Entropy Box: A Knowledge Compiler for Embodied AI

Yuqi Wang

August 2026

Abstract

Robotics engineering knowledge is fragmented across papers, repositories, software documentation, hardware manuals, and experiment records. Query-time retrieval can locate relevant passages, but prerequisites, interfaces, compatibility constraints, and evidence links must still be reconstructed for each request. We present Entropy Box, an agent-native knowledge compiler that converts heterogeneous sources into a persistent, typed capability substrate. Probabilistic workers propose bounded research artifacts; deterministic gates govern admission, identifier allocation, reference closure, and DAG acyclicity. A five-month production run spanning 15 robotics domains yielded 2,511 topics, 37,673 normalized capabilities, 11,397 registered assets, and a 54,602-edge dependency projection. Within 2,362 adjudicated high-similarity duplicate candidates, cosine score is poorly discriminative (AUC 0.509), showing that similarity can generate candidates but contextual LLM judgment is required before merging. On a controlled 84-query comparison spanning identity, concept, selection, function, negative, and multihop engineering intents, exact dense retrieval is the strongest tested ranker (nDCG@10 0.668); indiscriminate graph expansion and reranking reduce quality and increase latency. On 24 engineering-planning tasks, the full planner reduces unsupported claims from 100.0% under direct generation to 5.2% and raises mean hard-constraint coverage from 0.0% to 35.4%, but produces valid graphs in only 62.5% of cases. These mixed results motivate intent-dependent retrieval and fail-closed workflow validation. HTTP and Model Context Protocol interfaces expose typed lookup, evidence search, workflow assembly, and explicit gap reporting to downstream engineering agents.

Note to Practitioners— Robotics teams repeatedly search literature, repositories, package documentation, and issue trackers before assembling a system or diagnosing an integration failure. Entropy Box moves part of that recurring work into a persistent, typed capability substrate. Offline workers compile alternative technical routes under deterministic validation; online services return task structure, relevant capabilities and assets, dependency context, a candidate workflow, and explicit gaps ($\mathcal{C}_{\text{gap}}$). HTTP and MCP interfaces make the same substrate available to agentic development tools. The current system materially improves claim grounding, but its graph-serialization and constraint-satisfaction results also show that it should be used as a validated engineering aid rather than an autonomous authority.

Index Terms— robotics engineering knowledge, knowledge compiler, capability graph, agent systems, retrieval-augmented generation, task graphs, Model Context Protocol.

1 Introduction

1.1 The Robotics Knowledge Bottleneck and Engineering Isolation

Developing an autonomous robot requires integrating multidisciplinary components whose assumptions are distributed across different technical communities. A single system may combine sensor drivers, coordinate-frame conventions, state estimation, mapping, motion planning, control, manipulation, and safety monitors. The difficulty is not merely locating individual algorithms; it is recovering their dependency order, interface contracts, operating assumptions, and compatible implementations.

While foundation models have exhibited remarkable qualitative reasoning, when applied to robotic software engineering, autonomous coding agents and human engineers encounter a critical *knowledge fragmentation bottleneck*:

1. **Dispersed technical structure**: Algorithm descriptions, implementation guidance, and failure reports are split across publications, package documentation, repositories, and issue trackers.
2. **Hidden prerequisites and incompatible assumptions**: A component may depend on calibrated frames, timing guarantees, hardware limits, or a specific software distribution that a text-only answer does not expose.
3. **Unreliable software binding**: Generated plans can name unavailable, deprecated, or incompatible packages when claims are not checked against maintained asset records.

1.2 The Failure Modes of Ephemeral Query-Time Retrieval

The dominant paradigm for grounding language models in technical domains is query-time Retrieval-Augmented Generation (RAG) [6, 7] or dynamic agentic web search. These methods are useful for locating evidence, but a conventional query-time pipeline does not by itself provide three properties needed for reusable engineering structure (Table 1):

- **Persistent reuse**: Query-specific task structures are commonly discarded, so later requests repeat the same decomposition and source comparison.
- **Machine-checkable contracts**: Downstream agents benefit from explicit inputs, outputs, preconditions, invariants, and implementation bindings rather than prose alone.
- **Global identity and cross-topic linkage**: Equivalent capabilities described under different names should resolve to stable identities while genuinely distinct capabilities remain separate.

1.3 The Core Proposition: Source-Level Knowledge Compilation

To overcome these limitations, we introduce ENTROPY BOX, which shifts the burden of structural discovery from ephemeral query time to an **offline, quality-gated compilation lifecycle** (Fig. 1):

$$\text{Sources} \xrightarrow{\text{compile}} \mathcal{G}_{\text{sub}},$$
$$\mathcal{G}_{\text{sub}} \xrightarrow{\text{retrieve + assemble}} (\mathcal{D}_{\text{candidate}}, \mathcal{C}_{\text{gap}}).$$

Our method is governed by the **Symmetric Double-Gatekeeper Principle**:

> *At both build-time and run-time, probabilistic LLM workers are treated as strictly untrusted proposal generators. All state transitions, identifier allocations, topological closures, and output schemas are governed by deterministic programmatic algorithms.*

Terminology. In classical artificial intelligence, *knowledge compilation* refers to transforming a propositional theory into a representation such as OBDD or DNNF so that selected queries become tractable [5]. We use the term in a source-level engineering sense: compiling heterogeneous natural-language and software sources into a typed, persistent, queryable capability artifact. The two usages share the strategy of paying an offline transformation cost to support repeated downstream use, but ENTROPY BOX does not implement a classical target language or claim its tractability guarantees.

