Instructions for an intelligent agent

1 Essential Instructions for ATWL Workflow Extraction

1.1 Core Principles

1.1.1 Appropriate Abstraction Level

ATWL represents the analytical logic, not implementation details.
Capture:

- What analytical questions are asked
- What operations are performed and why
- How artifacts flow through the analysis
- Iterative refinement and assessment patterns

Omit:

- Specific algorithms (k-means, DBSCAN → "partition-based clustering", "density-based clustering")
- Tool names (Python, Tableau, D3.js)
- Exact parameters (k=5, ε=0.3 → handled as specifications or implicit)
- Implementation details (SQL queries, data formats, rendering methods)

Guideline: If comparing this workflow to 10 similar workflows, what level of detail helps identify patterns and differences in analytical **approach** without getting lost in implementation **specifics**?

1.1.2 Focus on Workflow Structure, Not Results

Artifact descriptions should describe ROLES and TYPES, not specific findings.

Too specific (describes results):

```
artifact movement_patterns : pattern(trajectories)
    description: "23% are local trips; three route types: peripheral,
        radial inward, radial outward"
```

Appropriate (describes role):

```
artifact movement_patterns : pattern(trajectories)
    representation form: "categorized patterns with characteristics"
    description: "Identified movement patterns: trip distance
        distributions, route types, spatial organization"
```

Why: Different datasets yield different results, but workflow structure remains comparable.

1.1.3 Tangible Artifact Representations

All artifacts must represent explicit, tangible outputs, not mental states.

The `representation form` field of knowledge, pattern, and other artifacts should describe a concrete, communicable form — not an analyst's internal understanding.

Wrong (mental state):

```
artifact insights : knowledge(data)
    representation form: "understanding of dynamics"
```

Correct (tangible output):

```
artifact insights : knowledge(data)
    representation form: "statements and explanations"
```

Acceptable representation forms include:

- `"statements and explanations"`
- `"quality judgment"`
- `"statements and diagrams"`
- `"statements and recommendations"`
- `"diagnostic statements"`
- `"categorized patterns with descriptions"`
- `"parameter settings"`
- `"recorded modelling interests and goals"`

Avoid:

- `"understanding"`, `"comprehension"`, `"awareness"`
- `"architectural understanding"`, `"diagnostic understanding"`
- `"synthesized understanding"`

1.1.4 Prefer Succinct Descriptions

Descriptions and manner fields should be **concise yet clear**. Include enough detail to convey the analytical role and method type, but omit redundant qualifiers, elaboration that restates what is already captured by the intent or artifact type, and unnecessary detail that does not aid workflow comparison.

Too verbose:

```
manner: "partition individual feature domains and feature-pair domains
    into disjoint regions; approximate local conditional target
    distributions; compute goodness-of-fit relevance measures at
    multiple partition complexities"
```

Succinct:

```
manner: "partition feature domains and compute multi-resolution
    relevance measures for conditional target distributions"
```

Guideline: If removing a phrase does not change the reader's understanding of what the transform does or what role the artifact plays, remove it.

1.2 Syntax Reference

This section summarises the key syntactic rules. Refer to the full ATWL definition document for complete details.

1.2.1 Workflow Declaration

```
workflow <workflow-ID>
    template: <intent> → <intent> → loop(...) → ...
    description: "..."
```

No trailing colon after the workflow identifier.

Wrong:

```
workflow MyWorkflow:
```

Correct:

```
workflow MyWorkflow
```

1.2.2 Comment Syntax

Use `#` for comments. Do **not** use `//`.

```
# This is a correct comment

// This is NOT valid ATWL
```

1.2.3 Transform Field Order

Transforms should follow this field order:

```
transform <ID> :
    intent: <generic-intent>
    manner: "..."                    # optional
    input: <artifact-IDs>
    output: <artifact-IDs>
    actor: human | machine | hybrid
    description: "..."               # required
```

Every transform **must** have a `description:` field.

1.2.4 Entities: Internal Structure Keywords

Use only these standard values for `internal structure:`:

- `elementary` — indivisible entity
- `group/cluster` — unordered collection of components
- `episode` — bounded time interval
- `region` — bounded spatial extent
- `sequence` — linearly ordered components
- `formation` — network, hierarchy, tree, graph, or other relational structure

Common mistakes:

- `network` → use `formation`
- `hierarchical` → use `formation`
- `group` (without `/cluster`) → use `group/cluster`

1.2.5 Entities: Embedment

Omit `embedment:` when the artifact represents a single entity (e.g., one aggregation tree, one projection space, one network snapshot). Embedment describes how *multiple* entities are positioned relative to each other; it is inapplicable when there is no collection of peer entities.

