

Auditable AI Data Analysis

Technical architecture · github.com/bluecatyellowdog/Analytical-pipeline-Agent

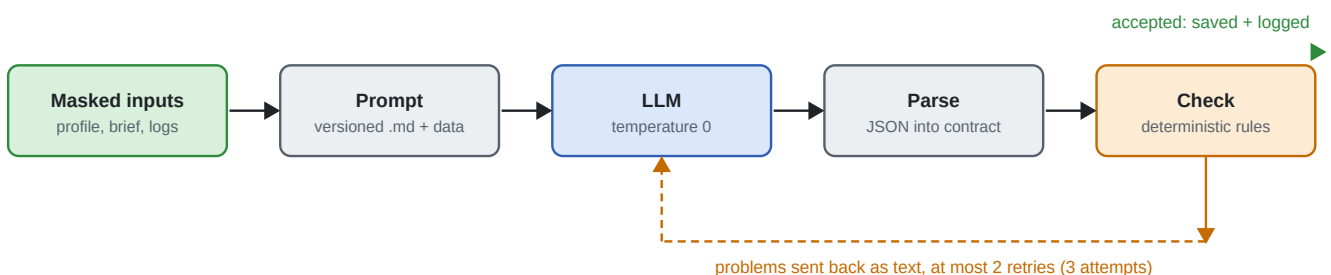
How the pipeline works inside: what each agent receives, how it works and is checked, what each gate tests, and where the limits are. Every limit quoted (rounds, retries, thresholds) is a constant in the code.

1 Principles and how they are enforced

Principle	Enforced by
The LLM decides, code executes	Agents only return JSON plans, tool requests or text. Data changes only through apply_op (library op or sandboxed custom_code); every number comes from a tool.
The LLM never sees rows	Agents receive the compact profile (pipeline/ingestion/compact.py), aggregate tool results and query answers. Personal values are reduced to their shape ('99999').
Every change is accounted for	apply_op hashes before and after, counts changed cells, new nulls and removed rows, and compares with what the step declared. Gate 2 reconciles and replays.
Raw data is read-only	Raw tables are hashed at load; verify() recomputes the hash. Every cleaning run starts again from raw.
No silent coercion	Files load as text; types are applied by logged cast steps; any new null needs a declared reason.
No silent judgment call	Each call is decided by the user, or by the AI from the brief with choice, confidence and reasoning, and can be changed in decisions.yaml.
Numbers in reports come from logs	Gate 4 traces every number in AI text to a logged tool result; reports are built from the run folder only.

2 The agent loop (shared by all five agents)

Every agent runs the same loop (pipeline/agents/common.py). The prompt is a versioned Markdown file plus the masked inputs; the reply must be JSON that parses into a typed pydantic contract (unknown fields are errors). Deterministic checks then test the content against the data and the brief. Problems go back to the model as plain text, at most twice.