1.4 Summary of Contributions

1. **Typed robotics engineering intermediate representation**: We formalize a multi-layer property graph $\mathcal{G} = \langle \mathcal{T}, \mathcal{K}, \mathcal{C}, \mathcal{A}, \mathcal{E}, \mathcal{R} \rangle$ that separates task order, capability identity, implementation assets, evidence, and project-specific gaps.
2. **Agent-native offline compiler**: A four-stage production pipeline lets language-model workers perform bounded research while 4,261 lines of deterministic validation code govern durable state transitions, graph invariants, and concurrent registry writes.
3. **A compiled capability substrate at operational scale**: The resulting artifact contains 2,511 topics across 15 domains, 37,673 normalized capabilities, 11,397 registered assets, and a 54,602-edge dependency projection.
4. **Evidence on identity resolution and retrieval routing**: A 2,362-pair duplicate-candidate audit shows why similarity must be followed by contextual LLM adjudication, while a controlled 84-query comparison across structured engineering intents shows that more retrieval stages do not necessarily improve entity ranking.
5. **Constrained workflow assembly and open interfaces**: A multi-view runtime and Integrate Planner expose typed lookup, evidence retrieval, candidate DAG assembly, and gap reporting through HTTP and MCP. A 24-task benchmark reports both grounding gains and remaining failures in graph validity and constraint satisfaction.

2 Related Work

Robot knowledge infrastructures. RoboEarth proposed a networked repository through which robots could share action recipes, environment models, and object knowledge [1]. KnowRob and CRAM connected symbolic knowledge and episodic reasoning to robot action, while openEASE provided queryable experiment and episode data [2, 3, 4]. These systems establish the value of reusable robot knowledge. ENTROPY BOX addresses a different layer: it compiles engineering knowledge about how systems are designed and integrated, represents alternative technical routes, and binds capabilities to software assets and evidence before a particular robot executes a task.

Retrieval and graph-grounded generation. RAG and dense retrieval ground generation in external corpora [6, 12]; reciprocal-rank fusion and neural reranking combine heterogeneous signals [13, 14]. GraphRAG organizes corpus-derived entities and summaries for local and global question answering [8]. Recent surveys examine LLM-assisted knowledge-graph construction and graph-enabled agents [17, 18]. ENTROPY BOX differs in its admission

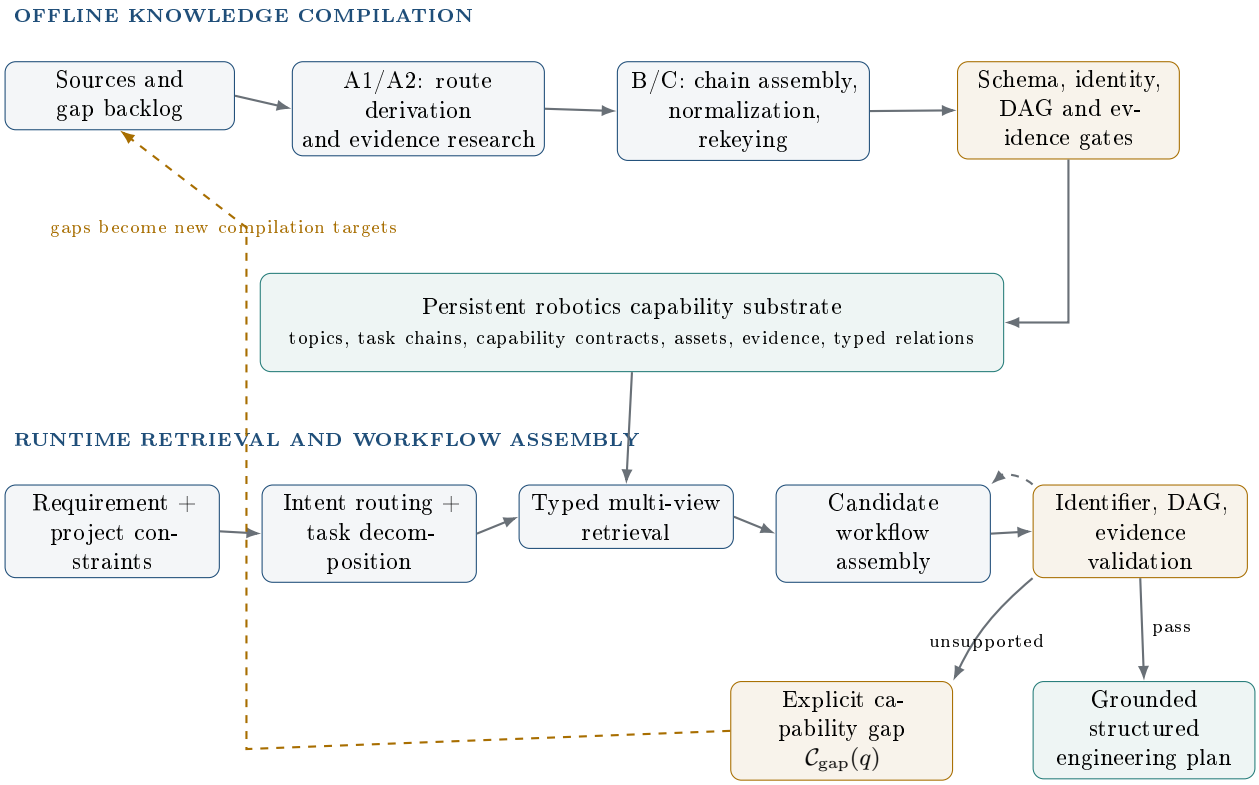

Figure 1: System architecture. Offline agents produce bounded research artifacts, while deterministic gates control admission to the persistent capability substrate. At runtime, a requirement is decomposed, retrieved against typed views, assembled into a candidate workflow, and validated. Structural failures return to assembly; unsupported requirements are emitted as explicit gaps and re-enter the offline compilation backlog.

boundary: the durable unit is a typed engineering artifact that must pass deterministic graph and registry checks, not only an extracted chunk, entity, or session report.

Language models for embodied planning and robot programming. SayCan grounds high-level language choices in skill affordances, Code as Policies generates robot-control programs, and ReAct interleaves reasoning with tool use [9, 10, 11]. Retrieval-Augmented Robotics retrieves external procedural information for physical task construction [15]; recent MCP-based work retrieves documentation, generates industrial robot programs, and corrects them in simulation [16]. ENTROPY BOX is complementary: it focuses on producing and maintaining the reusable, cross-project capability substrate that such agents query.

