isolate ratio.
- Classify severity using configured thresholds.
- Emit linter report for build gating.

Non-Functional Requirements

- Bounded runtime for large graphs.
- Deterministic linter outputs under fixed input.
- Actionable report format.

Test Equivalence Classes

- Fully connected graph.
- Multi-component graph.
- Threshold boundary classification.

Dependencies

- Stage 19.

Stage 21: Profile Compression and Bucketing

Functional Requirements

- Build time-bucketed profile curves.
- Apply compression/smoothing policy.
- Enforce or repair FIFO policy based on config.

Non-Functional Requirements

- Bounded model-size growth.
- Deterministic compression output.
- Preservation of core temporal signal.

Test Equivalence Classes

- Flat/noise-free profile.
- High-variance profile.
- FIFO violation detect-and-repair path.

Dependencies

- Stage 2, Stage 19.

Stage 22: Landmark Selection Algorithm

Functional Requirements

- Select strategic landmark nodes.
- Produce forward/backward distance precompute inputs.
- Emit reproducible selection metadata.

Non-Functional Requirements

- Stable selection under fixed seed.
- Practical preprocessing latency at scale.
- Controlled memory budget.

Test Equivalence Classes

- Small graph manual-verifiable selection.
- Dense vs sparse graph behavior.
- Seed reproducibility.

Dependencies

- Stage 19, Stage 21.

Stage 23: FlatBuffers Model Compilation

Functional Requirements

- Serialize validated topology, profiles, turns, spatial, landmarks.
- Fill model metadata and lineage fields.
- Produce final `.taro` artifact with identifier.

Non-Functional Requirements

- Binary integrity and reproducibility.
- Strict compile failure on critical contract break.
- Stable output size profile.

Test Equivalence Classes

- Minimal valid build.
- Full-feature build with all optional sections.
- Corrupt/missing required artifact handling.

Dependencies

- Stage 3, Stage 19, Stage 21, Stage 22.

Stage 24: KD Tree Serialization (Python)

Functional Requirements

- Build and serialize KD tree artifacts from coordinates.
- Validate leaf payload and index boundaries.
- Attach spatial index to final model.

Non-Functional Requirements

- Efficient build time at large node count.
- Deterministic tree layout under fixed build settings.
- Compact and load-safe binary encoding.

Test Equivalence Classes

- Minimal coordinate set.
- Balanced and skewed coordinate distributions.
- Invalid coordinate rows.

Dependencies

- Stage 3, Stage 19.

Stage 25: Java Model Loader and Hot Reload

Functional Requirements

- Load model with full header/schema validation.
- Materialize runtime stores (graph, profiles, turns, spatial, landmarks).
- Perform atomic model reload with continuity policy.

Non-Functional Requirements

- No partial visibility during reload.
- Fail-safe fallback to previous model on invalid reload.
- Bounded load latency and memory impact.

Test Equivalence Classes

- Valid model load.
- Invalid identifier/schema mismatch load failure.
- Atomic reload success/failure behavior.

Dependencies

- Stage 1, Stage 3, Stage 4, Stage 9, Stage 12, Stage 23, Stage 24.

Stage 26: HTTP API Layer (`/api/v1`)

Functional Requirements

- Serve route, matrix, live-update, telemetry, health, and admin endpoints.
- Validate typed request contracts.
- Normalize/validate time-unit compatibility.

Non-Functional Requirements

- Stable versioned contract behavior.
- Predictable error model and reason codes.
- High-concurrency safe request handling.

Test Equivalence Classes

- Valid route and matrix requests.
- Invalid schema and unit mismatch requests.
- Mixed addressing and trait-bound requests.

Dependencies

- Stage 7, Stage 13, Stage 14, Stage 15, Stage 16, Stage 17, Stage 18, Stage 25.

Stage 27: Telemetry Collection and Aggregation

Functional Requirements

- Capture predicted-vs-actual travel outcomes.
- Persist lineage fields (`model_hash`, `traits_hash`, etc.).
- Export telemetry for retraining pipelines.

Non-Functional Requirements

- Low overhead ingestion path.
- Privacy-safe and partitionable event records.
- Backpressure-aware buffering behavior.

Test Equivalence Classes

- Complete telemetry payload.
- Missing lineage field rejection/flagging.
- High-volume ingestion and flush behavior.

Dependencies

- Stage 26, Stage 4.

Stage 28: Observability and Metrics

Functional Requirements

- Expose runtime metrics and health indicators.

- Track query latency, parity health, overlay cleanup, reload outcomes.
- Provide operational alerts and dashboards.

Non-Functional Requirements

- Low cardinality metric labels.
- Minimal runtime overhead.
- Actionable and consistent observability semantics.

Test Equivalence Classes

- Metrics endpoint correctness.
- Alert threshold trigger behavior.
- Fault injection and recovery observability.

Dependencies

- Stage 26, Stage 27.

3. Cross-Stage Verification Gates

Mandatory release gates across stage boundaries: - FIFO gate: Stages 2, 9, 10, 21, 23. - Parity gate: Stages 10, 11, 13, 14, 25. - Determinism gate: Stages 7, 10, 13, 14, 23, 25, 26. - Reproducibility gate: Stages 19-24, 27.

This stage-wise breakup is intended to be used with the SSOT as the implementation planning checklist for each milestone.