



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

**DINFO**  
DIPARTIMENTO DI  
INGEGNERIA  
DELL'INFORMAZIONE

**DISIT**  
DISTRIBUTED SYSTEMS AND  
INTERNET TECHNOLOGIES LAB  
DISTRIBUTED DATA INTELLIGENCE  
AND TECHNOLOGIES LAB

<https://www.snap4City.org>

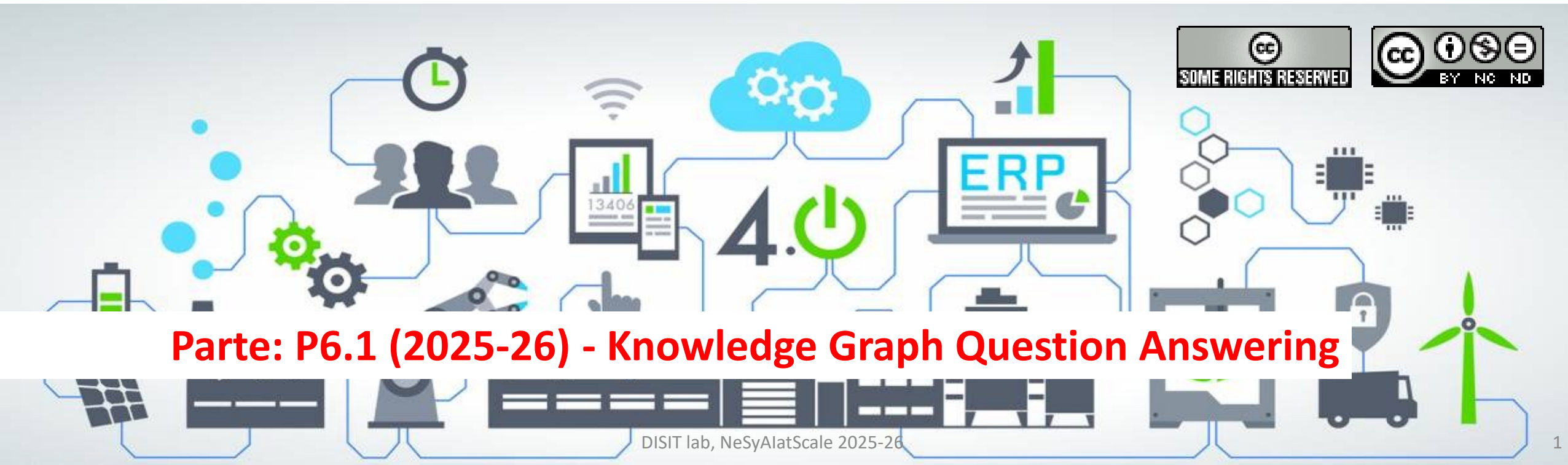
**SNAP4CITY**



# NeuroSymbolic Artificial Intelligence at Scale

Marco Fanfani, [marco.fanfani@unifi.it](mailto:marco.fanfani@unifi.it)

<https://www.disit.org/>



Parte: P6.1 (2025-26) - Knowledge Graph Question Answering

# Question Answering (QA) Problem

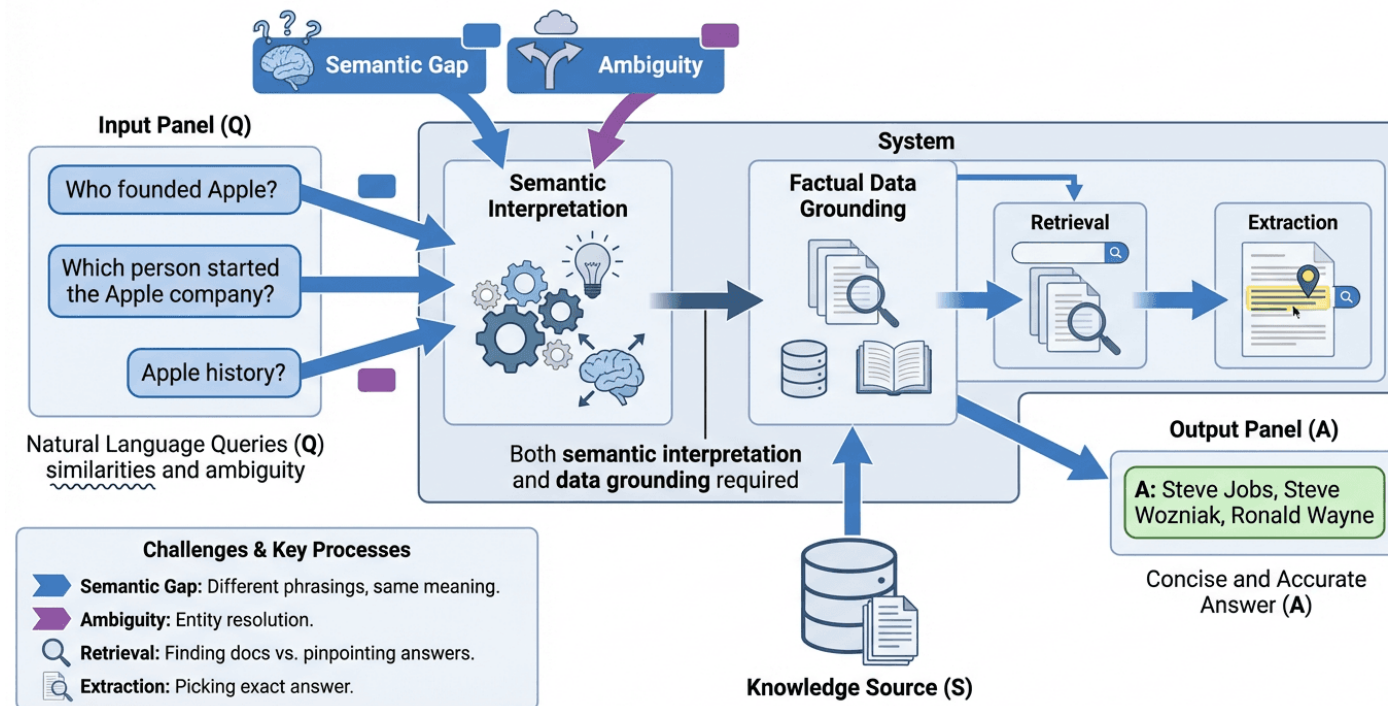
*Given a query  $Q$  expressed in natural language, the system must return a concise and accurate answer  $A$  derived from a knowledge source  $S$*

## Main challenges:

- **Semantic Gap:** Understanding that "Who founded Apple?" and "Which person started the Apple company?" require the same answer
- **Ambiguity:** Resolving entities (e.g., "Apple" the fruit vs. "Apple" the company)
- **Retrieval vs. Extraction:** Finding the right document vs. pinpointing the exact answer within it

**Semantic Interpretation**  
+  
**Factual Data Grounding**

## Semantic Interpretation and Factual Data Grounding in Question-Answering Systems



# LLM for QA

- LLMs have extraordinary **generative capabilities** and a remarkable capacity to act as versatile intelligent **agents**
- These models are trained on massive corpora of data, allowing them to **internalize general human knowledge** and sophisticated linguistic patterns
- Through techniques such as **prompt engineering** and **in-context learning**, LLMs can solve complex reasoning tasks:
  - variety of domains
  - no need for task-specific retraining
  - highly adaptable

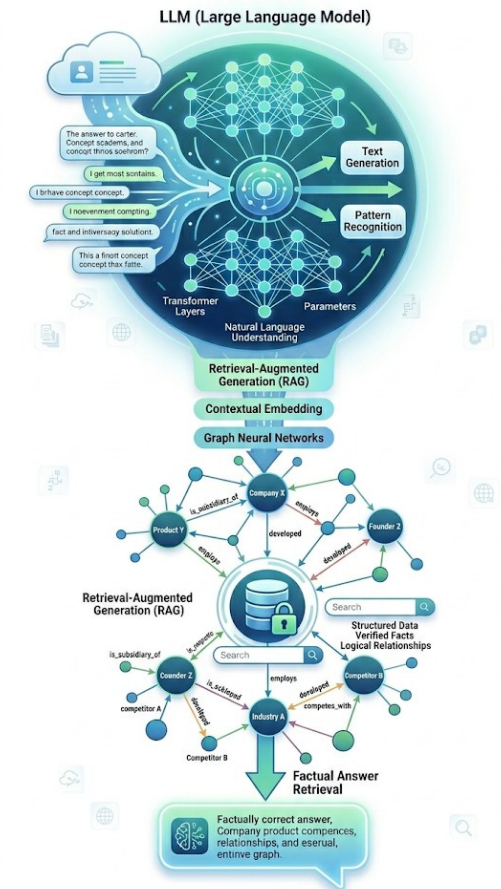


# LLM Limitation

- LLMs have **vulnerabilities** when applied to knowledge-intensive tasks
- Purely neural architectures, such as Large Language Models (LLMs), **require a systematic integration with structured symbolic knowledge bases**

The goal is to establish the necessity of a **hybrid approach** that bridges the gap between the **linguistic flexibility** of modern models and the **factual rigor** required for trustworthy AI applications

## HYBRID INTELLIGENCE SYSTEM: LLM & KNOWLEDGE GRAPH INTEGRATION



# Black Box Problem

- LLMs are characterized by an **inherent lack of transparency**, often described as the black box problem.
- Their internal decision-making processes are opaque:  
➔ **no interpretation and verification of their outputs**