3 Problem Formulation and Typed Intermediate Representation

3.1 Mathematical Formulation of the Capability Graph

We formalize robotics engineering knowledge as a typed, multi-layer directed property graph:

$$\mathcal{G} = \langle \mathcal{T}, \mathcal{K}, \mathcal{C}, \mathcal{A}, \mathcal{E}, \mathcal{R} \rangle$$

- $\mathcal{T} = \{T_0, T_1, T_2\}$: A 3-tier taxonomy tree spanning 15 L_0 domains, 142 L_1 sub-domains, and 2,511 L_2 topic nodes.
- $\mathcal{K} = \{K_t\}$: A set of compiled technical task chains for each topic t. Each chain $K = (V_K, E_K)$ is a Directed Acyclic Graph (DAG) of discrete engineering steps.
- $\mathcal{C} = \{c_j\}$: A global registry of normalized capability records. Each capability c_j is uniquely identified by an immutable surrogate key (`CAP_<hex>`) and bound to a formal typed contract:

  $$\sigma(c_j) = \langle \mathcal{I}(c_j), \mathcal{O}(c_j), \mathcal{P}(c_j), \Psi(c_j), \mathcal{H}(c_j) \rangle$$

  where $\mathcal{I}$ is the typed input set, $\mathcal{O}$ the output set, $\mathcal{P}$ preconditions, Ψ invariant operational constraints (e.g., maximum frequency, latency bounds), and $\mathcal{H}$ physical/hardware assumptions.
- $\mathcal{A} = \{a_k\}$: A registry of physical software assets (`AST_<hex>`), mapping capabilities to concrete GitHub repositories, ROS/ROS 2 packages, pretrained model weights, URDF models, or simulation environments.
- $\mathcal{E} = \{e_m\}$: Provenance evidence nodes containing source URLs, local document pointers, extraction timestamps, resolution status, and explicit `[unverified]` markers.

- $\mathcal{R}$: A typed relation set partitioned into five canonical edge families:

$$\mathcal{R} = \mathcal{R}_{\text{precedes}} \cup \mathcal{R}_{\text{impl}} \cup \mathcal{R}_{\text{requires}} \cup \mathcal{R}_{\text{hierarchy}} \cup \mathcal{R}_{\text{evidence}}$$

3.2 Cross-Paradigm Comparison

Table 1 provides a systematic comparison between ENTROPY BOX and existing knowledge representations.

3.3 Surrogate Keys Symbol Table and Decoupling

A major failure mode in early knowledge graphs is the tight coupling between human-readable entity names and their internal identifiers. When an engineer or LLM renames an entity to "Analytical 6-DOF IK Solver", all referencing task chains and graph edges break.

ENTROPY BOX introduces an **Immutable Surrogate Key Architecture**:

- When a capability is admitted, it is permanently assigned an opaque surrogate key: `CAP_<hex>`.
- The human-readable string is stored as a mutable display attribute with an associated alias set:

$$\text{Record}(c_j) = \langle \text{id}, \text{name}, \text{aliases} \rangle,$$
$$\text{id}(c_j) = \texttt{CAP\_90e22fca}.$$

- Any refactoring, renaming, or localization update triggers an **Atomic Topic Rewriting Transaction**: global references and Markdown views are rewritten simultaneously, followed by deterministic re-validation.

3.4 Formal Gap Formulation and Closed-Loop Feedback

A critical feature of ENTROPY BOX is the distinction between a registered capability and one that is supported for a concrete project. Let q denote project constraints, $V(a)$ a verified asset record, and $X_q(a)$ compatibility with platform, software version, compute budget, and hardware assumptions. Then

$$S_q(c) \Leftrightarrow \exists a \in \mathcal{A} : (c, a) \in \mathcal{R}_{\text{impl}} \wedge V(a) \wedge X_q(a).$$

For the capability requirements $\mathcal{C}_{\text{req}}(q)$ produced by task decomposition, the gap set is

$$\mathcal{C}_{\text{gap}}(q) = \{c \in \mathcal{C}_{\text{req}}(q) \mid \neg S_q(c)\}.$$

Thus, an asset may exist in the registry yet still yield a gap when its evidence is unresolved or its constraints are incompatible with the project. Gap records are returned with the candidate workflow and can be added to the compiler backlog for targeted research; they are not silently replaced by model-generated package names.

3.5 Amortization Economics and Structural Reuse

Across the public substrate of 2,511 topics:

- Total normalized capabilities: $|\mathcal{C}| = 37,673$.
- Total capability step usages across all task chains: $|U| = 58,989$.
- **Structural Amortization Factor**:

$$\alpha = \frac{|U|}{|\mathcal{C}|} = \frac{58,989}{37,673} \approx 1.566$$

The difference $|U| - |\mathcal{C}| = 21,316$ counts capability uses beyond one occurrence per normalized capability. It measures structural reuse opportunities, not time saved or derivations causally eliminated. Some foundational capabilities occur in more than 50 topics.

4 Offline Knowledge Production Compiler

4.1 The Production Challenge and the Handoff Rule

The artifact was produced by multi-agent workers under deterministic admission gates. The pipeline evolved through three internal iterations:

- **v1 (Monolithic Single-Agent)**: One agent researched, extracted, and formatted a topic graph in a single context. Long topics showed context crowding, later-chain degradation, and unsupported package names.
- **v2 (Specialist Relay Agents)**: Research, extraction, and validation were separated. Ambiguities were often lost because the handoff consisted of a summary rather than a validated structure.
- **v3 (Artifact-Centric Compilation)**: Agents communicate strictly through persisted, typed intermediate artifacts with deterministic validation gates.

The resulting design rule is: *keep stages in one context when the handoff object is reasoning; separate stages when the handoff object is a validated artifact.* Research and chain construction remain together because topology depends on source comparison. Alternative chains are isolated because the durable handoff is a chain document.

4.2 Four-Stage Multi-Agent Compilation Pipeline

The offline branch of Fig. 1 expands into four production stages:

1. **Phase A1 (Route Derivation)**: Given an L_2 topic, a coordinator analyzes its boundary and proposes at least three technically distinct route hypotheses.