Wrong:

```
artifact projection_space : entities
    internal structure: elementary
    embedment: space          # single reference frame, not a collection
```

Correct:

```
artifact projection_space : entities
    internal structure: elementary
    features:
      - id: axes
        value structure: atomic
        value type: numeric
        description: "Number of spatial dimensions (2)"
    description: "Two-dimensional projection space serving as
        reference frame"
```

1.2.6 Unified Feature Syntax

The same two-level scheme applies to both internal features (inside `entities` artifacts) and standalone `feature` artifacts:

```
# Internal feature (inside entities)

features:
  - id: <feature_id>
    value structure: <structure>
    value type: <type>              # optional
    description: "..."

# Standalone feature artifact

artifact <ID> : feature(<artifact-ID>)
    value structure: <structure>
    value type: <type>              # optional
    representation form: "..."      # optional
    description: "..."
```

Value structures:

- `atomic` — single value per entity
- `list` — ordered or unordered enumeration of values
- `vector` — fixed-length array of values
- `matrix` — two-dimensional array indexed by two dimensions
- `relational configuration` — irregular relational structure (graph, tree, network)

Value types:

- `numeric`, `ordinal`, `categorical`, `temporal`, `spatial`, `text`, `reference`

When components are of mixed types, use set notation: `value type: {numeric, temporal}`. May be omitted when types are evident from context.

Note: Use `reference` for entity pointers, `categorical` for binary indicators, `relational configuration` for structural features.

Feature representation form. The `representation form:` field on feature artifacts is **optional** and should be used only when `value structure:` alone is ambiguous about the concrete encoding. Typical uses:

- `relational configuration` — e.g., `"k-NN graph"`, `"directed flows with time series"`
- `vector` — e.g., `"time series per place"`, `"daily measurement profile"`
- `matrix` — e.g., `"pairwise similarity matrix"`, `"term-document matrix"`

Do **not** use `representation form:` as a substitute for `value structure:`, and omit it for `atomic` and `list` structures where `value type:` is sufficient.

1.2.7 Visualisation Artifact Syntax

Visualisation artifacts **must** use `layout:`, `form:`, and `encoding:` fields. Do **not** use `representation form:` for visualisations.

Wrong:

```
artifact my_viz : visualisation(data)
    representation form: "interactive scatterplot"
```

Correct:

```
artifact my_viz : visualisation(data)
    layout: "2D scatterplot"
    form: "colored points with connecting lines"
    encoding: "position from arrangement; color by cluster;
        lines connect consecutive items"
    description: "Interactive scatterplot showing ..."
```

1.2.8 Model Artifact Parameterisation

Model artifacts should declare what they are built from:

```
artifact my_model : model(training_data, model_spec)
    description: "..."
```

Given models (pre-trained, exogenous) may omit parameterisation:

```
artifact pretrained : model
    origin: given
    description: "..."
```

1.3 Artifact Types: Specification, Pattern, Knowledge

1.3.1 specification Artifact Type

Purpose: Parameters, settings, constraints that control how transforms operate.

When to use:

- Algorithm parameters being refined (distance thresholds, k values)
- User guidance/constraints (reference specifications, desired properties)
- Methodological choices (distance functions, aggregation strategies)
- Any setting that is iteratively adjusted or optimised

Creation:

- Use **assess** intent when derived from evaluation/optimisation
- Use **generate-knowledge** intent when analyst makes strategic decision

Required field: Every specification must have a `representation form:` field.
Example:

```
artifact clustering_params : specification
    origin: given
    representation form: "parameter settings"
    description: "Distance thresholds and sensitivity parameters
        for density-based clustering"
```

Note: The relationship between a specification and the transforms it controls is implicit from the dataflow — the specification appearing as input to those transforms. Do not add an `applies to:` field; this is not valid ATWL syntax.