In **high-stakes environments**—such as medical diagnostics, legal inquiry, or financial modeling—the inability of a system to **provide a symbolic** rationale or traceable evidence for its conclusions is a **major barrier** to widespread deployment



Reliability in reasoning is not guaranteed in purely neural models, whereas **symbolic systems are grounded in formal logics** that provide a proof for every derivation

# Hallucinations

- A central risk for LLMs is **hallucinations**:

*Generation of factually incorrect, nonsensical, or contextually irrelevant information*

- This occurs because LLMs operate on a **probabilistic basis**, predicting the next token in a sequence **based on statistical patterns** found in training data rather than by referencing a verified, structured fact
- Consequently, these models may present **false information**

# Knowledge Cutoffs

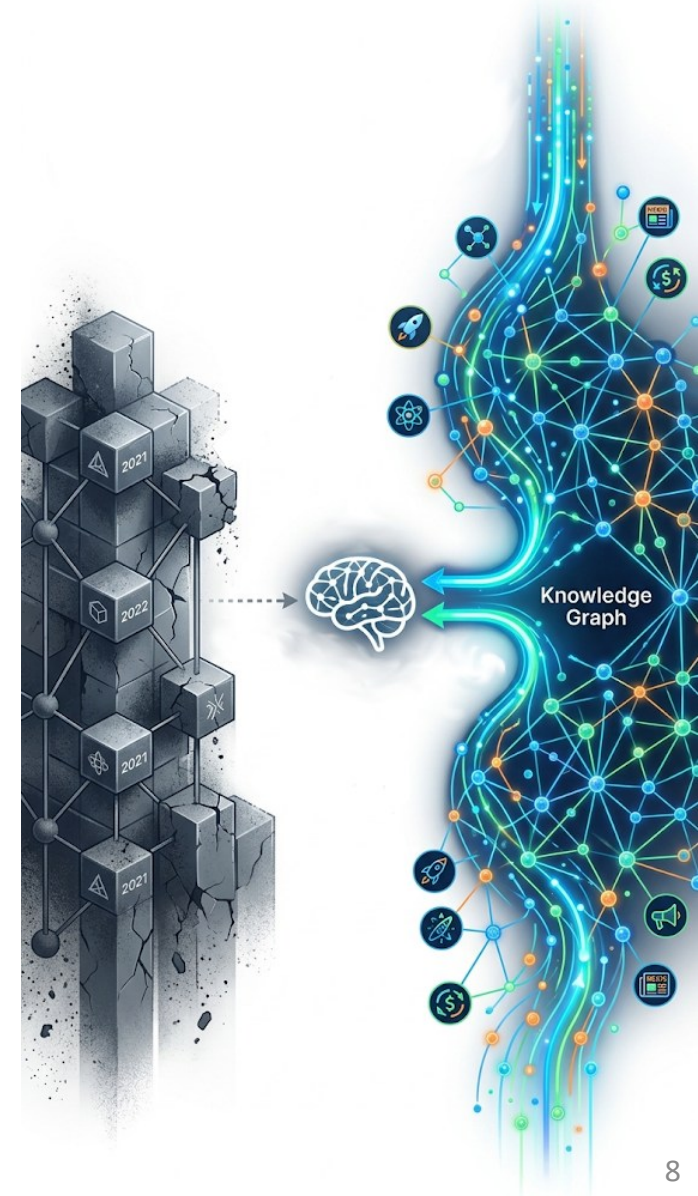
- Pure LLMs are fundamentally restricted by **knowledge cutoffs**

*LLMs internal understanding is strictly limited to the information available up until the end of their training phase*

- Maintaining an up-to-date model is a **continuous and monumental challenge**
- Full retraining or extensive fine-tuning require **immense computational resources and time**
- This static nature makes pure LLMs **impractical for use cases involving rapidly changing reality**, where facts may evolve on a daily or even hourly basis

# Knowledge Cutoffs

- The rapid evolution of the real world and the static snapshot represented by a model's training set is a **critical bottleneck** for intelligent systems
- While linguistic patterns evolve slowly, **factual truth is inherently dynamic and decentralized**
- Relying exclusively on **parametric memory**—*the knowledge stored within the weights of the network*—ensures an inevitable drift toward **obsolescence** unless the system is anchored to a **dynamic, external knowledge source**



# Specialized Domain Knowledge

- (General-purpose) LLMs frequently demonstrate subpar performance when tasked with inquiries requiring **highly specific technical expertise** or **niche domain knowledge**
  - Because their training data is primarily open-domain, they often fail to grasp the specialized terminology and complex relational dependencies found in fields such as aviation safety, malware analysis, or advanced scientific research.
  - They also struggle when a new Knowledge Graph is introduced without providing specific training (graph structure, class and properties names, interrogation logic, etc.)

# Catastrophic Forgetting

- Attempting to update an LLM through fine-tuning on new, domain-specific data often triggers a phenomenon known as **catastrophic forgetting**

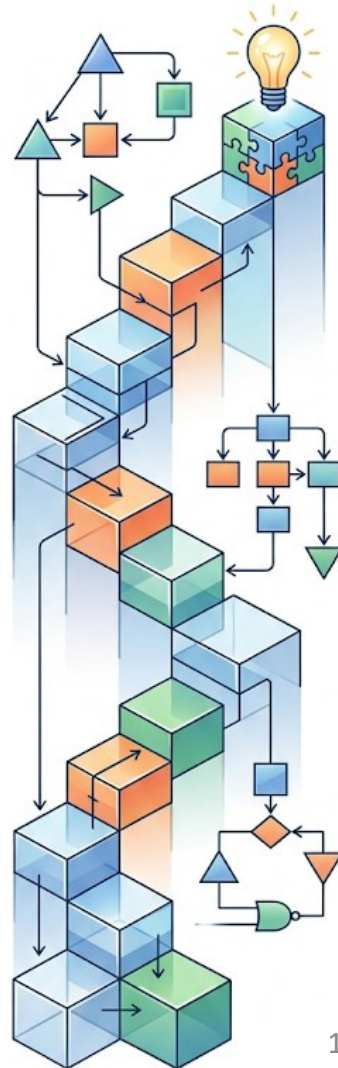
*In the process of adapting to specialized information, previously acquired general-purpose knowledge is inadvertently **overwritten** or **lost***

- This can lead to a **significant degradation** in the model's overall performance and the potential loss of fundamental dialogue and reasoning capabilities that were established during the initial pre-training

# Weaknesses in Multi-Step Deductive Reasoning



- While LLMs are proficient at inductive reasoning and pattern recognition, they often falter in tasks requiring **proof planning** and strict **logical chains**
- Purely neural models frequently **fail to maintain consistency** over long reasoning sequences
- This highlights a fundamental gap between *System 1* (fast, intuitive) and *System 2* (slow, logical)



# The Multi-Hop Reasoning Gap

- Multi-hop question answering requires a system to **recall**, **aggregate**, and **chain together** several discrete facts to reach a final conclusion
- Pure LLMs struggle significantly as the number of hops or relational links increases

*"What are the release years of the films in which Tom Hanks starred and which were directed by Steven Spielberg?"*

- **Hop 1:** Start with the *Tom Hanks* entity and follow the *acts\_in* relationship to find the movies
- **Hop 2:** Filter the films obtained by following the relationship *directed\_by* to the *Steven Spielberg* entity
- **Hop 3:** For films that meet both criteria, follow the *release\_year* report to get the final answer (the year)

# Failures in Numerical and Quantitative Reasoning

- Generative models are fundamentally **ill-equipped for precise symbolic or mathematical** computation
- They often **struggle with basic numerical tasks**, such as summing values, comparing quantitative attributes across entities, or performing joint compositional and numerical reasoning
- High error rates are common in applications requiring mathematical precision, where **symbolic reasoning systems excel** by handling symbols as modular, generalizable concepts

