Grounded Agency: A Capability Ontology for Safe, Auditable, and Composable AI Agents

Daniel Bentes[1]
[1]Synapti.ai
daniel@synapti.ai

January 2026

Abstract

As large language models transition from conversational assistants to autonomous agents capable of real-world actions, the gap between *what AI can do* (capability taxonomies) and *how AI operates reliably* (operational primitives) becomes critical. We present GROUNDED AGENCY, a comprehensive framework that bridges academic capability taxonomies with production-grade agent infrastructure through three core contributions: (1) a **capability ontology** of 99 atomic primitives organized across 8 functional layers with formal input/output schemas and 60 dependency edges; (2) a **world state schema** supporting real and digital system modeling with first-class uncertainty (epistemic, aleatoric, mixed), evidence anchors, and reversible state transitions; and (3) a **trust-aware conflict resolution** model with source authority weights, temporal decay, and Bayesian identity resolution. The framework enforces three invariants: every claim is grounded in evidence, uncertainty is explicit and typed, and every mutation is reversible and auditable. We introduce a design-time workflow validator that performs type inference across step bindings and suggests automatic coercions when mismatches occur. Evaluation on 5 reference workflows totaling 49 steps demonstrates 100% schema coverage, with the validator detecting all seeded type errors and suggesting patches from a registry of 5 coercion mappings. Our 18-class entity taxonomy with 57 subtypes and hierarchical namespace identifiers enables unambiguous cross-system entity references. We release the complete framework—ontology, schemas, workflows, and validator—as open source to establish a foundation for safe, auditable agentic AI.

Keywords: Large Language Models, Autonomous Agents, Capability Ontology, World Modeling, Trust Models, Type Systems, Safe AI

1 Introduction

The past two years have witnessed a fundamental shift in how large language models (LLMs) interact with the world. Systems like AutoGPT [1], Claude Computer Use [2], and Devin [3] demonstrate that LLMs can move beyond conversation to take autonomous actions: executing code, browsing the web, managing files, and interacting with external services. This transition from *assistant* to *agent* creates unprecedented opportunities—and unprecedented risks.

Current agent frameworks lack formal guarantees about three critical properties:

1. **Grounding**: What evidence supports the agent's beliefs and decisions?
2. **Uncertainty**: How confident is the agent, and what type of uncertainty applies?
3. **Reversibility**: Can actions be undone if something goes wrong?

Consider a digital twin synchronization workflow that monitors a payments service. The agent receives observability data, detects an anomaly, and must decide whether to trigger a rollback. Without explicit uncertainty modeling, it cannot distinguish between "the error rate is definitely 5%" (measurement) and "the error rate might be 5%" (inference from incomplete logs). Without evidence grounding, it cannot explain why it believes a rollback is necessary. Without reversibility guarantees, a mistaken rollback could cascade into a larger outage.

Academic capability taxonomies, such as the DIS '23 AI Capabilities framework [4], provide valuable classifications of what AI systems can perceive, model, and produce. However, these taxonomies focus on *feature-function* relationships rather than *operational primitives*—they classify capabilities without specifying how to compose them safely or how to handle failures.

We present Grounded Agency, a capability ontology and workflow framework that bridges this gap. Our contributions are:

1. **Capability Ontology** (§3): 99 atomic capabilities organized into 8 layers (Perception, Modeling, Reasoning, Action, Safety, Meta, Memory, Coordination) with formal input/output schemas, mutation flags, and 60 dependency edges.

2. **World State Schema** (§4): A canonical representation for modeling real and digital systems with entities, relationships, state variables, observations, and transition rules. Every element carries provenance records, evidence anchors, and typed uncertainty (epistemic, aleatoric, or mixed).

3. **Trust Model** (§5): A Bayesian conflict resolution system with source authority weights (hardware sensor: 0.95, human note: 0.55), temporal decay ($\tau_{1/2} = 14$ days), and field-specific expertise mapping.

4. **Identity Resolution** (§6): An 8-feature scoring system for entity disambiguation with merge/split policies, hard constraints, and confidence thresholds.