1.3.2 assess Intent

Purpose: Evaluate quality/adequacy of results; decide if refinement needed.
Produces: knowledge artifacts (quality judgments), optionally **specification artifacts** (derived parameters from evaluation or optimisation)
When to use:

- Evaluating cluster/topic/aggregation quality
- Checking model performance
- Assessing visualisation effectiveness
- Deciding whether to continue refinement
- Optimising parameters (output includes updated specification)

Do NOT use for:

- Discovering patterns in data (use **abstract**)
- Final pattern characterisation (use **abstract**)

Example (knowledge only):

```
transform T_assess_clusters :
    intent: assess
    manner: "evaluate cluster quality"
    input: cluster_viz, clusters
    output: cluster_assessment
    actor: human
    description: "Evaluate whether clusters are spatially coherent
        and whether refinement is needed"

artifact cluster_assessment : knowledge(clusters)
    representation form: "quality judgment"
    description: "Assessment of cluster spatial coherence and
        whether refinement is needed"
```

Example (knowledge + specification):

```
transform T_explore_thresholds :
    intent: assess
    manner: "adjust thresholds per strategy and evaluate trade-offs"
    input: fairness_viz, fairness_metrics, strategy_direction
    output: fairness_assessment, threshold_config
    actor: hybrid
    description: "Apply fairness optimisation strategy and evaluate
        its impact on per-slice performance"

artifact fairness_assessment : knowledge(trained_models)
    representation form: "fairness evaluation"
    description: "Assessment of fairness-performance trade-offs"

artifact threshold_config : specification
    representation form: "per-slice classification thresholds"
    description: "Optimised thresholds under chosen strategy"
```

1.3.3 generate-knowledge Intent and Specification Outputs

Purpose: Formulate knowledge artifacts from patterns, models, and visual evidence. May also produce specification artifacts when the analyst's primary act is to decide on a course of action.

Produces: knowledge artifacts (insights, conclusions), and/or **specification artifacts** (methodological decisions)

When to use for specification output:

- Analyst selects a modelling strategy
- Analyst chooses features to include
- Analyst sets parameters for the next iteration
- Analyst formulates investigation focus or query criteria

Example:

```
transform T_adjust :
    intent: generate-knowledge
    manner: "adjust clustering parameters based on assessment"
    input: cluster_assessment, cluster_viz, S_clustering
    output: S_clustering'
    actor: human
    description: "Adjust clustering parameters: modify number of
        clusters, select distance measure, or change focus"

artifact S_clustering' : specification
    representation form: "parameter settings"
    description: "Updated clustering parameters after refinement"
```

1.3.4 loop(...) in Templates

Purpose: Show iterative portions of workflow in template summary.

Syntax: `loop(intent -> intent -> ...)`

When to use:

- Iterative refinement is methodologically important
- Assessment-driven parameter adjustment
- Multiple transforms iterate together
- Helps readers understand structure

Constraint: Templates permit only one level of `loop(...)` notation. **Nested loops are discouraged in templates.** When a workflow has nested iteration (e.g., an inner exploration loop inside an outer improvement loop), retain the outer — more analytically significant — loop in the template and flatten the inner loop to a single representative pass. The inner loop remains fully specified in the workflow body.

Example — workflow with nested iteration:

```
# WRONG (nested loops in template):

template: loop(build-model → loop(visualise → abstract → assess)
             → generate-knowledge) → generate-knowledge

# CORRECT (inner loop flattened in template):

template: loop(build-model →
               visualise → abstract →
               visualise (explore) → abstract → assess →
               generate-knowledge (diagnose) →
               generate-knowledge (specify)) →
          generate-knowledge (finalize)
```

Annotation convention: When the same intent appears multiple times in a template with distinct analytical roles, use brief parenthetical annotations to distinguish them:

```
template: loop(generate-knowledge (specify) → build-model →
               visualise → abstract → assess →
               generate-knowledge (diagnose)) →
          generate-knowledge (select)
```

Annotations should capture the distinct role concisely (e.g., `specify`, `diagnose`, `select`, `explore`, `refine`).

1.4 Artifact Type Decision Guide

1.4.1 Is it specification, pattern, or knowledge?

Artifact	Type	Key Characteristics	Example
Parameters, settings	specification	Controls operations; prescriptive	"k=7", "use route similarity"
Quality judgments	knowledge	Evaluative statements; "is this good?"	"Clusters lack coherence"
Discovered structures	pattern	Observed regularities; "what exists?"	"Three cluster types identified"
Domain insights	knowledge	Interpreted understanding	"Peak usage at 8am"
Strategic decisions	specification	Methodological choices	"Use density-based clustering"

Table 1: Artifact type characteristics

1.4.2 Decision Tree for Ambiguous Cases

Question 1: Does it control how a transform operates?

- YES → **specification**
- NO → Continue

Question 2: Is it a judgment about quality/adequacy?

