

# A Neurosymbolic Benchmark for Temporal Knowledge-Graph Memory in Partially Observable Environments

Neurosymbolic Artificial Intelligence  
():1–16  
©The Author(s) 2026  
Reprints and permission:  
sagepub.co.uk/journalsPermissions.nav  
DOI: 10.1177/ToBeAssigned  
www.sagepub.com/

SAGE

Taewoon Kim<sup>1,2</sup>, Vincent François-Lavet<sup>2</sup>, and Michael Cochez<sup>3</sup>

## Abstract

Agents in partially observable environments require persistent memory to integrate observations over time. While KGs (knowledge graphs) provide a natural representation for such evolving state, existing benchmarks are not well suited to isolating long-term memory as the core challenge: they typically do not expose KG-shaped hidden states, do not require persistent tracking of time-varying facts, or do not permit direct inspection of what an agent remembers. We introduce the RoomKG Benchmark, a configurable neurosymbolic benchmark instantiated through a room environment whose hidden state is a Resource Description Framework (RDF) KG and whose observations are RDF triples. The agent may extend these observations into a temporal KG when storing them in long-term memory. The benchmark is easily adjustable in terms of grid size, number of rooms, inner walls, and moving objects.

To evaluate this benchmark, we define a lightweight temporal KG memory for agents in which every stored statement carries per-statement temporal annotations (:time\_added, :last\_accessed, :num\_recalled), following the annotation model of RDF 1.2, and evaluate several symbolic baselines that maintain and query this memory under different capacity constraints. Two neural sequence models (Long Short-Term Memory (LSTM) and Transformer) serve as contrasting baselines without explicit KG structure. Agents train on one environment and are evaluated on a held-out environment with the same dynamics but a different question order, exposing train–test generalization gaps in long-term memory rather than in perception. In this setting, temporal annotations lead to more stable performance, and the symbolic TKG (temporal knowledge graph) agent achieves more than three times higher test question answering (QA) accuracy than the neural baselines under the same benchmark and query conditions. RoomKG Benchmark is therefore a compact testbed for asking when explicit long-term memory matters and which memory representations support it. The benchmark definition, agent implementations, and experimental scripts are archived and released as open-source software for reproducible research.

## Keywords

neurosymbolic benchmark, partial observability, temporal knowledge graphs, agent memory

## 1. Introduction

Agents operating in partially observable environments must infer and maintain structured information about the world as it evolves over time (Kaelbling et al. 1998). Because the world evolves according to structured dynamics, agents must store and update facts observed many steps earlier. Knowledge graphs (KGs) provide a natural representational substrate for such settings: entities, relations, and annotations can express spatial structure, object locations, and temporal metadata in a uniform semantic framework (Ehrlinger and Wöß 2016; Hogan et al. 2021; Battaglia et al. 2018; Zambaldi et al. 2019). However, despite substantial work on relational and symbolic environments, most existing benchmarks are not designed to evaluate long-term memory itself. They do not jointly provide KG-shaped hidden states, temporally evolving facts that must be remembered over many steps, and an evaluation interface that makes the contents and use of memory inspectable (e.g., (Battaglia et al. 2018; Zambaldi et al. 2019; Côté et al. 2018; Hausknecht et al. 2020; Ammanabrolu and Hausknecht 2020; Wayne et al. 2018; Graves et al. 2014, 2016)). Recent discussions of benchmark quality in **neuro-symbolic AI** **neurosymbolic artificial intelligence (AI)** further emphasize the need for benchmarks

that clearly specify their task, metrics, data generation process, and intended uses (Manhaeve et al. 2025; Bortolotti et al. 2024; Hu et al. 2025).

This paper introduces the RoomKG Benchmark, a deterministic and fully configurable neurosymbolic benchmark designed to fill this gap. The underlying environment's hidden state is an **RDFa Resource Description Framework (RDF)** KG whose entities include rooms, objects, walls, and the agent, and whose relations encode spatial adjacency and object locations. At each timestep the agent receives a symbolic observation consisting of RDF triples describing the local room structure and the objects currently present. The benchmark is easily adjustable in terms of grid size, number of rooms, pattern of inner walls, and the number and motion patterns of moving objects. A further component of

<sup>1</sup>HumemAI, The Netherlands

<sup>2</sup>Vrije Universiteit Amsterdam, The Netherlands

<sup>3</sup>ELLIS Institute Finland and Åbo Akademi University, Finland

## Corresponding author:

Taewoon Kim, HumemAI, The Netherlands.

Email: taewoon@humem.ai

the benchmark is a query at each step requesting the current location of a named object. Accurate responses require maintaining an internal representation of the world that integrates partial observations over time. **The point of the benchmark is not merely to expose structured observations, but to force agents to demonstrate persistent long-term memory under controlled neurosymbolic conditions. A benchmark that only exposes structured observations does not test this.**

While KGs and temporal KGs are widely used for representing evolving knowledge (Hogan et al. 2021; Dell’Aglia et al. 2017; Moreau and Missier 2012; García-Durán et al. 2018), their application as *agent memory* in interactive environments remains underexplored. Prior work on gridworlds, symbolic environments, and relational reasoning benchmarks typically either (i) does not provide a KG-shaped hidden state, (ii) gives agents unstructured symbolic or text observations without a defined KG memory model, or (iii) uses neural representations that do not expose explicit triples or annotations for inspection (Côté et al. 2018; Hausknecht et al. 2020; Ammanabrolu and Hausknecht 2020; Kim et al. 2023; Battaglia et al. 2018; Zambaldi et al. 2019; Graves et al. 2014, 2016; Wayne et al. 2018; Ha and Schmidhuber 2018). **Crucially, these** These limitations make it hard to answer the benchmark question we care about: when an agent must remember a changing world over long horizons, what memory representation actually helps? As a result, there is limited understanding of how explicit temporal KGs function as internal state representations, how simple symbolic update rules perform relative to neural sequence models, and how temporal annotations influence generalization.

This benchmark gap is especially important because existing **The gap matters because** work on long-term memory has largely emphasized latent neural representations rather than explicit symbolic ones. Recent work on long-term memory in artificial intelligence has increasingly focused on neural vector-based representations, where stored information is encoded in continuous embeddings rather than explicit symbolic structures (Graves et al. 2014, 2016; Wayne et al. 2018; Ha and Schmidhuber 2018; Pritzel et al. 2017; Blundell et al. 2016). While such approaches are powerful, they often lack interpretability: the contents of memory and the basis of retrieved answers are not directly observable as explicit symbolic facts. One of the motivations for our contribution is to **which are powerful but whose contents cannot be read off directly. We therefore** provide a setting where both the hidden state of the world and the agent’s long-term memory are **represented as explicit KGs. This enables interpretable memory inspection, deterministic updates, and transparent reasoning** explicit KGs: the memory can be read at any timestep, and every answer the agent gives can be traced to the stored triples it came from.

We propose a lightweight temporal KG memory model for agents that extends standard RDF triples with **RDF 1.2 triple per-statement temporal** annotations capturing `:time_added`, `:last_accessed`, and `:num_recalled`. These annotation properties **provide a simple but expressive mechanism for tracking track** the recency and usage of stored facts, **enabling memory strategies that remain fully so the memory strategies built on them stay** interpretable (Missier et al. 2013). Based on this **TKG temporal knowledge graph (TKG)** framework, we implement several symbolic baselines that update and

query their memory using deterministic rules, with optional capacity limits and simple eviction heuristics. To provide a contrasting perspective, we include two neural sequence models (**LSTM Long Short-Term Memory, LSTM,** and Transformer) that receive exactly the same symbolic observations but maintain memory as a fixed-length sequence queue buffer without explicit KG structure.

We evaluate all agents on two deterministic layouts: a training environment and a held-out test environment with identical dynamics but a different question order. This setup **enables a controlled examination of isolates** train-to-test generalization **and highlights the importance of structured temporal : perception is identical in both environments, so any gap comes from memory.** Across capacities, temporal annotations contribute to more stable behavior, and the symbolic TKG agent **exhibits significantly higher test performance reaches more than three times higher test accuracy** than the neural baselines under the same benchmark and query conditions. **The benchmark is therefore intended not only as a new environment, but We therefore intend the benchmark** as a diagnostic instrument for studying long-term memory in neurosymbolic agents. The benchmark, agent implementations, and full reproducibility material are publicly available via Zenodo and GitHub.

## Contributions

- We introduce the RoomKG Benchmark, a deterministic and configurable neurosymbolic benchmark whose hidden state and observations are expressed as RDF knowledge graphs.
- We formalize a benchmark task, benchmark statistics, and benchmark metrics for object-location question answering under partial observability.
- We define a lightweight temporal KG memory with **RDF per-occurrence temporal annotations, encoded as RDF reification and released in RDF 1.2 triple annotations and form as well,** with deterministic update rules.
- We compare **symbolic TKG agents** two symbolic agents, one over plain KG memory and one over temporally annotated TKG memory, with neural sequence baselines under a shared interface and varying memory capacities.
- We release the benchmark artifact through a stable Zenodo record together with the benchmark software, baseline agents, and reproducibility scripts.

## 2. Background

### 2.1. RDF, RDF 1.2 Triple Annotations, and Knowledge Graph Representations

The Resource Description Framework (RDF) **RDF** models facts as triples  $(s, p, o)$  with subject, predicate, and object drawn from **IRIs Internationalized Resource Identifiers (IRIs)** or literals. An RDF graph therefore corresponds to a directed, edge-labeled multigraph in which relations capture structured connections between entities. This representation is well suited for spatially structured domains: rooms, objects, and adjacency relations can be expressed directly as triples without task-specific encodings.

RDF 1.2 extends RDF with triple terms, reifiers, and annotation syntax for *statements about statements* (Kellogg et al. 2026; Kellogg and Tomaszuk 2026). In concrete syntax, a triple such as  $\langle\langle s p o \rangle\rangle$  can be annotated with metadata such as `:time_added` or `:last_accessed`, giving a concise notation for temporal or provenance-style annotations on a base fact. **RDF 1.2 reifiers** allow several independent annotation sets to be attached to distinct occurrences of the same base triple; an agent memory needs this whenever the same fact is observed at different times. For agent memory, this provides a lightweight and standard-compatible way to associate temporal metadata with observed facts.

**Reification and RDF 1.2.** Each memory in the temporal KG model is one annotated occurrence of a base triple; the KG agent, defined later as a baseline, stores the bare triple with no annotations. The implementation predates RDF 1.2 and encodes each occurrence with standard RDF reification (`rdf:Statement` with `rdf:subject`, `rdf:predicate`, `rdf:object`), which also remains what the Python RDF ecosystem supports: `rdflib` does not parse RDF 1.2 syntax. RDF 1.2 has since specified the same pattern (Candidate Recommendation at the time of writing), a reifier identifying an occurrence of a triple term, and we release every memory snapshot in that form as well; Apache Jena parses and queries it. Reification suits the requirement that each memory be independently addressable, with its own annotation set and identifier, even when the same base triple is stored several times, with one reification node per memory playing the role of a reifier. The two encodings are one-to-one, as Listing 1 shows, and interchangeable for everything done in this paper: the agents read the base triple and the annotation values of each occurrence, which are identical in both, and perform no RDF entailment, which is where the two would part company. We therefore describe the memory in RDF 1.2 terms throughout and treat the reification as an implementation detail.

