CityData Harmonizer

System Architecture & Glossary

Technical Reference Document

MiMathon Harmonization Project | GEX Team

Version 1.0 — May 2026

Table of Contents

Table of Contents 2
1. Executive Summary 4
2. Glossary 5
3. Layered Architecture 9
3.1 Layer 1 — Configuration & Schema 9
3.2 Layer 2 — Ingestion 10
3.3 Layer 3 — Intelligence (LLM Agents) 10
3.4 Layer 4 — Knowledge Plane 10
3.5 Layer 5 — Quality Gate 11
3.6 Layer 6 — Execution & Validation 11
3.7 Layer 7 — Output & Provenance 12
4. Data Flow — The 9-Act Pipeline 13
4.1 ACT 1 — Ingestion 13
4.2 ACT 2 — Profiling 14
4.3 ACT 3 — Planning 14
4.4 ACT 4 — Resolution 14
4.5 ACT 5 — Knowledge Engine Verification 15
4.6 ACT 6 — Execution 15
4.7 ACT 7 — Validation 15
4.8 ACT 8 — Provenance Capture 16
4.9 ACT 9 — Output Persistence 16
5. Control Plane vs. Data Plane 17
5.1 Control Plane 17
5.2 Data Plane 18
6. Knowledge Plane 19
6.1 Knowledge Base Entry Lifecycle 19
6.2 Self-Learning Behaviour 20
7. MCP Integration Plane 21
7.1 MCP Tools Used 21
7.2 SchemaSelector 5-Step Pipeline (pre-pipeline Act 0) 21
7.3 SDMClient Architecture 22
8. Component Architecture 23
9. Data Structures Reference 24
9.1 FieldProfile 24
9.2 SourceIntelligenceFile (SIF) 24
9.3 FieldMapping 24
9.4 HarmonizationPlan 25
9.5 MappingEntry (Knowledge Base Record) 25
9.6 KEVerdict 26
9.7 ProvenanceRecord 26
10. Transform Registry 28
11. Configuration Reference 29

11.1 config/settings.yaml 29
11.2 config/llm.yaml 29
11.3 Manifest YAML (sources/{name}/manifest.yaml) 29
11.4 CLI Reference (demo/run_demo.py) 30
12. Appendix — Full Directory Structure 31

1. Executive Summary

CityData Harmonizer is an AI-assisted data integration pipeline designed to unify heterogeneous tabular data from multiple smart-city sources into validated NGSI-LD records conforming to OASC Smart Data Model (SDM) standards. The system tackles a pervasive interoperability challenge: different municipalities and data providers publish semantically identical information (air quality readings, weather observations, traffic counts) in incompatible formats, column names, and measurement units. Instead of requiring hand-crafted ETL scripts for every new source, CityData Harmonizer uses a hierarchy of LLM agents to analyse, map, and transform source fields automatically, backed by a self-learning Knowledge Base that amortises LLM costs over repeated runs.

The architecture is structured around three orthogonal planes — Control, Data, and Knowledge — layered across seven functional tiers from configuration to output. A deterministic Knowledge Engine acts as a mandatory quality gate between the LLM-driven planning phase and the record-level execution phase, ensuring that no AI-generated mapping reaches production data without passing four rule-based checks. Every output record carries a complete ProvenanceRecord that traces it back to its source row, all decisions made, and all transformations applied. Optional Model Context Protocol (MCP) integration with the FIWARE SmartDataModels catalogue enables automatic canonical schema discovery and server-side cross-validation, making the system fully self-configuring for new domains.

2. Glossary

All terms are listed alphabetically. Terms in bold within definitions are themselves defined in this glossary.

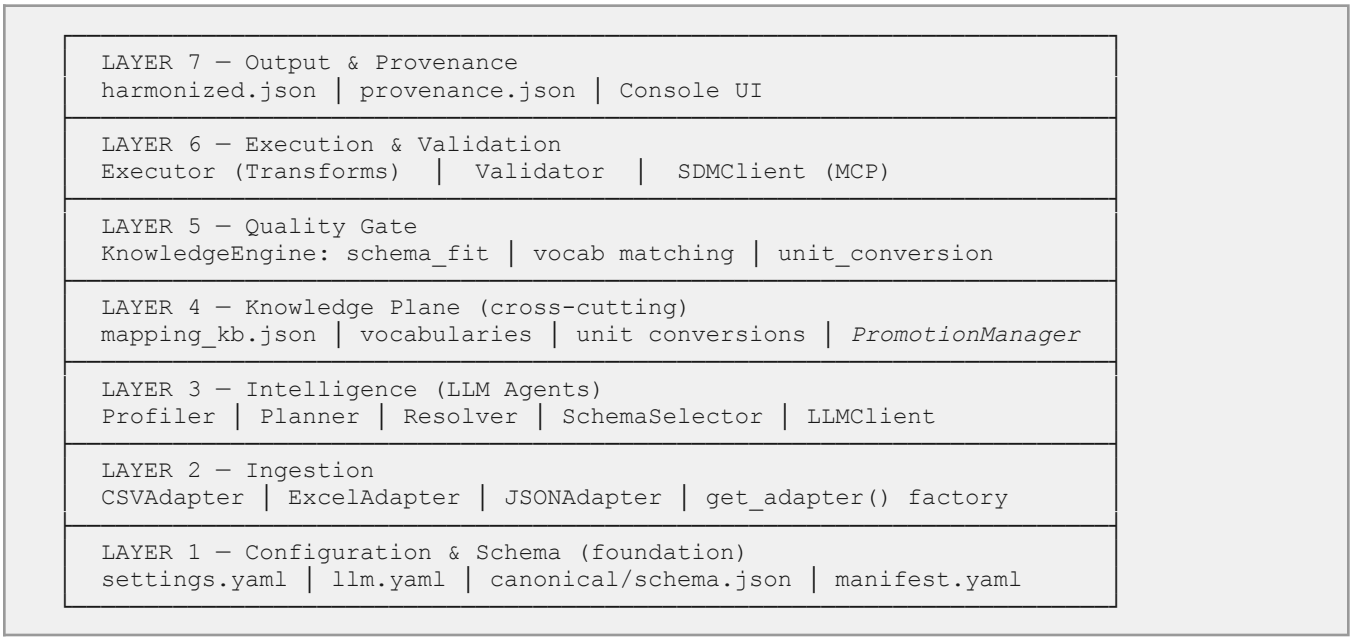
Term	Definition
ACT	One of 9 discrete, sequentially executed pipeline stages. Each Act has exactly one responsibility: Ingestion, Profiling, Planning, Resolution, KE Verification, Execution, Validation, Provenance Capture, or Output Persistence.
Adapter	A pluggable file-reader component (CSVAdapter, ExcelAdapter, JSONAdapter) that normalises raw source files into a uniform list of Python dicts. The factory function get_adapter(source_dir, fmt) returns the correct implementation based on the manifest format field.
Agent	An LLM-powered component responsible for one reasoning step. The four Agents are: Profiler, Planner, Resolver, and SchemaSelector. All share a single LLMClient and are stateless between calls.
Auto-approve threshold	A confidence score of 0.85 or above. When the Planner assigns a confidence at or above this threshold, the mapping is immediately marked INFERRED and queued for automatic KB promotion without human review.
Canonical Schema	The target JSON Schema that all harmonised output records must validate against. Loaded from canonical/schema.json or fetched automatically via the SchemaSelector and MCP.
Confidence score	A floating-point value in [0, 1] produced by LLM agents expressing their certainty that a proposed field mapping is correct. Drives the three-way routing decision: INFERRED (>=0.85) / UNCERTAIN (0.60-0.84) / UNMAPPED (<0.60).
Evidence	A list of human-readable strings produced by LLM agents to justify a mapping decision (e.g., 'Field name matches target', 'Unit conversion path verified'). Stored in KB entries for auditing and traceability.
Executor	The deterministic, LLM-free component that applies the verified HarmonizationPlan to raw records row-by-row. It looks up each transformation function in the TRANSFORM_REGISTRY, executes it, and injects required NGSI-LD fields (id, type).
HarmonizationPlan	A Pydantic model grouping all FieldMappings for one data source. Output by the Planner and updated by the Resolver. Consumed by the Knowledge Engine and Executor.
INFERRED	A mapping status assigned when confidence >= 0.85. The mapping requires no human review and is eligible for automatic promotion to the Knowledge Base.
KB short-circuit	An optimisation in the Planner: when an existing promoted KB entry is found for a (source_name, source_field) pair, the Planner reuses that mapping directly without spending any LLM tokens.