## Lack of Contextual Attribution

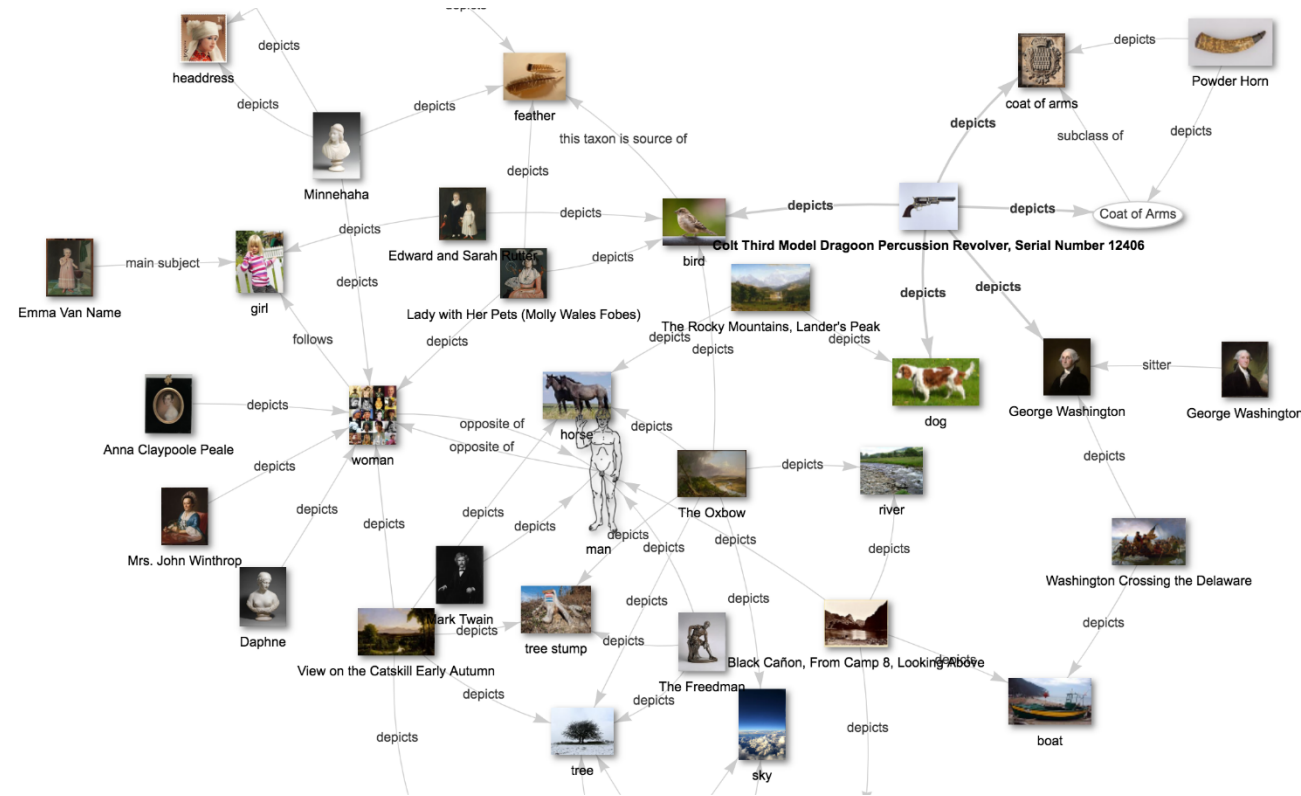
- A major limitation of purely neural models is their inability to accurately attribute the **sources of the information** they generate
- Without a built-in mechanism to **link a response back to a verifiable document or a specific triple in a knowledge base**, the reliability and transparency of the system are diminished
- Ensuring **accountability** is crucial in specialized fields; domain experts must be able to **validate that the system's logic** precisely reflects the intended relationships in the underlying data

# Knowledge Graphs

- Knowledge Graphs (KGs) have become the **backbone of modern AI**, used by major services like Google Search, Bing, Alexa, and Siri to **provide instant answers**
- KGs act as **formal representations of facts**, consisting of subjects of interest (entities) and the interactions between them (relations)
- Unlike unstructured text, KGs follow **well-defined formal syntax** and semantics, enabling high machine readability and interoperability across different sources
- The expressive nature of KGs allows for the **storage and retrieval of information through facts** organized as triples (subject, relation, object)

# Core Components of a Knowledge Graph

- A Knowledge Graph organizes **factual information** into a network of interlinked triples in the format of (Subject, Relation, Object)
  - Nodes are **real-world entities** (individuals) or **abstract concepts** (classes)
  - Edges represent the **semantic relationships** or roles between them
- This allows for information to be **machine-understandable** and globally identifiable through URIs



# Knowledge Graphs as Non-Parametric Memory

- In contrast to the parametric memory of an LLM, which is static and stored in weights, a **Knowledge Graph acts as non-parametric memory**
- It provides an **external, structured repository of ground truth** that can be updated in real-time without the need for model retraining
- This memory acts as an **authoritative and verifiable reference**, allowing the perception layer of an AI to ground its understanding in persistent, explicit facts

# Advantages of Structured Factual Knowledge

- Knowledge Graphs provide an **explicit and unambiguous representation of complex interactions**, including multi-hop relations and property characteristics
- By utilizing structured data, KGQA systems can perform exact searching and retrieval, ensuring that the evidence used to construct an answer is **factually grounded**
- This significantly **reduces the risk of hallucinatory responses** by forcing the system to operate within the constraints of verified, traceable data

## Semantic Web Standards in KGQA

- Many modern Knowledge Graphs adhere to **W3C** open standards, such as **RDF** for data modelling and **OWL** for defining rich semantics
  - These standards facilitate interoperability and promote the development of reusable software resources
- **SPARQL** serves as the standardized query language



W3C Semantic Web logo © W3C, used according to W3C guidelines.

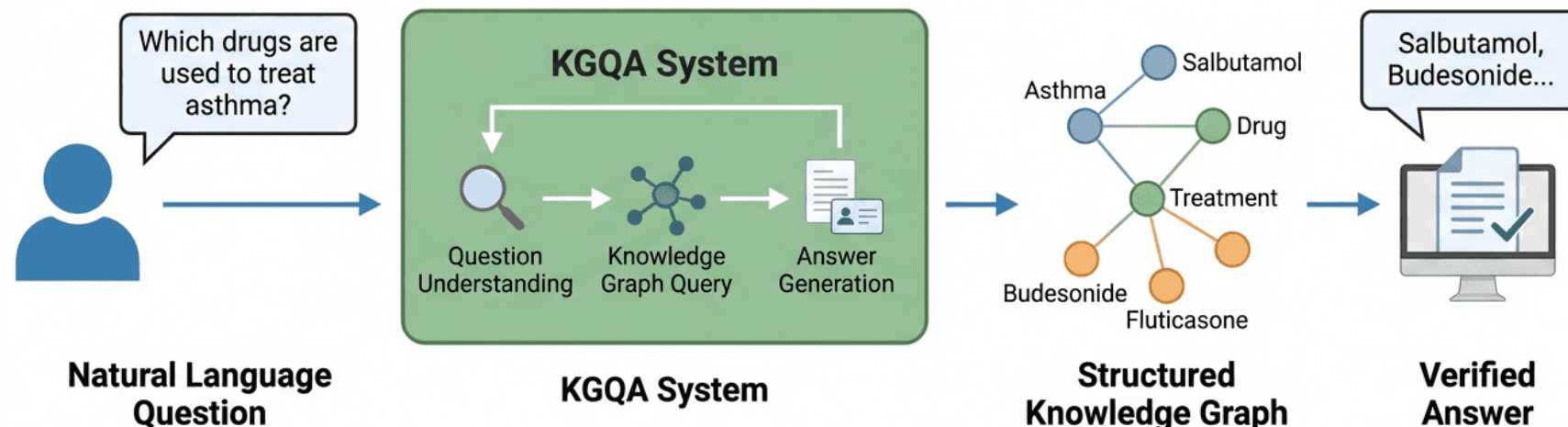
# Knowledge Graph Question Answering

- While KGs are highly expressive, they typically **require formal query languages like SPARQL or SQL** to retrieve information
- The requirement to understand the syntax of these formal languages and the underlying graph structure **poses a significant barrier** for non-expert users

*Knowledge Graph Question Answering (KGQA) aims to overcome this by providing a natural language interface where users can ask questions using their own terminology and obtain answers grounded on factual data*

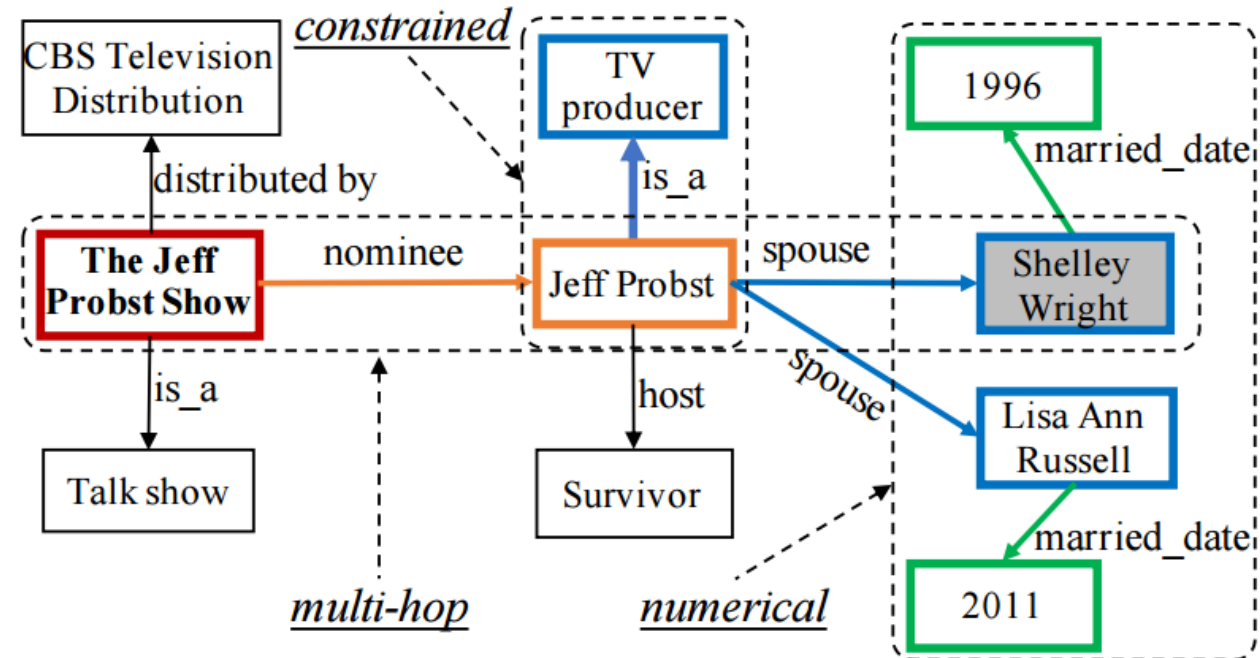
# Introduction to the KGQA Paradigm

- Knowledge Graph Question Answering (KGQA) is defined as the task of answering **natural language questions** by grounding the reasoning process in the information stored within a **structured Knowledge Graph (KG)**
- The paradigm aims to **bridge the gap** between human linguistic intent (the question) and the machine-readable, verified facts available in a graph's schema
- KGQA facilitates a cooperation between **human users** and **software agents**