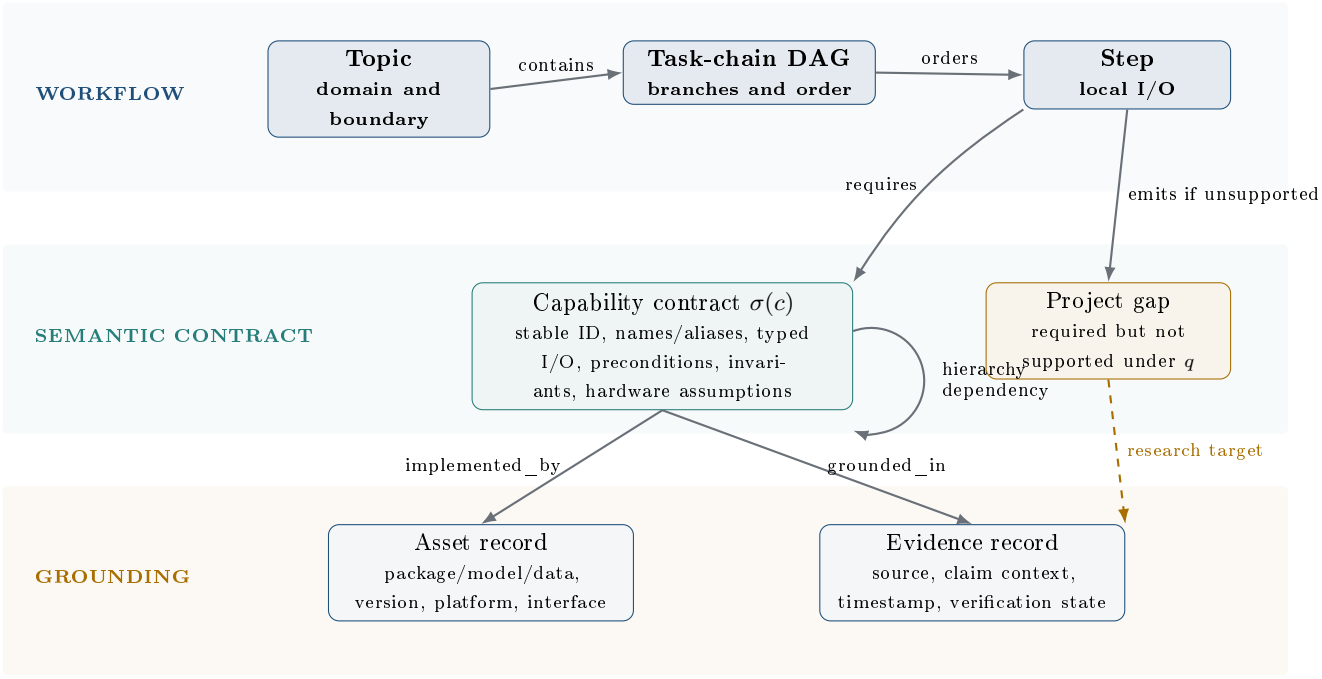

Figure 2: Typed intermediate representation. Topic scope, task order, capability semantics, implementation assets, provenance evidence, and project-specific gaps are separate objects joined by typed relations. Stable capability identities permit cross-topic reuse without conflating workflow order, implementation binding, and evidence status.

2. **Phase A2 (Isolated Research Workers)**: For each technical route, an isolated worker agent performs source and repository search, outputting a nine-section Markdown artifact containing problem formulation, step sequences, software bindings, and a mandatory retrieval ledger that records both supporting results and unsuccessful searches.

3. **Phase B (Topic Assembly)**: Evaluates pairwise capability overlap:

$$\text{overlap}(K_A, K_B) = \frac{|\mathcal{C}_A \cap \mathcal{C}_B|}{\min(|\mathcal{C}_A|, |\mathcal{C}_B|)}$$

   Chains with overlap ≥ 0.70 are consolidated into branching variants of a unified DAG.

4. **Phase C (Finalization & Rekeying)**: Assigns new opaque surrogate keys (`CAP_<hex>`) to unseen capabilities, updates references transactionally, and emits the candidate topic DAG.

4.3 Critical Sequencing Constraints

Through production deployments, we discovered two fundamental sequencing constraints:

1. **Normalize Identifiers Before Measuring Overlap**: Computing chain overlap using raw strings underestimates redundancy by over 60% due to phrasing variations.

2. **Rekey After Contextual Adjudication, Never Before**: Similarity first generates suspects; the LLM then compares natural-language names, typed contracts, graph context, and task usage. Surrogate-key rewriting occurs only as the final atomic commit.

4.4 Deterministic Quality Gatekeepers

The compiler incorporates 4,261 lines of deterministic Python code across validation modules:

- **DAG Acyclicity and Reachability Gate**: Implements Tarjan's strongly connected components and white/gray/black tri-color DFS to guarantee zero cycles and verify reachability from entry steps.
- **Bidirectional Hierarchy Closure**: Enforces parent–child reciprocity between related capabilities.
- **Physical Asset Gate**: Validates asset type, package-name syntax, required provenance fields, and URL format. Resolvability is tracked by a separate audit because network availability is time dependent.
- **Cross-Process Transactional Lock**: Registry admissions acquire atomic file locks to guarantee symbol uniqueness during high-concurrency builds (up to 20 parallel workers).

4.5 Why Similarity Cannot Decide Entity Merging

During production, the compiler accumulated 2,362 suspected duplicate pairs from two candidate-generation

Table 1: Relationship to adjacent paradigms. Entries describe the primary artifact, not every possible implementation.

Paradigm	Persistent unit	Typical structure	Admission boundary	Primary use
Vector RAG [6]	Text chunk and embedding	Document-local text	Indexing pipeline	Passage retrieval for generation
GraphRAG [8]	Corpus-derived entity/summary	Entity relations and communities	Extraction and clustering	Local/global corpus questions
Robot ontology [2]	Objects, actions, episodes, axioms	Description logic axioms/rules	Ontology consistency (OWL)	Robot cognition and action execution
Deep-research agent	Session trace or report	Query-specific plan and prose	Model/tool loop	One-off synthesis
Entropy Box	Topic, typed chain, capability (σ), asset, evidence	Task DAGs and cross-topic capability dependencies	Deterministic compiler gates and ledgers	Reusable engineering retrieval, assembly, and gap inspection

Table 2: Core intermediate-representation objects and their enforced role.