Term	Definition
KEVerdict	A Pydantic model containing the Knowledge Engine's decision for one field mapping. Fields: field, decision (APPROVE / REJECT / ESCALATE), checks_passed, checks_failed, and reason.
Knowledge Base (KB)	The persistent JSON file (knowledge_core/mapping_kb.json) storing all promoted, verified field mappings as MappingEntry objects. The primary long-term memory of the system, enriched on every successful pipeline run.
Knowledge Engine (KE)	A purely deterministic, LLM-free quality gate component that runs four checks on each proposed mapping and produces APPROVE, REJECT, or ESCALATE verdicts. It is the mandatory firewall between AI-generated plans and actual data transformation.
Knowledge Plane	The cross-cutting architectural plane composed of the Knowledge Base, vocabularies, unit conversion tables, KnowledgeCore loader, and PromotionManager. It is read by the Control Plane and written by the Quality Gate.
LLMClient	The shared asynchronous OpenRouter HTTP client used by all Agents. Supports three configurable model tiers: primary (deep reasoning), fast (quick utility), and fallback.
Manifest	A per-source YAML file (sources/{name}/manifest.yaml) describing the data file format, encoding, delimiter, sheet name, field descriptions, units, and ownership metadata. Required for every data source.
MCP (Model Context Protocol)	A standard JSON-RPC 2.0 protocol enabling LLM agents to call external tool servers. Used by CityData Harmonizer to query the FIWARE SmartDataModels MCP server for schema discovery and optional record validation.
Mapping Entry	A Pydantic model representing one record in the Knowledge Base. Stores the verified (source_name, source_field) to (target_field, transformation) mapping along with its lifecycle status and audit metadata.
NGSI-LD	The ETSI/FIWARE standard for representing smart-city entity data as JSON-LD linked data. All harmonised output records produced by CityData Harmonizer conform to NGSI-LD and include a URN id and a type field.
Planner	The LLM Agent responsible for proposing field-level mappings from source fields to canonical target fields. It applies the KB short-circuit for known fields and batches unknown fields into a single primary-tier LLM call.
Profiler	The LLM Agent responsible for analysing raw source fields and producing the SourceIntelligenceFile (SIF). It uses the fast LLM tier to extract dtype, sample values, format hints, semantic hints, and unit hints per field.
Promotion lifecycle	The state machine that governs KB entry progression: candidate (new proposal) -> reviewed (KE approved) -> approved (confidence threshold met) -> promoted (ready for KB short-circuit reuse).
PromotionManager	The component implementing the promotion lifecycle state machine. Decides when a mapping entry graduates to 'promoted' status based on KE verdict and confidence threshold.

Term	Definition
ProvenanceRecord	A Pydantic model capturing the complete audit trail for one harmonised output record. Fields: output_record_id, source_file, source_row_index, source_fields_used, mappings_applied, transforms_applied, ke_verdict, validator_result, timestamp, agent_versions.
Resolver	The LLM Agent that performs deep second-pass analysis on UNCERTAIN field mappings. It receives richer context than the Planner (full sample values, format hints, semantic hints, vocabularies) and produces a final INFERRED, UNMAPPED, or REVIEW decision.
SDMClient	The async MCP client (mcp_client/sdm_client.py) that communicates with the SmartDataModels MCP server. Uses JSON-RPC 2.0 over HTTPS with a 30-second timeout and local disk caching.
SchemaSelector	The LLM-powered Agent that auto-discovers the best matching canonical schema from the FIWARE SmartDataModels catalogue via MCP. Executes a 5-step pipeline: characterise sources -> build queries -> MCP search -> enrich candidates -> LLM ranking.
SIF (SourceIntelligenceFile)	The structured Pydantic model produced by the Profiler. It contains one FieldProfile per source field plus source-level metadata (name, file, format, row_count, profiled_at).
Smart Data Models	The open FIWARE/IUDX catalogue of NGSI-LD schemas covering domains such as environment, transportation, water management, and smart buildings. Served via the SmartDataModels MCP server.
Transform	A pure Python function applied to a single raw field value during the Execution Act. There are 9 registered transform types, each following the signature fn(value, params) -> Any.
UNCERTAIN	A mapping status assigned when confidence is between 0.60 and 0.84 (inclusive). The field is forwarded to the Resolver for deeper analysis before a final decision is made.
UNMAPPED	A mapping status assigned when confidence is below 0.60. The field is excluded from the harmonised output. It is recorded in provenance but no transformation is applied.
Unit conversion	A transform type (convert_unit) that converts numeric values between measurement units. Primary handler: the pint Python library. Fallback: domain-specific conversion factors defined in knowledge_core/units/conversions.yaml.
Validator	The component that validates each harmonised record against the canonical JSON Schema using jsonschema Draft7. Optionally cross-checks records against the SDM MCP server when --mcp-validate is passed.
Vocabulary	A YAML file (knowledge_core/vocabularies/*.yaml) listing canonical values and their synonyms for a categorical field. Used by the Knowledge Engine's vocabulary_membership check and provided as context to the Resolver.

3. Layered Architecture

The system is organised into seven functional layers stacked bottom-to-top. Each layer depends only on the layer(s) below it. The diagram below summarises the stack; the subsections that follow describe each layer in detail.



3.1 Layer 1 — Configuration & Schema

The foundation layer holds all static configuration that the rest of the system reads at startup. Nothing in this layer performs computation; it is purely declarative.

