10 NOVEMBER 2026 / VERSION 0.9

MDIL Kide
Declared Reasoning AI Architecture

AI that reasons with declared knowledge.

Technical summary. Detailed methods are reserved for controlled evaluation.

Abstract

MDIL is a declared reasoning AI: its knowledge, relationships and permitted operations are explicit. It uses them to derive conclusions, explore possible developments and construct decisions, plans, language and data. The knowledge behind a result remains available to check, revise and use again. MDIL Kide is the first release of MDIL; where this summary says MDIL, it means this release. Current Core capabilities include declared reading, evidence reasoning, controlled decisions, attention, trajectories, shared-state projections, finite continuation, supported revision and language generation. Working demos and studies extend the approach to exact reconstruction and communication, and a model has carried out one compression and reconstruction task through the tool interface. MDIL is used in three steps: declare, reason, keep. A model or a person declares a task once. MDIL reasons, computing the answers in order by itself and checking each one. The result is kept, so that a change redoes only what moved and a later session, another AI model or a small local model can continue from it. This summary explains the relationship between these capabilities, their distinct result contracts and the opportunity created by retaining useful knowledge across operations.

1. Knowledge the AI can reason with

A fact can support a conclusion. A condition can change which actions are possible. Several facts can become a new explanation. A shared construction can recreate a much larger artifact. MDIL makes these relationships explicit and puts them to work.

A declaration specifies the relevant knowledge, relationships, conditions and permitted operations. A request supplies the task and current context. Supported reasoning operations derive a result from this basis, with its context and the evidence checked against its contract. The application can inspect what was produced and use the knowledge again.

A retained construction can answer questions about alternatives and continuations. Evidence can change which facts contribute to a conclusion. A changed fact can change which linguistic expression is appropriate. An installed component can give a small later transfer enough meaning to reconstruct a large artifact. The reusable asset is the structure that makes these operations possible.

A model can work through MDIL. It declares a task's structure once; MDIL reasons, computes the answers, checks them and keeps them; and a later session or another model continues from the published structure. Models can also help people develop declarations, interpret open requests or propose new material. Supported declared execution can operate without a model call, including on a local CPU. The application configures the placement of models; model-free execution is one capability of the architecture.

2. Several reasoning operations, connected

Attention asks what needs examination. A configured selection uses declared objectives and bounds to determine which candidates can affect the result. Its evidence distinguishes work performed from assumptions trusted. This is computational selection, distinct from Transformer attention.

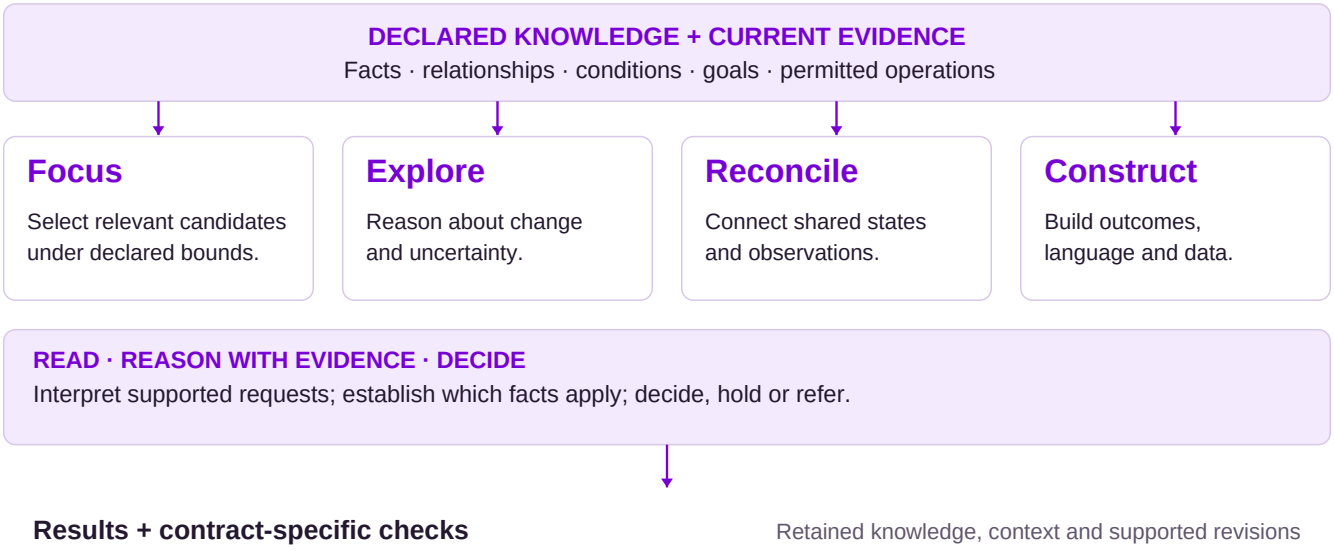
Evidence reasoning asks what applies. Facts carry status, validity and scope. Known, unknown, stale, conflicting and revoked information are distinct. A fact contributes when its conditions hold; disagreement is preserved rather than silently resolved by choosing the newest value.