## Simple vs Complex QA

- Simple QA refers to questions that involve only a **single factual statement** (a single triple)
  - E.g., *Where was Dante Alighieri born?*
- Complex QA involves questions that require more than a single fact, often involving **multi-hop reasoning** or **multiple subjects**
  - E.g., *Who is the first wife of TV producer that was nominated for The Jeff Probst Show?*



# The Goal of Hybrid KGQA

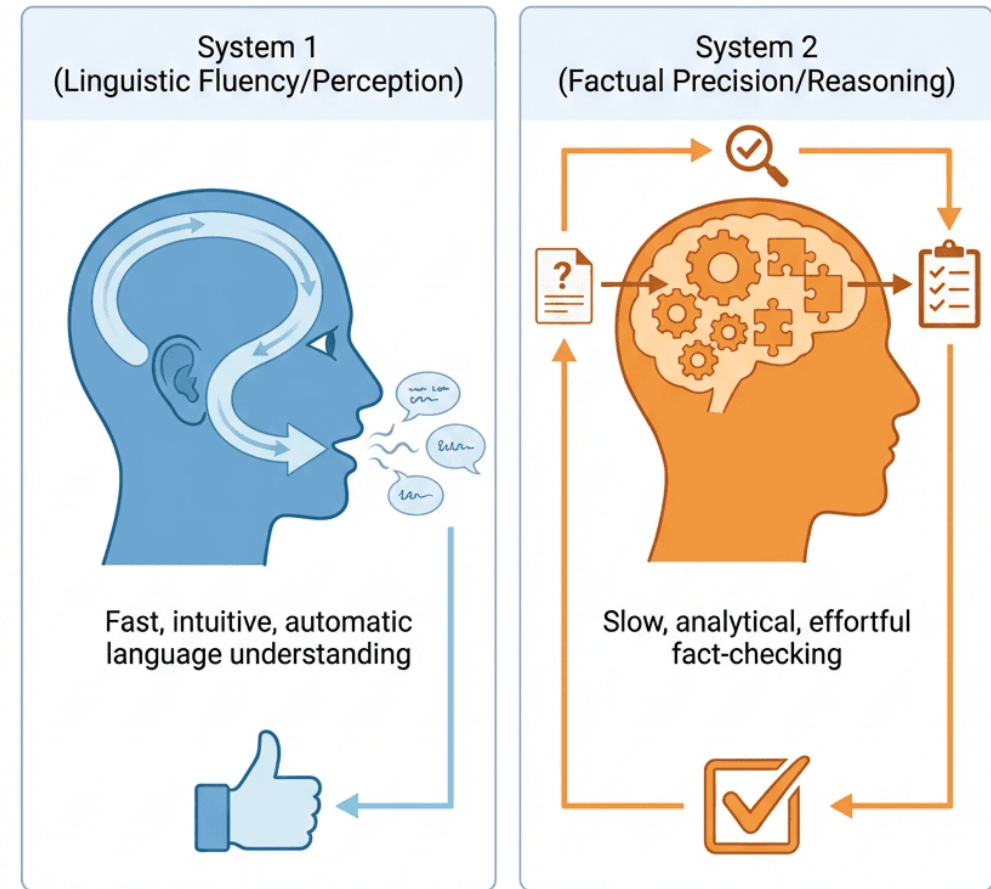
- The objective of hybrid KGQA is to

Synthesize the natural language understanding capabilities of LLMs



Rigorous structural reasoning of Knowledge Graphs

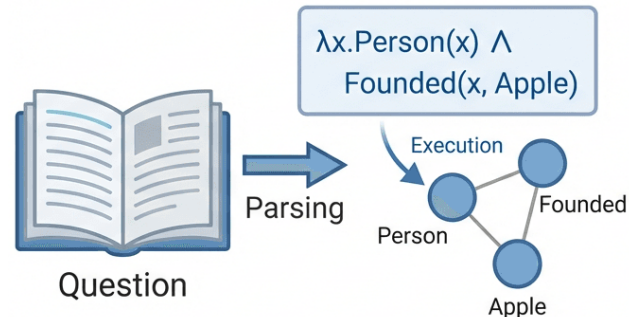
- This synergy creates systems that can
  - interpret the **nuances of human questions**
  - accurately **navigate a graph** to extract specific, verified facts
- By combining these worlds, we achieve a balance between linguistic fluency (**System 1 perception**) and factual precision (**System 2 reasoning**)



# Semantic Parsing vs. Information Extraction

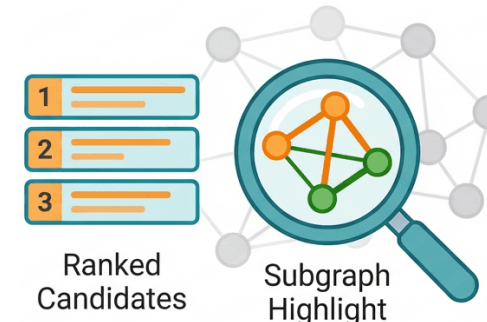
- The KGQA is largely divided into two approaches:

The **Semantic Parsing (SP)** paradigm focuses on translating the question into a symbolic logic form that can be executed against the graph



- + Explainability
- + Logical Precision

The **Information Extraction (IE)** paradigm focuses on constructing a question-specific subgraph and ranking all candidate entities within it



- + Robustness
- + End-to-end

## Semantic Parsing (SP)

- Semantic Parsing is the task of **mapping a natural language question to an executable meaning representation**, also known as a logical form
- A correct logical form must satisfy two conditions:
  - it must yield the correct answer upon execution and
  - accurately capture the full meaning of the question
- This paradigm is often referred to as *Parse-then-Execute* because the reasoning is explicitly modelled in the query itself
- SP systems **prioritize interpretability**, as the generated query provides an explicit trace of the reasoning steps taken by the model

# SP Workflow (1): Natural Language Understanding

- The first step in the SP workflow is to conduct **semantic and syntactic analysis** of the user's input
- The system utilizes a **question understanding module**, often implemented via neural networks, to create an **encoded question representation** capturing key components such as **entities and relations** and **question focus**
- Example: "*Who is the CEO of Apple?*"
  - Entity = {Apple}
  - Relation = {is the CEO}
  - Focus = {Who}

## SP Workflow (2): Logical Parsing

- In this stage, the encoded question is transformed into an uninstantiated logical form

*An **Uninstantiated Logical Form** is a formal representation (structure) of a question (e.g., how many relations or constraints are needed) that hasn't yet been linked to specific data in the KG*

- Example: "Who is the CEO of Apple?"

$$\text{query}(x) \leftarrow \text{is\_a}(E_1, \text{Organization}) \wedge R_1(x, E_1)$$

that means:

"Find  $x$  where an unknown Entity ( $E_1$ ) has an unknown Relationship ( $R_1$ ) with  $x$ ."

## SP Workflow (2): Logical Parsing

$$query(x) \leftarrow is\_a(E_1, Organization) \wedge R_1(x, E_1)$$

equivalent to

```
SELECT ?x WHERE {  
    ?entity    ?relation    ?x .  
    ?entity    rdf:type     ex:Organization .  
}
```

## SP Workflow (3): KB Grounding

- KB Grounding is the process to instantiate the logical form with specific entities and relations from the KG
- The system must map natural language phrases (e.g., "is the CEO") to the exact predicates defined in the KG ontology (e.g., `ex:ceo_of`)
- In some sophisticated workflows, logical parsing and KB grounding are performed simultaneously to prune invalid paths early in the search
- Example:

$$query(x) \leftarrow is\_a(Apple, Organization) \wedge ceo\_of(x, Apple)$$

## SP Workflow (3): KB Grounding

$query(x) \leftarrow is\_a(Apple, Organization) \wedge ceo\_of(x, Apple)$

equivalent to

```
SELECT ?x WHERE {  
  ex:Apple    ex:ceo_of    ?x .  
  ex:Apple    rdf:type    ex:Organization .  
}
```

# SP Workflow (3): KB Grounding

KB Grounding is composed by two main steps:

- Entity Linking
- Relation Extraction

# Entity Linking

- Entity Linking is the task of **identifying which specific entity** in a KG is being referred to by a phrase in a natural language question.
- Spans of text in the user's question are **mapped to unique identifiers (URIs)** in the KG
- This is a **non-trivial task**, as it requires the system to handle ambiguity
  - e.g., determining if the word "Apple" refers to the fruit or the technology corporation based on the surrounding context.