File / Component	Purpose
config/settings.yaml	Confidence thresholds (auto_approve: 0.85, review_required: 0.60), KE check list, transform registry, MCP server URL and timeout, profiler batch sizes, output paths
config/llm.yaml	LLM provider (OpenRouter), API key env var, model tiers: primary (DeepSeek v3), fast (Mistral-small), fallback (Gemini 2.5 Flash), temperature and max_tokens
canonical/schema.json	Target the canonical Smart Data Model JSON Schema. Defines required fields (id, type), property types, formats, and value constraints that all output records must satisfy
sources/*/manifest.yaml	Per-source metadata: format, encoding, delimiter, sheet name, field descriptions, units, unit_codes, owner, and date column declarations

3.2 Layer 2 — Ingestion

The ingestion layer abstracts file-format differences. Every adapter reads its source file using pandas and returns a uniform list[dict[str, Any]] regardless of format. The manifest.yaml drives all format-specific parameters.

Component	Input	Output
CSVAdapter	*.csv + manifest (delimiter, encoding)	list[dict]
ExcelAdapter	*.xlsx / *.xls + manifest (sheet_name)	list[dict]
JSONAdapter	*.json (array or root object)	list[dict]
get_adapter()	source_dir, format string	Correct Adapter class

3.3 Layer 3 — Intelligence (LLM Agents)

The intelligence layer contains all LLM-powered reasoning. Four stateless Agents share one LLMClient and are invoked sequentially by the orchestrator. The Knowledge Base short-circuit in the Planner allows this layer to be bypassed for already-known fields.

Agent	LLM Tier	Input	Output
Profiler	Fast	Raw records + manifest	SourceIntelligenceFile (SIF)
Planner	Primary	SIF + Canonical Schema + KB	Harmonization Plan
Resolver	Primary	UNCERTAIN mappings + SIF + Schema	Updated Harmonization Plan
SchemaSelector	Primary	Source characterisation + MCP results	Canonical schema path
LLMClient	Configurable	Prompt + model tier	JSON response

3.4 Layer 4 — Knowledge Plane

The Knowledge Plane is a cross-cutting layer accessed by both the Intelligence layer (reads) and the Quality Gate layer (writes). **It provides the self-learning memory of the system** and enables **progressive cost reduction as the KB grows**.

Component	Role
mapping_kb.json	Persistent store of promoted MappingEntry objects; primary lookup key is (source_name, source_field)
KnowledgeCore loader	Deserialises KB, vocabularies, and unit conversion tables into runtime objects at pipeline startup

Component	Role
PromotionManager	Implements the candidate → reviewed → approved → promoted state machine for KB entries
vocabularies/*.yaml	Categorical synonym tables; used by KE vocabulary_membership check and as Resolver context
units/conversions.yaml	Domain-specific unit conversion equations; used by KE unit_compatibility check and Executor convert_unit

3.5 Layer 5 — Quality Gate

The Quality Gate is the deterministic firewall between AI-generated proposals and actual data transformation. The Knowledge Engine runs four independent checks on every proposed mapping and issues a binding verdict. No LLM is involved.

Check	What it verifies	Fail verdict
schema_fit	target_field exists as a property in the canonical JSON Schema	REJECT (field excluded from output)
vocabulary_membership	If the mapping references a vocabulary, the mapped value appears in that vocabulary	ESCALATE (flagged for review)
unit_compatibility	A conversion path exists between source and target units (via pint or conversions.yaml)	ESCALATE (flagged for review)
provenance_completeness	source_field, evidence list, and confidence score are all present and non-empty	ESCALATE (flagged for review)

3.6 Layer 6 — Execution & Validation

The execution layer is entirely deterministic. It takes the verified HarmonizationPlan from the Quality Gate and applies it to every raw record. The Validator then checks each harmonised record against the canonical schema.

Component	Role
Executor	Iterates raw records; for each active mapping applies the registered transform function; catches per-field exceptions; injects required NGSI-LD fields
Transform Registry (9 fns)	rename_field, convert_unit, map_value, parse_date, normalize_text, restructure, parse_geojson, combine_lat_lon, normalize_range
Validator	onally cross-checks against SDM MCP server (--mcp-validate)

Component	Role
SDMClient	Async MCP client for optional server-side schema discovery (SchemaSelector) and record validation (Validator)

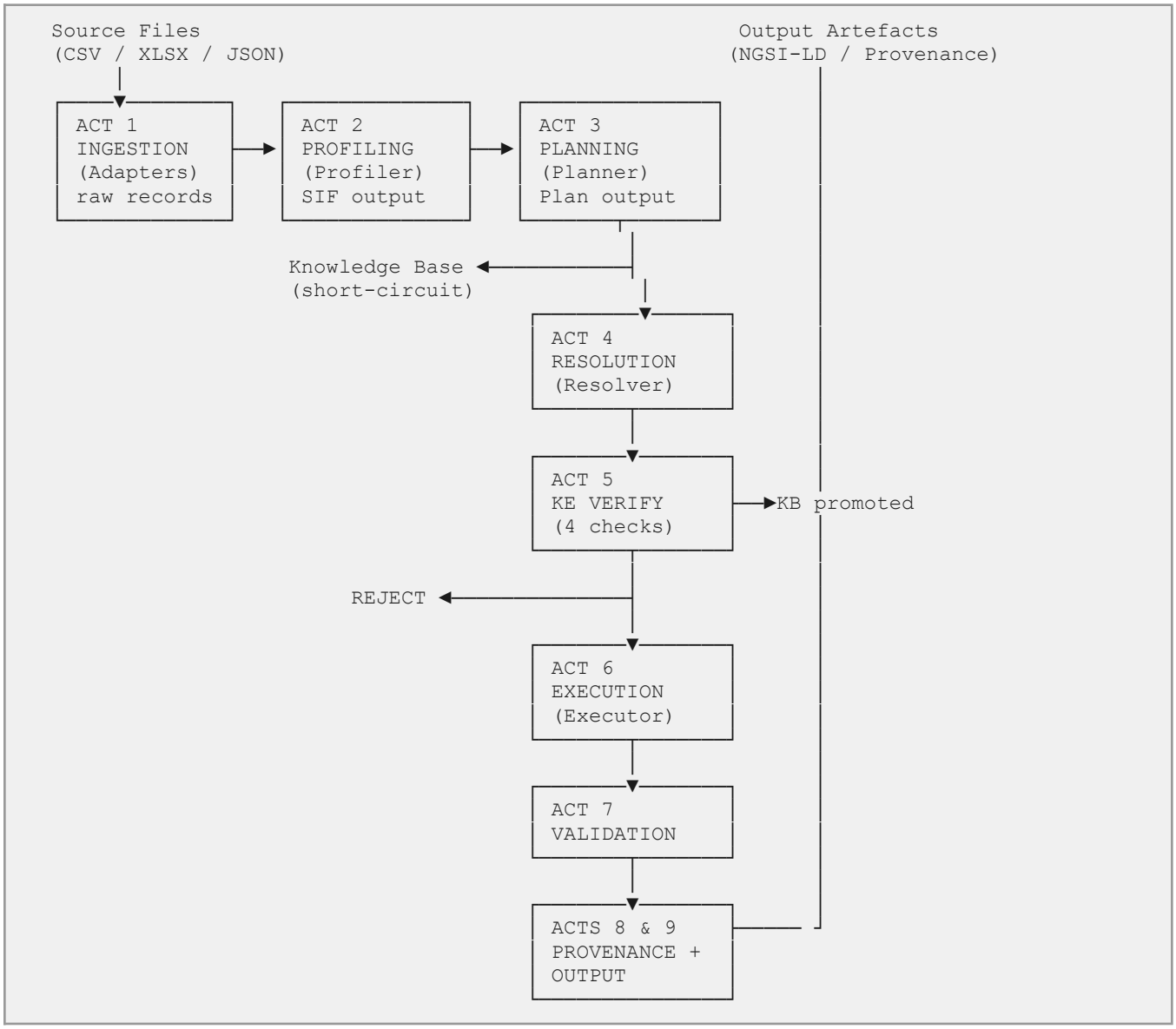
3.7 Layer 7 — Output & Provenance

The output layer persists all artefacts produced by the pipeline and provides human-readable reporting. The Knowledge Base is also updated at this stage with any newly promoted mappings.