Trajectories ask what could happen. Declared states and transitions support reasoning about possible developments and uncertainty. A policy that works with information available at decision time is different from a plan that becomes feasible only with hindsight. A prohibited action remains prohibited rather than becoming an unfavourable preference.

Projections ask whether views agree. Overlapping windows connect through explicitly shared state. An observation retains its identity across views, so duplication does not create independent evidence. A projected state remains distinct from an observed fact.

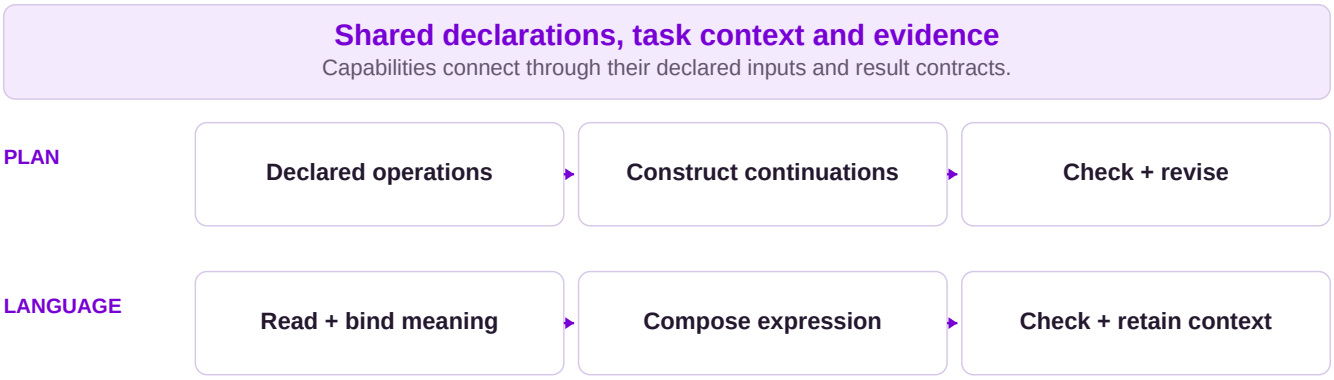
Construction asks what can be produced. Declared parts and operations support plans and sequences, linguistic expressions and exact reconstructed artifacts. Each form has its own result contract.

Reading and controlled decisions connect these operations to a request. A declared reader produces a supported explicit interpretation. Facts, relationships and priorities govern whether the application receives a decision, hold or review outcome. Accepted observations and declared context can be retained for later turns.



Architecture map. Different operations share declared knowledge and evidence; a task uses the capabilities it needs.

These operations combine according to the task. In current Core, the planning path constructs continuations, checks them and supports revision under tightened conditions. The language path binds a reading to evidence and meaning, constructs an expression, checks it through an independent reading and retains declared context. These paths share declarations and evidence while keeping their own construction and checking contracts. Reconstruction and communication retain their own working demo and study implementations. Compression and reconstruction has also been run by a model through the tool interface, and is part of the beta of 17 November 2026.



Two implemented paths. Other capabilities retain their own contracts and implementation forms.