Object	Representative fields	Deterministic checks	Downstream use
Topic	identifier, domain, names, task chains, status	schema, unique ID, at least one accepted chain	discovery and scope
Chain step	step ID, predecessors, successors, capability IDs, inputs, outputs, constraints	local reference closure, edge reciprocity, acyclicity	ordered plan and branch navigation
Capability	opaque ID, names/aliases, contract σ, category	global ID/type conflict, candidate-duplicate ledger	cross-topic reuse and dependency expansion
Asset	opaque ID, type, repository or source URL, implementation note	type/URL syntax, hard repository conflicts	software/model/dataset selection
Evidence	source URL, local document pointer, claim context, verification marker	required pointer fields; resolvability audit	provenance inspection
Gap	required capability, reason, status	explicit unresolved state rather than invented asset	missing-component reporting

paths. The embedding path contributed 1,867 pairs at $s \geq 0.862$; a lexical fallback contributed 495 pairs. Each candidate was then judged in context by an LLM using the entity contracts and graph neighborhood, with uncertain decisions retained as separate entities. Confirmed decisions were recorded before any transactional merge. This is a deliberately difficult, range-restricted candidate pool rather than a random sample of all entity pairs.

Finding: Within the high-similarity embedding candidate set, cosine score has little ranking value (AUC 0.509, 95% CI $[0.453, 0.564]$). Raising the threshold from 0.862 to 0.942 increases precision only from 5.5% to 6.9%, while recall of the adjudicated duplicates in this set falls from 100% to 67.0%. Similarity remains useful for retrieving a manageable suspect set, but the experiment shows that scalar algorithmic similarity is insufficient to decide whether two engineering capabilities are semantically interchangeable.

Engineering response: We use a three-part decision pipeline in which algorithmic similarity proposes, an LLM judges, and deterministic code commits:

1. *Algorithmic candidate generation*: Embedding and lexical signals retrieve suspect pairs and deterministic rules remove obvious non-duplicates. Neither path is authorized to merge entities.

2. *Contextual LLM adjudication*: For each remaining pair, the adjudicator reads canonical names, aliases, typed I/O contracts, hierarchy, dependency neighborhood, and local task-chain usage. It must decide `same`, `distinct`, or `uncertain` with a structured rationale. Micro-batches ($N = 5$, temperature 0.1) and a retain-when-uncertain rule limit cross-pair interference and false merges.

3. *Deterministic atomic commit*: Only a `same` decision that passes identity and reference checks triggers transactional rewriting across chains, registries, and generated views. `distinct`, `uncertain`, or failed validation leaves both entities unchanged.

5 Runtime Multi-Facet Retrieval and Topological Reasoning

The lower branch of Fig. 1 separates retrieval, workflow assembly, validation, and gap feedback. In particular, a

Table 3: Durable compilation lifecycle. Status changes are deterministic even when stage contents are model-generated.

State	Persisted product	Admission condition	Failure handling
Queued	Topic inventory and anchor card	Topic is in taxonomy, unclaimed, and not accepted	Expire lease; retain current stage
A1 scoped	Three route hypotheses and representative leads	Routes parse, are nonempty, and are technically distinct	Repair list format or return to queue
A2 researched	Isolated chain documents and source ledgers	Evidence sections, capability/asset fields, and chain gates pass	Reject only failing chain; preserve siblings
B transcribed	Incremental typed topic DAG	Local references close; edges are reciprocal and acyclic	Return machine-readable report; append only missing chain
C finalized	Candidate topic plus final validation	Blocking checks clear after one bounded repair	Record structural debt; do not loop indefinitely
Registered	Global identifiers, conflicts, suspect ledger	No hard conflict; rewrites validate as a transaction	Abort without partial global write
Published	Lookup, lexical, vector, and graph views	Accepted records and view manifest agree	Rebuild views from accepted records

Table 4: Audit of duplicate-candidate generators ($n = 2,362$). AUC intervals use 2,000 bootstrap resamples.

Path	Pairs	Confirmed Dup.	Precision	AUC [95% CI]
Embedding ($s \geq 0.862$)	1,867	103	.055 (5.5%)	.509 [.453, .564]
Lexical fallback	495	8	.016 (1.6%)	.689 [.572, .791]

structural validation failure returns to workflow assembly rather than restarting or altering the original requirement; a missing supported capability is emitted to the offline backlog.

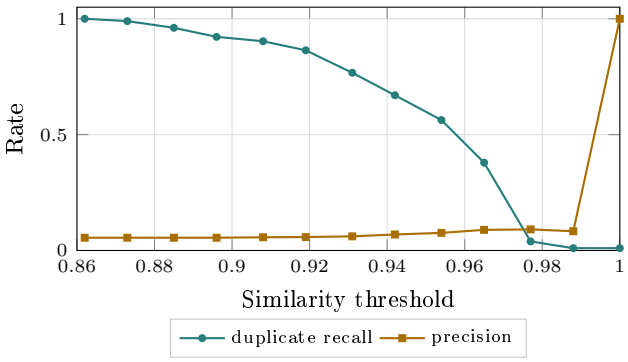

Figure 3: Threshold sweep within the embedding-flagged candidate set. Similarity is useful for candidate generation but insufficient for the merge decision: precision remains below 9% through the practically useful range, while raising the threshold sharply reduces duplicate recall. Remaining candidates therefore require contextual LLM adjudication.

5.1 6-Dimensional Semantic Views and Dynamic Intent Weighting

Rather than collapsing an entity into one embedding, ENTROPY BOX projects every capability $c \in \mathcal{C}$ into six complementary semantic views. The views are field-specific representations; no statistical orthogonality is assumed.