5. **Workflow DSL & Validator** (§7): A domain-specific language for composing capabilities with input bindings, parallel groups, gates, and recovery loops, plus a design-time validator that performs type inference and suggests coercions.

We evaluate the framework on 5 reference workflows (§8), demonstrating that the design-time validator catches 100% of seeded type errors. The complete framework is released as open source.[1]

2 Background and Related Work

2.1 Capability Taxonomies for AI Systems

The systematic classification of AI capabilities has a rich history. Early work focused on cognitive architectures [5, 6] that decomposed intelligence into perception, memory, and action modules. More recently, the DIS '23 framework [4] proposed a taxonomy of AI capabilities in design contexts, identifying verbs like *Detect*, *Identify*, *Estimate*, *Discover*, *Generate*, *Forecast*, *Act*, and *Compare*.

While valuable for understanding AI's functional scope, these taxonomies do not address operational concerns: How should capabilities compose? What happens when one fails? How should conflicts between data sources be resolved? Grounded Agency extends capability taxonomies with operational semantics.

[1] `https://github.com/synapti-ai/grounded-agency`

2.2 Agent Architectures

The emergence of LLM-based agents has spawned diverse architectures. ReAct [7] interleaves reasoning and action in a single prompt chain. Reflexion [8] adds self-reflection loops for error correction. AutoGPT [1] and BabyAGI [9] pursue fully autonomous task decomposition.

Framework libraries like LangChain [10], LlamaIndex [11], and Semantic Kernel [12] provide abstractions for tool use, memory, and chaining. However, none of these frameworks provide:

- Explicit uncertainty typing (epistemic vs. aleatoric)
- Evidence anchors for every claim
- Trust-aware conflict resolution
- Design-time type checking for workflow composition

2.3 World Modeling and Digital Twins

The digital twin concept originated in manufacturing [13] and has expanded to encompass any cyber-physical system where a virtual model mirrors a real-world counterpart [14]. NASA's digital twin vision [15] emphasized high-fidelity simulation for predictive maintenance.

Knowledge graphs [16] provide a complementary approach, representing entities and relationships as typed edges. Google's Knowledge Vault [17] demonstrated extraction at scale, while Wikidata [18] showed the value of community-maintained structured knowledge.

Grounded Agency's world state schema combines elements of both: entities and relationships (from knowledge graphs) with time-indexed state variables and transition rules (from digital twins), unified by a provenance model that grounds every claim in evidence.

2.4 Type Systems for Workflows

Workflow orchestration systems like Apache Airflow [19], Temporal [20], and Prefect [21] provide DAG-based task composition with retry policies and failure handling. These systems focus on job scheduling rather than semantic typing of data flow.

Dataflow type systems [22] ensure that producer outputs match consumer inputs. Gradual typing [23] allows mixing typed and untyped code. Our workflow validator applies these ideas to agent capabilities, inferring types from schemas and suggesting coercions when mismatches occur.

2.5 Safe and Auditable AI

Constitutional AI [24] and RLHF [25] focus on aligning model outputs with human values. These approaches address content safety (what the model says) rather than operational safety (what the agent does).

Process-level safety research examines tool use risks [26], sandboxing [27], and human oversight [28]. Grounded Agency contributes a complementary layer: structural safety through capability dependencies, checkpointing requirements, and reversibility guarantees.

3 Capability Ontology

The capability ontology defines **99 atomic capabilities** that agents can invoke, organized into 8 functional layers with explicit dependencies and contracts.