- YES → **knowledge**
- NO → Continue

Question 3: Is it a discovered structure/regularity in the data?

- YES → **pattern**
- NO → **knowledge** (likely synthesised insight)

1.4.3 Concrete Entities vs. Patterns

When a transform identifies sets or groups of concrete objects (e.g., partitioning graph nodes by structural equivalence, grouping records by shared properties), the output is **entities** with `internal structure: group/cluster`, not a pattern. Patterns represent *abstracted regularities* (e.g., "morning peak," "three route types"), not the concrete objects themselves.

Wrong:

```
artifact structural_templates : pattern(graph)
    description: "Sets of graph nodes with identical structure"
```

Correct:

```
artifact structural_templates : entities
    internal structure: group/cluster
    embedment: set
    description: "Sets of graph nodes with identical nested
        structure, representing repeated module types"
```

1.4.4 Visualisation Inputs and References

`visualise` transforms consume data, feature, arrangement, pattern, and specification artifacts — **not other visualisation artifacts**. Visualisation artifacts are consumed by `abstract` and `assess` transforms, where the human interprets the display. When two views share layout or are visually coordinated, model this through shared data inputs, not by piping one visualisation into another.

Wrong:

```
transform T_visualize_overlay :
    intent: visualise
    input: base_visualization, new_data
```

Correct:

```
transform T_visualize_overlay :
    intent: visualise
    input: base_data, new_data, shared_features
```

Corollary: Parenthetical references on visualisation artifacts (e.g., `visualisation(...)`) should reference what is depicted — entities, features, patterns — not other visualisations.

1.4.5 Arrangement Context

The `context:` field of an arrangement identifies the reference structure *within which* entities are positioned. It must be a **separate entity**, not the same entity being arranged.

Wrong (self-referential):

```
artifact graph_arrangement : arrangement(graph)
    context: graph
```

Correct:

```
artifact layout_space : entities
    internal structure: elementary
    features:
      - id: axes
        value structure: atomic
        value type: numeric
        description: "Number of spatial dimensions (2)"
    description: "Two-dimensional display space"

artifact graph_arrangement : arrangement(graph)
    context: layout_space
    principle: "similarity-preserving projection"
    description: "..."
```

1.5 Intent Selection Guide

1.5.1 assess vs. abstract vs. generate-knowledge

Intent	Purpose	Typical Output	Example Transform
assess	Evaluate quality; decide refinement	knowledge (judgment) + optional specification	"Assess cluster coherence"
abstract	Discover patterns in data	pattern	"Identify route types"
generate-knowledge	Formulate insights or strategic decisions	knowledge + optional specification	"Synthesise findings" or "Decide clustering approach"

Table 2: Intent comparison

1.5.2 characterise vs. define-unit for Cluster Labels

When clustering produces both groups (entities) and per-entity membership labels (feature), both outputs come from the same `define-unit` transform. Cluster labels are an inherent by-product of creating clusters.

However, when a pre-existing model or grouping is *applied* to assign labels without creating new entities (e.g., assigning documents to their dominant pre-computed topic), the intent is `characterise` — it computes a feature, not new units.

define-unit (creates groups + labels):

```
transform T_cluster :
    intent: define-unit
    output: clusters, cluster_labels
```

characterise (assigns labels from existing grouping):

```
transform T_assign_topics :
    intent: characterise
    input: documents, topic_weights
    output: topic_assignments
```

1.5.3 Common Intent Mistakes

Using abstract for assessment:

```
# WRONG

transform T_assess :
    intent: abstract  # This evaluates, not discovers
```

Using characterise for user decisions:

```
# WRONG

transform T_specify_guidance :
    intent: characterise  # This is strategic decision-making
    output: user_guidance  # specification
```

Using define-unit for pattern discovery:

```
# WRONG

transform T_find_patterns :
    intent: define-unit  # Not creating entities
    output: patterns
```

Using define-unit for feature computation:

```
# WRONG

transform T_assign :
    intent: define-unit  # Computing a feature, not creating entities
    output: topic_assignments  # feature artifact

# CORRECT

transform T_assign :
    intent: characterise
    output: topic_assignments
```

1.6 Common Workflow Patterns

1.6.1 Pattern 1: Progressive Refinement with Assessment

Structure:

```
loop L_refinement:
    compute (uses parameters_spec) → visualise →
    assess (produces quality_assessment: knowledge) →
    if needed:
        adjust parameters (assess or generate-knowledge
                           produces parameters_spec')
```