- **Identity View ($\vec{v}_{\mathbf{id}}$)**: Canonical name, multilingual translation, and verified aliases.
- **Concept View ($\vec{v}_{\mathbf{con}}$)**: Algorithmic principle and mathematical overview.
- **Function View ($\vec{v}_{\mathbf{fn}}$)**: Structured contract fields: Inputs, Outputs, Invariants, and Assumptions.
- **Relation View ($\vec{v}_{\mathbf{rel}}$)**: Hierarchical taxonomy context (parent, siblings, child specializations).
- **Context View ($\vec{v}_{\mathbf{ctx}}$)**: Topological dependency neighborhood (c_{precedes} and $c_{\text{succeeded\_by}}$).
- **Implementation View ($\vec{v}_{\mathbf{impl}}$)**: Supported physical assets and package bindings.

An intent classifier categorizes the query into one of six canonical intents, dynamically assigning view weights (Table 5):

$$S(Q, c) = \sum_{k \in \{\text{id,con,fn,rel,ctx,impl}\}} w_k(Q) \cdot \cos\left(\mathbf{E}(Q), \vec{v}_k(c)\right)$$

5.2 Native GPU PyTorch Exact Dense Index

The dense retriever uses an exact PyTorch matrix index on a single GPU:

- The six 2560-dimensional float16 view matrices and associated buffers reside in GPU memory (approximately 5 GB in the evaluated snapshot on an NVIDIA RTX 4090).

Table 5: Dynamic Intent Weighting across 6 Semantic Views.

Query Intent	w_{id}	w_{con}	w_{fn}	w_{rel}	w_{ctx}	w_{impl}
Identity (e.g., "MoveIt", exact pkg)	**0.55**	0.15	0.05	0.05	0.05	0.15
Concept (e.g., "Kinodynamic planning")	0.15	**0.40**	0.20	0.10	0.05	0.10
Function (e.g., "Numerical IK solver")	0.10	0.15	**0.50**	0.05	0.10	0.10
Relation (e.g., "sampling vs optimization")	0.10	0.10	0.05	**0.55**	0.15	0.05
Context (e.g., "Prerequisites for MPC")	0.05	0.10	0.10	0.15	**0.55**	0.05
Implementation (e.g., "ROS 2 MPC pkg")	0.15	0.10	0.10	0.05	0.05	**0.55**

- Query scoring executes as batch matrix multiplication: $\mathbf{S} = \mathbf{M}_{\text{GPU}} \cdot \mathbf{q}_{\text{GPU}}$.
- The matrix-scoring kernel takes less than 2 ms over the evaluated 10^5-scale view-vector collection and is exact with respect to exhaustive cosine scoring. This kernel measurement excludes query encoding, host–device transfer, result materialization, and service overhead; Table 7 reports end-to-end latency.

5.3 Lexical BM25F with 124+ Domain Synonym Dictionaries

The lexical engine uses field-weighted BM25F with dedicated tokenizers for ASCII symbols, camelCase names, and CJK characters. It incorporates a curated dictionary of **124+ robotics synonym groups** (e.g., `IK` ↔ `Inverse Kinematics`). Exact symbol and repository matches receive deterministic score boosts (+5.0).

5.4 Multi-Stage Fusion, Topological Expansion, and Reranking

1. **Reciprocal Rank Fusion (RRF)**: Lexical and dense candidates are fused using RRF ($K = 60$).
2. **1-Hop Topological Expansion**: Traverses $\mathcal{G}_{\text{substrate}}$ along hierarchy, precedes, and implementation edges, appending structural context.
3. **Cross-encoder reranking**: Candidates are formatted as structured natural-language contracts and scored by a cross-encoder. The benchmark in Section 8 shows that this stage is not beneficial for every intent.
4. **Chain Completion**: When a capability step is retrieved, its parent task chain is attached to preserve end-to-end topological continuity.

6 LLM End-to-End Workflow Assembly and Scientific Advisory

6.1 The Integrate Planner Engine

Served via `/api/consult`, the **Integrate Planner** takes an engineering specification, performs multi-intent subquery decomposition, executes multi-facet retrieval over $\mathcal{G}_{\text{substrate}}$, and asks an assembly model to propose a structured workflow:

- **Typed candidate graph**: The output schema records step identifiers, dependencies, I/O descriptions, and applicable constraints. A deterministic validator rejects malformed references and cycles; the result is a design artifact, not directly executable robot code.
- **Prerequisite expansion**: Retrieved task-chain context supplies upstream candidates before assembly. Closure is checked against the bounded retrieved subgraph rather than claimed globally.
- **Binding or gap**: Each required capability must reference an admissible asset for the project or be emitted as $\mathcal{C}_{\text{gap}}(q)$.

6.2 Automated Gap Reporting and Scientific Advisory

When no admissible asset satisfies a requirement, the output schema requires the planner to:

1. Emits a formal **Engineering Gap ($\mathcal{C}_{\text{gap}}$)** declaring missing dependencies.
2. Provides an evidence-linked trade-off note when the retrieved sources support one; otherwise, the recommendation remains explicitly unverified.

6.3 FastMCP Server and OpenAPI Ecosystem

The runtime registers four standardized agent tools:

1. `consult_robotics_solution`: decomposes a request, performs multi-view retrieval, and returns a candidate task graph with gap analysis.
2. `lookup_entity`: retrieves capability contracts (σ), asset metadata, or topic summaries by identifier or name.
3. `search_evidence`: queries source records, extraction provenance, and negative retrieval ledgers.
4. `search_knowledge_base`: exposes direct hybrid retrieval over topic, capability, and asset layers.

MCP-compliant development clients can connect through standard tool configuration, while the same functions remain available through HTTP endpoints.

7 Artifact Snapshot

Table 6 reports a frozen substrate snapshot used by the experiments. The 2,511 topics span 15 top-level domains covering the robotics software stack, from perception and mapping through planning, manipulation, safety, simulation, and systems infrastructure. Registry counts and the dependency projection are reported separately because not every registered capability appears in the projected cross-topic graph.

Table 6: Frozen artifact snapshot used for evaluation, 28 August 2026.