Artefact	Contents
output/harmonized.json	Array of NGSI-LD records ready for ingestion into FIWARE-compatible platforms
output/provenance.json	One ProvenanceRecord per output record — full audit trail from source row to final verdict
knowledge_core/mapping_kb.json	Updated Knowledge Base with newly promoted MappingEntry objects
Console display (display.py)	Rich-formatted terminal output showing each Act's progress and key metrics

4. Data Flow — The 9-Act Pipeline

Every data source is processed through nine sequential Acts. Each Act has a single responsibility, a defined input contract, and a defined output contract. The orchestrator (demo/run_demo.py) executes the Acts in order and passes artefacts between them.



4.1 ACT 1 — Ingestion

Field	Value
Component	Adapters
Input	sources/{name}/manifest.yaml + data file (CSV, XLSX, or JSON)

Field	Value
Process	The factory function get_adapter() selects CSVAdapter, ExcelAdapter, or JSONAdapter based on the manifest format field. The adapter reads the file using pandas, respecting format-specific parameters (delimiter, encoding, sheet name). The manifest column definitions inform the Profiler in the next stage.
Output	list[dict[str, Any]] — one dict per source row, with all column values as Python native types

4.2 ACT 2 — Profiling

Field	Value
Component	Profiler Agent
Input	Raw records (list of dicts) + manifest metadata
Process	The Profiler batches all source fields into a single LLM call. For each field it produces: dtype, nullable flag, unique_count, sample_values (up to 10), format_hints (e.g., 'ISO8601'), semantic_hints (e.g., 'air quality measurement'), and unit_hint. The Profiler does NOT propose mappings; it only characterises the data.
Output	SourceIntelligenceFile (SIF) — a Pydantic model containing one FieldProfile per source field

4.3 ACT 3 — Planning

Field	Value
Component	Planner Agent (primary LLM tier)
Input	SIF + canonical schema properties + Knowledge Base promoted entries
Process	For each field, the Planner first checks the KB for a promoted (source_name, source_field) entry. On a hit, it reuses the mapping (KB short-circuit, zero LLM cost). For unknown fields, it batches them into one primary-tier LLM call requesting: target_field, transformation_type, transformation_params, confidence, and evidence. Confidence routing: >= 0.85 ⟶ INFERRED; 0.60–0.84 ⟶ UNCERTAIN; < 0.60 ⟶ UNMAPPED.
Output	HarmonizationPlan with status-tagged FieldMapping for every source field

4.4 ACT 4 — Resolution

Field	Value
Component	Resolver Agent (primary LLM tier)
Input	UNCERTAIN-status FieldMappings + full SIF + canonical schema + available vocabularies

Field	Value
Process	If any UNCERTAIN mappings exist, the Resolver issues one primary-tier LLM call with richer context than the Planner: full sample values, all format_hints and semantic_hints, and the contents of relevant vocabulary files. The Resolver produces a refined decision for each UNCERTAIN field: INFERRED, UNMAPPED, or REVIEW. The plan is patched in-place.
Output	Updated HarmonizationPlan with UNCERTAIN fields resolved to INFERRED, UNMAPPED, or REVIEW

4.5 ACT 5 — Knowledge Engine Verification

Field	Value
Component	KnowledgeEngine + PromotionManager
Input	FieldMapping list from the HarmonizationPlan
Process	For each non-UNMAPPED mapping, the KE runs four deterministic checks in sequence: (1) schema_fit — target_field exists in canonical schema; (2) vocabulary_membership — if a vocabulary applies, the mapped value is valid; (3) unit_compatibility — a conversion path exists; (4) provenance_completeness — all required audit fields are populated. Decision: APPROVE (all pass), REJECT (schema_fit failed), or ESCALATE (other check failed). APPROVE triggers the PromotionManager: if confidence >= 0.85, the entry progresses to 'promoted' status in the KB.
Output	List[KEVerdict] per field + updated mapping_kb.json

4.6 ACT 6 — Execution

Field	Value
Component	Executor + Transform Registry
Input	Raw records + verified HarmonizationPlan + canonical schema
Process	The Executor iterates every raw record. For each APPROVE or ESCALATE mapping it looks up the transform function in the TRANSFORM_REGISTRY by name, constructs the parameter dict (including partner-field injection for combine_lat_lon), and calls fn(value, params). Per-field exceptions are caught and stored as {_transform_error, _original_value} rather than failing the record. After all transforms, required NGSI-LD fields not yet present are injected: id as urn:ngsi-ld:{type}:{uuid4()}, type from the schema's first enum value.
Output	list[dict[str, Any]] harmonised records in NGSI-LD shape

4.7 ACT 7 — Validation

Field	Value
Component	Validator + SDMClient (optional)
Input	Harmonised records + canonical JSON Schema
Process	The Validator runs jsonschema Draft7 validation on every record locally. Each record receives a ValidationResult (passed: bool, errors: list[str]). When --mcp-validate is active, an additional cross-check call is made to the SDM MCP server's validate_data tool for each record. Both local and MCP results are combined.
Output	list[ValidationResult] — one per record with pass/fail status and error details