Listing 1: The same memory entry in RDF 1.2 annotation syntax with a named reifier (top) and in the RDF 1.0-compatible reification serialization that current tooling can load and query (bottom). The released artifacts include the memory snapshots in both serializations. The reifier `:m42` and the statement node play the same role: they make this particular occurrence of the base triple addressable and annotatable. For readability we show step indices; the artifacts serialize these values as `xsd:dateTime` literals.

```
# RDF 1.2 annotation syntax
:nightstand :at_location :library ~ :m42
  {} :time_added 17 ;
    :last_accessed 42 ;
    :num_recalled 3 {} .

# RDF 1.0-compatible reification
:m42 a rdf:Statement ;
  rdf:subject :nightstand ;
  rdf:predicate :at_location ;
  rdf:object :library ;
  :memory_id 42 ;
  :time_added 17 ;
  :last_accessed 42 ;
  :num_recalled 3 .
```

The released artifacts include serialized Turtle snapshots of agent memories in both serializations (reification for loading with current tooling, RDF 1.2 annotation syntax for inspection), so the representation can be examined directly without re-running the code.

## 2.2. Temporal Knowledge Graphs

**Temporal knowledge graphs (TKGs)** TKGs represent facts whose validity or relevance changes over time. Common approaches include interval-based, event-based, and update-based models, all of which associate each triple with temporal metadata reflecting when it was introduced or last observed. Several lines of Semantic Web work address time in RDF: Temporal RDF attaches validity time to triples (Gutierrez et al. 2005), validity time can be given a logic-based semantics in RDF and the Web Ontology Language (OWL) (Motik 2012), named graphs and quads carry statement-level context and provenance (Carroll et al. 2005), spatiotemporal RDF (stRDF) and its query language stSPARQL extend RDF for spatio-temporal data (Koubarakis and Kyzirakos 2010), OWL-Time provides a vocabulary for temporal entities (Cox and Little 2017), and RDF stream processing handles continuously arriving triples (Dell’Aglio et al. 2017). RDF 1.2 can express such per-statement metadata by attaching annotation key–value pairs to annotated triples, enabling fine-grained annotations such as `:time_added`, `:last_accessed`, or `:num_recalled` (Kellogg et al. 2026; Kellogg and Tomaszuk 2026; Moreau and Missier 2012). Temporal KGs have been widely used in the Semantic Web to model evolving data, streaming updates (Dell’Aglio et al. 2017), and temporally indexed knowledge sources

An important distinction for this paper: our annotations record *memory-management metadata* (when a fact was observed, last retrieved, and how often it was retrieved), not *temporal validity* (during which interval a fact holds in the world). A stored `:at_location` statement with an old `:time_added` may or may not still be true; deciding which stored occurrence to trust is the job of the memory policies introduced below. This places our representation closer to provenance-style annotation (Moreau and Missier 2012) than to validity-time models (Gutierrez et al. 2005; Motik 2012).

In interactive settings, temporal annotations allow agents to track the recency and usage of stored information, to maintain evolving internal models, and to reason over past observations. These properties align naturally with the needs of partially observable environments, where an agent’s internal state must integrate information gathered over many timesteps.

## 2.3. Symbolic Memory and Semantic State Tracking

Research on long-term memory for artificial agents has increasingly explored neural vector-based representations, where stored information is encoded implicitly in embeddings or hidden states (Graves et al. 2014, 2016; Wayne et al. 2018; Ha and Schmidhuber 2018; Pritzel et al. 2017; Blundell et al. 2016). While effective for high-dimensional signals, such representations are typically opaque: the contents of memory cannot be inspected directly as noted in neural-symbolic systems (Besold et al. 2017), and the basis for retrieval is difficult to trace.

In contrast, symbolic memory systems store explicit facts that can be examined, queried, and updated deterministically. KGs offer a particularly suitable substrate for this purpose, as they represent entities and relations at a semantic level and support principled operations for insertion, deletion, and reannotation. When temporal annotations are incorporated, symbolic memory becomes fully transparent: each fact carries a clear history of when and how it was observed.

Heuristic update strategies such as recency-based selection, frequency-based selection, and simple eviction rules such as first-in, first-out (FIFO), least recently used (LRU), and least frequently used (LFU) can operate directly over annotation values attached to embedded triples. Such heuristics preserve interpretability while providing meaningful signals for deciding which facts to retain when memory capacity is limited.

### 2.3. Semantic Representations for Agents and Interactive Environments

Semantic Web research has long explored structured representations, logical reasoning, and knowledge-driven decision processes. Several interactive or agent-based settings use symbolic descriptions of the world, but the underlying environments are rarely expressed directly as KGs, and the agent’s internal memory is typically not a temporal KG. Existing gridworld-style or relational reasoning benchmarks often provide symbolic observations without specifying how those observations should be integrated into a structured, evolving memory.

As a result, there is limited empirical understanding of how temporal KGs function as internal state representations for agents, how simple annotation-driven heuristics compare to unstructured sequence-based memory, and how temporal metadata affects generalization in deterministic but partially observable domains. The benchmark and memory model introduced in this paper address these gaps by providing a setting in which both the hidden state of the world and the agent’s memory are explicit, inspectable knowledge graphs represented in standard RDF and RDF 1.2 annotation syntax.

## 3. RoomKG Benchmark

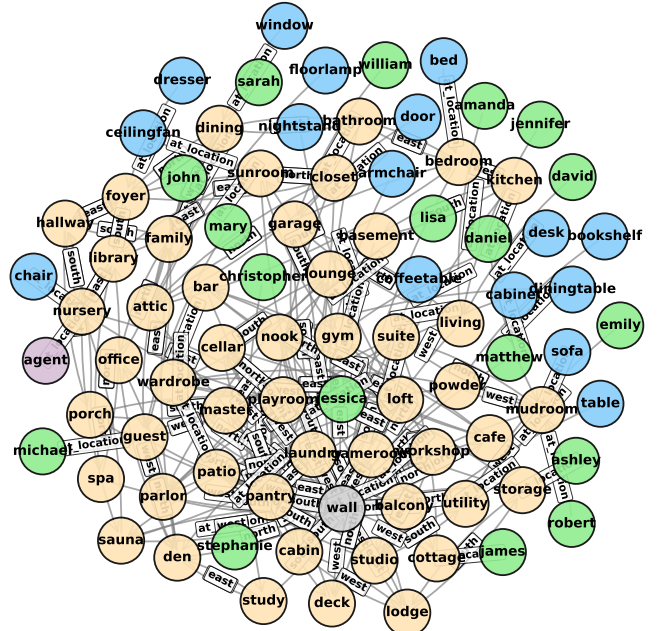
The RoomKG Benchmark (Figures 1 and 2) is instantiated through a configurable  $g \times g$  grid world populated with named rooms, static and moving objects, and periodically changing inner walls. All benchmark components, namely (hidden dynamics, observations, and the QA loop, question answering (QA) loop) are defined in RDF triples, making the benchmark a minimal but fully KG-shaped testbed for studying temporal memory.

The benchmark deliberately couples navigation and question answering. Because objects move and visibility is local, correct answers require exploration, accumulation of partial observations, and persistent memory of object trajectories. This design prevents trivial solutions: without navigation the task reduces to static KG lookup, and without queries exploration has no semantic purpose. Their combination creates a controlled setting in which temporal memory is both necessary and measurable.

The benchmark generalizes the earlier setting introduced by Kim et al. (Kim et al. 2023): instead of a single fixed layout, our benchmark supports arbitrary grid sizes, heterogeneous wall schedules, multiple moving objects with independent motion rules, and exposes the entire hidden state as an RDF KG; when used as long-term memory inside the agent, this can be extended into a temporal KG via annotations.

|                                       |                                     |                       |                  |                           |                          |                             |
|---------------------------------------|-------------------------------------|-----------------------|------------------|---------------------------|--------------------------|-----------------------------|
| living<br>David<br>jennifer<br>amanda | kitchen                             | bedroom<br>bed        | bathroom         | office                    | den<br>john<br>jessica   | study<br>christopher        |
| garage<br>door<br>william             | basement                            | attic<br>floorlamp    | dining<br>window | family<br>sarah           | guest<br>mary<br>michael | master                      |
| laundry<br>armchair                   | pantry                              | closet                | foyer<br>dresser | hallway                   | porch                    | deck                        |
| patio<br>wardrobe                     | balcony                             | sunroom<br>ceilingfan | library          | nursery<br>chair<br>agent | playroom                 | gameroom<br>coffeetable     |
| studio<br>table                       | workshop                            | gym<br>desk           | spa              | sauna<br>wall             | cellar                   | storage<br>ashley           |
| utility<br>emily                      | mudroom<br>sofa<br>robert<br>daniel | powder<br>bookshelf   | wardrobe         | loft<br>nightstand        | cabin<br>matthew         | lodge                       |
| cottage<br>james                      | suite                               | parlor<br>cabinet     | lounge           | bar                       | cafe<br>stephanie        | nook<br>dininatable<br>lisa |

**Figure 1.** Bird’s-eye schematic of the hidden state at  $t = 99$  ( $s_{t=99}$ ), showing spatial layout and entity placement. This view is only a schematic for intuition: the actual environment state and agent-facing world are represented in the RDF knowledge-graph world (Figure 2). The agent does not directly observe  $s_t$ ; its observation  $o_t$  is the induced RDF subgraph of the current room and visible adjacency relations.



**Figure 2.** Knowledge-graph view of the same hidden state at  $t = 99$  ( $s_{t=99}$ ), expressed as RDF structure. Node colors indicate semantic categories: rooms (yellow), agent (purple), static objects (blue), moving objects (green), and walls (grey). The figure conveys the overall scale and structure; a readable fragment of this same state in Turtle, at the same timestep, is given in Listing 2.

### 3.1. Benchmark Metadata, Scope, Agent Interface and Intended Use Task Protocol

The public benchmark artifact is archived at Zenodo as a software release with DOI , publication date 2026-02-21, and version `benchmark`

defines a minimal observation–query–action protocol. At every timestep  $t$ , an agent (i) receives a symbolic observation  $o_t$ : the RDF triples describing its current room (defined in “Observations as RDF Graph Fragments” below); (ii) receives one *location query* ( $X, :at\_location, ?$ ) for a named object  $X$ ; this is the sole query class in the current benchmark version; (iii) returns an answer (a room) and a movement action from  $\{\text{north, east, v3.0.6.}$  The archived benchmark software is based on the public repository , which is released under the MIT license; the baseline-agent implementations used south, west, stay}. The benchmark maintains only the hidden state  $s_t$  and the emitted observations; it does not prescribe any internal memory structure. Agents may keep any private state they choose: an explicit KG, a temporal KG, a token buffer, a latent vector, or nothing at all.