Template:

```
template: loop(compute → visualise → assess) →
          abstract → generate-knowledge
```

1.6.2 Pattern 2: User-Guided Interactive Refinement

Structure:

```
loop L_user_guided:
    compute (uses user_spec) → visualise →
    assess (produces quality_judgment: knowledge) →
    if needed:
        generate-knowledge (produces updated user_spec: specification)
```

Template:

```
template: loop(visualise → assess → generate-knowledge (specify)) →
          abstract → generate-knowledge
```

1.6.3 Pattern 3: Multi-Phase Progressive Analysis

Structure:

```
loop L_phase1:
    cluster by property A → assess → refine

generate-knowledge (decide next approach: specification) →

loop L_phase2:
    cluster by property B → assess → refine

abstract (discover final patterns) → generate-knowledge
```

Template:

```
template: loop(define-unit (phase1) → assess) →
          generate-knowledge (strategy) →
          loop(define-unit (phase2) → assess) →
```

```
          abstract → generate-knowledge
```

1.6.4 Pattern 4: Hypothesis Generation and Verification

Structure:

```
analyze subset → abstract (hypothesis: pattern) →
generate-knowledge (formulate verification approach: specification) →
verify on full dataset → assess →
exclude/refine based on findings → analyze remaining
```

Template:

```
template: define-unit (subset) → abstract (hypothesis) →
          generate-knowledge (verification strategy) →
          define-unit (verify on all) → assess →
          define-unit (refine) → abstract → generate-knowledge
```

1.7 Loop Termination Patterns

Two legitimate termination patterns exist, depending on whether any transforms are conditionally gated.

1.7.1 Pattern A: Conditional with Gated Refinement Step

Use when a specific transform (typically refinement or specification) should execute only if the analyst decides to continue. One branch contains the gated transforms; the other branch contains `exit loop`.

```
loop L_refinement:
    purpose: "..."
    until: "..."
    body:
        transform T_compute : ...
        transform T_visualise : ...
        transform T_assess : ...
            output: assessment

        if assessment indicates refinement needed:
            then:
                transform T_refine : ...
                    output: updated_spec
                assign: spec := updated_spec
            else:
                exit loop L_refinement
end loop L_refinement
```

Note: `exit loop` may appear in *either* branch — whichever is more natural given how the condition is phrased. For example:

```
# Also valid:

if assessment indicates satisfactory:
    then:
        exit loop L_refinement
    else:
        transform T_refine : ...
```

Post-loop transforms. Transforms that execute once after the loop terminates (e.g., final model selection, knowledge synthesis) may be placed either inside the exit branch (before `exit loop`) or after `end loop`. Placing them after `end loop` is preferred when the same transform would otherwise be duplicated across multiple exit points, or when the transform is conceptually a separate analytical phase rather than the conclusion of the iterative process.

Important: `continue loop` is **not** valid ATWL syntax. Only `exit loop <ID>` is defined. When the analyst decides to continue iterating, the loop body simply falls through to the next iteration. If the gated work is in the `then` branch, leave the `else` branch empty (or with a comment) rather than writing `continue loop`.

```
# WRONG:

      else:
        continue loop L_selection

# CORRECT:

      else:
        # Continue exploring
```

1.7.2 Pattern B: No Conditional; until: Governs Termination

Use when all transforms execute every iteration and there is no refinement step to gate. The `until:` clause alone governs termination. This is appropriate when the stopping criterion reflects a gradual, holistic change in the analyst's understanding that cannot be naturally reduced to a single assessable artifact.

```
loop L_exploration:
    purpose: "..."
    until: "..."
    body:
        transform T_explore : ...
        transform T_identify : ...
        transform T_assess : ...
end loop L_exploration
```

Choosing between them: Ask "Is there a specific transform that should only execute when the analyst decides to continue?" If yes, use Pattern A. If all transforms naturally

execute every iteration, use Pattern B.

1.8 Loop Assignment Rules

When artifacts are updated across loop iterations, ATWL requires explicit `:=` assignment statements.

1.8.1 Rule 1: Pre-Loop Declaration and Initialisation

The target of a `:=` assignment must be a previously declared artifact — either `origin: given`, produced by a transform **before** the loop, or initialised by a pre-loop `assign` statement. Do not create placeholder artifacts with artificial descriptions; ensure the initial version has proper provenance.