3.1 Layer Taxonomy

Capabilities are assigned to layers based on their primary function:

- PERCEPTION (4 capabilities): Interface with external data sources. Examples: `retrieve`, `inspect`, `search`, `receive`.
- MODELING (45 capabilities): Construct and maintain world representations. Examples: `world-state`, `state-transition`, `causal-model`, `identity-resolution`, `grounding`.
- REASONING (20 capabilities): Analyze, plan, and decide. Examples: `plan`, `prioritize`, `compare`, `critique`, `decompose`.
- ACTION (12 capabilities): Execute changes in the world. Examples: `act-plan`, `transform`, `send`, `constrain`.
- SAFETY (8 capabilities): Ensure correctness and enable recovery. Examples: `verify`, `audit`, `checkpoint`, `rollback`.
- META (6 capabilities): Discover and compose other capabilities. Examples: `discover-entity`, `discover-pattern`, `invoke-workflow`.
- MEMORY (1 capability): Persistent state across invocations. Example: `remember`.
- COORDINATION (3 capabilities): Multi-agent interaction. Examples: `delegate`, `synchronize`, `negotiate`.

3.2 Capability Schema

Each capability node specifies:

Listing 1: Capability node schema (example: `verify`)

```
id: verify
type: DIS.Level4Verb
layer: SAFETY
risk: medium
mutation: false
requires_checkpoint: false
requires_approval: false

input_schema:
  type: object
  required: [target, criteria]
  properties:
    target: {type: [string, object]}
    criteria: {type: array, items: {type: string}}
    evidence_policy: {type: string, default: anchors_required}

output_schema:
  type: object
  required: [verdict, evidence_anchors, confidence]
  properties:
    verdict: {type: string, enum: [PASS, FAIL, INCONCLUSIVE]}
    failures: {type: array, items: {type: string}}
    evidence_anchors: {type: array, items: {type: string}}
    confidence: {type: number, minimum: 0, maximum: 1}

requires: [inspect]
soft_requires: [search]
```

Key design decisions:

1. **100% Schema Coverage**: All 99 capabilities have both `input_schema` and `output_schema`, enabling static type checking.
2. **Evidence by Default**: Every output schema requires `evidence_anchors` and `confidence`.
3. **Risk Classification**: Capabilities are labeled `low` (88), `medium` (8), or `high` (3) risk. High-risk capabilities (e.g., `act-plan`) require checkpoints.
4. **Mutation Tracking**: 7 capabilities are marked as mutating; these modify external state and require explicit safety measures.

3.3 Dependency Graph

The ontology defines **60 dependency edges** between capabilities:

- **requires** (44 edges): Hard prerequisite. The dependent capability cannot execute unless the required capability has produced output.
- **enables** (11 edges): The source capability produces outputs that make the target more effective.
- **governed_by** (2 edges): Policy or constraint relationship.
- **verifies** (1 edge): The source validates outputs of the target.
- **soft_requires** (1 edge): Preferred but not mandatory.
- **documented_by** (1 edge): Specification linkage.

This graph enables the validator to check that workflows satisfy all hard prerequisites before a capability is invoked.

4 World State Schema

The world state schema provides a canonical representation for modeling both real-world and digital systems. It is designed for agent workflows that require evidence grounding, uncertainty quantification, and reversible transitions.

4.1 Design Principles

The schema is guided by six principles:

1. **Everything is time-indexed**: State variables carry timestamps and validity windows.
2. **Every claim is grounded**: Evidence anchors link assertions to their sources.
3. **Uncertainty is explicit**: Epistemic, aleatoric, and mixed uncertainty are first-class.
4. **Identity is stable and aliasable**: Entities have canonical IDs with alias resolution.
5. **State is separate from observations**: Raw data and derived state are distinct.
6. **Transitions are reversible**: State changes carry rollback specifications.

4.2 Schema Structure

A world state snapshot contains:

Listing 2: World state top-level structure

```
meta:
  world_id: example:payments-service
  as_of: 2026-01-24T00:00:00Z
  version_id: sha256:abc123...
  parent_version_id: sha256:def456...
  lineage: [sha256:def456..., sha256:ghi789...]

entities: [...]         # 18 entity classes, 57 subtypes
relationships: [...]    # Typed edges with uncertainty
state_variables: [...] # Time-indexed measurements
observations: [...]     # Append-only event log
transition_rules: [...] # State machine specifications
actions: [...]          # Executed/planned changes
indexes: {...}          # by_entity, by_variable, by_time
retention_policy: {...} # Lifecycle management
```

4.3 Uncertainty Typing

The framework distinguishes three types of uncertainty:

$$\text{Uncertainty} ::= \text{Epistemic}(c, n) \mid \text{Aleatoric}(c, [l, h]) \mid \text{Mixed}(c, D) \tag{1}$$

where $c \in [0, 1]$ is confidence, n is a textual note, $[l, h]$ is a credible interval, and D is an optional distribution specification.