4.8 ACT 8 — Provenance Capture

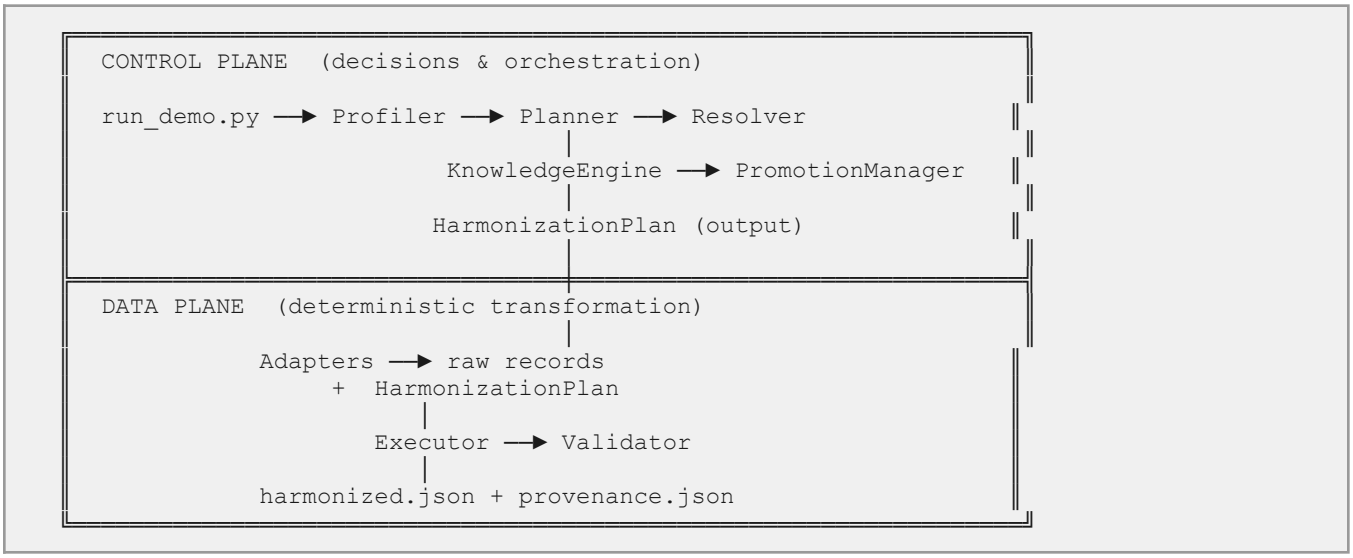
Field	Value
Component	Provenance module
Input	Decisions from all prior Acts + validation results
Process	A ProvenanceRecord is constructed for every output record capturing: source_file, source_row_index, source_fields_used, mappings_applied (KB entry UUIDs), transforms_applied (name:source->target(params) strings), ke_verdict, validator_result (PASS / FAIL), agent_versions dict, and ISO-8601 timestamp.
Output	output/provenance.json — array of ProvenanceRecord objects

4.9 ACT 9 — Output Persistence

Field	Value
Component	Orchestrator
Input	Harmonised records + ProvenanceRecords + updated KB state
Process	All artefacts are written to the output directory: harmonized.json, provenance.json, and pipeline_report.md. The updated mapping_kb.json (with newly promoted entries) is written back to knowledge_core/. A rich console summary is printed.
Output	Final persisted artefacts + enriched Knowledge Base ready for the next run

5. Control Plane vs. Data Plane

The system is divided into two orthogonal planes of operation: the Control Plane, which governs decisions and orchestration, and the Data Plane, which performs deterministic record-level processing. This separation means the Data Plane is fully reproducible and LLM-independent: given the same HarmonizationPlan, the same input records always produce the same output.



5.1 Control Plane

The Control Plane governs WHAT happens and WHEN. It does not touch source records directly; it produces the plan that the Data Plane executes.

Component	Decision it owns
run_demo.py (Orchestrator)	Sequences the 9 Acts; passes artefacts between stages
Profiler Agent	Which semantic hints and unit hints are attached to each field
Planner Agent	Which target field each source field maps to; at what confidence; via which transform
Resolver Agent	Final determination for UNCERTAIN fields
Knowledge Engine	APPROVE / REJECT / ESCALATE for each proposed mapping
PromotionManager	When a mapping graduates to 'promoted' status in the KB
SchemaSelector	Which canonical Smart Data Model schema to use for the pipeline run
config/settings.yaml	Numeric thresholds and behavioural rules for all of the above

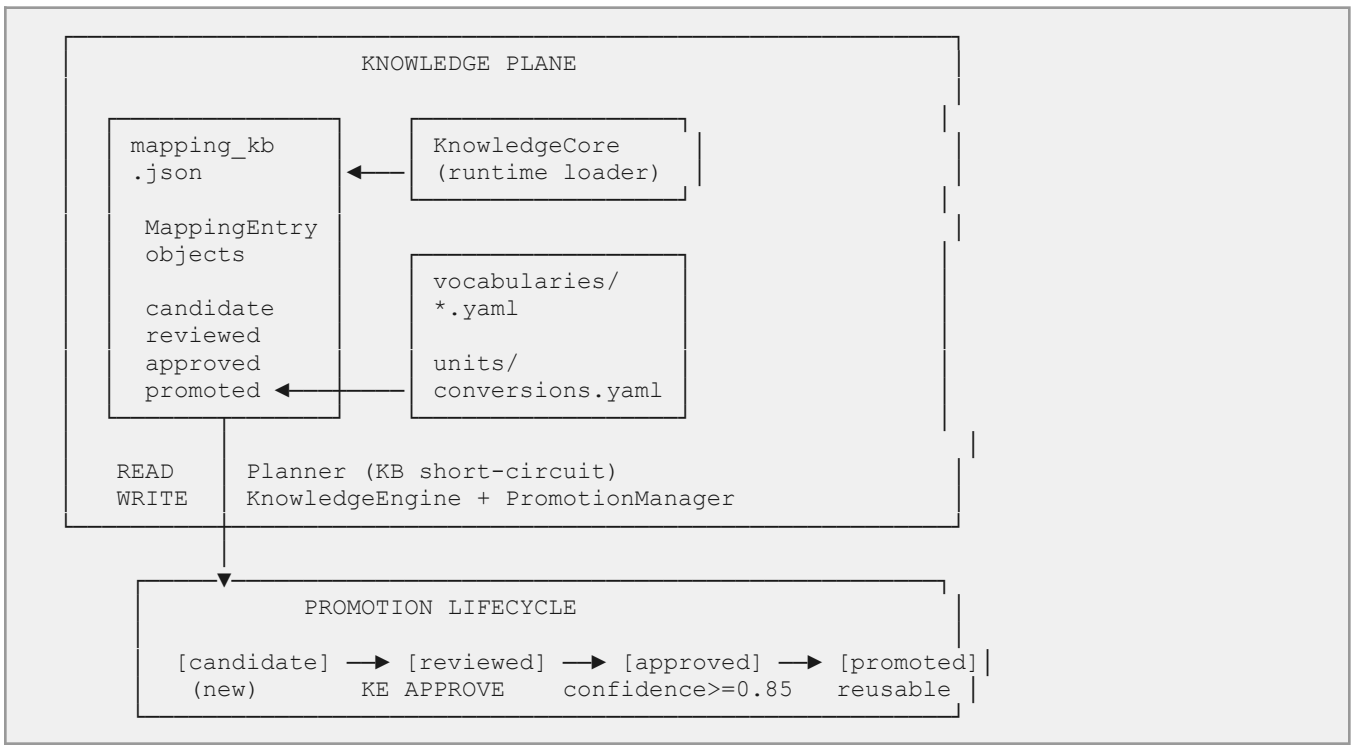
5.2 Data Plane

The Data Plane is deterministic and LLM-free. Given the same HarmonizationPlan and the same raw records, it always produces identical output. This property enables reproducibility auditing and makes the system safe for production data pipelines.