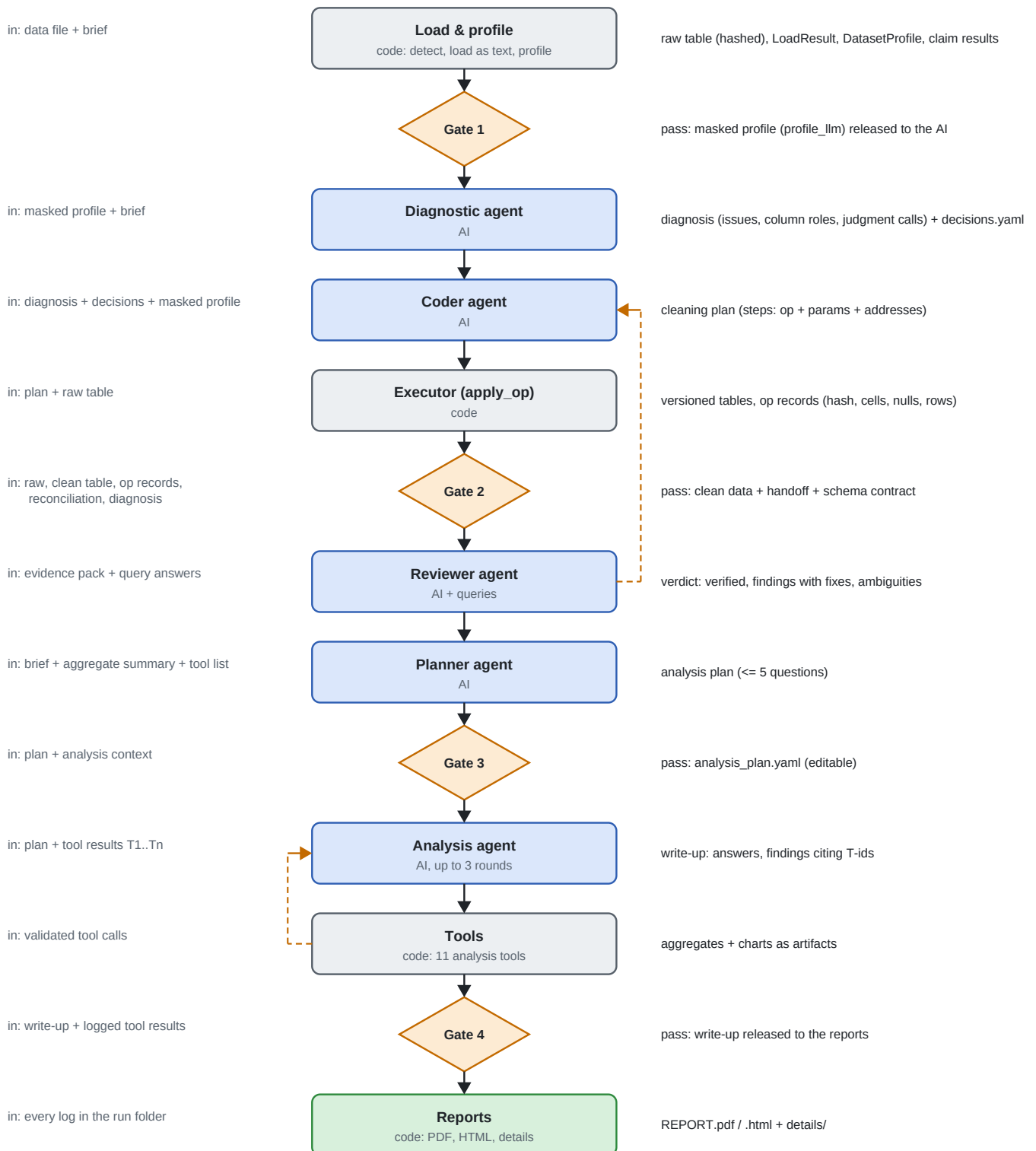


Every attempt (model, tokens, cost, seconds, problems) is logged; after 3 failed attempts the step stops with an error.

- **Models:** one per agent from settings.yaml, any OpenAI-compatible provider. On OpenRouter, calls are routed only to providers that neither store nor train on prompts.
- **Audit:** each attempt is logged with model, tokens, cost, time and problems; the manifest records the model and the prompt version per agent (a hash of the full system prompt, so it changes with the prompt, the op list or the output schema).
- **Tracing:** optionally, each call is also sent to Langfuse as a generation named after its agent.

3 Data flow: what each step hands on, and what each gate receives

Boxes: blue AI, grey code, green you; diamonds are gates. Left: what the step takes in. Right: what it hands on. Dashed: the reviewer's fix requests to the coder, and the analysis agent's tool rounds.



4 Loading, profiling and Gate 1 (code only)

Input	The data file and the brief.
Mechanism	Detects encoding and delimiter from the first 1 MiB (50 sample lines; delimiters , ; tab). Loads every value as text into DuckDB's raw schema and hashes it. Unparseable lines are rejected and stored (up to 1,000, 500 characters each). Profiles each column (sample up to 100,000 rows): inferred type, missing share, codes for missing values (e.g. ", NA, null, -), top 5 values, formats, outliers (3 x IQR), candidate keys, duplicates. Detects personal data by pattern (e-mail, Danish CPR with a valid date, phone, IBAN, IP; likely at 50%, possible at 10% of values). Tests each claim in the brief (e.g. 'customerID is unique') against the data.
Output	Raw table, LoadResult, DatasetProfile, claim results, and the compact profile for the AI: one or two lines per column (about 100-150 tokens), at most 3 examples, personal values masked to their shape.

Gate 1: is what was loaded faithfully the file that was handed in?