Epistemic uncertainty represents knowledge gaps—reducible with more evidence. Example: “We infer this service depends on Postgres from config files, but haven’t verified at runtime.”

Aleatoric uncertainty represents inherent randomness—irreducible by gathering more data. Example: “The error rate fluctuates between 0.3% and 0.6% due to traffic variance.”

Mixed uncertainty combines both components. Most real-world measurements involve both measurement error (epistemic) and natural variation (aleatoric).

4.4 Provenance Records

Every entity, relationship, and state variable carries provenance:

Listing 3: Provenance record structure

```
provenance:
  - claim_id: c1
    created_at: 2026-01-24T00:00:00Z
    agent: world-state-builder
    capability: inspect
    anchors:
      - ref: file:services/payments/README.md:12
        kind: file
        excerpt: "Dependencies: postgres-main"
    transformations:
      - "Parsed service metadata from repo docs"
    assumptions:
      - "Repository docs reflect deployed reality"
```

Anchors can reference files (`file:path:line`), URLs, tool outputs, logs, sensors, APIs, or human notes.

4.5 Transition Rules

State transitions are explicit and reversible:

Listing 4: Transition rule example

```
transition_rules:
  - rule_id: tr-error-spike
    trigger:
      event_type: anomaly_detected
      match: {message_contains: '500'}
    guards:
      - error_rate_5m > 0.01
    effects:
      - "set status=degraded for svc:payments-api"
      - "increase alert_level=high"
    allowed_next_states: [healthy, degraded, down]
    rollback:
      strategy: set_previous_status
      inverse_effects:
        - "restore prior status"
```

5 Trust-Aware Conflict Resolution

When multiple sources provide conflicting information about the same entity or state variable, the framework applies a trust-weighted resolution function.

5.1 Source Authority Ranking

Six source types are ranked by default trustworthiness:

Source Type	**Trust Weight**
Hardware Sensor	0.95
System of Record	0.92
Primary API	0.88
Observability Pipeline	0.80
Derived Inference	0.65
Human Note	0.55

Table 1: Default source authority weights

5.2 Temporal Decay

Information trustworthiness decays over time unless refreshed:

$$\text{recency}(t) = \exp\left(-\frac{t - t_{\text{observed}}}{\tau_{1/2}}\right) \tag{2}$$

where $\tau_{1/2} = 14$ days by default. Trust never decays below a minimum threshold of 0.25.

5.3 Conflict Resolution Function

When sources conflict, the winning claim is determined by:

$$\text{score}(\text{claim}) = \text{trust}(\text{source}) \times \text{confidence} \times \text{recency}(t) \tag{3}$$

Tie-breakers apply in order: (1) prefer authoritative source for the specific field, (2) prefer higher confidence, (3) escalate to human.

5.4 Field-Specific Authority

Some fields have designated authoritative sources:

- `serial_number`: Hardware sensor, System of record
- `deployment_version`: Primary API, System of record
- `error_rate_5m`: Observability pipeline

Field authority overrides default source ranking for those specific attributes.

6 Identity Resolution

Entities may appear under different names or identifiers across data sources. The identity resolution policy determines when to merge aliases and when to split incorrectly merged entities.

6.1 Entity Taxonomy

The framework defines **18 entity classes** (10 digital, 8 real-world) with **57 subtypes**:

Digital: service (api, worker, scheduler, gateway), database (postgres, mysql, redis, vectorstore), host (vm, k8s-node, baremetal), container, artifact, document, dataset, job, queue, endpoint.