Component	Responsibility
Adapters (Ingestion)	Read raw source files into uniform list[dict]
Executor	Apply the verified HarmonizationPlan transform by transform, row by row
Transform Registry (9 functions)	Stateless pure functions: rename, convert units, parse dates, normalise text, GeoJSON, etc.
Validator	Assert each harmonised record against the canonical JSON Schema
Provenance module	Record all decisions, transforms, and verdicts for every output row

6. Knowledge Plane

The Knowledge Plane is a cross-cutting subsystem that spans both the Control Plane (read path) and the Quality Gate (write path). It provides the system's long-term memory, enabling self-improvement across pipeline runs. As more sources are processed, the Knowledge Base grows, the KB short-circuit eliminates more LLM calls, and the cost-per-record approaches zero for stable source formats.



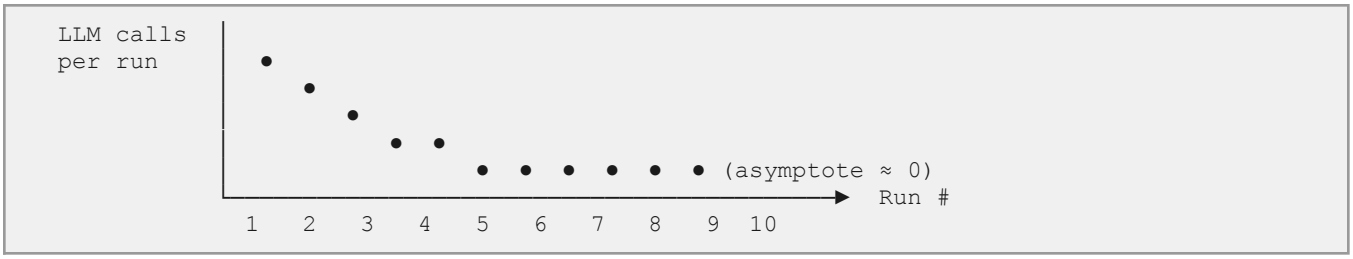
6.1 Knowledge Base Entry Lifecycle

Every field mapping begins as a candidate and may progress to promoted through the following states. Only promoted entries participate in the KB short-circuit.

State	Trigger	Effect
candidate	Planner proposes a new mapping not yet in KB	Entry created; not reused on next run
reviewed	Knowledge Engine returns APPROVE verdict	Entry flagged as valid; awaits confidence check
approved	confidence >= auto_approve_threshold (0.85)	Entry ready for promotion; auto-approved without human review
promoted	PromotionManager confirms approval	Entry is reused by Planner on future runs (KB short-circuit); LLM call skipped

6.2 Self-Learning Behaviour

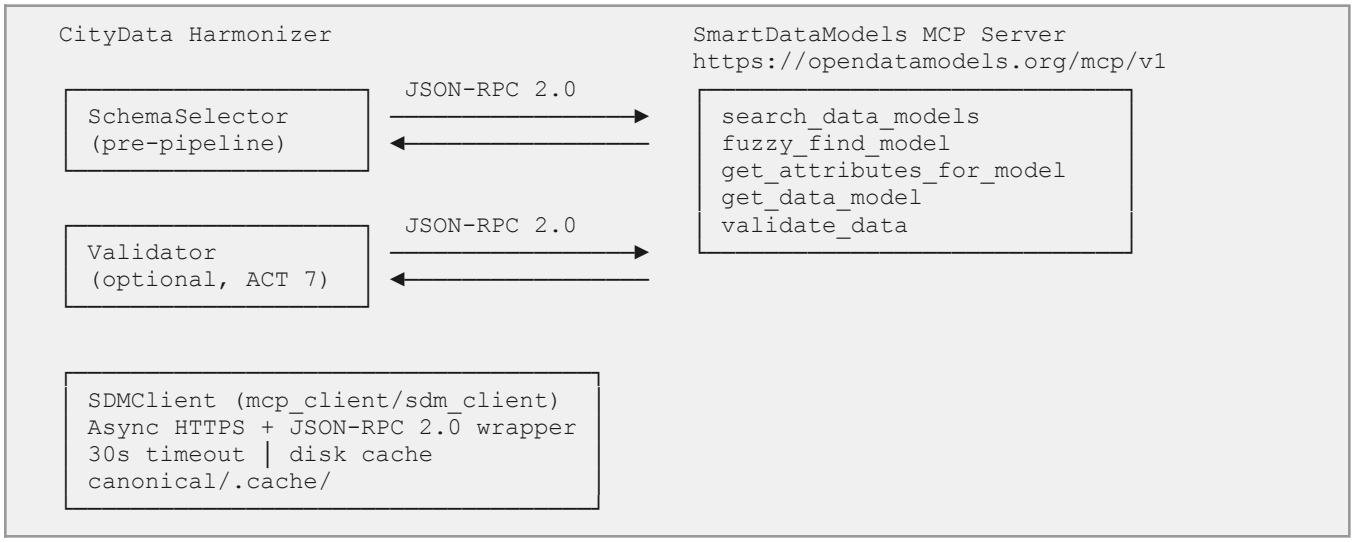
The Knowledge Plane implements a convergence property: each pipeline run enriches the KB, which makes subsequent runs faster and cheaper. The graph below illustrates the expected cost trajectory over repeated runs on the same set of sources.



On the first run for a new source, every field is unknown and goes through the full LLM chain (Profiler + Planner + Resolver). On the second run of the same source, all fields with confidence >= 0.85 are served from the KB. By the third or fourth run, the marginal LLM cost is typically zero for stable sources.

7. MCP Integration Plane

The Model Context Protocol (MCP) is a JSON-RPC 2.0 standard that allows LLM agents to call external tool servers. CityData Harmonizer integrates with the FIWARE SmartDataModels MCP server to enable two capabilities: automatic canonical schema discovery (SchemaSelector) and optional server-side record validation (Validator).



7.1 MCP Tools Used

Tool	Called by	Purpose
search_data_models	SchemaSelector	Keyword search of the SDM catalogue to find candidate schemas
fuzzy_find_model	SchemaSelector	Fuzzy fallback when keyword search returns no results
get_attributes_for_model	SchemaSelector	Fetch attribute list for top candidate schemas to compare coverage
get_data_model	SchemaSelector / SDMClient	Fetch full JSON Schema for a specific model; fallback if GitHub fetch fails
validate_data	Validator (--mcp-validate)	Server-side cross-validation of a harmonised record against the authoritative schema

7.2 SchemaSelector 5-Step Pipeline (pre-pipeline Act 0)

Step	Action
1. Characterise	LLM (primary tier) summarises all sources → entity_nouns, domain, key_concepts
2. Build queries	LLM generates 3–5 search query strings for the SDM catalogue based on characterisation
3. MCP search	SDMClient calls search_data_models for each query; collects candidate model identifiers
4. Enrich	SDMClient calls get_attributes_for_model for top candidates; fetches attribute lists
5. Rank	LLM selects best match by comparing attribute overlap with source field semantics; caches schema to canonical/.cache/

7.3 SDMClient Architecture

Property	Value
Protocol	HTTPS + JSON-RPC 2.0
Server URL	https://opendatamodels.org/mcp/v1
Timeout	30 seconds per call
Schema fetch strategy	1) MCP get_data_model 2) GitHub raw URL (raw.githubusercontent.com) 3) Fallback: MCP response property list
Caching	Fetched schemas cached to disk under canonical/.cache/ for reproducibility and offline re-runs

Note: The MCP integration is available in a separate branch of the codebase. Although the implementation was completed successfully, we decided to defer the official integration due to issues with the SDM MCP server.