## Relation Extraction

- While EL finds the nouns (nodes), **Relation Extraction finds the verbs or properties** (edges) that connect them
- Unlike entities, which are usually specific noun phrases, relations are typically expressed through noun or verb phrase patterns that use less specific language (e.g., "*X was the birthplace of Y*" maps to the predicate `ex:bornIn`)
- The relation can be directly expressed in the text ("**Who directed Titanic?**") or inferred through context ("*Which Americans have been on the moon?*", *Americans* imply `ex:bornIn ex:USA`)

# Disambiguation

- Disambiguation is the process of selecting the **most probable interpretation** from a set of ambiguous candidates identified during question analysis and phrase mapping
- **Local Disambiguation** uses features like **string similarity** or **type consistency checks** to rank candidates independently of the wider graph structure

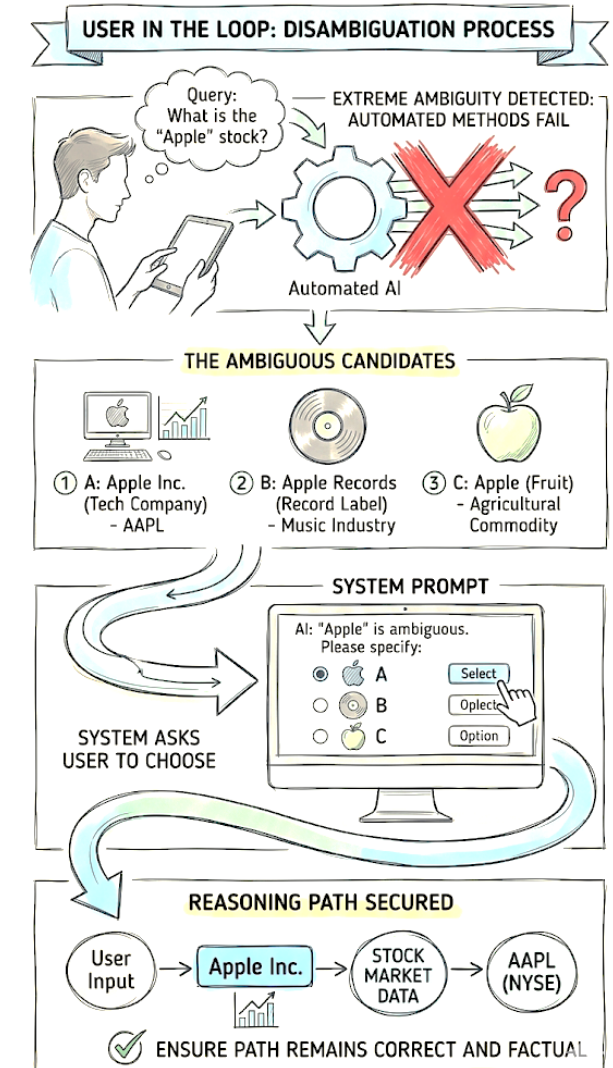
# Disambiguation

- Other approach look to the **global context**
- **Graph-based Search** leverages the KB's topology to resolve ambiguity by searching for **subgraphs that are semantically isomorphic** to the question's structure
  - The system looks for a "shape" in the KG that matches the "shape" of the question
- **Probabilistic Models** uses techniques like **Hidden Markov Models (HMM)** looking for the most likely "path" of entities that explains the whole sentence at once
  - Given "*Jordan played for the Bulls*", the model calculates that the probability of *Michael Jordan* being in the same sentence as *Chicago Bulls* is much higher than a Country (*Jordan*) being in a sentence with an Animal (*bull*).

# Disambiguation

When everything fails, ask to the user!

- **User in the Loop:** When automated methods fail due to extreme ambiguity, some systems ask the user to choose from a list of potential resources to ensure the reasoning path remains correct



## SP Workflow (4): Execution

- The final stage involves sending the fully instantiated executable query to a KB execution module
- The KB executor (e.g., a SPARQL engine) **runs the formal query against the KG** and retrieves the final denotations, or answers
- This stage is generally **parameter-free** and relies on the accuracy of the preceding parsing and grounding steps
- If the parser is successful, the output is a concise set of entities or literal values that resolve the user's request

## SP Strengths

- The primary strength of the SP paradigm is its ability to handle complex logical operators like **negation**, **conjunction**, and **disjunction**
- It excels at processing **numerical operators** (COUNT, SUM, MAX, MIN) and **comparative filters** (e.g., "older than 40")
- Logic-based approaches are highly precise because they translate vague human language into **strict formal constraints**
- This allows the system to resolve intricate questions like "*Which actors born in Rome have won more than two Oscars?*" by building a query with multiple triples and filters

## SP Strengths

- Unlike black-box models, SP systems provide a **transparent view of the reasoning** process through the symbolic logic form
- Developers and expert users can inspect the generated SPARQL queries to understand why a specific answer was returned
- This interpretability makes it easier to **diagnose errors**, such as whether the system failed during entity linking or if it mapped a relation incorrectly
- In industrial and conversational scenarios, this allows for meaningful user feedback and system rectification

# Information Extraction/Retrieval (IE/IR)

- IE views the task as a **binary classification** or **ranking problem** over candidate answer entities
- Instead of writing code, the system calculates a **matching score** between the natural language question and the candidate **entities in a vector space**
- This paradigm is often described as *Retrieve-and-Rank* because it navigates the graph structure without explicitly generating symbolic queries
- IE methods are **naturally suited for end-to-end training** and often show greater robustness when the underlying KG is incomplete

# IE/IR 1: Natural Language Understanding

- As in SP solution, at first the NL question must be parsed to extract:
  - entities
  - relations
  - question focus

## IE/IR 2: Retrieval Source Construction

- The retrieval process begins by extracting a topic-centric subgraph from the KG.
- Starting from topic entities identified in the question, the system gathers facts (i.e. nodes/arcs) within a certain number of "hops" (relation traversals), using **Entity Linking** and **Relation Extraction**
- This extracted subgraph becomes the **search space** from which all possible answers will be drawn

## IE/IR Workflow (3): Representation Learning

- Both the natural language question and the entities in the subgraph **are mapped into a low-dimensional vector space**
- Sub-graph entities and relations are **transformed into embeddings** to capture their semantic properties
- The system encodes also the question to produce **reasoning instructions**, which are often represented as vectors pointing to the most interesting areas in the sub-graph
- This numerical representation allows the system to **compute semantic relevance** using differentiable mathematical operations

## IE/IR Workflow (4): Semantic Matching

- The system conducts a semantic **matching process** to propagate information across the graph structure
- It updates the **reasoning status** of candidate nodes by calculating their compatibility with the instruction vector derived from the question
- The matching is often achieved through distance or similarity functions, such as dot products or cosine similarity
- Advanced IR models may **repeat this process multiple times** to simulate multi-step reasoning hops across the graph

## IE/IR Workflow (5): Answer Ranking

- The final stage of the IE workflow is to rank all entities contained in the question-specific graph based on the final reasoning status
- Since the embedding space is a continuous space while the KG is discrete, the reasoning process can identify multiple promising candidates.
- A ranking of candidates is used to filter out the better the top-ranked entities that are predicted as the answers to the user's question

# IE/IR Reasoning and Ranking

- **Reasoning Instruction:** Defines the selection criterion (the "compass").
- **Reasoning Status:** Maintains the context of the path (the "memory").
- **Ranking:** It is the final decision mechanism that transforms mathematical probabilities into a specific graph entity
- Example: "*Who is the CEO of Apple?*"
  - After identifying "Apple" as the starting entity, the **instruction vector** points toward the relation "CEO" and tells the system to move toward nodes connected by leadership or executive roles. It defines the semantic direction of the next hop.
  - The **status** records that we started at the node Apple and that we are currently looking for a person associated with a CEO role. Fundamental for multi-hop.
  - The "CEO direction" might lead to several candidate nodes: Tim Cook, Steve Jobs, ... All of them are "CEOs of Apple" in the KG. The **ranking** compares the reasoning instruction (and the reasoning status) against these candidates calculating a similarity score (e.g., dot product) for each.

# Robustness to Incompleteness

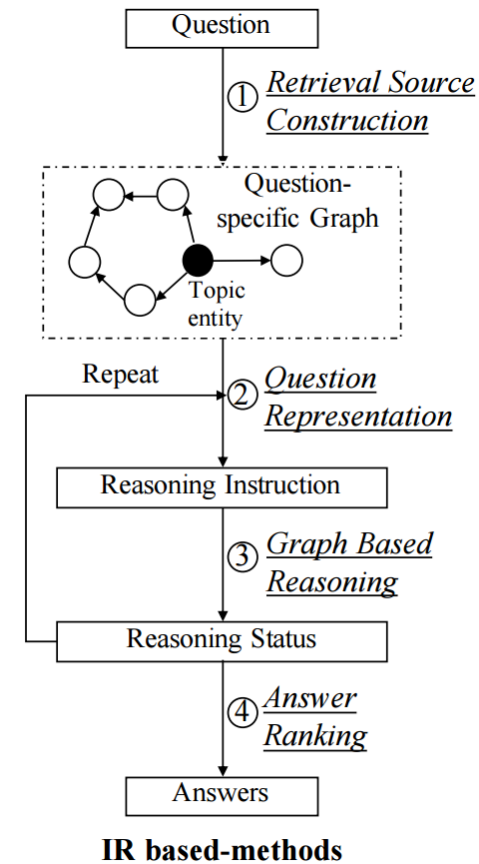
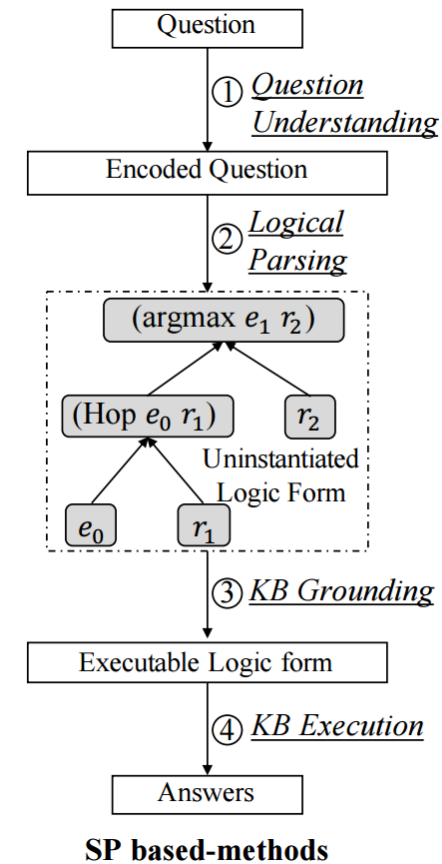
- KGs can be incomplete: facts from the real world may be missing as triples
- SP-based methods are brittle in this scenario:
  - if a single triple in the logical chain is missing, the **formal query returns an empty result**
- IE-based methods are more robust because they can leverage **semantic similarity** and **representation learning** to infer missing links:
  - By projecting queries and entities into embedding spaces, IE models can find potential answers by **identifying the nearest entity to a query vector**, even if the explicit link is absent