Real-world: person (employee, customer, contractor), organization (company, team, department), location (site, room, gps-area), asset (machine, vehicle, tool, building), sensor, material, process, event.

6.2 Hierarchical Namespace IDs

Entity identifiers follow a hierarchical convention:

```
<class>:<org>/<domain>/<env>/<system>/<local_id>                (4)
```

Examples:

- `svc:acme/payments/prod/api`
- `db:acme/payments/prod/postgres-main`
- `sensor:acme/factory/line-3/temp-02`

6.3 Alias Confidence Scoring

When two entity references might refer to the same entity, confidence is computed from 8 weighted features:

Feature	Weight	Rule
Exact ID match	1.00	String equality
Namespace match	0.25	Same org/domain/env/system
Type match	0.20	Same entity type
Attribute value match	0.20	High-signal attrs (IP, serial)
Relationship context	0.15	Shared neighbors
Label overlap	0.10	Jaccard similarity
Attribute keys overlap	0.10	Shared attribute keys
Temporal coherence	0.10	Events overlap plausibly

Table 2: Identity resolution feature weights

6.4 Merge/Split Thresholds

Based on confidence score:

- > 0.90: Auto-merge
- 0.75–0.90: Suggest merge (human review)
- 0.60–0.75: No action
- < 0.40: Force split review

Hard constraints prevent catastrophic merges:

1. Never merge different entity types (unless subtype alias)
2. Never merge different org namespaces without exact ID match
3. Never merge if high-signal attributes conflict (e.g., different serial numbers)

7 Workflow DSL and Type Validator

The workflow DSL enables composing capabilities into multi-step processes with data flow, conditions, and error recovery.

7.1 Step Schema

Each workflow step specifies:

Listing 5: Workflow step schema

```
- capability: transform
  purpose: Normalize raw signals into canonical events
  risk: low
  mutation: false
  requires_checkpoint: false
  store_as: transform_out
  input_bindings:
    source: ${receive_out.messages}
    target_schema: canonical_event
  mapping_ref: docs/schemas/transform_mapping.yaml
  output_conforms_to: docs/schemas/event_schema.yaml#/event
  gates:
    - when: ${transform_out.confidence} < 0.5
```

```
      action: stop
      message: Low confidence transformation
  failure_modes:
    - condition: Parse errors exceed threshold
      action: request_more_context
      recovery: Ask for format specification
```

7.2 Data Flow via Bindings

The `${ref}` syntax enables referencing outputs from earlier steps:

- `${receive_out.messages}`: Field access
- `${transform_out.transformed[0]}`: Array indexing
- `${verify_out.failures: array<string>}`: Explicit type annotation

7.3 Advanced Features

Parallel Groups: Steps with the same `parallel_group` execute concurrently:

```
- capability: search
  parallel_group: context_gathering
  join: all_complete
  store_as: search_out
```

Gates: Runtime conditions that halt execution:

```
gates:
  - when: ${checkpoint_out.created} == false
    action: stop
    message: No checkpoint created. Do not mutate.
```

Recovery Loops: Failure handling with evidence injection:

```
failure_modes:
  - condition: Verdict == FAIL
    recovery:
      goto_step: plan
      inject_context:
        failure_evidence: ${verify_out.failures}
      max_loops: 3
```

7.4 Design-Time Type Validator

The validator performs five passes:

1. **Structural**: Capability exists in ontology
2. **Prerequisite**: Dependency edges satisfied
3. **File**: `mapping_ref` and `output_conforms_to` files exist
4. **Reference**: `${store.path}` expressions resolve
5. **Type**: Producer output type $\subseteq$ Consumer input type

The type system supports:

$$\text{Type} ::= \texttt{string} \mid \texttt{number} \mid \texttt{boolean} \mid \texttt{object} \mid \texttt{array<}T\texttt{>} \mid \texttt{nullable<}T\texttt{>} \mid \texttt{map<}K, V\texttt{>} \tag{5}$$

When type mismatch occurs, the validator looks up the coercion registry and suggests inserting a `transform` step with the appropriate mapping.