Pre-loop assignment for initialisation:

```
artifact D_all : entities
    ...

assign:
    D_current := D_all

loop L1:
    purpose: "Progressively analyse subsets"
    until: "All relevant subsets have been examined"
    body:
        transform T_process :
            input: D_current, ...
            output: D_next
            ...
        assign:
            D_current := D_next
end loop L1
```

Wrong — orphan artifact with no provenance:

```
artifact approach : knowledge(dataset)
    description: "Initial state: no prior attempts"

loop L_iteration:
    ...
    assign: approach := updated_approach
```

Correct — initial version produced by a pre-loop transform:

```
transform T_explore_data :
    intent: abstract
    ...
    output: data_characteristics, analytical_direction

artifact analytical_direction : knowledge(dataset)
    description: "Analyst's initial modelling interests based
```

```
        on data exploration"

loop L_iteration:
    ...
    assign: analytical_direction := updated_analytical_direction
```

1.8.2 Rule 2: Consumption Requirement

Every artifact updated via `:=` must appear as input to at least one transform in the loop body. The assignment is only meaningful if the updated value feeds back into subsequent computation. If an artifact produced inside a loop's `then` branch feeds back to the next iteration, verify that the corresponding `:=` target appears as input to a transform in the loop body.

1.8.3 Rule 3: Features Not Needed When Not Consumed

Internal features of entities artifacts (e.g., `cluster_size`) should be declared only when they are consumed by a downstream transform — for example, when cluster size affects visualisation encoding, assessment criteria, or selection decisions. Do not add features purely for documentation if no transform uses them.

1.9 Extraction Guidelines

1.9.1 Step 1: Identify Main Analytical Phases

Read paper's methodology section. Look for:

- Data preparation steps
- Iterative refinement procedures
- Assessment/evaluation steps
- Interactive user guidance
- Pattern discovery and interpretation
- Knowledge synthesis

Create workflow template showing main phases with `loop(...)` where appropriate.

Tool-focused papers: When a paper describes a tool rather than a single methodology, construct an assumed composite workflow by:

1. Identifying the tool's core analytical workflows or modes
2. Examining case studies or usage scenarios for concrete analytical progressions

3. Synthesising a representative workflow that captures the typical analytical process the tool enables

4. Noting in the workflow description that it is an assumed composite workflow

Prioritise case studies that demonstrate the fullest use of the tool's capabilities, and choose a linearisation that reflects the most analytically coherent progression.

1.9.2 Step 2: Map Paper Operations to ATWL Intents

Paper Language	**Likely ATWL Intent**
"compute features", "aggregate", "calculate"	characterise
"extract", "filter", "cluster", "segment", "group"	define-unit
"arrange", "position", "layout", "project"	contextualise
"display", "visualise", "render", "show"	visualise
"identify patterns", "discover", "find structures"	abstract
"assess quality", "evaluate", "check adequacy"	assess
"train model", "fit", "learn"	build-model
"formulate insights", "conclude", "explain"	generate-knowledge
"assign labels", "classify (existing model)"	characterise

Table 3: Mapping paper language to ATWL intents

1.9.3 Step 3: Identify Artifact Types

For each artifact created, determine:

Is it entities?

- Collection of analysis units (documents, trajectories, episodes, clusters)
- Has internal structure (elementary, sequence, episode, formation, etc.)
- Has embedment when there are multiple entities to relate (omit for single entities)
- Internal features use `value structure:` + optional `value type:`

Is it feature?

- Properties computed from entities

- Describes characteristics of entities
- Uses `value structure:` + optional `value type:`
- Do **not** use `representation form:` as a substitute for `value structure:`

Is it arrangement?

- Positioning of entities in a context
- References a context (a **separate** entities artifact serving as reference structure)

Is it visualisation?

- Visual representation for human perception
- References the data artifacts being depicted (entities, features, patterns — not other visualisations)
- Uses `layout:`, `form:`, `encoding:` fields (not `representation form:`)
- Consumed by **abstract** and **assess** transforms, not by other **visualise** transforms

Is it pattern?

- Discovered *abstracted* regularities/structures IN the data
- Typically produced by abstract intent
- Must be an abstraction, not concrete objects (use entities for concrete groups)

Is it knowledge?

- Judgments ABOUT results (from assess)
- Domain insights/understanding (from generate-knowledge)
- Interpreted statements, not just discovered patterns
- Must have tangible `representation form:` (not mental states)

Is it specification?

- Controls how operations behave
- Parameters, settings, constraints, user guidance
- Must have `representation form:`
- Relationship to controlled transforms is implicit from dataflow

Is it model?