Check	Tests	Fails as
rows_accounted_for	Lines in the file = rows loaded + rows rejected	block
reject_rate_within_threshold	Share of rejected lines	flag above 0.1%, block above 1%
unique_column_names	No duplicate or blank header after normalising	block
provenance_hashes_present	File and raw-table hashes recorded	block
profile_matches_loaded_rows	The profile describes the rows that were loaded	block
type_coverage_or_flagged	Every column has an inferred type, or a flag saying why not	block
brief_claims_hold	Each checkable claim in the brief holds in the data	flag; the failed claim is shown in the report

A block stops the run before any AI call, so a broken file costs nothing.

5 Diagnostic agent

Input	The compact profile and the brief (description, goal, target, column notes, known issues, exclusions, guidance, judgment_calls mode).
Mechanism	Returns a Diagnosis: a role and target type for every column (feature, target, identifier, excluded or drop), issues (kind, column, evidence, suggested op) and judgment calls (question, options, and in 'decide' mode its own decision with confidence and reasoning). In 'decide' mode it takes the least destructive option with low confidence when the brief does not settle a call.
Checked by	check_diagnosis: every column listed exactly once and no unknown columns; issues and calls refer to real columns; suggested ops exist; at most one target, and it is the brief's; the target is never imputed; excluded columns get role 'excluded'; a column stored as text that should become a number or date has an issue that converts it; in 'decide' mode every call has a decision.
Output	diagnosis.json, and edit/decisions.yaml with each decision (ai_choice records the AI's; decision is what will be carried out). A user's change is never overwritten by a re-run.

Can

- Read a column's meaning from its name, examples and the brief, e.g. that 999 or a blank is a code for 'unknown'.
- Link problems across columns (11 blank TotalCharges, tenure starting at 0).
- Turn the user's context into methodological decisions, and say how sure it is.

Cannot / limits

- Sees only the masked profile: a problem visible only in a few individual rows can be missed.
- Its reasoning is checked for consistency, not for correctness: a wrong but well-formed decision passes, so it is shown in the report and marked when confidence is low.

6 Coder agent and the executor

Input	The diagnosis, the decisions, the compact profile and the brief; on a loop-back, the reviewer's fix requests and the previous plan.
Mechanism	Writes a CleaningPlan (up to 60 steps). Each step is a library op with parameters, or custom_code, and names the issues or decisions it addresses; anything not handled goes in 'skipped' with a reason. Library ops (15): cast, parse_dates, normalize_decimal, strip_currency, trim_whitespace, normalize_case, standardize_categories, tokens_to_null, replace_sentinel, impute, drop_duplicates, drop_rows, clip_outliers, drop_column, rename_column.
Checked before running	check_plan: steps only address real issues, decided calls or fix requests (never an undecided call); every issue with a suggested op and every decided call is addressed or skipped with a reason; the target is never imputed or dropped; step parameters validate against each op's schema.
Executor	pipeline/ops/plan.py runs the plan from raw through apply_op: each step writes a new versioned table; apply_op records hashes, cells changed, new nulls and removed rows (kept with their reason). A step that changes nothing is reported. A refused step stops the plan; the error goes back to the coder with data values masked. A plan that only drew warnings is kept as a fallback, so a worse retry cannot lose it.
custom_code	A transform(df) function that must declare every column it changes, adds or drops, and any new nulls or removed rows with a reason; apply_op refuses any difference between the declaration and what changed. It runs in a sandbox: a static check first (imports only from an allowlist such as pandas, numpy, re, math, datetime; no file, network, eval/exec, dunder access or pandas readers/writers), then a separate process with no secrets in its environment, restricted builtins, an audit hook blocking file writes, network and new processes, and a 120 s timeout.
Output	Clean table, op records (cleaning_log.json), cleaning_plan.json; the code of every step is stored, so the log can be replayed.

Can

- Handle the common problems with tested ops, and the long tail with its own code.
- Work from the decisions, so the user's choices are carried out exactly.

Cannot / limits

- The Python sandbox is a guard rail, not a security boundary: a determined attacker could escape it. The hard boundary would be a container with no network.
- Accounting proves the data is what the log says, not that each step was the right choice; that is the reviewer's and the user's job.

7 Reviewer agent: how the review works

Gate 2 proves the cleaning is accounted for; the reviewer looks for what is wrong anyway: a decision carried out with the wrong scope, day and month swapped, categories merged that should not be, problems left in the clean data.