7.5 Coercion Registry

Five type coercions are pre-defined:

From → To	**Strategy**
`string` → `number`	Parse or null
`number` → `string`	Stable repr
`object` → `string`	JSON stringify (sorted keys)
`array<object>` → `array<string>`	Project field
`array<any>` → `array<object>`	Wrap value

Table 3: Type coercion registry

Each coercion mapping includes a **determinism contract**: pure (same input → same output), no payload dropping, and evidence anchors required when inference occurs.

8 Evaluation

We evaluate Grounded Agency on three dimensions: coverage, validation effectiveness, and comparison with existing frameworks.

8.1 Framework Coverage

Component	**Count**	**Notes**
Capabilities	99	100% schema coverage
Layers	8	Perception to Coordination
Dependency edges	60	44 requires, 11 enables
Entity classes	18	10 digital, 8 real-world
Entity subtypes	57	Extensible taxonomy
Core relations	13	depends_on, owns, causes...
Source types	6	Ranked by trust weight
ID features	8	For alias scoring
Coercions	5	Type transform mappings

Table 4: Framework component coverage

8.2 Workflow Catalog

We developed 5 reference workflows:

The `digital_twin_sync_loop` demonstrates the full capability:

1. `receive`: Ingest signals from logs/APIs

Workflow	Steps	Risk	Parallel	Recovery
debug_code_change	10	medium	0	✓
world_model_build	11	low	0	—
capability_gap_analysis	7	low	0	—
digital_twin_sync_loop	19	high	1	—
digital_twin_bootstrap	2	high	0	—
Total	49	—	1	1

Table 5: Reference workflow characteristics

2. `search`: Parallel context gathering
3. `transform`: Normalize to canonical events
4. `integrate`: Merge with existing twin
5. `identity-resolution`: Resolve entity collisions
6. `world-state`: Produce updated snapshot
7. `state-transition`: Apply transition rules
8. `detect-anomaly`: Identify drift
9. `estimate-risk`: Score severity
10. `forecast-risk`: Project trajectory
11. `plan`: Generate remediation plan
12. `constrain`: Apply safety policies
13. `checkpoint`: Save recovery point
14. `act-plan`: Execute if policy allows
15. `verify`: Check success criteria
16. `audit`: Record provenance
17. `rollback`: Revert on failure
18. `summarize`: Produce human report

8.3 Validation Effectiveness

We seeded 50 errors across the workflow catalog:

- 15 missing capability references
- 12 unresolved `${ref}` expressions
- 10 type mismatches (e.g., `string` where `number` expected)
- 8 missing prerequisite capabilities
- 5 missing file references

The validator detected **100% of seeded errors** (50/50). For the 10 type mismatches, it suggested coercions from the registry for 8 (80%), with the remaining 2 requiring custom transform mappings.

8.4 Comparison with Existing Frameworks

No existing framework provides all seven features. LangChain and AutoGPT focus on task execution without formal grounding. Claude Code Skills provide partial evidence via tool output logging. Semantic Kernel offers partial typing through its plugin schema system but lacks runtime type checking.

Framework	**Uncertainty**	**Evidence**	**Reversible**	**Trust**	**Types**	**Identity**	**World Mo**
LangChain	—	—	—	—	—	—	—
AutoGPT	—	—	—	—	—	—	—
Claude Code Skills	—	Partial	—	—	—	—	—
Semantic Kernel	—	—	—	—	Partial	—	—
Grounded Agency	✓	✓	✓	✓	✓	✓	✓

Table 6: Feature comparison with existing agent frameworks

9 Discussion

9.1 Limitations

Manual capability implementation: While the ontology defines contracts, actual capability implementations must be written manually. Future work could explore capability synthesis from execution traces.

Trust weight calibration: Default weights are based on domain expertise; production deployments may require empirical calibration from historical conflict resolutions.

Coercion completeness: The registry covers common type mismatches but is not exhaustive. Custom domains may require additional mappings.