## Ease of Training

- A major advantage of IE-based methods is that they naturally fit into the **end-to-end training** paradigm
  - The parameters of the neural network can be directly optimized using **ground truth question-answer pairs**
  - Common loss functions (e.g., cross-entropy, KL divergence) can be utilized to train
- This avoids the complexity of generating intermediate symbolic logic forms

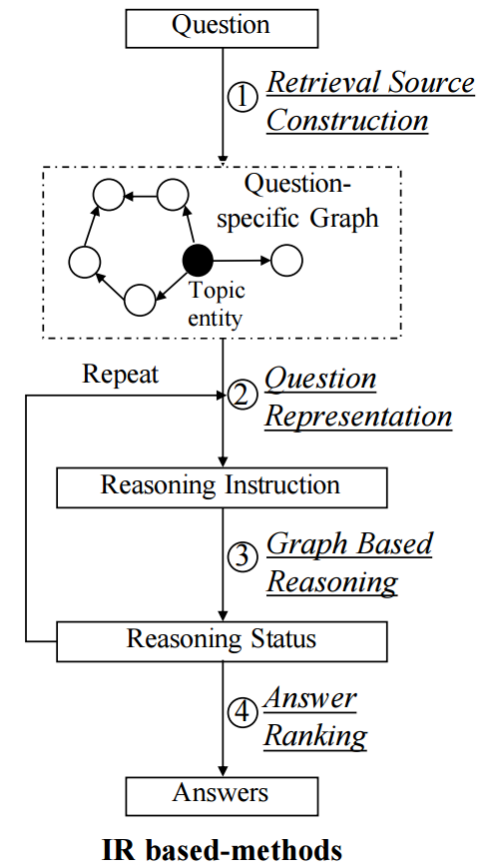
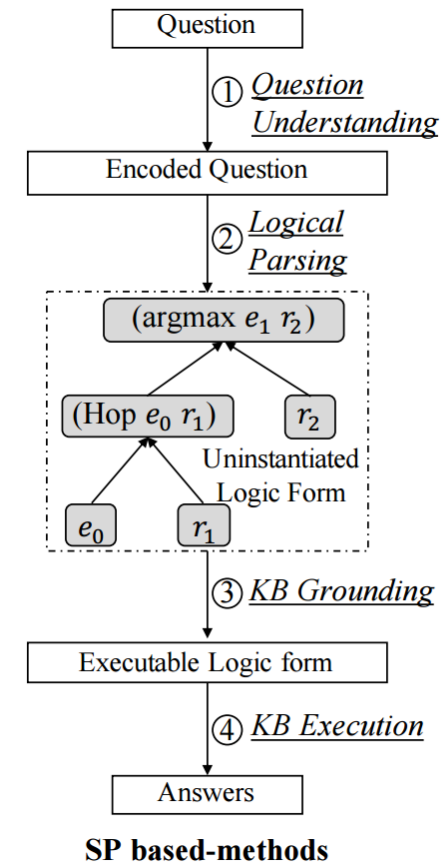
# Information Extraction vs Semantic Parsing

- Semantic Parsing is the preferred choice for tasks requiring **logical precision and explainability**. It excels at handling complex constraints (negation, counts) but is sensitive to noise and graph incompleteness
- Information Extraction is the preferred choice for **robustness and scalability**. It handles messy, incomplete graphs well through vector-based similarity but often operates as an uninterpretable "black box"
- Both paradigms rely on accurate **Entity Linking** and **Relation Identification** to bridge the lexical gap between language and data
- The ultimate goal is to unify these paradigms—**combining the symbolic interpretability of SP with the neural flexibility of IE**



# Information Extraction vs Semantic Parsing

Feature	Information Extraction (IE)	Semantic Parsing (SP)
<b>Core Logic</b>	<b>Similarity:</b> Ranks entities and relations based on how well they "match" the question's semantics.	<b>Logic:</b> Translates the question into a rigid code structure (e.g., SPARQL) for a database engine.
<b>Training Needs</b>	Requires <b>Question-Answer pairs</b> to train neural networks to rank candidates.	Requires <b>Question-Query pairs</b> for fine-tuning or few-shot prompting.
<b>Complex Logic</b>	<b>Struggles</b> with multi-hop questions, counting, comparisons, or qualifiers.	<b>Excels</b> at multi-hop and complex logical constraints, provided the query is correct.
<b>Failure Mode</b>	<b>Noise &amp; Superficiality:</b> May pick an answer that <i>sounds</i> right (superficial correlation) but is factually irrelevant.	<b>Fragility:</b> A single syntax or semantic error in the query results in an execution error and zero results.
<b>Process Flow</b>	Retrieves a candidate subgraph → Ranks → Extracts.	Translates to logic → Executes on graph engine → Returns result.
<b>Outcome</b>	Always provides a "best guess" answer, even if the question is messy.	Returns a perfect answer or nothing at all; notoriously "brittle."

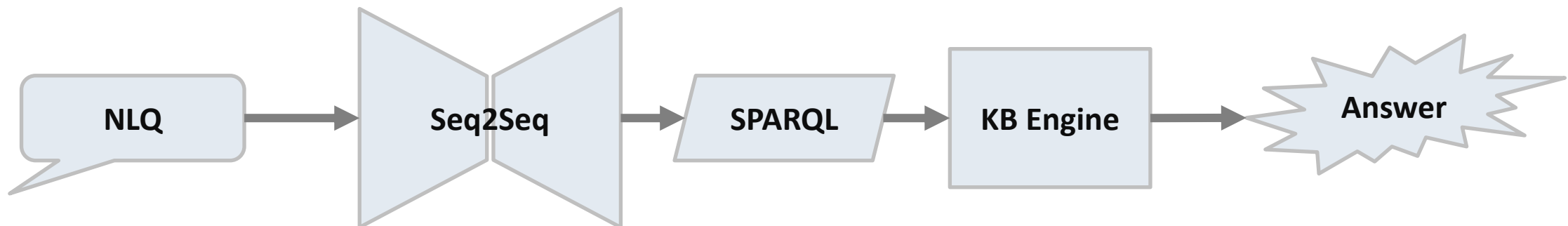


# Neural Semantic Parsing

- Traditional semantic parsing relied heavily on **manually defined rules and template extraction**, which required extensive linguistic knowledge and lacked the scalability to handle the diversity of natural language
- **Neural Semantic Parsing (NSP)** represents a paradigm shift toward data-driven approaches where neural networks **learn the mapping** from questions to logical forms directly from data
- By utilizing trainable parameters, NSP-based methods enhance both the **parsing capability** and the **flexibility** to generalize across different domains and large-scale knowledge graphs

# Seq2Seq for Semantic Parsing

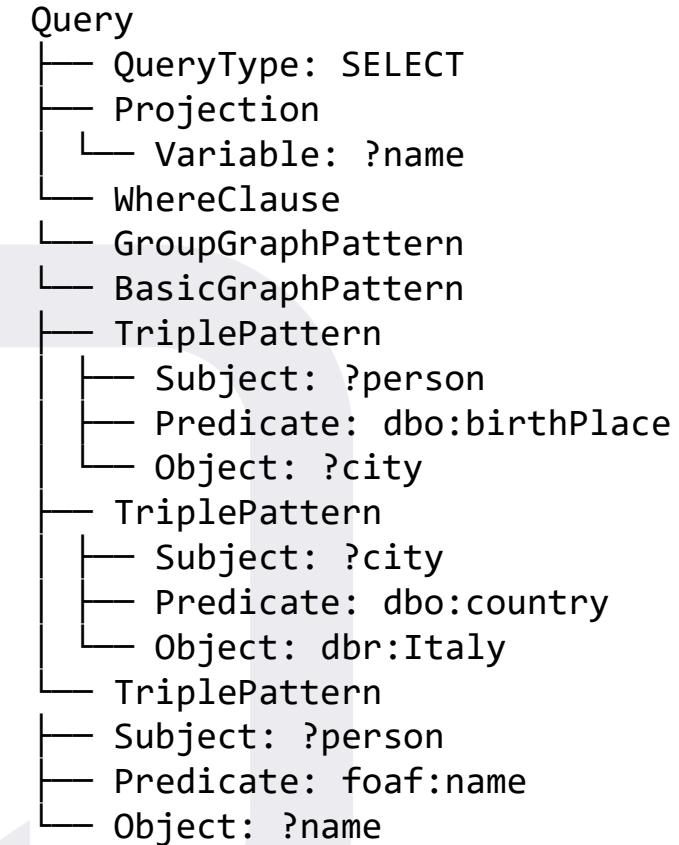
- Initial NSP often models semantic parsing as a **machine translation task**, where a natural language question (source) is translated into a formal query language (target) like SPARQL
- The standard architecture for this is the **Sequence-to-Sequence (Seq2Seq)** model, consisting of an **encoder** that creates a context-dependent representation of the question and a **decoder** that generates the output query