Evidence pack	The brief, the diagnosis, the decisions, the plan with what each step actually changed (measured), Gate 2's result, the schema contract, and masked profiles of the raw and the clean data.
Queries	It may ask aggregate questions of the raw or clean table: count (rows matching conditions), value_counts, summary, group_count (also per year or month of a date) and compare (how raw values became clean ones). At most 8 queries per reply, 2 query rounds, 4 conditions per query, 20 result rows, 1,500 characters per answer. Queries are structured requests, never SQL: columns are checked and quoted, values bound as parameters. Personal or identifier columns are shown only as their shape and can be filtered only by is_null / is_not_null.
Verdict	verified (claims with evidence), findings (a problem, optionally with a fix: an op and parameters for the coder), ambiguities (questions for the user, never decided by the reviewer).
How the verdict is checked	Every verified claim and every finding must quote a number that appears in the evidence it was shown or in its query answers; claims that do not are dropped. This catches invented evidence, not a wrong reading of a real number. Findings must name real steps, columns and ops. At least 2 verified claims are needed, otherwise the review counts as weak (the run is flagged). 'accepted' is derived by code (no finding carries a fix), never taken from the model.
The loop	Fix requests go back to the coder, which writes a new plan from raw; Gate 2 runs again; the reviewer reviews again. At most 2 loop-backs. A budget (\$0.50 by default) stops the loop between rounds. If the coder cannot apply the fixes, or the fix makes Gate 2 block, the previous result is kept and the run is flagged.
Output	review_log.json with every round, query and answer; the review status (accepted / flagged) in the manifest and the report.

Can

- Check the cleaning with its own numbers, independently of the coder, ideally on a different model.
- Turn concrete problems into fixes, and raise open questions instead of guessing.

Cannot / limits

- The evidence check matches numbers, not meaning: a real number misread still passes.
- The second most expensive agent (25% of the Telco run's cost); its added value over Gate 2 is not yet measured.

8 Gate 2: is the clean data what the diagnosis said it should become?

Input: the raw table, the clean table, every op record, the reconciliation results, the plan, the diagnosis, the brief and the null tokens per column. Layers run in order: contract, invariant, judgment.

Check	Layer	Tests
schema_types_match_diagnosis	contract	Each column's DuckDB type fits the type the diagnosis set (DATE and TIMESTAMP told apart)
columns_match_diagnosis	contract	Columns kept, dropped and added as the diagnosis and the steps say
primary_key_intact	invariant	The key column is still unique and complete
target check	invariant	The target was never imputed
row_loss_within_threshold	invariant	Rows removed: flag above 5%, block above 20%
row_accounting	invariant	Raw rows = clean rows + rows each step removed
dropped_rows_have_reasons	invariant	Every removed row is stored with the step and reason

Check	Layer	Tests
cell_diff_accounted_for	invariant	Cells that differ from raw = cells the steps report changing
untouched_columns_unchanged	invariant	Columns no step touched are identical to raw
no_unexplained_new_nulls	invariant	Every new null was declared with a reason
replay_reproduces_output	invariant	Replaying the op log from raw gives the same table (hash)
distribution_shift_within_threshold	judgment	Population stability index per numeric column; flag above 0.2

Output when it does not block: clean_data.parquet/.csv, handoff.json (types, roles, clean profile) and schema_contract.yaml. On a block, none of these are written, and old ones are deleted first. What Gate 2 cannot show is that the decisions were right; that is what the reviewer and the user are for.

9 Planner agent and Gate 3

Input	The brief; a summary of the clean data built from aggregates only (no rows, no identifier or personal values); the tool list generated from the registry; the columns a check found to predict the target almost perfectly (possible leakage); the limitations cleaning recorded (up to 15).
Mechanism	Returns an AnalysisPlan: objective, task type (with reason), target, up to 5 questions (each with kind, columns, methods, priority 1-5, success criteria, feasible or an infeasible reason), excluded features with reasons, assumptions and limitations.
Output	edit/analysis_plan.yaml, approved automatically when Gate 3 passes. The user may edit it; an edited plan is checked by Gate 3 as the user's and never overwritten.

Gate 3: is the plan executable, safe and feasible on this data?