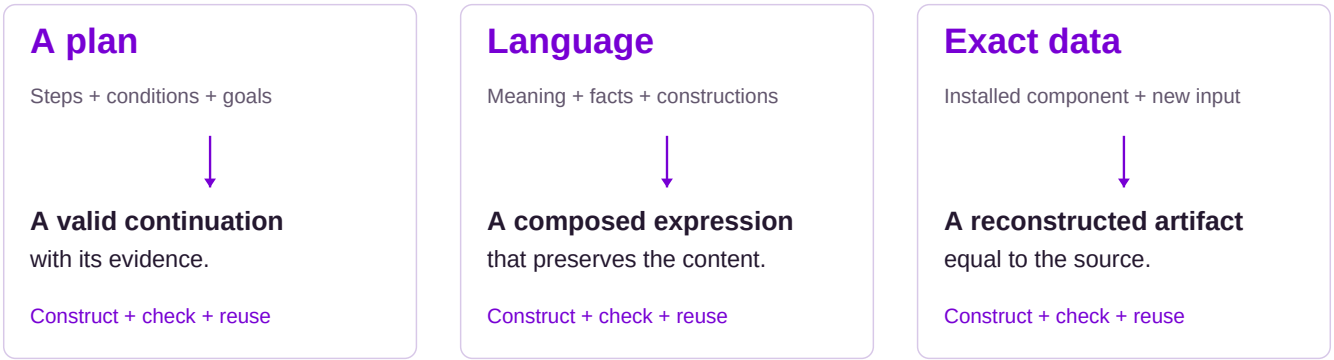
Implemented planning and language paths share declarations and evidence while retaining their own checks.

3. Three forms of generation

Constructive results. Operations, prerequisites, resources, goals and a horizon define possible continuations. MDIL can compute a valid continuation, count feasible alternatives and return a witness, or establish that no continuation meets the declared conditions. Retained construction records support further questions about remaining possibilities.

Declared language. Requested meaning, available facts and permitted constructions determine what can be said. In the recorded service-price example, equal prices produce “Pump Larch and Valve Juniper each cost €210 to service.” Different values require distinct statements. Expired or conflicting evidence changes the type of answer. An independent reading checks what the constructed expression means.

Exact reconstruction. Reusable components and new input reconstruct an artifact equal to its source. The installed structure is part of the receiver's prior knowledge and of the accounting. This contract is byte equality; the language contract concerns supported meaning, and the planning contract concerns a valid constructed outcome.



Three generation contracts: constructive results, declared language and exact reconstruction.

The common contribution is an explicit relationship between knowledge, permitted construction and a checkable result. A plan, a sentence and a file are different outputs. Each becomes reusable through the knowledge and operations that produce it.

4. Change and observation are part of the computation

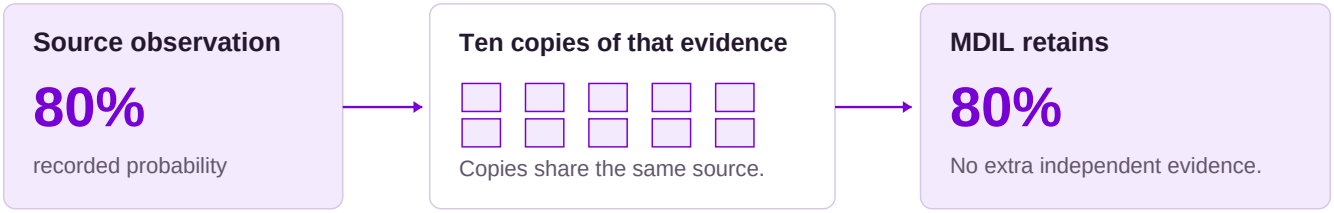
A useful capability must remain connected to the conditions that make its results valid. MDIL retains declarations, evidence and context so updates can be handled explicitly.

The current planning implementation supports repair when conditions tighten: an availability window closes, a cost rises, a requirement becomes stricter, a resource ceiling falls or a state becomes forbidden. It retains and updates construction records, with results matching full recomputation under the supported contract. Other changes, including a changed goal or operation set, explicitly require a fresh construction. Functional revision and a measured saving from revision are separate claims.