# Seq2Tree Models

- Formal query languages are **inherently tree-structured**
- Standard sequence decoders (Seq2Seq) often fail to exploit these **hierarchical dependencies** typical of complex compositional queries
- **Seq2Tree** models address this by introducing **structure-aware decoding strategies**
  - uses a **hierarchical tree-structured decoder** that generates queries in a top-down or bottom-up manner

*Find the names of  
people born in a city  
located in Italy*



```
SELECT ?name WHERE {  
  ?person dbo:birthPlace ?city .  
  ?city dbo:country dbr:Italy .  
  ?person foaf:name ?name .  
}
```

# Tree-structured Decoder

- A Tree-structured Decoder does not generate one word after another, but builds the query by **expanding the nodes of a logical tree**
- Example: "Who is the CEO of Apple?"
- The Seq2Tree decoder will not write the query from left to right but will follow a top-down approach, expanding inner branches:
  - Root: The decoder generates the SELECT root node.
  - Sub-nodes: Knowing that SELECT requires a condition, it generates a child node WHERE.
  - Branch expansion: The WHERE node is expanded into a triple structure:
    - Subject: ?x
    - Predicate: dbo:ceo
    - Object: dbr:Apple\_Inc
  - Closure: Once the "holes" in the structure are filled, the decoder finishes expanding.
- This allow to avoid introduction of syntax errors (e.g., missing parenthesis)

# Improving Multi-hop Query Generation

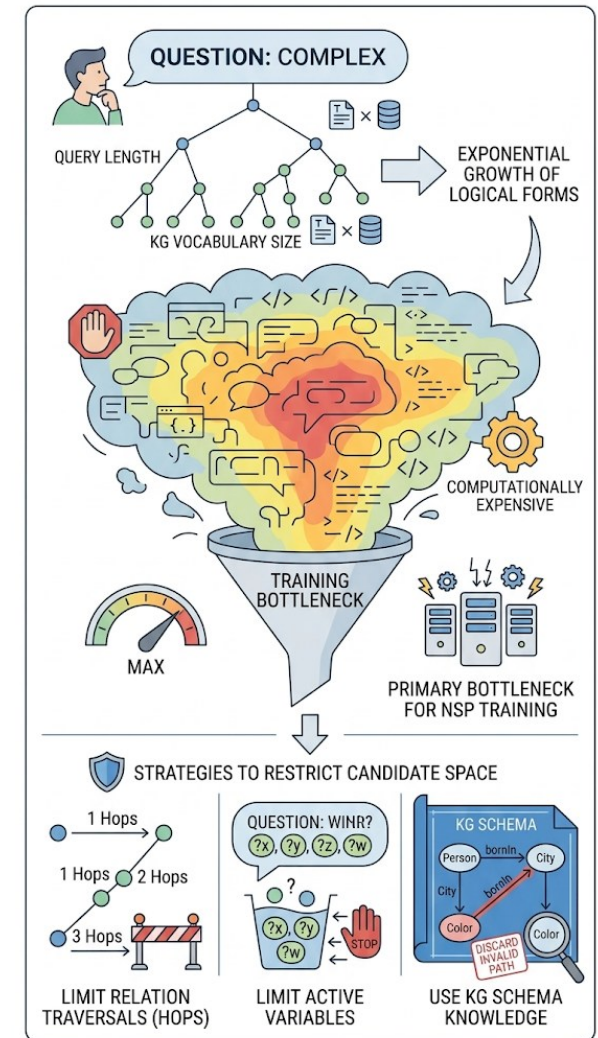
- The adoption of the **Transformer** architecture has significantly enhanced NSP by replacing recurrent layers with **self-attention mechanisms**
- Transformers are typically used as **encoders** that map a question into a rich semantic representation that is consumed by a decoder to generate a logical form
- Transformer helps to
  - bridge the gap between **natural language expressions** and **structured graph predicates**
  - **contextual understanding**, fundamental for multi-hop queries

# Exponential Explosion Problem

- As questions increase in complexity, the **number of possible logical forms** grows exponentially with the length of the query and the size of the KG vocabulary

Searching for consistent logical forms in this massive space is computationally expensive and is a primary bottleneck for NSP systems

- To make the task manageable, KGQA systems must employ **search strategies** to restrict the candidate space



# Decompose-Execute-Join

One effective strategy for handling the search space of complex questions is the **decompose-execute-join** approach

1. The system **decomposes** a complex multi-part question into several simple, single-triple sub-questions

*Where was the wife of the US president born?*



*Who is the US president?, Who is his wife?, and Where was she born?*

2. Each sub-question is **parsed and executed independently** against the KG to retrieve intermediate results
3. The final answer is then computed using **symbolic operations like intersection or union** over the sub-answers retrieved from the graph

## Decompose-Execute-Join

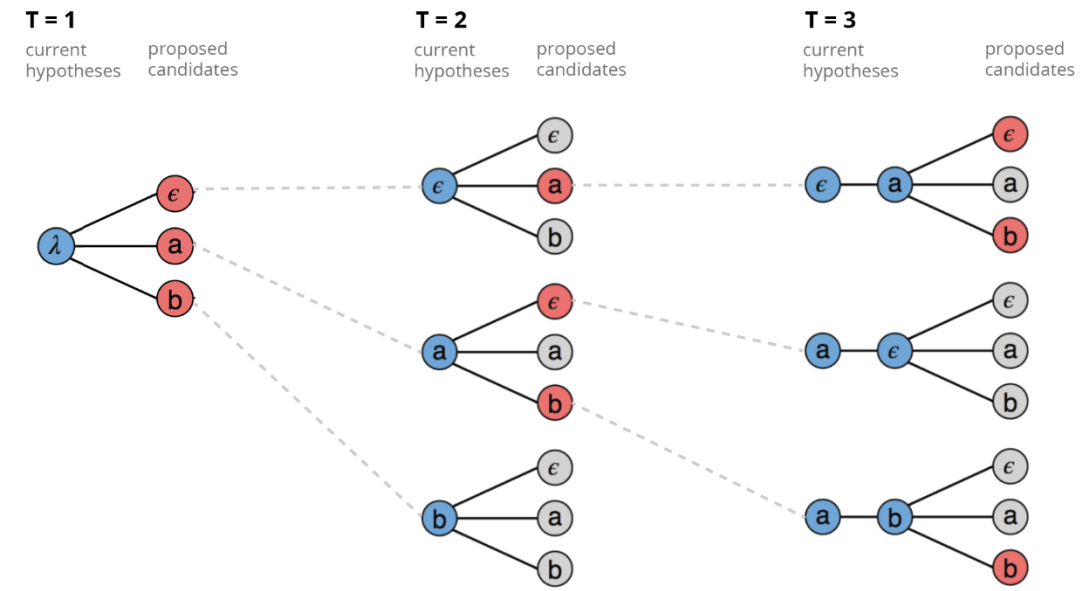
- Once partial queries are executed, the system must logically **recombine** the results, typically using **intersection** and **union**

*This symbolic merging ensures that the final result remains interpretable and verifiable, since the composition is based on **explicit KG facts***

- Advanced composition frameworks can also handle **numerical literals** and **temporal constraints**:
  - Set-based filtering: **logical filters** like  $>$ ,  $<$ ,  $=$ , ...
  - Ranking: **argmax()** or **argmin()** to identify the *youngest, first, largest, ...*

# Expand-and-Rank Strategy

- An alternative to decomposition is the **iterative expand-and-rank** strategy that
  - generates a large set of candidate queries upfront and
  - rank them based on their likelihood of correctly answering the question
- The query is **built hop-by-hop** starting from a seed entity
  - The system begins by identifying topic entities and collecting all 1-hop query graphs in their immediate neighbourhood as initial **multiple** candidates
  - These candidates are **ranked** using semantic similarity, and only the most confident partial graphs (the "beam") are expanded into more complex 2-hop or 3-hop graphs
- This iterative search ensures that the system **avoids exploring the entire graph**, significantly reducing the computational cost while maintaining high precision



Standard beam search algorithm with an output alphabet  $\{\epsilon, a, b\}$  and a beam size of three.