Check	Tests
columns_exist, methods_exist	Every column is in the clean data; every method is a registered tool
target_as_briefed, task_type_fits_target	The target is the brief's; the task type (e.g. yes/no vs number) fits it
usable_columns, exclusions_listed	No identifier, personal, free-text or excluded column is analysed; every exclusion is listed with its reason
methods_fit_target	Each method can run on this target
leaky_columns_excluded	A column predicting the target almost perfectly (strength 0.95 or more) is excluded
feasible	Enough rows: 30 overall, segments of 30 rows, 10 rows of the rarer outcome per regression feature; otherwise marked infeasible
something_to_do	At least one question can be answered
methods_fit_question	Whether each method suits its question's kind (a note only)

For the AI's plan every failure blocks and goes back to the planner (at most twice). For a plan the user edited, leakage and feasibility only warn: the user's judgment is kept.

10 Analysis agent, tools and Gate 4

Input	The approved plan (the user's edit if there is one) and the analysis context: the clean Parquet re-checked against its schema contract, the column roles and kinds.
Mechanism	Works in rounds (at most 3; at most 8 calls per round, 20 in total). Each round it requests tool calls with parameters and a reason; code validates and runs them and shows the results back under ids T1, T2, ... It then writes a WriteUp: a summary, an answer per plan question with its status, findings (headline, detail, evidence = T-ids, strength strong/moderate/weak/none, caveats), limitations, doubts, recommendations and next questions.
Tools (11)	summary_table, distribution, missingness, correlation, target_by_group (with own ranges), target_by_two_groups, target_by_ratio, trend_over_time, compare_by_target, linear_regression, logistic_regression (statsmodels). All return aggregates only and are logged as artifacts with charts.

Tool safeguards	A call is refused, with the reason sent back, if it uses the target, an ID, an excluded or personal column as a feature, or a column that does not exist. Groups under 10 rows are merged or hidden. A 'subset' is allowed only if it leaves 30 rows. A fit with R ² or AUC above 0.95, or a perfect prediction, is flagged as possible leakage.
-----------------	---

Gate 4: is the write-up complete, grounded and honest about what the tools found?

Check	Tests
every_question_handled	Every plan question has an answer; a question the plan marked infeasible stays infeasible
evidence_exists	Every finding cites tool calls that ran
numbers_traceable	Every number in the text is one the cited calls reported, read at the precision written (74% matches 0.742). A number the AI computed itself (a difference, a ratio) matches nothing and is reported. Whole numbers up to 10 and conventions (95%, 0.05, 0.01, 0.001) are exempt; a summary or limitation may use any logged number.
no_overclaiming	A finding resting only on models no better than no model (p 0.05 or more) cannot be called strong or moderate
model_warnings	Suspiciously good fits, too few rows, fits that did not settle: flagged at the top of the report
Failures block and go back to the agent (at most twice); warnings continue and are shown. A passed write-up becomes artifacts that point to their plan question and their calls.	

Can

- Follow up on what is interesting: a later round can split a finding by a subgroup or check it with a regression.
- Mark its own doubts, and say when a question cannot be answered with this data.

Cannot / limits

- Traceability checks the numbers, not the interpretation: a correct number in a misleading sentence passes.
- 'Goes with, never causes' is an instruction in the prompt, not yet a code check.
- Deliberately simple statistics: no cross-validation, no complex models.

11 Reports and the trust lights

Reports are rebuilt from the run folder only (pipeline/reports): one summary object drawn as REPORT.pdf and REPORT.html, plus the detail documents. Text written by the AI is marked; numbers come from the logs. The trust lights are set by fixed rules, never by the AI: **Data** is yellow when rows were rejected or removed or a column is missing 10% or more; **Checks** is red when a gate blocked and yellow when one flagged or the reviewer left points open; **Decisions** is yellow when a call is undecided or was decided by the AI with low confidence.

12 System-level limitations

- **Checks prove accounting and consistency, not correctness.** A wrong but documented decision, or a misleading sentence with correct numbers, can pass every gate; this is why decisions are shown and reversible.
- **What the AI sees is limited by design.** The masked profile hides rare row-level problems; the profiling and the reviewer's queries reduce, not remove, this risk.
- **Text from the data reaches the prompt.** Column names, category values and masked examples are shown to the AI, so data could carry instructions (prompt injection). Contracts, checks and the no-SQL query tool limit the damage; a dedicated test is still to do.
- **Not deterministic.** Temperature 0 reduces but does not remove variation between runs; repeated runs would measure stability.
- **Validation is informal so far** (two public datasets compared with common practice); a scored evaluation set and an ablation of each agent's contribution are the next steps.
- **Prototype scope.** Local, single user, files only; the custom-code sandbox is process-level; summaries go to the AI provider (and to Langfuse when tracing is on).