The environment evaluates each answer against  $s_t$  and emits a per-step correctness signal  $r_t \in \{0, 1\}$ . For all agents,  $r_t$  is the *evaluation* metric (QA accuracy is the episode sum, at most 100). Whether an agent additionally uses  $r_t$  as a *training* signal is up to the agent: the symbolic agents in this paper are hosted separately at under the Apache License rule-followers that never consume the reward, whereas the neural baselines are trained on it with a reinforcement-learning objective (see “Agents”). The benchmark itself is therefore agnostic: it supports symbolic agents, reinforcement learning (RL) agents, neural sequence models, large language model (LLM) based agents, or any other system that satisfies the observation–query–action protocol (although the present comparison deliberately excludes LLM-based agents; see the scope discussion below).

### 3.2. Benchmark Metadata, Scope, and Intended Use

Both artifacts are public and archived. The benchmark environment is <https://github.com/humemai/room-env> (MIT License), archived as v3.0.6 under <https://doi.org/10.5281/zenodo.18721817>; the baseline agents reported here are <https://github.com/humemai/roomkg-baselines> (Apache 2.0), archived as v0.1.5 under <https://doi.org/10.5281/zenodo.21967778>. Each Digital Object Identifier (DOI) resolves to the exact version used for the reported results. Because the benchmark is procedurally specified rather than collected from external annotators, its source data consists of author-designed layouts, periodic wall patterns, object inventories, movement rules, and deterministic query schedules. Maintenance is therefore software-centered: new benchmark versions are created through versioned repository releases and archived snapshots.

In scope, RoomKG Benchmark targets neurosymbolic systems that must integrate symbolic observations over time, maintain explicit or implicit world models, and answer semantic queries under partial observability. The primary task is object-location question answering during navigation. The primary evaluation metric is QA accuracy; the benchmark also reports room coverage, triple coverage, and memory-state visualizations to expose how different memory

representations support or hinder the task. We therefore intend the benchmark as a compact testbed for temporal memory, symbolic state tracking, and generalization studies, rather than as a full embodied-AI benchmark covering language, continuous control, or open-ended planning. For the same reason, the present comparison does not include large-language-model-based agents: their performance would conflate bounded memory effects with pretraining, prompt design, context-window management, and optional tool or retrieval configurations, which would blur the paper’s focus on controlled long-term memory mechanisms under fixed symbolic observations.

With this scope in place, we next specify the benchmark dynamics and observation model that create these long-term memory demands.

### 3.3. Hidden State and Dynamics

At timestep  $t$ , the hidden state  $s_t$  consists of the on/off status of all inner walls, the positions of all static and moving objects, and the agent’s current room.

Each room is a node, and cardinal directions are modeled as labeled edges (north, east, south, west). For a given room, a direction can point either to an adjacent room or to a wall, e.g., (`roomAlibrary`, `northeast`, `roomBnursery`) or (`roomAlibrary`, `southnorth`, `wall`). This representation keeps blocked and unblocked directions explicit in the graph.

Every moving object is equipped with a fixed, ordered preference list drawn from  $\{\text{north, east, south, west, stay}\}$ . At each step it executes the *first feasible* move in its list; if none are feasible, it remains in place. This rule, combined with wall schedules, makes object motion *deterministic*.

The inner walls follow fixed periodic schedules: if inner wall  $i$  has a pattern of length  $|p_i|$ , then the joint wall configuration repeats every  $P_w = \text{lcm}(|p_1|, \dots, |p_W|)$  steps, where  $\text{lcm}$  denotes the least common multiple. Moving objects induce an additional period  $P_m$ . After an initial transient  $T_{\text{settle}}T_{\text{settle}}$ , the global state obeys

$$s_{t+P} = s_t, \quad P = \text{lcm}(P_w, P_m),$$

so the benchmark environment is a finite deterministic system with a compact periodic orbit.

Although deterministic, the state space is combinatorially large. With  $R = g^2$  rooms,  $M$  moving objects, and  $W$  inner walls with pattern lengths whose open/closed schedules have periods  $|p_1|, \dots, |p_W|$ , a coarse upper bound on the state space is the moving parts of the state are bounded by

$$|S| \leq R^M \prod_{i=1}^W |p_i|.$$

configurations. For our running configuration ( $g=7$ ,  $R=49$ ,  $M=18$ ,  $W=36$ ), this exceeds  $6.6 \times 10^{33}$ , wall periods between 2 and 10) that bound is  $3.4 \times 10^{57}$ . Exact planning is infeasible; agents must instead reason over their own growing memory  $M_t$ .

The periodic wall schedule and deterministic object motion are intentional design choices: they ensure that the hidden state evolves in a structured but nontrivial way,

making it impossible for an agent to answer location queries from the current observation alone. Because objects move and visibility is local, accurate question answering requires integrating partial observations over many steps. Periodicity keeps the benchmark fully deterministic and replayable while still enforcing genuine long-term memory demands.

A detailed description of the periodicity analysis, including closed-form bounds and deterministic replay examples, is provided in the public benchmark artifacts. Listings 2 and 3 excerpt one such orbit: the same room at  $t=99$  and  $t=0$ , its northern edge open in the first and a wall in the second. Replay scripts reproducing the alternation step by step are in the released code.

Listing 2: A fragment of the hidden state  $s_t$  in Turtle at  $t=99$ : room adjacency, walls as explicit blocked directions, and an object placement. These components are fixed by the configuration and the timestep alone, independently of the agent’s trajectory. Unlike the observation in Listing 3, the full state is never visible to the agent, whose observation is the subgraph induced by its current room alone. Comparing `:library :north` here with the same edge at  $t=0$  in Listing 3 shows the periodic wall dynamics: the northern wall of `:library` is closed at  $t=0$  and open at  $t=99$ .

```
:library :north :foyer ;
        :east  :nursery ;
        :south :spa ;
        :west  :sunroom .
:spa     :north :library ;
        :east  :wall ;
        :south :wall ;
        :west  :wall .
:chair :at_location :nursery .
```

Listing 3: A complete observation  $o_t$  in Turtle, from the released memory snapshots ( $t=0$ , reformatted with a prefix for readability): local adjacency (walls make blocked directions explicit) plus the agent and any objects currently in the room.

```
:library :north :wall ;
        :east  :nursery ;
        :south :spa ;
        :west  :sunroom .
:agent :at_location :library .
```

### 3.4. Observations as RDF Graph Fragments

At timestep  $t$ , the agent receives an observation  $o_t$  consisting of the RDF subgraph induced by its current room: the room node for the agent’s location, any one outgoing direction-labeled edges to adjacent rooms that are not blocked by walls for each of the four cardinal directions, pointing either to the adjacent room or to `:wall` where the direction is blocked, and object–location triples for all objects currently in that room. This yields 5–6 triples per step. The very first observation of the released trace, with the agent starting in the `library` room, is Listing 3.

Observations contain no information beyond the agent’s immediate neighborhood, so long-horizon queries require integrating partial observations over time.

The benchmark couples these observations with an explicit query stream. Each timestep also issues a *location* query of the form:  $(X, :at\_location, ?)$ . The agent must

identify the current room of the queried object. This requires maintaining a temporal model of object motion as observed indirectly through partial glimpses, echoing aspects of KG-based QA (question-answer) question answering (Yih et al. 2015).

### 3.5. Deterministic Question–Move Loop

At each timestep the agent first answers the query by returning the room of the specified object (reward +1 if correct, otherwise 0), and then executes a movement action chosen from `{north, east, south, west, stay}`. Each timestep  $t$  executes the following fixed sequence:

1. the environment poses the query  $(X_t, :at\_location, ?)$ ;
2. the agent answers with a room; correctness is evaluated against the *current* hidden state  $s_t$ , yielding  $r_t \in \{0, 1\}$  (as defined in “Agent Interface and Task Protocol”);
3. all moving objects execute their deterministic motion rules, through the wall layout in force at  $t$ ;
4. the agent’s movement action is applied;
5. the wall schedules advance, and the step counter is incremented;
6. the environment emits the next observation  $o_{t+1}$ .

The order above is fixed by the environment. When an agent writes to memory is left to the agent: the benchmark constrains what it observes and what it must answer, not how or when it stores anything. The agents studied here update on receiving  $o_{t+1}$ . Answers are thus graded against the state at query time, before any movement occurs.

Episodes last 100 steps, a length chosen to balance tractability with sufficient temporal depth. The full query sequence for each episode is pre-generated and fixed across all agents, ensuring strict comparability in evaluation.

### 3.6. Configurable Layouts and Test Split

The benchmark is fully parameterized by the grid length  $g$ , the number and placement of static and moving objects, the number and periodic patterns of inner walls, and the episode length.

We use two deterministic `layouts` `environment` instances: a training `layout` with fixed walls and object preferences, `environment` and a held-out test `layout` with identical dynamics but a different query order. This difference in question order `environment`. Table 1 makes explicit which components are shared and which differ: the spatial layout, wall schedules, object inventories, and motion rules are *identical*; the only difference is the order of the query stream. This yields a controlled train–test generalization setting in which agents must transfer their learned memory strategy rather than memorize a fixed query sequence.

### 3.7. Memory Interface (Agent-Agnostic)

The benchmark maintains only the hidden state  $s_t$  and the symbolic observation  $o_t$  emitted at each timestep. It does not prescribe any internal memory structure: agents may keep any private state they choose. The benchmark simply provides RDF observations, receives an action and an answer, and evaluates correctness against  $s_t$ .

In our experiments, we compare four agents. The

| Component                    | Train                                | Test             |
|------------------------------|--------------------------------------|------------------|
| Grid and room layout         | identical ( $7 \times 7$ , 49 rooms) |                  |
| Walls and periodic schedules | identical (36 walls)                 |                  |
| Objects and motion rules     | identical (18 + 18)                  |                  |
| Episode length               | identical (100 steps)                |                  |
| Query pool                   | identical                            |                  |
| Query order                  | fixed order A                        | permuted order B |
| Agent seeds (learned)        | 5 seeds each, same set               |                  |

**Table 1.** [Added in this revision] Train/test protocol: shared vs. differing components. Environment dynamics are deterministic and identical in both instances; only the question order differs.

### 3.7. Default Configuration

Unless stated otherwise, experiments use a `grid_length` of 7, with 18 static objects, 18 moving objects, 36 inner walls following periodic patterns, an episode length of 100, and a fixed query list shared across all agents. The reward is simply the number of correct queries (maximum 100). All benchmark code, layouts, and deterministic replay scripts are included in the public benchmark artifacts for full reproducibility and layouts are released with the artifacts.

## 4. Agents

We evaluate four agents: two *symbolic* agents that store explicit KG structures, and two *neural* agents that store tokenized observation histories. All agents receive the same query sequence and operate in the same benchmark, but their exploration behaviors cause them to generate different observation histories. We evaluate them under memory capacities. Each agent maintains a private memory  $M_t$ ; for the symbolic agents we write  $M_t = (M_t^{\text{short}}, M_t^{\text{long}})$ , where the short-term part holds only the current observation and the long-term part holds retained facts.