8. Component Architecture

This section provides the complete inventory of all system components, their source locations, and their inter-dependencies.

Component	Source Path	Depends On
Orchestrator	demo/run_demo.py	All components
CSVAdapter	adapters/csv_adapter.py	pandas, manifest.yaml
ExcelAdapter	adapters/excel_adapter.py	pandas, openpyxl, manifest.yaml
JSONAdapter	adapters/json_adapter.py	pandas, manifest.yaml
Adapter Factory	adapters/__init__.py	CSVAdapter, ExcelAdapter, JSONAdapter
LLMClient	agents/llm_client.py	OpenRouter API, config/llm.yaml
Profiler	agents/profiler/profiler.py	LLMClient, SourceIntelligenceFile model
Planner	agents/planner/planner.py	LLMClient, KnowledgeCore, HarmonizationPlan model
Resolver	agents/resolver/resolver.py	LLMClient, HarmonizationPlan model
SchemaSelector	agents/schema_selector/schema_selector.py	LLMClient, SDMClient
KnowledgeCore	knowledge_core/loader.py	mapping_kb.json, vocabularies/, units/conversions.yaml
KnowledgeEngine	engine/knowledge_engine.py	KnowledgeCore, HarmonizationPlan, canonical schema
PromotionManager	engine/promotion.py	KnowledgeCore, KEVerdict
Executor	executor/executor.py	Transform Registry, HarmonizationPlan, canonical schema
Transform Registry	executor/transforms.py	pint, dateutil
Validator	validator/validator.py	jsonschema, SDMClient (optional)
SDMClient	mcp_client/sdm_client.py	aiohttp/httpx, JSON-RPC 2.0
Provenance	provenance/provenance.py	ProvenanceRecord model
Schema Utils	utils/schema.py	jsonschema, canonical schema

9. Data Structures Reference

All inter-component contracts are expressed as Pydantic models. This section documents each model's fields and the Act in which it is produced.

9.1 FieldProfile

Produced by: Profiler (ACT 2). One instance per source column.

Field	Type	Description
name	str	Source column name as it appears in the raw file
dtype	str	Pandas dtype string (e.g., object, float64, int64, datetime64)
nullable	bool	True if any null / NaN values exist in the sample
unique_count	int	Number of unique values observed in the full column
sample_values	list[Any]	Up to 10 representative raw values shown to the LLM
format_hints	list[str]	LLM-inferred format clues (e.g., 'ISO8601', 'Unix timestamp ms')
semantic_hints	list[str]	LLM-inferred semantic clues (e.g., 'air quality measurement')
unit_hint	str | None	LLM-inferred measurement unit (e.g., 'fahrenheit', 'ppb')
value_range	dict | None	{min, max} for numeric fields; null otherwise

9.2 SourceIntelligenceFile (SIF)

Produced by: Profiler (ACT 2). One instance per data source.

Field	Type	Description
source_name	str	Stable source identifier from manifest.yaml name field
source_file	str	Absolute or relative path to the data file
format	str	One of: csv, xlsx, json
row_count	int	Total number of rows in the source file
fields	list[FieldProfile]	One FieldProfile per source column
profiled_at	str	ISO-8601 timestamp of when profiling completed

9.3 FieldMapping

Produced by: Planner (ACT 3), updated by Resolver (ACT 4). One instance per source field.

Field	Type	Description
source_field	str	Name of the field in the source data
target_field	str | None	Name of the property in the canonical schema; None if UNMAPPED
status	str	One of: KNOWN (from KB), INFERRED (>=0.85), UNCERTAIN (0.60-0.84), UNMAPPED (<0.60)
transformation_type	str | None	One of the 9 registered transform names; None if UNMAPPED
transformation_params	dict	Parameters passed to the transform function (e.g., {from_unit, to_unit})
confidence	float	Score in [0, 1] expressing the Planner's or Resolver's certainty
evidence	list[str]	Human-readable justifications for the mapping decision
kb_entry_id	str | None	UUID of the KB MappingEntry used (KNOWN status) or None

9.4 HarmonizationPlan

Produced by: Planner (ACT 3), updated by Resolver (ACT 4). One instance per data source per pipeline run.

Field	Type	Description
source_name	str	Matches the source manifest name
target_schema	str	Identifier or path of the canonical schema being targeted
mappings	list[FieldMapping]	One FieldMapping per source field
created_at	str	ISO-8601 creation timestamp

9.5 MappingEntry (Knowledge Base Record)

Persisted in: knowledge_core/mapping_kb.json. One instance per verified source field mapping.

Field	Type	Description
id	str (UUID)	Unique identifier for this KB entry; referenced in ProvenanceRecord
source_name	str	Stable source name from manifest; primary KB lookup key
source_field	str	Source column name; combined with source_name for lookup

Field	Type	Description
target_field	str	Canonical schema property name
transformation_type	str	Verified transform name
transformation_params	dict	Verified transform parameters
confidence	float	Confidence score at time of promotion
evidence	list[str]	Justification strings preserved from the original agent call
status	str	Current lifecycle state: candidate | reviewed | approved | promoted
created_at	str	ISO-8601 timestamp of entry creation
updated_at	str	ISO-8601 timestamp of last status change

9.6 KEVerdict

Produced by: Knowledge Engine (ACT 5). One instance per non-UNMAPPED FieldMapping.

Field	Type	Description
field	str	Source field name this verdict applies to
decision	str	One of: APPROVE (all checks passed), REJECT (schema_fit failed), ESCALATE (other check failed)
checks_passed	list[str]	Names of KE checks that passed
checks_failed	list[str]	Names of KE checks that failed
reason	str	Human-readable explanation of the decision

9.7 ProvenanceRecord

Produced by: Provenance module (ACT 8). One instance per harmonised output record.

Field	Type	Description
output_record_id	str (UUID)	Matches the id field in the harmonised NGSI-LD record
source_file	str	Source directory or file path
source_row_index	int	Zero-based row index in the original source file
source_fields_used	list[str]	All source field names that contributed to this record
mappings_applied	list[str]	KB entry UUIDs for all active mappings applied

Field	Type	Description
transforms_applied	list[str]	Human-readable transform descriptions: name:src->tgt(params)
ke_verdict	str	Overall KE decision for this record: APPROVE | ESCALATE | REJECT
validator_result	str	PASS or FAIL from the Validator
timestamp	str	ISO-8601 timestamp when provenance was captured
agent_versions	dict[str,str]	Version strings for each agent: {profiler, planner, resolver, ...}

10. Transform Registry

All transform functions are registered in executor/transforms.py under TRANSFORM_REGISTRY. Every function is a pure stateless function with the signature fn(value: Any, params: dict[str, Any]) -> Any. They are called by the Executor in ACT 6.