Artifact field	Count
Published topics	2,511
Registered physical assets ($\mathcal{A}$)	11,397
Registered capability contracts ($\mathcal{C}$)	37,673
Capability-dependency projection nodes	26,441
Directed dependency edges ($\mathcal{R}_{\text{precedes}}$)	54,602
Total capability step usages ($\lvert U \rvert$)	58,989
Amortization factor (α)	1.566

8 Experimental Evaluation

8.1 Entity-Retrieval Benchmark

The controlled comparison contains 84 engineering queries across six intent categories (34 identity, 24 concept, 13 selection, 6 multihop, 5 function, and 2 negative). It was assembled to compare retrieval configurations on structured and comparatively demanding engineering intents, rather than to serve as a census of the continuously expanding public query suite. Eighty-three queries have scorable positive judgments; the comparison also contains 17 hard-negative assertions that must not appear in the top 10. We evaluate the same frozen corpus under six retrieval configurations. Ranking intervals use 5,000 query-level bootstrap resamples. Latency is measured end to end from service receipt through query encoding, retrieval stages, and result serialization; it is therefore not comparable to the isolated matrix-kernel timing reported above.

Result: Dense retrieval is the strongest tested entity ranker (nDCG@10 0.668). In paired query-level analysis, the full graph-plus-reranker pipeline is lower than dense by 0.301 nDCG@10 (95% CI $[-0.393, -0.209]$) and 0.217 Recall@10 (95% CI $[-0.325, -0.108]$), while its median latency is approximately twelve times higher. Removing graph expansion from that full configuration changes nDCG@10 by only 0.016 (95% CI $[0.004, 0.030]$) in favor of the simpler variant. The benchmark therefore supports routing direct entity lookups to dense retrieval and reserving structural expansion for requests whose output actually requires neighboring nodes or complete chains. It does not establish a benefit for graph expansion on multihop tasks because that subgroup contains only six queries.

8.2 Plan Synthesis and Grounding Benchmark

We evaluated 24 multidisciplinary robotics engineering tasks from a development/candidate set under five matched conditions, yielding 120 prediction records (Table 8). All conditions were run once with the fixed seed 20260828; the reported bootstrap intervals reflect paired variation across tasks, not variation across repeated model runs. This design controls cost while providing a reproducible comparison, but it does not estimate run-to-run stochasticity. The harness is fail closed: an unparseable field, unresolved identifier, or unsupported assertion is scored as failure rather than repaired after evaluation. Unsupported-claim rate is computed over atomic package, asset, or capability assertions that lack traceable support. Hard-constraint violation is the fraction of plans violating at least one task constraint; constraint coverage is the mean fraction of requested constraints explicitly represented. Graph validity is the fraction of outputs with closed step references and an acyclic dependency graph. Capability F1 is micro-F1 against the development set's expected capability labels.

Result: The full planner reduces unsupported claims from 100.0% to 5.2% (a −94.8 percentage-point change) and raises explicit hard-constraint coverage from 0.0% to 35.4%. These gains do not constitute end-to-end planning success: 66.7% of full-planner outputs still violate at least one hard constraint, graph validity falls to 62.5%, and capability F1 remains 0.008. The current contribution is therefore strongest as a grounding and gap-reporting mechanism. Exact capability selection, constraint satisfaction, and reliable graph serialization remain unresolved runtime problems.

9 Discussion, Engineering Lessons, and Threats to Validity

9.1 Core Lessons for Agent-Native Systems

1. **Durable artifacts make agent work inspectable**: Typed intermediate files preserve evidence and topology across worker boundaries more reliably than conversational summaries alone.

2. **Deterministic gates protect state, not truth**: Cycle checks, locks, and transactional rewrites prevent structural corruption, but they cannot establish that a technical claim is scientifically correct. Evidence status and human audit remain necessary.

3. **More retrieval stages are not automatically better**: The entity benchmark favors exact dense retrieval over the tested fusion, graph, and reranking configurations. Routing must be justified by intent-specific evidence rather than pipeline complexity.

4. **Grounding and planning quality are distinct**: The full planner greatly reduces unsupported claims yet still performs poorly on exact capability selection, hard constraints, and graph validity. A grounded plan can remain incomplete or structurally invalid.

9.2 Limitations and Threats to Validity

- **Provenance completeness**: In audited chain documents, 81.99% of references resolve to physical artifacts and 2,388 references remain explicitly marked

Table 7: Entity retrieval over 84 queries (83 with scorable positives). All 17 hard-negative assertions had zero hits at rank 10.

Condition	nDCG@10	Recall@10	Recall@20	MRR	P50 ms	P95 ms
Lexical (BM25)	.563 [.458, .664]	.639 [.530, .735]	.735 [.639, .831]	.535 [.434, .635]	90.6	138.5
Dense (PyTorch GPU)	**.668** [.572, .759]	**.771** [.675, .855]	**.843** [.771, .916]	**.604** [.512, .697]	336.4	377.8
Hybrid fusion	.605 [.508, .700]	.723 [.627, .819]	.831 [.747, .904]	.550 [.453, .644]	413.8	456.5
Hybrid + graph	.599 [.499, .696]	.711 [.614, .807]	.771 [.675, .855]	.545 [.448, .640]	527.7	630.2
Hybrid + reranker	.382 [.296, .471]	.590 [.494, .699]	.783 [.687, .867]	.302 [.228, .376]	4,038.5	6,345.4
Full graph + reranker	.366 [.280, .458]	.554 [.446, .663]	.771 [.675, .855]	.296 [.222, .370]	4,122.1	6,485.3

Table 8: Plan synthesis and context grounding on a 24-task development/candidate set under five conditions (120 prediction records; one fixed-seed run per condition). Lower is better for unsupported claims and constraint violations; higher is better otherwise. Paired task-level intervals use 2,000 bootstrap resamples.

Condition	Unsupported claims ↓	Hard-constraint violation ↓	Constraint coverage ↑	Graph validity ↑	Capability F1 ↑
LLM direct	100.0%	100.0%	0.0%	83.3%	.025
Lexical RAG	100.0%	100.0%	0.0%	75.0%	.020
Hybrid RAG	100.0%	100.0%	0.0%	79.2%	.008
Entropy Box context only	100.0%	83.3%	16.7%	75.0%	.008
Entropy Box Full	**5.2%**	**66.7%**	**35.4%**	62.5%	.008
Full − Hybrid RAG	−94.8 [−97.4, −92.1] pp	−33.3 [−54.2, −16.7] pp	+35.4 [+16.7, +54.2] pp	−16.7 [−37.5, +4.2] pp	.000 [.000, .000]

`[unverified]`. Registration therefore does not imply verification or project compatibility.