**Capacity semantics.** All agents are evaluated under long-term memory capacities  $K$  from 0 to 512 entries, which covers the range where capacity meaningfully constrains what can be stored, where  $K$  counts stored entries. One entry is one observed triple together with whatever the agent keeps alongside it: for the TKG agent an annotated occurrence of that triple, for the KG agent the bare triple, and for the neural agents one tokenized triple in a buffer of length  $K$ . The counting unit is common to all three, so  $K$  is directly comparable; what differs is how much each entry carries and how fast entries accumulate. Since an episode produces only  $\sim 500$ – $600$  observation triples in total, capacities of  $K \geq 512$  approach exhaustive retention (see “Calibration Bounds”).

The same  $K$  is not equally scarce for the two symbolic agents, however. The KG agent stores plain triples in a set, so re-observing a fact collapses onto the entry already held: over a full episode its long-term memory retains roughly 300 distinct entries, below  $K = 512$ , and it therefore never evicts, which is why its scores at  $K = 512$  and  $K = 1024$  coincide exactly. The TKG agent instead records each observation as a separate annotated occurrence, so repeated observations of the same triple accumulate as distinct entries (visible as the occurrence counts in Figure 5) and the capacity bound is reached within an episode. A capacity of 0 means that no long-term memory is retained across timesteps, so agents

must act only on the current observation and any non-persistent internal computation available within that step. For the TKG agent, the candidate query and exploration policies are

**Three policy families.** The TKG agent’s behavior is defined by three independent policy choices, all ranking stored statements by their annotation values: (a) a *QA policy* selects which matching statement answers the current query: most recently added (MRA), most recently used (MRU), and or most frequently used (MFU), while the candidate eviction policies are FIFO, LRU, and LFU; (b) an *exploration policy* decides, using the same annotation signals, which stored edge statement to trust when several describe the same direction, and so determines the map the agent searches; and (c) an *eviction policy* selects which statement to drop at capacity: first in first out (FIFO), least recently used (LRU), or least frequently used (LFU). These are orthogonal because every stored statement carries all three annotations. Table 2 summarizes the four agents.

### 4.1. Symbolic agents

Both symbolic agents store explicit triples. Their difference is whether they attach temporal annotations.

**KG agent (plain RDF triples).** Memory is a bounded set of RDF triples  $(s, p, o)$ . Because these entries carry no timestamps or usage counts, the agent has no basis for ordering them. Therefore, when memory is full, the only principled choice is to evict a *uniformly random* triple.

**QA rule.** Answers are obtained by plain SPARQL (SPARQL Protocol and RDF Query Language) pattern matching:

```
SELECT ?o WHERE { <s> <r> ?o . }
ORDER BY RAND() LIMIT 1
```

If multiple candidates match, tie-breaking is uniform because the agent lacks temporal metadata.

**Exploration.** The current map is reconstructed from the triples stored so far. The agent performs BFS (breadth-first search) to the nearest unvisited room. BFS is used because the memory itself is a graph and the memory is itself a graph, so BFS is the simplest deterministic traversal aligned with that structure.

**TKG agent (RDF 1.2 annotations temporally annotated occurrences).** Each observed triple  $(s, r, o)$  is stored together with annotation assertions over its annotated form, written here using the concise notation  $\ll s r o \gg$  with :

```
:time_added, :last_accessed, :num_recalled.
```

```
:time_added, :last_accessed, and
:num_recalled. This creates a temporal knowledge
graph  $M_t M_t^{\text{long}}$ , one annotated occurrence per observation
(Listing 1).
```

**Memory-operation semantics.** The life cycle of a memory is fully deterministic:

- **Insertion:** when an observed triple is retained at step  $t$ , a new annotated occurrence is created with  $:time\_added = t$ ,  $:last\_accessed = t$ , and  $:num\_recalled = 0$ .

| Agent              | Memory representation    | Annotations | QA rule         | Eviction     |
|--------------------|--------------------------|-------------|-----------------|--------------|
| Symbolic-KG        | RDF triples              | No          | SPARQL          | Random       |
| Symbolic-TKG       | Annotated RDF statements | Yes         | MRA/MRU/MFU     | FIFO/LRU/LFU |
| Neural-LSTM        | Observation history      | No          | Prediction head | FIFO         |
| Neural-Transformer | Observation history      | No          | Prediction head | FIFO         |

**Table 2.** [Revised] Agent comparison. MRA = most recently added, MRU = most recently used, MFU = most frequently used.

- **Repeated observation:** re-observing the same  $(s, r, o)$  at a later step creates a *new* statement instance with fresh annotations; existing instances are never overwritten. Multiple annotated occurrences of the same base triple therefore coexist (the counts in Figure 5 show this).
- **Short-term precedence:** if the queried object is in the agent’s current room, the answer is read from the current observation and returned directly. The QA policy is not consulted and no annotation is updated. This is why an agent with no long-term memory at all ( $K=0$ ) still answers a few queries correctly, which is what the observation-only bound in Table 3 measures.
- **Query-time update:** otherwise, when the QA policy selects a stored statement to answer a query, that statement’s `:last_accessed` is set to the current step and its `:num_recalled` is incremented; non-selected statements are untouched.
- **Traversal update:** the same refresh is applied to each edge statement the exploration step relies on, so access counts reflect use for navigation as well as for answering. This is the only access that occurs without a query.
- **Contradictions:** conflicting facts, such as two `:at_location` statements placing the same object in different rooms, are *not* resolved at insertion time. They coexist in memory, and the QA policy resolves them at query time (e.g., MRA answers with the most recently added occurrence). Stale facts are thus superseded functionally rather than deleted.
- **Eviction and deletion:** when  $|M_t^{\text{long}}| > K$ , the eviction policy removes the argmin statement (below); this is the only deletion mechanism.

*QA rules.* Each query  $(s_q, r_q, ?)$  is answered by selecting the object  $o$  whose annotation value is maximal:

MRA : `arg max :time_added,`  
 MRU : `arg max :last_accessed,`  
 MFU : `arg max :num_recalled.`

A concrete SPARQL 1.2 realization of the MRU rule is:

```
SELECT ?o WHERE {
  << <s> <r> ?o >> :last_accessed ?t .
}
ORDER BY DESC(?t)
LIMIT 1
```

*Eviction.* Using annotations permits structured removal: . Each rule is a cascade over the annotations, least first, and

falls through to the next when the current one ties:

FIFO : `:time_added,`

LRU : `:last_accessed → :time_added,`

LFU : `:num_recalled → :last_accessed → :time_added.`

Any statements still tied after the cascade are separated uniformly at random. The cascade matters in practice: most stored statements are never recalled, so LFU without it would degenerate to a random choice over that majority. The QA and exploration policies break ties the same way, MRU and MFU falling through to MRA.

*Exploration.* The agent ranks frontier rooms using MRA/MRU/MFU on edge triples reconstructs the map from its stored edge statements and performs BFS to the top-ranked frontier least-visited room. The exploration policy does not pick the target: it ranks competing statements about the same (room, direction) pair by MRA, MRU or MFU, exactly as the QA policy ranks answer candidates. It shapes the *map* searched, not the frontier sought.

Thus, the TKG agent is entirely deterministic and fully traceable: every every TKG decision (answer, movement, eviction) is reproducible from a traceable function of  $M_t$ : the selected statement is always the extremum of an explicit annotation value, and the selecting rule is inspectable. The single residual source of randomness is tie-breaking: when several statements share the extremal annotation value, or when the BFS frontier is empty, one candidate is drawn uniformly at random. Ties are the only reason the symbolic rows in Table 3 carry a non-zero across-seed standard deviation even though the environment itself is deterministic. Fixing the seed makes a TKG run reproducible; the KG agent additionally depends on the iteration order of its triple store, as documented with the released results.

## 4.2. Neural agents

The neural agents store a fixed-length *tokenized observation history* without explicit graph structure. They differ only in the sequence encoder used.

*Joint decision structure.* Unlike the symbolic agents, which decompose question answering and exploration into two independent rules, the neural agents produce a *single joint operation* that simultaneously selects the answer and the movement. This results in a large option set (49 possible room answers  $\times$  5 movement choices) from which the prediction head chooses one element. Because answers referring to rooms that have not yet been observed are invalid, the option set is dynamically *masked* at each timestep so that only admissible answers remain. Consequently, the neural agents must learn end-to-end how to resolve queries and how to navigate, without access to explicit graph structure, annotations, or symbolic update rules. The neural

agents learn this joint policy using a standard reinforcement-learning objective implemented via a Deep Q-Network (DQN) (Mnih et al. 2015).

*Tokenization.* Each triple  $(s, p, o)$  is converted to an embedding vector

$$\mathbf{e}_i = \mathbf{W}\mathbf{W}_{\text{emb}} [\text{emb}(s) \parallel \text{emb}(p) \parallel \text{emb}(o)],$$

and the last  $L$  such tokens form the memory buffer. When full, the oldest token is removed (FIFO), reflecting the queue structure of a fixed-length sequence buffer.

### LSTM neural agent

The sequence  $\mathbf{E} = [\mathbf{e}_1, \dots, \mathbf{e}_L]$  is encoded by:

$$\mathbf{H} = \text{LSTM}(\mathbf{E}).$$

Because LSTMs process data sequentially, temporal order is included implicitly (Hochreiter and Schmidhuber 1997).

*Question-conditioned pooling.* Given a question triple  $(s_q, r_q, ?)$ , embed it as  $\mathbf{q}$  and compute attention weights:

$$\alpha_i = \frac{\exp(\mathbf{q}^\top \mathbf{h}_i)}{\sum_j \exp(\mathbf{q}^\top \mathbf{h}_j)}, \quad \mathbf{c} = \sum_i \alpha_i \mathbf{h}_i.$$

A feedforward prediction head maps  $\mathbf{c}$  to the 245 possible operation options (answer  $\times$  movement).

### Transformer neural agent

Here,  $\mathbf{E}$  is fed to a Transformer encoder:

$$\mathbf{H} = \text{Transformer}(\mathbf{E} + \text{PE}),$$

where PE are sinusoidal positional encodings that supply the temporal order (Vaswani et al. 2017), enabling long-range temporal reasoning. The same question-conditioned pooling and prediction head are used as in the LSTM agent.

*Interpretability.* Neural agents have no explicit triples, annotations, or traceable update rules, similar to as in episodic-control agents (Pritzel et al. 2017; Blundell et al. 2016). Their internal state is an opaque vector, and ; only soft attention weights give partial insight into indicate which past observations were influentialmattered. Symbolic agents , in contrast, expose every stored fact and every eviction decision.