- Trained/calibrated for prediction/simulation
- Generalises beyond observed data
- Parameterised by training inputs: `model(training_data, spec)`

1.9.4 Step 4: Handle Iterative Refinement

Paper describes parameter adjustment?

→ Create explicit specification artifact (`origin: given` for initial values); create loop with assess or generate-knowledge producing updated specification; close the feedback with `assign`

Paper describes user guidance?

→ Use generate-knowledge to create specification artifacts representing user input

Paper describes quality evaluation?

→ Use assess intent, producing knowledge artifacts with judgments (and optionally specification artifacts)

Loop requires iteratively updated artifacts?

→ Use `:=` assignment; ensure the target is declared before the loop with proper provenance

Loop has a gated refinement step?

→ Use conditional Pattern A with `exit loop` in one branch; place one-time terminal transforms after `end loop`

Loop has no gated step; all transforms execute every iteration?

→ Use Pattern B; let `until:` govern termination

1.9.5 Step 5: Maintain Appropriate Generality

Remove from descriptions:

- Specific numeric results ("23% are local trips")
- Dataset-specific details ("red cluster", "northwest region")
- Exact parameter values in descriptions (put in specifications if iteratively refined)
- Tool/algorithm names (abstract to conceptual approach)

Keep in descriptions:

- Analytical roles and purposes
- Types of patterns/structures
- Evaluation criteria
- Methodological approaches

1.10 Common Mistakes and Corrections

1.10.1 Putting Results in Artifact Descriptions

Wrong:

```
artifact clusters : entities
    description: "Five clusters representing morning, afternoon,
        evening, night, and weekend patterns"
```

Correct:

```
artifact clusters : entities
    internal structure: group/cluster
    embedment: set
    description: "Temporal clusters grouping time periods with
        similar activity patterns"
```

1.10.2 Using pattern for Specifications

Wrong:

```
artifact user_constraints : pattern(topics)
    description: "User-specified desired topic characteristics"
```

Correct:

```
artifact user_constraints : specification
    representation form: "desired topic properties"
    description: "User-specified constraints defining desired
        topic properties"
```

1.10.3 Using pattern for Quality Judgments

Wrong:

```
artifact quality_assessment : pattern(clusters)
    description: "Assessment that clusters lack coherence"
```

Correct:

```
artifact quality_assessment : knowledge(clusters)
    representation form: "quality judgment"
    description: "Assessment that clusters lack spatial coherence
        and require refinement"
```

1.10.4 Over-Specifying Parameters

Too specific:

```
transform T_cluster :
```

```
    manner: "k-means clustering with k=5, max_iter=300,
            init='k-means++'"
```

Appropriate abstraction:

```
transform T_cluster :
    manner: "partition-based clustering"
    input: data, clustering_spec
```

1.10.5 Missing Loop Notation in Template

Unclear:

```
template: define-unit → visualise → assess → generate-knowledge

# Hidden: there's iterative refinement in the body
```

Clear:

```
template: loop(define-unit → visualise → assess) →
          generate-knowledge
```

1.10.6 Visualisation Artifacts as Inputs to Visualise Transforms

Wrong:

```
transform T_visualize_overlay :
    intent: visualise
    input: base_visualization, new_data
```

Correct:

```
transform T_visualize_overlay :
    intent: visualise
    input: base_data, new_data, shared_features
```

1.10.7 Orphan Artifacts as Loop Pre-Declarations

Wrong:

```
artifact direction : knowledge(dataset)
    description: "Initial state: no prior attempts"

loop L_iteration:
    ...
    assign: direction := updated_direction
```

Correct:

```
transform T_initial_analysis :
    intent: abstract
```

```
    ...
    output: initial_direction

artifact initial_direction : knowledge(dataset)
    description: "Analyst's initial analytical direction"

loop L_iteration:
    transform T_specify :
        input: ..., initial_direction
        ...
    ...
    assign: initial_direction := updated_direction
```

1.10.8 Using representation form for Visualisations

Wrong:

```
artifact my_viz : visualisation(data)
    representation form: "interactive scatterplot with clusters"
```

Correct:

```
artifact my_viz : visualisation(data)
    layout: "2D scatterplot"
    form: "colored points"
    encoding: "position from projection; color by cluster"
    description: "Interactive scatterplot showing clusters"
```