Between the members of a project, a change redoes only what it reaches: MDIL re-derives what depends on the changed fact and keeps the rest as stored. In one recorded task six changed readings moved a dependent count from 3 to 4 without the model recomputing it. In another, a second version of a 1.97 MB software repository needed 26 new objects beside 1,452 kept, 12,560 bytes where the first version took 401,983. Measured in Core on a supply chain of 12, 24 and 48 stations, a surge in customer demand was taken in a single call and only the stations whose incoming demand moved were computed again: 4 of 12, 14 of 24 and 35 of 48. Section 5 gives the chain of 48 stations run by a model.

Shared observations illustrate a different kind of reasoning about change. Copying one observation into several views must not make it several independent observations. In the saved synthetic example, the supported probability remains 80% after ten copies; an incorrect independence assumption would inflate it above 99.9999%. Across the study's 5,000 duplicate events, duplication caused no probability changes. Contradictory observations were rejected rather than averaged into the accepted state.

ONE OBSERVATION. SEVERAL VIEWS.



Treating the copies as independent would inflate this example above 99.9999%.

Recorded synthetic shared-observation example. Duplicate observations do not create independent evidence.

Retained context also has declared meaning. A repeated statement, a new observation and a projection are different inputs. The application can preserve these distinctions across turns instead of reconstructing the situation from an undifferentiated transcript.

5. Evidence that illuminates different capabilities

This section holds two kinds of evidence: tasks carried out by a model through the tool interface in November 2026, and recorded results by capability. Each picture below sets one result beside what it is measured against. The first three are one task, a calculation of 320 steps whose answer is 511 digits long, set as three jobs: build it, pick it up, change it. The jobs were done by the AI model alone, then by the same model through MDIL, and the work was then continued by a small local model. The last three are the serving measurement: the same twelve requests answered by a local AI model and by prepared MDIL.

The AI model alone

builds it · picks it up · changes it

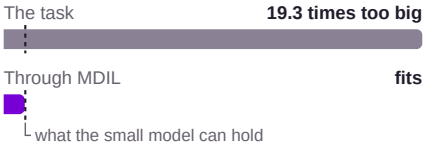
0 of 3 jobs right

The same model through MDIL

builds it · picks it up · changes it

3 of 3 jobs right, each checked by a separate program

Then a small local model



The task is 19.3 times its window of 4,096 tokens; through MDIL its whole session is 3,981 tokens. The checked answer came back in 8 seconds.

Time to return an answer



68 times faster (median, the same twelve requests)

Energy for each answer



99.5% less gross chip energy per answer attempted

Complete answers



12 of 12 against 6 of 12

Results at a glance. Grey is the AI model alone or a local AI model; violet is the same work through MDIL. Every number stands in this section beside what it is measured against.

Tasks run by a model through the tool interface

Claude Sonnet 5.5 carried out the tasks through the tool interface. In each, the comparison is the same model doing the same task alone. A separate program graded each run by reading the stored results, not the model's account of them.

The calculation task. A workflow of 320 steps whose answer is 511 digits long, set in three roles, each in its own session: one builds the structure, a second picks it up, a third changes it. Working alone, the model got none of the three roles right, in 24 turns and with 17,120 output tokens. Through MDIL the same model got three of three right, in 18 turns and 24 requests, with 8,715 output tokens: 25% fewer turns and 49% less writing, about half. The second session confirmed the published result, the 511-digit count included, in two turns and one request, in 9 seconds.

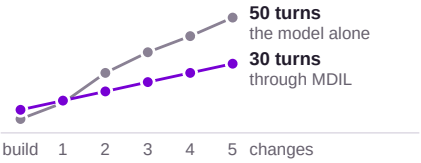
A small local model continued the work. The large model is needed once, to build the structure. Counted in the tokens of the small model tested, the raw calculation task is 79,075 tokens, 19.3 times that model's window of 4,096. Through MDIL the small model's whole session fits: its opening prompt measures 634 tokens, and the session read 3,981 tokens in all. The frontier model published the structure in a fresh session of 8 turns and 108 seconds. The small model took it up in two turns and one request, in 8 seconds, and returned the exact reference of the stored, checked answer. The separate grading program confirmed the answer behind that reference: the right cost and the 511-digit count.