Design-time only: The validator runs before execution; runtime type checking would catch dynamic errors but adds overhead.

9.2 Broader Impact

Grounded Agency enables a new class of auditable AI agents where:

- Every decision can be traced to evidence
- Uncertainty is communicated rather than hidden
- Actions can be reversed when errors are detected
- Conflicts between data sources are resolved transparently

This transparency is critical for deploying agents in high-stakes domains (healthcare, finance, infrastructure) where opaque decision-making is unacceptable.

9.3 Future Work

Runtime enforcement: Extend the validator to check types at execution time with gradual typing semantics.

Learned trust weights: Use reinforcement learning from conflict resolution feedback to calibrate source authority.

Multi-agent protocols: Extend the Coordination layer with formal protocols for delegation, negotiation, and consensus.

Capability synthesis: Generate capability implementations from natural language specifications using code generation models.

10 Conclusion

We presented Grounded Agency, a capability ontology and workflow framework for building safe, auditable, and composable AI agents. The framework bridges academic capability taxonomies with production infrastructure through 99 atomic capabilities, a world state schema with first-class uncertainty, trust-aware conflict resolution, and a design-time type validator.

Key contributions:

- A capability ontology with 100% schema coverage and 60 dependency edges
- A world state schema distinguishing epistemic, aleatoric, and mixed uncertainty
- A Bayesian trust model with temporal decay and field-specific authority
- An identity resolution policy with 8 weighted features and hard constraints
- A workflow validator that infers types and suggests coercions

The framework enforces three invariants that we believe are essential for trustworthy agentic AI: grounding (every claim has evidence), uncertainty (confidence is typed and explicit), and reversibility (every mutation can be undone).

We release the complete framework—ontology, schemas, workflows, validator, and reference implementations—as open source to support the research community in building agents that are not just capable, but trustworthy.

References

[1] Significant-Gravitas. AutoGPT: An autonomous GPT-4 experiment. GitHub Repository, 2023.

[2] Anthropic. Claude computer use. Technical Report, 2024.

[3] Cognition Labs. Devin: The first AI software engineer. Technical Report, 2024.

[4] Q. Yang, A. Steinfeld, and J. Zimmerman. Exploring AI capabilities for design: A framework of AI-enabled design. In *Proc. DIS '23*, pages 1–15, 2023.

[5] J. E. Laird. *The Soar Cognitive Architecture*. MIT Press, 2017.

[6] J. R. Anderson and C. Lebiere. The Newell test for a theory of cognition. *Behavioral and Brain Sciences*, 26(5):587–601, 2003.

[7] S. Yao, J. Zhao, D. Yu, et al. ReAct: Synergizing reasoning and acting in language models. In *ICLR*, 2023.

[8] N. Shinn, F. Cassano, E. Berman, et al. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

[9] Y. Nakajima. BabyAGI: Task-driven autonomous agent. GitHub Repository, 2023.

[10] LangChain. LangChain: Building applications with LLMs. Documentation, 2023.

[11] LlamaIndex. LlamaIndex: Data framework for LLM applications. Documentation, 2023.

[12] Microsoft. Semantic Kernel: Integrate AI into your apps. Documentation, 2023.

[13] M. Grieves and J. Vickers. Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. In *Transdisciplinary Perspectives on Complex Systems*, pages 85–113. Springer, 2017.

[14] F. Tao, J. Cheng, Q. Qi, et al. Digital twin-driven product design, manufacturing and service with big data. *Int. J. Adv. Manuf. Technol.*, 94:3563–3576, 2018.

[15] E. H. Glaessgen and D. S. Stargel. The digital twin paradigm for future NASA and U.S. Air Force vehicles. In *53rd AIAA/ASME/ASCE/AHS/ASC Structures Conf.*, 2012.

[16] A. Hogan, E. Blomqvist, M. Cochez, et al. Knowledge graphs. *ACM Computing Surveys*, 54(4):1–37, 2021.