## 5. Experiments

We evaluate all agents in the deterministic RoomKG Benchmark under varying long-term memory capacities, from 0 to 512. Each configuration is trained and tested on 5 random seeds, and we report the mean and standard deviation. All raw results, hardware specifications, Each neural architecture is trained in two sizes (embedding dimension 16 with 2 layers, and hyperparameters are provided in the public benchmark 32 with 4 layers; the Transformer uses 2 and 4 attention heads, respectively), which is 11,573 and 51,061 parameters for the LSTM and 13,781 and 68,085 for the Transformer, with a DQN objective for 200 episodes per configuration; raw results for the neural and TKG runs are included in the released artifacts. In particular, a long-term

memory capacity of 0 denotes the no-memory setting: agents do not retain any long-term memory across timesteps.

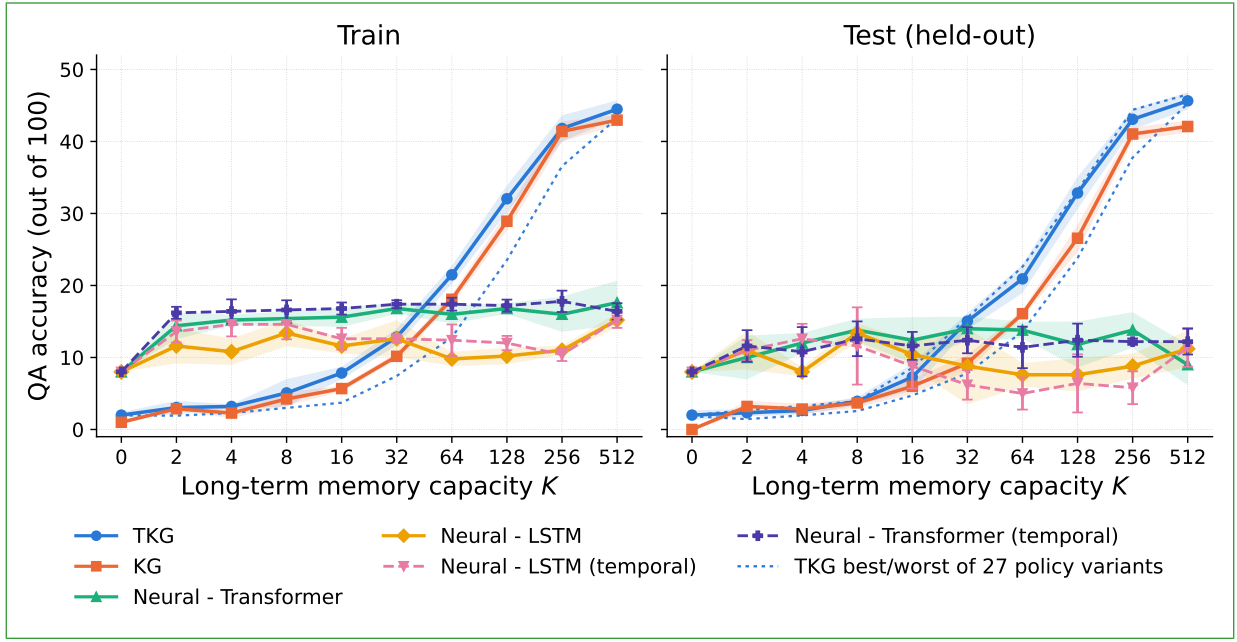
The primary task is object-location question answering under partial observability. We report two quantitative views of performance: QA accuracy across memory capacities and coverage over time, measured in terms of visited rooms and stored triples. In addition, we provide a qualitative visualization of memory-state evolution for the TKG agent to illustrate how temporal symbolic memory develops over an episodeAll experiments run on commodity central processing unit (CPU) hardware. Training a single neural configuration takes on the order of one to ten CPU-hours depending on architecture size and memory capacity, while the symbolic agents require no training and evaluate in minutes.

### 5.1. QA Accuracy Across Long-Term Memory Capacities

Figure 3 shows the train and test QA accuracy for all agents. Neural agents learn exploration and question answering jointly under a single end-to-end policy, yielding a large discrete action space and difficult credit assignment. As memory capacity increases, these agents also receive longer observation histories, making the underlying learning problem more computationally demanding; in principle, higher-capacity models may require substantially more training to benefit from the additional information. In our experiments, however, all memory-capacity settings were trained for the same number of episodes (200) to ensure a fair comparison, which likely contributes to the minimal performance gains observed in the higher-capacity regime.

Across almost all capacities, the Transformer agent slightly outperforms the LSTM agent, consistent with empirical trends in sequence modeling. However, neither architecture meaningfully exploits larger memory We do not claim that either the LSTM or the Transformer is the better architecture: which of them leads varies with capacity and environment, and no result here depends on it. What separates them from the symbolic agents is common to both: their accuracy is nearly flat in capacity, so the symbolic agents overtake them once memory begins to matter, and both exhibit substantial train-test gapslose about a fifth to a quarter of their accuracy from train to test. The symbolic agents show no such degradation (the TKG agent scores 44.48 on train and 45.64 on test at the same capacity), indicating that the generalization gap exposed by the permuted query order is specific to the learned policies, not to the benchmark difficulty. In low-memory regimes (0–16), neural agents outperform symbolic ones—but this reflects random trial-and-error ones, but this reflects reward-driven guessing rather than informed reasoning. At capacity 0, where agents have no long-term memory at all, performance is predictably poor across the board: the agents often guess object locations without provenance and only gradually reinforce correct answers through reward.

Symbolic agents begin to outperform neural agents at capacity 32. Their The neural baselines are already temporally aware by construction: the LSTM encodes order through recurrence, the Transformer through sinusoidal positional encodings. To test whether the symbolic advantage rests on something further, we add controls whose tokens also carry an explicit arrival timestep, the sequence analogue of :time\_added. Each architecture is trained at two sizes, with and without



**Figure 3.** Train–test QA accuracy for all agents across long-term memory capacities. Mean on the training environment (left) and standard deviation are computed the held-out test environment (right). Lines show means across 5 seeds, with  $\pm 1$  standard deviation shown as shading for the four main series and as capped error bars for the two temporal ones. Raw values can be found in Series marked “temporal” carry an explicit arrival-timestep feature. The dotted blue lines bound the public benchmark artifacts best and worst of all 27 TKG policy variants, computed per capacity. On the training panel the TKG line coincides with the upper bound by construction, since the reported variant is selected by training accuracy; on the held-out test panel it falls away from that bound wherever the variants differ, showing that the selection never uses test performance.

the feature, under the identical protocol; the temporal variant adds only the timestep embedding (288 or 1,088 parameters, at most 2.5% of the base model), so each pair is size-matched.

Across all eight configurations, no neural agent reaches more than 18.6 on train or 14.2 on test at any capacity. Capacity does little for them: the strongest configuration gains about three points from  $K=2$  to  $K=512$  on train, and none gains more than 0.6 points on the held-out environment. Arrival-time features move configurations within this band without lifting any out of it, and their effect is not reliably positive: averaged over capacities it gains 1.2 points (LSTM) and 0.8 (Transformer) on train, and loses 0.8 and 0.3 on the held-out environment, every difference inside the across-seed spread. We therefore report the neural family by this bound rather than agent by agent, and we read the result as showing that access to temporal information is not what separates the two families: what matters is the ability to query and maintain those annotations.

Of the three annotations, only arrival time transfers to a sequence model as an input feature: `:last_accessed` and `:num_recalled` are functions of the agent’s own retrieval history rather than properties of the world, so supplying them would mean giving the neural agent the store the comparison is meant to test.

Both symbolic agents eventually exceed that bound: the TKG agent from  $K=64$  on train and  $K=32$  on test, the KG agent from  $K=128$  and  $K=64$ . Symbolic accuracy increases with memory because operations such as BFS and graph queries add computation but not statistical difficulty. Neural agents, by contrast, , whereas neural agents must learn these structures

from data and therefore require much need far more training and capacity to benefit from larger memory a larger store.

The TKG agent achieves better QA accuracy in most memory capacities leads every other agent from  $K=64$  upward on train and from  $K=32$  upward on test. The TKG agent family consists of 27 variants, arising from all combinations of three annotation-based QA rules (MRA, MRU, MFU), three exploration priorities based on the same annotation properties, and three eviction heuristics (FIFO, LRU, LFU); these choices are independent because each stored fact carries the annotation properties `:time_added`, `:last_accessed`, and `:num_recalled`. We report the best-performing variant, selected by training variant selected by training accuracy and evaluated identically once at test time. All 27 train/test combinations for every memory capacity are provided in the public benchmark artifacts, i.e., standard model selection without access to test performance. The neural side is held to the same discipline: each run keeps its best-validation checkpoint, and the family bound above is the maximum over all eight configurations, including those Figure 3 does not plot. The symbolic agents are thus represented by one train-selected variant and the neural agents by their strongest. At capacity 512, the training-selected the selected variant uses MRU for QA, and MRU for exploration, and LFU for eviction; the eviction rule is immaterial there, since MRU/MRU with FIFO, LRU or LFU all reach 44.48 on train, and we break the tie in favour of LFU. This suggests that recency is the most useful signal both for answer selection and for frontier prioritization, while LFU eviction helps preserve repeatedly useful facts under a fixed memory budget whereas the eviction rule separates variants only under memory scarcity, consistent

with “Calibration Bounds”, where eviction is shown not to bind at high capacity.

We also report the spread over all 27 TKG variants. On test, the (worst / median / best) scores are (7.80 / 9.68 / 15.52) at  $K=32$ , (23.80 / 30.56 / 33.20) at  $K=128$ , and (45.28 / 45.72 / 46.52) at  $K=512$ . Policy choice matters most under memory scarcity; at high capacity all 27 variants converge to within 1.3 points, so the reported result is robust to the selection. The policy configuration is a hyperparameter, and like any hyperparameter it is tuned per operating point, so the selected variant differs across capacities. What varies is instructive: LFU is the eviction rule of the selected variant at eight of the ten capacities, winning outright at six, sharing the top score with LRU at  $K=0$  and tying with FIFO and LRU at  $K=512$ . The exceptions are  $K=4$ , where FIFO is selected, and  $K=256$ , where LRU is. Eviction is therefore the most stable of the three dimensions without being fixed, whereas the QA and exploration rules change from capacity to capacity without a simple pattern. Selecting on training accuracy is also conservative: the best variant at test time reaches the 46.52 above, higher than the 45.64 we report.

This setup favors the plain KG agent. Long-term memory capacity is counted in stored entries, so : as “Capacity semantics” notes, at the same nominal capacity the KG agent can retain more main triples it retains more distinct facts than the TKG agent. By contrast, the TKG agent uses each entry for an annotated triple, combining the base fact with temporal metadata, which spends each entry on one annotated occurrence. Even with this storage disadvantage, that storage disadvantage the TKG agent performs better scores higher at every capacity from  $K=32$  upward, on train and on test, indicating that temporal annotations make the memory state more informative and expressive than a larger store of unannotated triples.