Planning and evidence. A maintenance plan over three sessions: a first plan; ten incoming documents, four of them copies; then a capacity cut and a repair. Through MDIL three of three sessions were correct end to end, and 25 of 25 stored answers are sealed, so each number can be traced and checked afterwards. One request solved all eight members of the project, and six changed readings moved a dependent count from 3 to 4 without the model recomputing it.

A supply chain, built once and then changed. The chain has 48 work stations. Each station's order depends on what its neighbours order, so one change at the head moves the orders behind it. Through MDIL the session that built the chain took 10 turns, and each of the five changes after it was stated in one accepted request: 4, 4, 4, 4 and 4 turns, 20 in all. The same model working alone, with a shell of its own, also reached the right orders; it worked the chain out again for each change and took 7, 13, 9, 7 and 8 turns, 44 in all. Over the build and the five changes that is 30 turns through MDIL against 50. Stored data behaves the same way: the second version of the software repository of section 4 needed 26 new objects beside 1,452 kept.

THE SUPPLY CHAIN, RUN BY A MODEL.

Build once, then change



30 turns for the build and five changes through MDIL; the model alone 50

What a change costs



20 turns for five changes through MDIL; the model alone 44

Only what moved is redone

26 new objects beside 1,452 kept as they were

The supply chain, run by a model. Grey is the AI model alone, with a shell of its own; violet is the same model through MDIL. The last picture is the second version of the software repository of section 4.

Recorded results by capability

These examples answer different questions. Their population, result contract and comparison remain attached to the result; the system card supplies detailed methods and accounting.

Capability	Recorded result	What it establishes
Tasks through the tool interface	On a 320-step calculation Claude Sonnet 5.5, working alone, got none of the three roles right; through MDIL the same model got three of three, graded by a separate program that reads the stored results, in 18 turns against 24 and with about half the writing (November 2026; one run of the model alone and one through MDIL). In a planning-and-evidence task three of three sessions were correct and 25 of 25 stored answers are sealed.	The calculation done right where the model alone was wrong, in fewer turns, with every answer checked and traceable to a stored result.
Decisions with control	Twelve equipment-service cases produced four dispatch decisions and eight holds or reviews, matching independently defined outcomes.	Relationships and priorities determine the decision; a missing permit or hazard changes the permitted outcome.
Trajectories and information timing	Among 2,835 finite policy cases, 597 were robustly feasible and 66 appeared feasible only with hindsight; decisions matched explicit policy enumeration.	Future knowledge is not silently granted to a decision that must be made now.
Shared-state projections	151,257 overlapping-state comparisons had no state disagreement; 5,000 duplicate observation events caused no probability change.	Shared state and evidence retain their identity across windows and updates in the named studies.
Constructive generation	All 1,306,368 operator strings across 48 small instances were independently replayed, with no disagreement with compiled feasibility, counts and selected solutions.	The study checks a family of possible constructions, beyond one selected example.

Capability	Recorded result	What it establishes
Language serving	Prepared MDIL: 3.09 ms median returned response and 0.047 J gross chip energy per attempt; local Qwen2.5-1.5B: 209.14 ms and 9.30 J. That is 68 times faster, with 99.5% less energy. Complete answers: twelve of twelve versus six of twelve under one rule.	The cost of serving the same twelve explicit service-price requests after preparation, on one machine.
Reconstruction and reuse	A 944-file, 30.5 MB workspace restored from 3,804 newly transferred bytes, using a 401,983-byte component installed once.	Reusable structure turns a small subsequent transfer into exact larger output. The repository of the compression task in section 4 is the same base data, so the two results are one data set, not two.
Query-aware communication	67.05% fewer application bytes while preserving the tested query decisions.	Stable queries and retained synchronized state can change how much information needs to travel.