Transform Name	Input -> Output	Key Parameters	Notes
rename_field	Any -> Any	none	No-op value pass-through; the Executor writes the value to the target_field name
convert_unit	number -> number	from_unit, to_unit	Primary: pint library (handles offset units such as Fahrenheit to Celsius). Fallback: factor from conversions.yaml
map_value	str -> str	mapping (dict)	Categorical lookup; raises KeyError if value not in dict (caught by Executor)
parse_date	str -> str (ISO-8601)	format (strptime string)	Falls back to dateutil.parser for ambiguous formats; output is always ISO-8601
normalize_text	str -> str	lowercase (bool)	Strips whitespace, applies NFC Unicode normalisation, optionally lowercases
restructure	Any -> dict	none	Wraps value in NGSI-LD Property envelope: {type: 'Property', value: <input>}
parse_geojson	str -> dict	none	Parses 'lat, lon' or 'lat lon' string into GeoJSON Point {type, coordinates: [lon, lat]}
combine_lat_lon	number -> dict	partner_field, partner_value	Combines separate lat/lon column values into GeoJSON Point; partner_value injected by Executor
normalize_range	number -> number	in_min, in_max, out_min, out_max	Linear interpolation: maps value from [in_min, in_max] to [out_min, out_max]

11. Configuration Reference

11.1 config/settings.yaml

Controls all numeric thresholds, KE check list, profiler batch sizes, and MCP settings.

```
thresholds:
  auto_approve: 0.85       # confidence >= this -> INFERRED, auto-promote
  review_required: 0.60    # confidence < this -> UNMAPPED

knowledge_engine:
  checks:                  # run in this order
    - schema_fit
    - vocabulary_membership
    - unit_compatibility
    - provenance_completeness

profiler:
  sample_size: 10          # sample values per field shown to LLM
  max_fields_per_call: 20  # batch size per LLM call

mcp:
  smartdatamodels:
    enabled: false         # true when --auto-schema flag is set
    server_url: 'https://opendatamodels.org/mcp/v1'
    timeout_seconds: 30
    cache_dir: 'canonical/.cache'
```

11.2 config/llm.yaml

Configures the LLM provider, API key environment variable, and model tiers.

```
provider: openrouter
api_key_env: OPENROUTER_API_KEY    # environment variable name

models:
  primary:  'deepseek/deepseek-v3'          # deep reasoning (Planner, Resolver,
SchemaSelector)
  fast:     'mistral/mistral-small'          # quick analysis (Profiler)
  fallback: 'google/gemini-2.5-flash'        # ultra-low cost fallback

parameters:
  temperature: 0.1    # near-deterministic output
  max_tokens: 4096
```

11.3 Manifest YAML (sources/{name}/manifest.yaml)

Required for every data source. The name field is the stable KB lookup key and must not be changed between runs.

```yaml
name: 'Valencia Air Quality Station'   # STABLE identifier used as KB key
format: csv                            # csv | xlsx | json
encoding: utf-8
filename: air_quality_2025.csv         # optional; auto-detected if omitted
delimiter: ','                         # CSV only
sheet_name: 'Data'                     # Excel only
description: 'Hourly air quality readings from Valencia monitoring network'
owner: 'City of Valencia - Environment Dept'
date_columns: ['datetime']             # columns to parse as dates
columns:
  - name: datetime
    description: 'Measurement timestamp in local time'
    type: object
    unit: null
  - name: so2
    description: 'Sulphur dioxide concentration'
    type: float64
    unit: micrograms_per_m3
    unit_code: GQ
```

11.4 CLI Reference (demo/run_demo.py)

Argument	Default	Description
--sources	required	One or more source directory paths; each must contain manifest.yaml and a data file
--schema	canonical/schema.json	Path to canonical JSON Schema file; overridden by --auto-schema
--settings	config/settings.yaml	Path to settings YAML
--llm-config	config/llm.yaml	Path to LLM configuration YAML
--knowledge-core	knowledge_core/	Path to the knowledge_core directory
--output	output/	Directory where harmonized.json, provenance.json, and report are written
--auto-schema	disabled	Activates SchemaSelector to discover canonical schema via MCP before the pipeline runs
--mcp-validate	disabled	Enables MCP cross-validation in ACT 7; requires --auto-schema to be active
--domain-hint	none	Free-text hint to SchemaSelector (e.g., 'parking', 'weather observation')

12. Appendix — Full Directory Structure

```
citydata-harmonizer/
├── config/
│   ├── settings.yaml          # thresholds, KE checks, MCP settings, batch sizes
│   └── llm.yaml               # LLM provider, model tiers, API key env var
│
├── canonical/
│   ├── schema.json            # Active canonical FIWARE Smart Data Model
│   └── .cache/                # MCP-fetched schema cache (auto-populated)
│
├── sources/
│   ├── source_a/              # Real air quality CSV data + manifest
│   ├── source_b/              # Real weather Excel data + manifest
│   ├── dev_source_a/          # Development placeholder source
│   └── dev_source_b/          # Development placeholder source
│
├── adapters/
│   ├── base.py                # Abstract SourceAdapter base class
│   ├── csv_adapter.py
│   ├── excel_adapter.py
│   ├── json_adapter.py
│   └── __init__.py            # get_adapter() factory
│
├── agents/
│   ├── llm_client.py          # Shared async OpenRouter client
│   ├── profiler/profiler.py
│   ├── planner/planner.py
│   ├── resolver/resolver.py
│   └── schema_selector/schema_selector.py
│
├── engine/
│   ├── knowledge_engine.py    # 4-check deterministic quality gate
│   └── promotion.py           # KB entry lifecycle state machine
│
├── executor/
│   ├── executor.py            # Row-by-row transform application
│   └── transforms.py          # 9 pure transform functions + TRANSFORM_REGISTRY
│
├── knowledge_core/
│   ├── loader.py              # KnowledgeCore runtime loader
│   ├── mapping_kb.json        # Persistent Knowledge Base
│   ├── vocabularies/          # Categorical synonym YAML files
│   └── units/conversions.yaml # Domain unit conversion factors
│
├── models/
│   ├── source_intel.py        # FieldProfile, SourceIntelligenceFile
│   ├── harmonization_plan.py  # FieldMapping, HarmonizationPlan
│   ├── mapping_entry.py       # MappingEntry (KB schema)
│   ├── ke_verdict.py          # KEVerdict
│   └── provenance_record.py   # ProvenanceRecord
│
├── validator/
│   └── validator.py           # jsonschema Draft7 validation + optional MCP
│
├── provenance/
│   └── provenance.py          # ProvenanceRecord construction
│
├── mcp_client/
│   └── sdm_client.py          # Async SDM MCP client
│
```

```
├── utils/
│   └── schema.py              # JSON Schema property extraction, $ref resolution
│
├── demo/
│   ├── run_demo.py            # Main orchestrator (CLI entry point)
│   ├── display.py             # Rich console display helpers
│   └── dashboard.py           # Optional Streamlit UI
│
├── output/
│   ├── harmonized.json        # Final NGSI-LD records
│   ├── provenance.json        # Audit trail
│   └── pipeline_report.md     # Execution summary
│
├── tests/                     # pytest test suite
├── scripts/                   # Utility scripts (e.g., probe_sdm_mcp.py)
├── requirements.txt
└── pytest.ini
```