At capacity 512, this MRU/MRU /LFU TKG variant reaches a QA accuracy of 44.48 on train and 45.64 on test, whereas : more than twice the best neural agent (LSTM) reaches 11.2, a four-fold difference on test. These results highlight that explicit symbolic memory is considerably more effective for long-horizon, partially observable semantic reasoning. Statistical variation is shown as shaded regions score on train (18.6) and more than three times it on test (14.2).

## 5.2. Calibration Bounds

To make the absolute QA scores interpretable, we anchor them between the reference bounds in Table 3. The upper anchor needs no experiment, so we give it here in the text: an agent with direct access to  $s_t$  answers all 100 queries correctly by construction, since correctness is evaluated against that same state (see “Deterministic Question–Move Loop”). The symbolic agents are reported at the capacity where retention saturates. An episode emits only  $\sim 500$ – $600$  observation triples, and doubling the budget to  $K=1024$  leaves the reported TKG variant unchanged and moves no variant of the 27 by more than 0.16 points; the KG agent never evicts at  $K=512$  in the first place, since its store holds roughly 300 distinct entries (see “Capacity semantics”). Eviction is therefore not the binding constraint at high capacity. The observation-only baseline ( $K=0$ ) bounds the problem from below: with no long-term memory an agent can answer only when the queried object happens to be in

| Reference                      | Train QA     | Test QA      |
|--------------------------------|--------------|--------------|
| Hidden-state oracle (analytic) | 100          | 100          |
| TKG at $K=512$                 | 44.48 (1.23) | 45.64 (1.21) |
| KG at $K=512$                  | 42.96 (0.70) | 42.08 (0.85) |
| Best neural                    | 18.60 (2.87) | 14.20 (2.14) |
| Observation-only ( $K=0$ )     | $\leq 2$     | $\leq 2$     |

**Table 3.** [Added in this revision] Calibration bounds and reference scores (mean, standard deviation in parentheses), ordered from the upper bound down to the lower. Symbolic rows follow the selection protocol described above and are unchanged at  $K=1024$ ; the neural row is the maximum over all eight trained configurations and all capacities. The  $K=0$  row is the lower bound, taken over both symbolic agents, neither of which retains anything across timesteps there. The oracle needs no experiment: an agent reading  $s_t$  answers every query correctly by construction.

the room it currently occupies, which holds for at most 2 of the 100 queries.

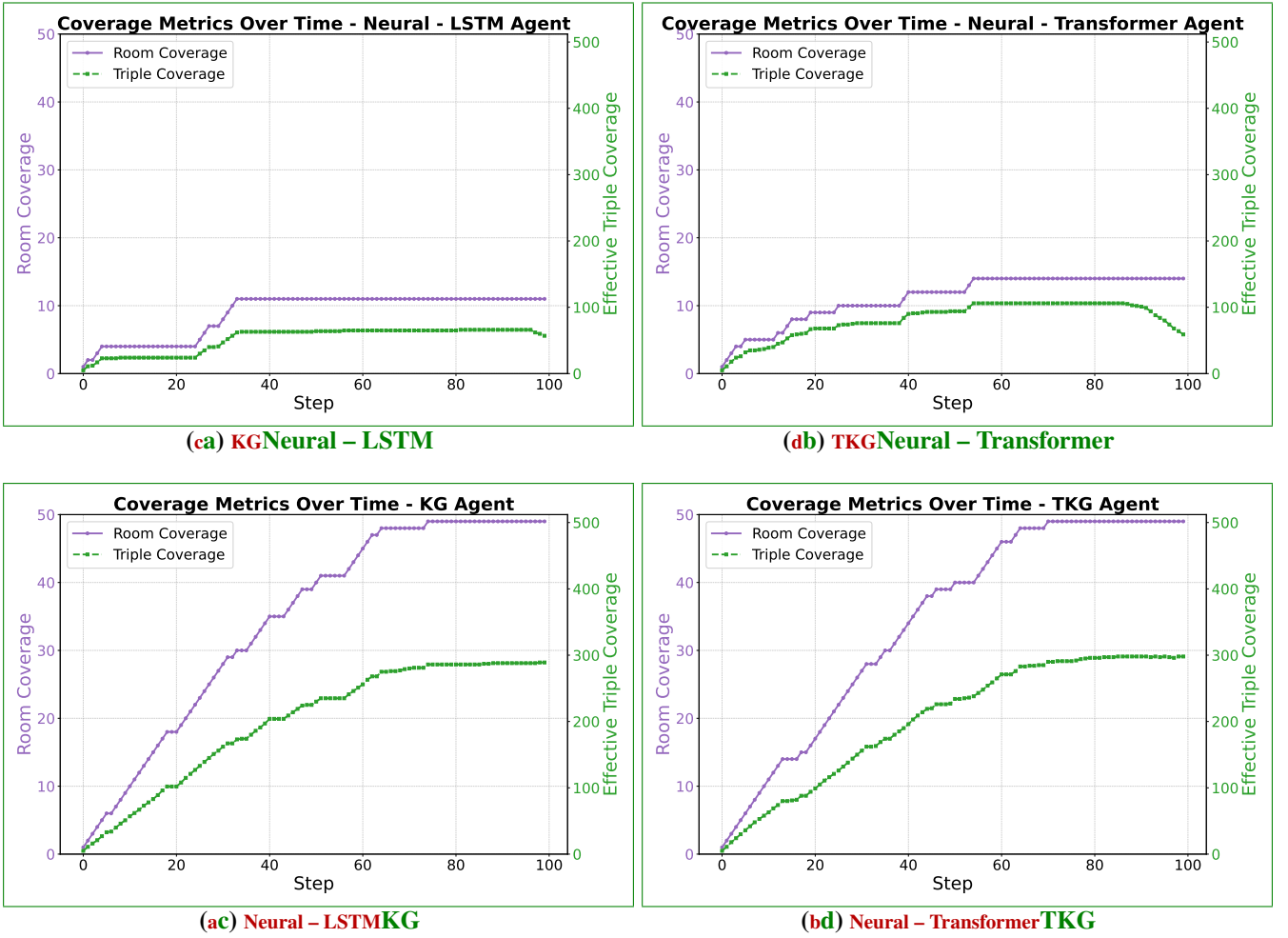
The gap between the best saturated symbolic score ( $\approx 46$ ) and the analytic upper bound (100) quantifies the irreducible cost of partial observability here: even an agent that retains every observation must revisit rooms to refresh facts about moving objects, and those facts go stale between visits. The gap between  $K=0$  (at most 2) and  $K=512$  isolates the contribution of long-term memory itself. The regime in between, where capacity is scarce and management policies decide which facts survive, is what this benchmark targets.

## 5.3. Coverage Metrics: Room and Triple Coverage

Figure 4 shows, for the long-term memory capacity of at capacity 512, the coverage metrics of the four agents: KG, TKG, LSTM, and Transformer. Each subplot includes the cumulative number of unique rooms visited and the number of unique  $(s, p, o)$  triples stored in memory at each timestep. Raw values are provided in the public benchmark artifacts. by each agent at every timestep.

**Symbolic agents.** **Symbolic agents.** Both symbolic agents achieve full coverage of all 49 rooms in the benchmark. Moreover, the TKG agent reaches full room coverage slightly earlier (timestep 70) than the KG agent (timestep 74). The temporal annotations in the TKG memory allow it to track time-varying wall configurations more accurately, yielding , which is consistent with its temporal annotations supporting a more up-to-date inferred map of the benchmark environment and enabling more efficient exploration time-varying wall configurations. We do not read that gap as direct evidence of memory quality, since the two agents also differ in how their searches are implemented. Coverage is not independent of memory either: the set of visited rooms a search consults is read from long-term memory, so an agent with no long-term memory cannot tell where it has already been. Triple-coverage results follow the same pattern: symbolic agents accumulate nearly complete internal maps because they continue to explore until all reachable structure has been observed.

**Neural agents.** **Neural agents.** In contrast, the LSTM and Transformer agents stop increasing room coverage long before the end of the episode (around step 50 for the Transformer and step 30 for the LSTM). Note that this is not stochasticity: the environment is deterministic, and at test



**Figure 4.** Coverage metrics for the four agents, at the long-term memory capacity of 512. Neural agents halt exploration early, while symbolic agents visit all 49 rooms and accumulate nearly all unique triples. *Raw values can be found in the public benchmark artifacts.*

time the neural agents are deterministic too (greedy argmax over Q-values, no  $\epsilon$ -exploration). What differs is the *policy itself*. By design, we do not equip the neural agents with symbolic search: the symbolic agents' BFS exploration is a hand-designed procedure, whereas the neural agents must learn any exploration behavior end-to-end; how well agents cope without built-in symbolic machinery is part of what the benchmark measures. A plausible explanation is that neural policies *must* couple exploration and answering within a single high-cardinality action space. Once training converges to a locally rewarding but suboptimal behavior, such as repeatedly answering queries with moderate reward or oscillating among familiar states, the agents no longer discover new rooms. Because the reward signal, and that a reward signal which is sparse and not explicitly tied to exploration, the policies collapse into low-entropy, lets training settle into locally rewarding but non-exploratory behaviors. These plateauing exploration patterns also explain behavior. The same plateau explains the limited triple coverage: neural agents simply these agents encounter fewer states from which to build internal structure.

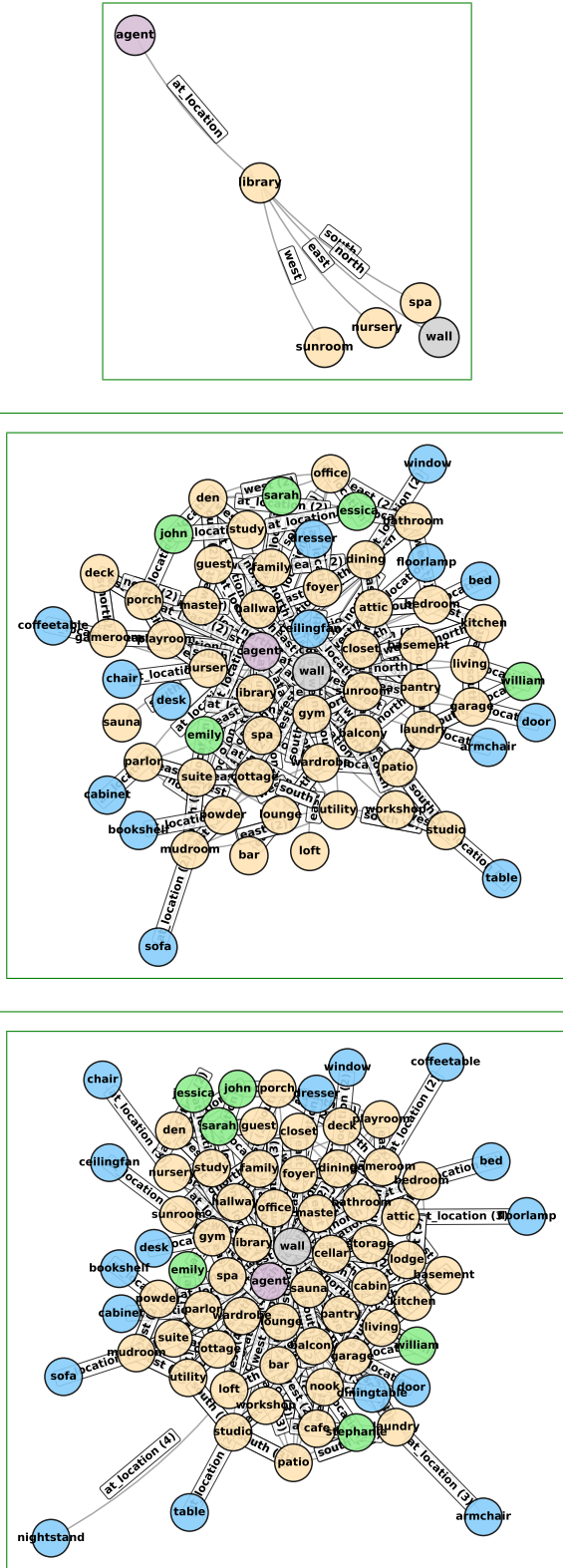
#### 5.4. Memory-State Evolution (TKG)

To illustrate the expressive advantage of temporal symbolic memory, Figure 5 shows the TKG agent's long-term memory at timesteps  $t = 0$ ,  $t = 50$ , and  $t = 99$ , using capacity 512. Each edge label includes the number of associated memories

when more than one annotated triple exists. Node colors correspond to semantic categories (rooms, static objects, moving objects, walls, and agent).