The serving measurement counts already interpreted requests after preparation. Open-language intake, preparation, updates and fallback belong in whole-application accounting. Its energy measure is gross chip energy per attempt. Other examples have their own tasks and do not inherit these timing or energy figures.

Working applications provide another form of evidence. Compression and small-to-large reconstruction demos recover exact output. An operational scenario demo explores steel-order plans, constraint trade-offs, event replay and replanning after changed resources. These are implemented demonstrations, distinct from a deployed industrial installation. Product owners identify the capabilities and environments demonstrated by REMAP and Airut.

6. From a reasoning capability to shared infrastructure

A developer, or an AI model working through the tool interface, declares the knowledge, conditions and desired result; MDIL reasons, computes the result, checks it and keeps it; the application uses the output and its evidence. An agent can declare a structure once, return to its stored results, state a change and publish the structure for another agent to continue. A structure that a frontier model declares is then used by a small local model inside a context window of 4,096 tokens: the raw calculation task of section 5 is 19.3 times that window, and through MDIL the small model's whole session fits in it. A service can compose grounded language. A planner can evaluate continuations. A receiver can reconstruct data from a shared component. Capability and interface releases specify the supported combinations.

MDIL works best when it is used as designed: the model states the task and makes the declarations, and MDIL reasons, computes the answers, checks them and keeps them. Used as designed, one request runs a whole project, so the model's turns follow the declarations a task needs, not the number of steps in it. The structure is declared once and shared by reference, so any model, even a small local one, continues from it. Every answer is checked, traceable and reusable: stored answers are sealed, a separate grading program re-reads them, and a later session continues from them.

Conventional solvers, rules engines and databases provide valuable exact operations; models provide interpretation and open-ended generation. MDIL's proposition is to make explicit knowledge useful across several reasoning and construction forms, with retained conditions and contract-specific checks. Comparisons should evaluate the application work this supports, including what must be authored, integrated, maintained and executed.

Knowledge becomes shared infrastructure when a reviewed contribution becomes a capability other devices, teams and applications can invoke and improve. Local execution can reduce repeated inference work. At the level of a whole task the model's own turns are counted as well: on the recorded calculation task the model working through MDIL took 18 turns against 24 for the model alone and wrote about half as much, with all three roles graded correct. Explicit information boundaries help applications retain control of local data. Reusable constructions can reduce repeated computation and communication. These benefits arise from useful coverage and integration; each is measured at its corresponding task and resource boundary.

MDIL Kide is the first release of MDIL. MDIL launches on 17 November 2026 at 17:11 at Allas Pool in Helsinki, with three products built on it. MDIL Portal is the development environment where developers, integrators and enterprises build on MDIL, with the MDIL SDK and MDIL MCP opening in beta; access to the beta is by agreement, and the Portal's own availability is announced separately. REMAP turns a goal stated in one sentence into declared AI agents and loops that run a company's work on its own systems; it is in closed beta by invitation. Airut is an AI assistant for businesses and people, in the chat they already use; its waitlist is open. Release manifests identify capabilities available through each interface. The beta includes compression and reconstruction.

7. Evaluation and relation to established methods

The research programme evaluates selective computation, causal and constrained planning, shared-state reasoning, declared language and exact reconstruction. Each has relevant established methods. Bound-based selection relates to branch-and-bound; constructive planning to constrained search and synthesis; evidence handling to explicit reasoning systems; language composition to grammar-based generation; reconstruction to compression with shared dictionaries or components. MDIL's architectural question concerns their relationships through declared knowledge and result contracts.

Evaluation has two levels: check the operation against its contract, and measure the useful application that composes it with input preparation, models, tools and updates. Correct refusals and missed useful results both count. Reuse accounting names what the receiver already has. A receipt establishes the conditions its checker examines; application permissions and physical execution remain responsibilities at the actual resource.

The system card records the first task runs through the tool interface, completed component comparisons, current functional evidence and the wider programme. The controlled evidence package preserves source configurations, results and replay methods for independent evaluation. Relevant method references include Drools rule-engine documentation, Programming Z3, OR-Tools CP-SAT, grammar-constrained decoding and the Brotli specification.