[17] X. Dong, E. Gabrilovich, G. Heitz, et al. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *KDD*, pages 601–610, 2014.

[18] D. Vrandečić and M. Krötzsch. Wikidata: A free collaborative knowledgebase. *CACM*, 57(10):78–85, 2014.

[19] Apache Software Foundation. Apache Airflow. Documentation, 2015.

[20] Temporal Technologies. Temporal: Durable execution platform. Documentation, 2020.

[21] Prefect. Prefect: Modern workflow orchestration. Documentation, 2020.

[22] B. C. Pierce. *Types and Programming Languages.* MIT Press, 2002.

[23] J. G. Siek, M. M. Vitousek, M. Cimini, and J. T. Boyland. Refined criteria for gradual typing. In *SNAPL*, 2015.

[24] Y. Bai, S. Kadavath, S. Kundu, et al. Constitutional AI: Harmlessness from AI feedback. arXiv:2212.08073, 2022.

[25] L. Ouyang, J. Wu, X. Jiang, et al. Training language models to follow instructions with human feedback. In *NeurIPS*, 2022.

[26] M. Kinniment, L. Sato, H. Du, et al. Evaluating language-model agents on realistic autonomous tasks. arXiv:2312.11671, 2024.

[27] Z. Xi, W. Chen, X. Guo, et al. The rise and potential of large language model based agents: A survey. arXiv:2309.07864, 2023.

[28] S. R. Bowman, J. Hyun, E. Perez, et al. Measuring progress on scalable oversight for large language models. arXiv:2211.03540, 2022.

A Capability Layer Distribution

B World State Schema (Excerpt)

Listing 6: Uncertainty type definition

```
Uncertainty:
  type: object
  required: [type, confidence]
  properties:
    type:
      type: string
      enum: [epistemic, aleatoric, mixed]
    confidence:
      type: number
      minimum: 0
      maximum: 1
    interval:
      type: object
```

Layer	Count	Key Capabilities
MODELING	45	world-state, state-transition, causal-model, identity-resolution, grounding, simulation
REASONING	20	plan, schedule, prioritize, compare, critique, decompose, infer
ACTION	12	act-plan, transform, send, constrain, mitigate
SAFETY	8	verify, audit, checkpoint, rollback, monitor
META	6	discover-entity, discover-pattern, invoke-workflow
PERCEPTION	4	retrieve, inspect, search, receive
COORDINATION	3	delegate, synchronize, negotiate
MEMORY	1	remember

Table 7: Capability distribution by layer

```
      properties:
        low: {type: number}
        high: {type: number}
    distribution:
      type: object
      description: "e.g. {name: normal, mean: x, stdev: y}"
    notes:
      type: string
```

C Validator Algorithm

Algorithm 1 Workflow Validation

```
Require: Workflow $W$, Ontology $O$, Coercion Registry $C$
Ensure: Errors $E$, Suggestions $S$
 1: $E \leftarrow []$, $S \leftarrow []$
 2: $schemas \leftarrow \{\}$                                    ▷ Store output schemas by step name
 3: for each step $s$ in $W.steps$ do
 4:     if $s.capability \notin O.nodes$ then
 5:         $E$.append(“Unknown capability”)
 6:     end if
 7:     for each $r$ in $O.nodes[s.capability].requires$ do
 8:         if $r$ not yet executed then
 9:             $E$.append(“Missing prerequisite”)
10:         end if
11:     end for
12:     for each binding $(k, v)$ in $s.input\_bindings$ do
13:         $actual \leftarrow$ InferType($v$, $schemas$)
14:         $expected \leftarrow O.nodes[s.capability].input\_schema[k]$
15:         if $\neg$Compatible($expected$, $actual$) then
16:             $E$.append(“Type mismatch”)
17:             if $(actual, expected) \in C$ then
18:                 $S$.append(SuggestTransform($C[actual, expected]$))
19:             end if
20:         end if
21:     end for
22:     $schemas[s.store\_as] \leftarrow O.nodes[s.capability].output\_schema$
23: end forreturn $E$, $S$
```