The graphs show the internal map becoming increasingly complete over time. Static objects and room nodes accumulate in a way that reflects what the agent has actually observed by each timestep. At  $t = 0$ , the visualization contains only the current short-term observation: four room nodes (yellow), one wall node (grey), and one agent node (purple), with no stored object nodes yet. By  $t = 50$ , after both short-term and long-term memories have accumulated, the graph contains 43 room nodes, 14 static object nodes (blue), 5 moving object nodes (green), together with the wall and agent nodes. By  $t = 99$ , the graph contains all 49 room nodes, 16 static object nodes, and 6 moving object nodes, again together with the wall and agent nodes. This final snapshot shows that the agent has visited all rooms and has retained all room nodes in memory, while still The agent has thus visited every room and retained every room node, while storing fewer moving than static objects because moving objects, which are harder to observe consistently over time.

Full memory evolution images for all agents (from  $t = 0$  to  $t = 99$ ) are available in the public benchmark. The full series, one frame per timestep, can be found in the released artifacts.



**Figure 5.** Memory state of the TKG agent at (capacity 512); from top to bottom:  $t = 0$ ,  $t = 50$ , and  $t = 99$  (capacity 512). Counts in parentheses (e.g., “at location (3)”) indicate how many TKG memories share the same main underlying triple but differ in their temporal annotation values.

## 6. Related Work

*Temporal and hyper-relational knowledge graphs.* The Semantic Web stack provides a mature basis for representing structured data as RDF graphs and querying them with

SPARQL (Seaborne and Harris 2013). RDF 1.2 and SPARQL 1.2 extend this model with triple terms, reifiers, and annotation syntax for *statements about statements* (Kellogg et al. 2026; Kellogg and Tomaszuk 2026; Hartig et al. 2026). Temporal and hyper-relational knowledge graphs build on these ideas by attaching time and other annotations to edges and have been extensively studied for tasks such as link prediction and event forecasting (e.g., (García-Durán et al. 2018)). In contrast, we use a temporal KG not primarily as a prediction target but as the *internal state* of an agent: every observation becomes an RDF 1.2-annotated statement with temporal annotations, and downstream decisions (a temporally annotated occurrence, and answering and exploration ) are implemented as are queries and updates over this KG.

*Memory under partial observability.* In partially observable environments, agents must aggregate information over time. A large body of work uses neural memories to maintain an internal state proxy, including recurrent architectures, memory-augmented networks such as Neural Turing Machines and Differentiable Neural Computers (Graves et al. 2014, 2016), and long-context models with compression or retrieval (e.g., MERLIN and related world-model agents (Wayne et al. 2018; Ha and Schmidhuber 2018)). These approaches typically store history in dense vectors or key-value tensors, which makes it difficult to inspect what exactly is remembered at a given time, a transparency gap long noted in neural-symbolic systems (Besold et al. 2017). Our work instead treats the agent’s long-term memory as an explicit temporal knowledge graph in RDF 1.2 annotation syntax: which each stored fact is a verifiable triple with annotations, and carrying its own annotations, so the effect of capacity limits or update rules on question answering can be inspected directly at the graph level. Alongside this, we include neural baselines that operate on observation histories without explicit KG structure, highlighting the behavioural differences between which lets us compare how latent and symbolic memories behave (Kimura et al. 2021). What is missing from this literature is a benchmark expressly constructed to make long-term memory failures visible, measurable, and comparable across symbolic and neural representations. This same issue appears when one looks at interactive environments and KG-based agents.

*Graph-structured environments and KG-based agents.* Several interactive environments expose graph-structured or relational state (Chaplot et al. 2020), and neural architectures explicitly designed for graph-structured reasoning (Yun et al. 2019). TextWorld and Jericho provide text-based games where an evolving symbolic world model can be extracted and used for control, leading to agents that query or learn over dynamic knowledge graphs (Côté et al. 2018; Hausknecht et al. 2020; Ammanabrolu and Hausknecht 2020; Oh et al. 2017). Agents with multiple KG-based memory stores have also been explored (Kim et al. 2023). Other systems use KGs as world models for planning, question answering, or recommendation (Hogan et al. 2021; Ehrlinger and Wöß 2016; Bordes et al. 2015; Yih et al. 2015). More recent benchmark-oriented work in neuro-symbolic AI has argued for explicit task definitions, benchmark metadata, and diverse evaluation views, as illustrated by benchmarking surveys and suites

such as those of Manhaeve et al. (Manhaeve et al. 2025), Bortolotti et al. (Bortolotti et al. 2024), and Hu et al. (Hu et al. 2025). Compared to these settings, our focus is deliberately narrow: we design the Room KG Benchmark where (i) the hidden state and observations are *already* KGs, (ii) the question format is fixed, and (iii) the benchmark parameters (number of rooms, walls, objects, episode length) are easily adjusted. This allows us to isolate the role of temporal KG memory and to compare symbolic and neural baselines under controlled conditions rather than to optimize a full task pipeline. Existing environments and benchmark suites are valuable for their own aims, but they are not sufficient when the goal is to evaluate whether an agent can build, maintain, and use explicit long-term memory about a changing symbolic world.

*Semantic Web, agents, and explainability.* Within the Semantic Web community, KGs are increasingly used to support decision-making and explainable reasoning in multi-step processes, e.g., planning, data analytics, and interactive decision support. Traceability and provenance are central: KG representations allow one to link conclusions back to the underlying facts (Anyanwu and Sheth 2003). Our design follows this line by making the agent’s long-term memory itself a temporal KG; question answering is implemented as SPARQL 1.2 a declarative retrieval over that memory, and expressible as a SPARQL 1.2 triple-term query over the RDF 1.2 form of the snapshots (see “Reification and RDF 1.2”), while the agents evaluate the equivalent selection over the reified store; memory contents can be visualized as graphs over time. This provides gives a compact testbed for studying how temporal annotations (e.g., recency and recall counts) interact with partial observability and capacity limits, and how symbolic KG memories compare to neural baselines in terms of generalization from training memories generalize to held-out question orders compared with neural baselines.

## 7. Limitations

RoomKG Benchmark is intentionally narrow. First, its hidden dynamics are deterministic, procedurally specified, and fully symbolic, which makes the benchmark well suited for suits controlled studies of memory but is less representative of noisy embodied settings. Second, the current benchmark task is restricted to object-location question answering; it does not yet cover , not richer query families such as temporal comparison, counterfactual reasoning, or multi-hop compositional questions. Third, our released benchmark views are derived from synthetic layouts rather than from the layouts are synthetic rather than human-authored or naturally occurring datasets, so benchmark , so diversity is controlled by parameterization rather than by broad real-world variation. Finally, although we include both symbolic and neural baselines, future work can evaluate a wider range of agents, training regimes, and evaluation protocols than those studied here.

## 8. Conclusion

We presented the RoomKG Benchmark in which both . Both the hidden state and the agent’s memory are represented as knowledge graphs, enabling a controlled study of temporal memory under partial observability. Using a lightweight RDF 1.2-based memory with temporal annotations, symbolic agents achieved full room coverage and substantially higher QA accuracy than neural sequence baselines

readable at any timestep. We also release baselines for both families: symbolic agents over a plain KG and over a temporally annotated TKG, and neural sequence models reading the same observations under the same benchmark and query conditions. The results highlight the effectiveness and interpretability of temporal knowledge graphs as agent memory , and show why a dedicated benchmark is needed: without a setting that makes long-term memory explicit, structured, and inspectable, it is difficult to distinguish whether an agent succeeds because it remembers well or merely because the task under-tests memory. RoomKG Benchmark provides a compact testbed for future work on semantic representations, temporal update rules, and neuro-symbolic memory models, and is intended to serve as a reference point for evaluating long-term memory in neurosymbolic agents. protocol. Both symbolic agents cover every room. Both far outscore the neural baselines on question answering. The annotated memory is the stronger of the two, ahead of the plain KG at every capacity from  $K=32$  upward, on train and on test. Annotations, not a bigger store, are what make the difference.