1.10.9 Non-Standard Internal Structure Keywords

Wrong:

```
artifact graph : entities
    internal structure: network

artifact tree : entities
    internal structure: hierarchical

artifact groups : entities
    internal structure: group
```

Correct:

```
artifact graph : entities
    internal structure: formation

artifact tree : entities
    internal structure: formation

artifact groups : entities
    internal structure: group/cluster
```

1.10.10 Embedment on Single Entities

Wrong:

```
artifact agg_tree : entities
    internal structure: formation
    embedment: {set, time}    # single tree, not a collection
```

Correct:

```
artifact agg_tree : entities
    internal structure: formation
    # no embedment --- single entity
```

1.10.11 Using continue loop

Wrong:

```
      else:
        continue loop L_selection
```

Correct:

```
      else:
        # Continue exploring
```

1.10.12 Intangible Representation Forms

Wrong:

```
artifact insights : knowledge(data)
    representation form: "comprehensive understanding"
```

Correct:

```
artifact insights : knowledge(data)
    representation form: "statements and explanations"
```

1.11 Quality Checklist

Before finalising ATWL representation, verify:

- **Template** shows main analytical progression with `loop(...)` where appropriate; **no nested loops**; parenthetical annotations distinguish repeated intents
- **All transforms** have a `description:` field (required) and optionally `manner:`
- **Transform field order:** `intent -> manner -> input -> output -> actor -> description`

- **Intents** match purposes: assess for evaluation, abstract for discovery, characterise for feature computation (not define-unit)
- **Assessment outputs** are **knowledge** artifacts (judgments), optionally **specification**; not pattern
- **Parameters/settings** are **specification** artifacts when iteratively refined; have `representation form:`; no `applies to:` field
- **Discovered structures** are **pattern** artifacts from abstract intent (not concrete groups — those are entities)
- **Internal structure keywords** are standard: `elementary`, `group/cluster`, `episode`, `region`, `sequence`, `formation`
- **Embedment** omitted for single-entity artifacts
- **Feature syntax** uses `value structure:` + optional `value type:` (both internal and standalone); no `representation form:` as substitute
- **Visualisation syntax** uses `layout:`, `form:`, `encoding:` (not `representation form:`)
- **Model artifacts** parameterised: `model(training_data, spec)` (given models may omit)
- **Knowledge representation forms** are tangible (not mental states)
- **Descriptions** are succinct: focus on roles/types, not specific results; omit unnecessary detail
- **Abstractions** appropriate: conceptual approaches, not algorithm names
- **Actor types** specified (human, machine, hybrid) for all transforms
- **Artifact types** follow decision guides (specification vs. pattern vs. knowledge)
- **Generic terminology** used (not dataset-specific findings)
- **Visualisation inputs**: visualise transforms consume data artifacts, not other visualisations
- **Visualisation references**: parenthetical references cite depicted data artifacts, not other visualisations
- **Arrangement context**: references a distinct entity, not self-referential
- **Loop `:=` targets** declared before the loop with proper provenance (no orphan placeholders)
- **Loop `:=` targets consumed** by at least one transform in the loop body
- **No `continue loop`** — only `exit loop <ID>` is valid

1.12 Summary: Key Distinctions

Concept	Definition	Created By	Example
pattern	Abstracted regularity IN data	abstract	"Three route types: peripheral, inward, outward"
knowledge (from assess)	Quality judgment ABOUT results	assess	"Clusters lack coherence; refinement needed"
knowledge (from generate-knowledge)	Domain insights or strategic decisions	generate-knowledge	"Peak usage at 8am" or "Use density clustering"
specification	Control parameters FOR operations	assess or generate-knowledge	"k=7" or "route similarity distance function"

Table 4: Key concept distinctions

1.13 Final Guidance

When in doubt:

1.13.1 For artifact types:

Ask "What is its purpose?"

- Controls operations → specification
- Quality judgment → knowledge
- Abstracted regularity → pattern
- Concrete grouped objects → entities (group/cluster)
- Domain insight → knowledge

1.13.2 For intents:

Ask "What is the primary goal?"

- Evaluate quality → assess
- Discover patterns → abstract
- Formulate insights/decisions → generate-knowledge

- Create/modify units → define-unit
- Compute properties → characterise
- Assign labels from existing grouping → characterise (not define-unit)

1.13.3 For abstraction level:

Ask "What helps workflow comparison?"

- Keep: analytical approach, workflow structure, artifact roles
- Remove: specific algorithms, tools, exact parameters, dataset results

ATWL captures analytical recipes, not computational cookbooks.