- **Duplicate-label validity**: The duplicate analysis is retrospective and range restricted to generated candidates. Decisions come from the production LLM-adjudication process and lack a random-pair sample plus independently reported inter-annotator agreement. The AUC therefore evaluates similarity against these operational decisions and should not be generalized to all entities or embedding models.
- **Benchmark size and construction**: The retrieval comparison contains 84 queries and only six multihop queries; the planner development/candidate set contains 24 tasks and was evaluated once per condition with a fixed seed. Both were built around the current substrate and may favor its vocabulary. The reported bootstrap intervals do not measure model-run stochasticity. Larger external benchmarks, repeated runs where resources permit, and blinded expert judgments are needed.
- **Runtime immaturity**: The full planner's 62.5% graph validity, 66.7% hard-constraint violation rate, and 0.008 capability F1 preclude autonomous deployment. Deterministic post-validation catches some failures but does not repair semantic omissions.
- **No physical execution claim**: The present evaluation measures compilation artifacts, retrieval, and structured planning. It does not establish safe physical robot execution, transfer across hardware, or improvements in engineering time and cost.
- **Model and infrastructure dependence**: Results may change with the worker model, embedding model, cross-encoder, GPU, prompt version, and source availability. The frozen snapshot and configuration must accompany any reproducibility release.

10 Conclusion

Entropy Box treats robotics engineering knowledge production as compilation: probabilistic workers extract and organize technical routes, while deterministic code controls admission into a persistent typed substrate. The resulting artifact spans 2,511 topics, 37,673 capabilities, 11,397 registered assets, and a 54,602-edge dependency projection. The evaluation supports three bounded conclusions. First, similarity is useful for duplicate-candidate generation but insufficient for entity merging in the audited ambiguous region; contextual LLM adjudication is needed before deterministic commit. Second, dense retrieval is the strongest tested method for direct entity ranking, so graph and reranking stages should be invoked selectively. Third, typed context and fail-closed assembly substantially improve claim grounding but do not yet solve constraint satisfaction or graph generation. These findings position Entropy Box as reusable robotics knowledge infrastructure and establish a concrete agenda for external benchmarking, stronger validators, and execution-grounded evaluation.

Code, Data, and Live System

Code, public artifact indices, evaluation materials, and project documentation are available at `https://github.com/chenli-yy/entropy-box-public`. The live system is available at `https://xiangshang.ngrok.app/`. The public deployment is continuously updated as new pa-

pers, methods, models, and software assets appear; consequently, its current counts may differ slightly from the frozen 28 August 2026 snapshot used for the experiments in this manuscript.

Acknowledgment

OpenAI ChatGPT/Codex was used for manuscript restructuring, language editing, consistency checks, and LaTeX preparation. It was not treated as a source of experimental evidence. The authors reviewed the resulting text and remain responsible for the code, data, analyses, citations, and claims.

References

[1] M. Waibel et al., *RoboEarth*, IEEE Robotics & Automation Magazine, vol. 18, no. 2, pp. 69–82, 2011.

[2] M. Tenorth and M. Beetz, *KnowRob: A Knowledge Processing Infrastructure for Cognition-Enabled Robots*, Int. J. Robot. Res., vol. 32, no. 5, pp. 566–590, 2013.

[3] M. Beetz, L. Mösenlechner, and M. Tenorth, *CRAM—A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments*, in Proc. IEEE/RSJ IROS, 2010.

[4] M. Beetz et al., *openEASE—A Knowledge Processing Service for Robots and Robotics/AI Researchers*, in Proc. IEEE ICRA, 2015.

[5] A. Darwiche and P. Marquis, *A Knowledge Compilation Map*, Journal of Artificial Intelligence Research, vol. 17, pp. 229–264, 2002.

[6] P. Lewis et al., *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*, in Advances in Neural Information Processing Systems, 2020.

[7] Y. Gao et al., *Retrieval-Augmented Generation for Large Language Models: A Survey*, arXiv:2312.10997, 2023.

[8] D. Edge et al., *From Local to Global: A Graph RAG Approach to Query-Focused Summarization*, arXiv:2404.16130, 2024.

[9] M. Ahn et al., *Do As I Can, Not As I Say: Grounding Language in Robotic Affordances*, in Proc. Conf. Robot Learning, 2022.

[10] J. Liang et al., *Code as Policies: Language Model Programs for Embodied Control*, in Proc. IEEE ICRA, 2023.

[11] S. Yao et al., *ReAct: Synergizing Reasoning and Acting in Language Models*, in Proc. ICLR, 2023.

[12] V. Karpukhin et al., *Dense Passage Retrieval for Open-Domain Question Answering*, in Proc. EMNLP, 2020.

[13] G. V. Cormack, C. L. A. Clarke, and S. Büttcher, *Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods*, in Proc. ACM SIGIR, 2009.

[14] R. Nogueira and K. Cho, *Passage Re-Ranking with BERT*, arXiv:1901.04085, 2019.

[15] I. Temiraliev, D. Yang, and Y. Zhang, *Retrieval-Augmented Robots via Retrieve–Reason–Act*, arXiv:2603.02688, 2026.

[16] Z. Zhou, S. Chen, O. Salunkhe, E. T. Bekar, J. Stahre, and A. Skoogh, *Retrieval-Grounded Robot Program Generation and Simulation-Based Correction via Model Context Protocol*, arXiv:2608.21417, 2026.

[17] S. Pan et al., *LLM-Empowered Knowledge Graph Construction: A Survey*, arXiv:2510.20345, 2025.

[18] B. Jin et al., *Graphs Meet AI Agents: Taxonomy, Progress, and Future Opportunities*, arXiv:2506.18019, 2025.