## References

- Ammanabrolu P and Hausknecht M (2020) Graph constrained reinforcement learning for natural language action spaces. In: *International Conference on Learning Representations*. URL <https://openreview.net/forum?id=B1x6w0EtW.H>.
- Anyanwu K and Sheth A (2003)  $\rho$ -queries: Enabling Querying and Ranking Complex Relationships. In: *Proceedings of the 12th International World Wide Web Conference (WWW '03)*, WWW '03. New York, NY, USA: Association for Computing Machinery. ISBN 1581136803, pp. 690–699. DOI:10.1145/775152.775249. URL <https://doi.org/10.1145/775152.775249>.
- Battaglia PW, Hamrick JB, Bapst V, Sanchez-Gonzalez A, Zambaldi V, Malinowski M, Tacchetti A, Raposo D, Santoro A, Faulkner R, Gulcehre C, Song F, Ballard A, Gilmer J, Dahl G, Vaswani A, Allen K, Nash C, Langston V, Dyer C, Heess N, Wierstra D, Kohli P, Botvinick M, Vinyals O, Li Y and Pascanu R (2018) Relational inductive biases, deep learning, and graph networks. URL <https://arxiv.org/abs/1806.01261>.
- Besold TR, d'Avila Garcez A, Bader S, Bowman H, Domingos P, Hitzler P, Kuehnberger KU, Lamb LC, Lowd D, Lima PMV, de Penning L, Pinkas G, Poon H and Zaverucha G (2017) Neural-symbolic learning and reasoning: A survey and interpretation. URL <https://arxiv.org/abs/1711.03902>.
- Blundell C, Uria B, Pritzel A, Li Y, Ruderman A, Leibo JZ, Rae J, Wierstra D and Hassabis D (2016) Model-free episodic control. URL <https://arxiv.org/abs/1606.04460>.
- Bordes A, Usunier N, Chopra S and Weston J (2015) Large-scale simple question answering with memory networks. URL <https://arxiv.org/abs/1506.02075>.
- Bortolotti S, Marconato E, Carraro T, Morettin P, van Krieken E, Vergari A, Teso S and Passerini A (2024) A neuro-symbolic benchmark suite for concept quality and reasoning shortcuts. In: *Proceedings of the 38th International Conference on Neural Information Processing Systems*. pp. 115861–115905. URL <https://dl.acm.org/doi/10.5555/3737916.3741595>.
- Carroll JJ, Bizer C, Hayes P and Stickler P (2005) Named graphs, provenance and trust. In: *Proceedings of the 14th International Conference on World Wide Web (WWW)*. ACM, pp. 613–622. DOI:10.1145/1060745.1060835.
- Chaplot DS, Salakhutdinov R, Gupta A and Gupta S (2020) Neural topological SLAM for visual navigation. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. pp. 12875–12884. URL [https://openaccess.thecvf.com/content\\_CVPR\\_2020/html/Chaplot\\_Neural\\_Topological\\_SLAM\\_for\\_Visual\\_Navigation\\_CVPR\\_2020\\_paper.html](https://openaccess.thecvf.com/content_CVPR_2020/html/Chaplot_Neural_Topological_SLAM_for_Visual_Navigation_CVPR_2020_paper.html).
- Cox S and Little C (2017) Time ontology in OWL. W3C Recommendation. URL <https://www.w3.org/TR/owl-time/>.
- Côté MA, Ákos Kádár, Yuan X, Kybartas B, Barnes T, Fine E, Moore J, Tao RY, Hausknecht M, Asri LE, Adada M, Tay W and Trischler A (2018) Textworld: A learning environment for text-based games. In: *Proceedings of the Workshop on Computer Games*. pp. 41–75. URL <https://arxiv.org/abs/1806.11532>. Presented at CGW@IJCAI 2018, Stockholm.
- Dell'Aglío D, Della Valle E, van Harmelen F and Bernstein A (2017) Stream reasoning: A survey and outlook—a summary of ten years of research and a vision for the next decade. *Data Science* 1(1-2): 59–83. DOI:10.3233/DS-170006.
- Ehrlinger L and Wöß W (2016) Towards a definition of knowledge graphs.
- García-Durán A, Dumančić S and Niepert M (2018) Learning sequence encoders for temporal knowledge graph completion. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, pp. 4816–4821. DOI: 10.18653/v1/D18-1516. URL <https://aclanthology.org/D18-1516/>.
- Graves A, Wayne G and Danihelka I (2014) Neural turing machines. Technical report, arXiv preprint arXiv:1410.5401.
- Graves A, Wayne G, Reynolds M, Harley T, Danihelka I, Grabska-Barwińska A, Colmenarejo SG, Grefenstette E, Ramalho T, Dyer C et al. (2016) Hybrid computing using a neural network with dynamic external memory. *Nature* 538: 471–476.
- Gutierrez C, Hurtado CA and Vaisman AA (2005) Temporal RDF. In: *The Semantic Web: Research and Applications, Lecture Notes in Computer Science*, volume 3532. Springer, pp. 93–107.
- Ha D and Schmidhuber J (2018) World models. *CoRR* abs/1803.10122. URL <http://arxiv.org/abs/1803.10122>.
- Hartig O, Seaborne A, Taelman R, Williams G and Tanon TP (2026) SPARQL 1.2 query language. W3C working draft, W3C. <https://www.w3.org/TR/2026/WD-sparql12-query-20260323/>.
- Hausknecht M, Ammanabrolu P, Côté MA and Yuan X (2020) Interactive fiction games: A colossal adventure. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34. pp. 7903–7910. DOI:10.1609/aaai.v34i05.6297. URL <https://doi.org/10.1609/aaai.v34i05.6297>.
- Hochreiter S and Schmidhuber J (1997) Long short-term memory. *Neural Computation* 9(8): 1735–1780. DOI:10.1162/neco.1997.9.8.1735.
- Hogan A, Blomqvist E, Cochez M, D'amato C, Melo GD, Gutierrez C, Kirrane S, Gayo JEL, Navigli R, Neumaier S, Ngomo ACN, Polleres A, Rashid SM, Rula A, Schmelzeisen L, Sequeda J, Staab S and Zimmermann A (2021) Knowledge graphs. *ACM Computing Surveys* 54(4): 1–37. DOI:10.1145/3447772. URL <http://dx.doi.org/10.1145/3447772>.
- Hu M, Chen T, Zou Y, Lei Y, Chen Q, Li M, Mu Y, Zhang H, Shao W and Luo P (2025) Text2world: Benchmarking large language models for symbolic world model generation. In: *Findings of the Association for Computational Linguistics: ACL 2025*. Vienna, Austria: Association for Computational Linguistics, pp. 26043–26066. DOI:10.18653/v1/2025.findings-acl.1337. URL <https://aclanthology.org/2025.findings-acl.1337/>.
- Kaelbling LP, Littman ML and Cassandra AR (1998) Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101(1): 99–134. DOI: [https://doi.org/10.1016/S0004-3702\(98\)00023-X](https://doi.org/10.1016/S0004-3702(98)00023-X). URL <https://www.sciencedirect.com/science/article/pii/S000437029800023X>.

- Kellogg G, Hartig O, Champin PA and Seaborne A (2026) RDF 1.2 concepts and abstract data model. W3C candidate recommendation snapshot, W3C. <https://www.w3.org/TR/rdf12-concepts/>.
- Kellogg G and Tomaszuk D (2026) RDF 1.2 turtle. W3C working draft, W3C. <https://www.w3.org/TR/2026/WD-rdf12-turtle-20260407/>.
- Kim T, Cochez M, François-Lavet V, Neerinx M and Vossen P (2023) A machine with short-term, episodic, and semantic memory systems. In: *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23. AAAI Press. ISBN 978-1-57735-880-0. DOI:10.1609/aaai.v37i1.25075. URL <https://doi.org/10.1609/aaai.v37i1.25075>.
- Kimura D, Ono M, Chaudhury S, Kohita R, Wachi A, Agravante DJ, Tatsubori M, Munawar A and Gray A (2021) Neuro-symbolic reinforcement learning with first-order logic. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, pp. 3505–3511. DOI:10.18653/v1/2021.emnlp-main.283. URL <https://aclanthology.org/2021.emnlp-main.283/>.
- Koubarakis M and Kyzirakos K (2010) Modeling and querying metadata in the semantic sensor web: The model stRDF and the query language stSPARQL. In: *The Semantic Web: Research and Applications (ESWC), Lecture Notes in Computer Science*, volume 6088. Springer, pp. 425–439. DOI: 10.1007/978-3-642-13486-9\_29.
- Manhaeve R, Giannini F, Ali M, Azzolini D, Bizzarri A and Borghesi A (2025) Benchmarking in neuro-symbolic ai. In: *Learning and Reasoning: 4th International Joint Conference on Learning and Reasoning, IJCLR 2024, and 33rd International Conference on Inductive Logic Programming, ILP 2024, Proceedings*. DOI:10.1007/978-3-032-09087-4\_17. URL [https://doi.org/10.1007/978-3-032-09087-4\\_17](https://doi.org/10.1007/978-3-032-09087-4_17).
- Missier P, Belhajjame K and Cheney J (2013) The w3c prov family of specifications for modelling provenance metadata. In: *Proceedings of the 16th International Conference on Extending Database Technology, EDBT '13*. New York, NY, USA: Association for Computing Machinery. ISBN 9781450315975, p. 773–776. DOI:10.1145/2452376.2452478. URL <https://doi.org/10.1145/2452376.2452478>.
- Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S and Hassabis D (2015) Human-level control through deep reinforcement learning. *Nature* 518(7540): 529–533. DOI:10.1038/nature14236. URL <https://doi.org/10.1038/nature14236>.
- Moreau L and Missier P (2012) Prov-dm: The prov data model.
- Motik B (2012) Representing and querying validity time in RDF and OWL: A logic-based approach. *Journal of Web Semantics* 12-13: 3–21. DOI:10.1016/j.websem.2011.11.004.
- Oh J, Singh S and Lee H (2017) Value prediction network. In: *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., pp. 6120–6130. URL <https://dl.acm.org/doi/10.5555/3295222.3295360>.
- Pritzel A, Uria B, Srinivasan S, Badia AP, Vinyals O, Hassabis D, Wierstra D and Blundell C (2017) Neural episodic control. In: *Proceedings of the 34th International Conference on Machine Learning, Proceedings of Machine Learning Research*, volume 70. PMLR, pp. 2827–2836. URL <https://proceedings.mlr.press/v70/pritzel17a.html>.
- Seaborne A and Harris S (2013) SPARQL 1.1 query language. W3C recommendation, W3C. <https://www.w3.org/TR/2013/REC-sparql11-query-20130321/>.
- Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L and Polosukhin I (2017) Attention is all you need. In: *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., pp. 5998–6008. URL <https://papers.nips.cc/paper/7181-attention-is-all-you-need>.
- Wayne G, Hung C, Amos D, Mirza M, Ahuja A, Grabska-Barwinska A, Rae JW, Mirowski P, Leibo JZ, Santoro A, Gemici M, Reynolds M, Harley T, Abramson J, Mohamed S, Rezende DJ, Saxton D, Cain A, Hillier C, Silver D, Kavukcuoglu K, Botvinick MM, Hassabis D and Lillicrap TP (2018) Unsupervised predictive memory in a goal-directed agent. *CoRR* abs/1803.10760. URL <http://arxiv.org/abs/1803.10760>.
- Yih Wt, Chang MW, He X and Gao J (2015) Semantic parsing via staged query graph generation: Question answering with knowledge base. In: Zong C and Strube M (eds.) *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 1321–1331. DOI:10.3115/v1/P15-1128. URL <https://aclanthology.org/P15-1128/>.
- Yun S, Jeong M, Kim R, Kang J and Kim HJ (2019) Graph transformer networks. In: *Advances in Neural Information Processing Systems 32*. DOI: 10.5555/3454287.3455360. URL <https://proceedings.neurips.cc/paper/2019/hash/9d63484abb477c97640154d40595a3bb-Abstract.html>.
- Zambaldi V, Raposo D, Santoro A, Bapst V, Li Y, Babuschkin I, Tuyls K, Reichert D, Lillicrap T, Lockhart E, Shanahan M, Langston V, Pascanu R, Botvinick M, Vinyals O and Battaglia P (2019) Deep reinforcement learning with relational inductive biases. In: *7th International Conference on Learning Representations (ICLR 2019)*. URL <https://openreview.net/forum?id=HkxaFoC9KQ>.