# Staged Query Generation (STAGG)

- STAGG is a representative approach that tackles semantic parsing through a **staged pipeline**
- Rather than generating a full query in one step, the **process is divided into manageable phases**:
  - First, entity linking identifies candidate entities mentioned in the question
  - Next, the system constructs a core chain representing the main relational path
  - Finally, additional constraints such as filters or aggregations are added.
- This staged design **simplifies the search space** and allows each component to be optimized independently, improving both efficiency and accuracy.
- STAGG is essentially an **expand-and-rank style search over query graphs**, where candidate query structures are incrementally expanded and scored at each step, but implemented within a **learned neural model that acts as a scoring function**

# Learning with Weak Supervision

- Training semantic parsing models is **challenging** because annotated logical forms are often unavailable
- Instead, systems are trained using **weak supervision**, where only the final answer is known.
- The logical form is obtained as a **latent variable**
- Possible techniques includes:
  - Maximum Marginal Likelihood (MML)
  - Reinforcement Learning (REINFORCE) + Reward shaping

# Learning with Weak Supervision

- **Maximum Marginal Likelihood (MML)** it does not search for a single correct query, but considers all possible queries that lead to the right answer.
- It is commonly used to maximize the probability of generating any logical form that leads to the correct answer
  - The model generates many candidate queries
  - They run on the knowledge graph
  - All those that produce the correct answer contribute to learning

$$\log p(y \mid x) = \log \sum_{z \in Z(x,y)} p(z \mid x)$$

- $x$  = Question
- $y$  = correct answer
- $z$  = logical query (latent)
- $Z(x, y)$  = all queries that produce  $y$

# Learning with Weak Supervision

- Reinforcement Learning approaches, such as REINFORCE, treat query generation as a **sequential decision process**
- It produces a sequence of decisions:
  - choose an initial entity (e.g., Rome)
  - choose a relationship (e.g., capitalOf, locatedIn, etc.)
  - maybe it adds another hop
  - constructs a complete logical query (SPARQL)
- The obtained query is executed and **get a reward** if the query produces the right results
- Problems:
  - reward are very rare
  - the model doesn't know which part of the query is right or wrong

# Learning with Weak Supervision

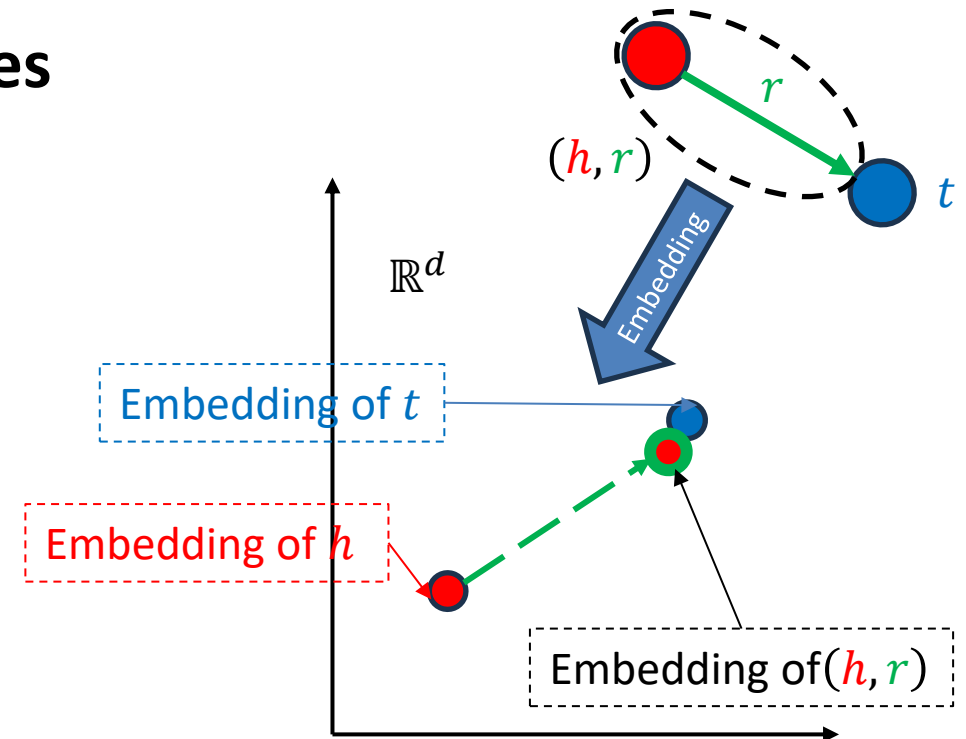
- With pure REINFORCE the signal only comes at the end and most sampled queries are incorrect during early training
- **Reward shaping** adds intermediate rewards (**proxy signal**) to guide the search:
  - is the query executable?
  - is the query coherent with the KG structure?
  - are selected entities/relations plausible w.r.t. the question?
- The model still **does not use gold logical forms** or gold intermediate annotations
- Intermediate rewards are computed automatically using **heuristic signals** such as:
  - string matching / lexical similarity
  - embedding similarity
  - KG-based constraints and exploration heuristics
  - type consistency and connectivity in the KG

## Reasoning in Vector Space

- Unlike semantic parsing, IR-based KGQA operates directly in a continuous vector space
- The system **retrieves relevant entities or subgraphs** based on similarity measures
- Questions and graph components are embedded into the **same vector space**
- Answering becomes a **similarity-based retrieval** task rather than logical execution performed through geometric operations
- This approach enables **end-to-end differentiable training** and often scales better to large graphs
- However, the **lack of explicit logical forms** can make reasoning **less interpretable** and harder to control

# Knowledge Graph Embeddings and GNNs

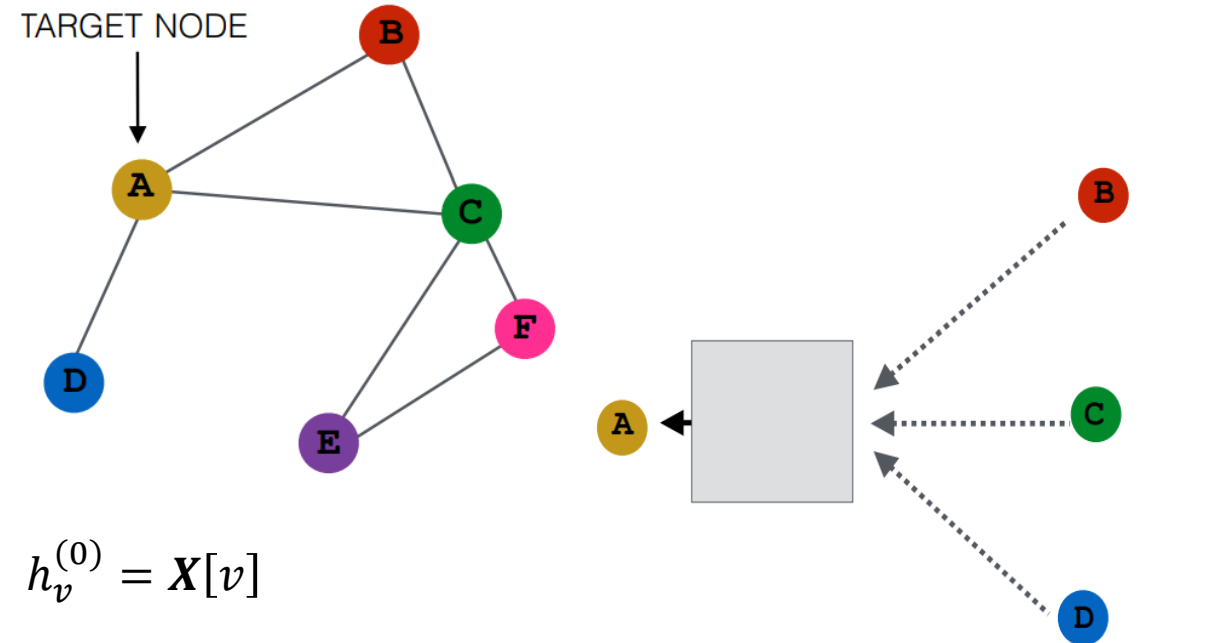
- Knowledge Graph Embeddings (KGE) and Graph Neural Networks (GNNs) extend vector-space IR by **incorporating the structure of the knowledge graph**
- Knowledge Graph Embeddings represent **entities and relations as vectors**, capturing relational patterns through geometric transformations:
  - Entities  $\rightarrow$  Vectors
  - Relations  $\rightarrow$  Geometric transformations
- The retrieval is not symbolic (match on labels), but **geometric** (nearest neighbours in the embedding space)



# Knowledge Graph Embeddings and GNNs

- **Graph Neural Networks (GNNs)** extend KGE by explicitly modelling graph structure:
    - Through message passing, **nodes aggregate information from their neighbours**, enabling multi-hop reasoning
  - KKG and GNN allow models to infer **implicit relationships** and generalize beyond explicitly observed triples. If you have seen:
    - (France, capital, Paris)
    - (Italy, capital, Rome)
- you can infer:
- (Spain, capital, Madrid)

because you **have learned the "capital relation" pattern** in vector space.



$$h_v^{(0)} = X[v]$$

$$h_v^{(l+1)} = \sigma \left( W_l \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B_l h_v^{(l)} \right), \forall l \in \{0, \dots, L-1\}$$

$$z_v = h_v^{(L)}$$

# Multi-step Reasoning with External Memory

- Memory-augmented architectures introduce an explicit mechanism to store and retrieve facts during reasoning
- The knowledge graph is treated as a **differentiable memory bank**:
  - each memory slot stores a fact (typically a triple)
  - the model performs iterative retrieval over memory
- Instead of constructing or executing symbolic multi-step queries, the model performs **iterative attention-based retrieval** over a memory bank of triples

# Multi-step Reasoning with External Memory

- The knowledge graph is encoded as a set of fact embeddings (memory slots)
  - Each slot represents a triple (h, r, t)
- The model does not execute logical steps, but repeatedly:
  - computes attention over all memory slots
  - retrieves a weighted combination of relevant facts updating an internal state

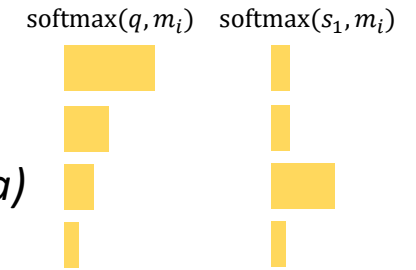
*q*="Who is the president of the country  
where Rome is located?"

*m*<sub>1</sub>=(Rome,locatedIn,Italy)

*m*<sub>2</sub>=(Italy,capital,Rome)

*m*<sub>3</sub>=(Italy,president,Mattarella)

*m*<sub>4</sub>=(France,capital,Paris)



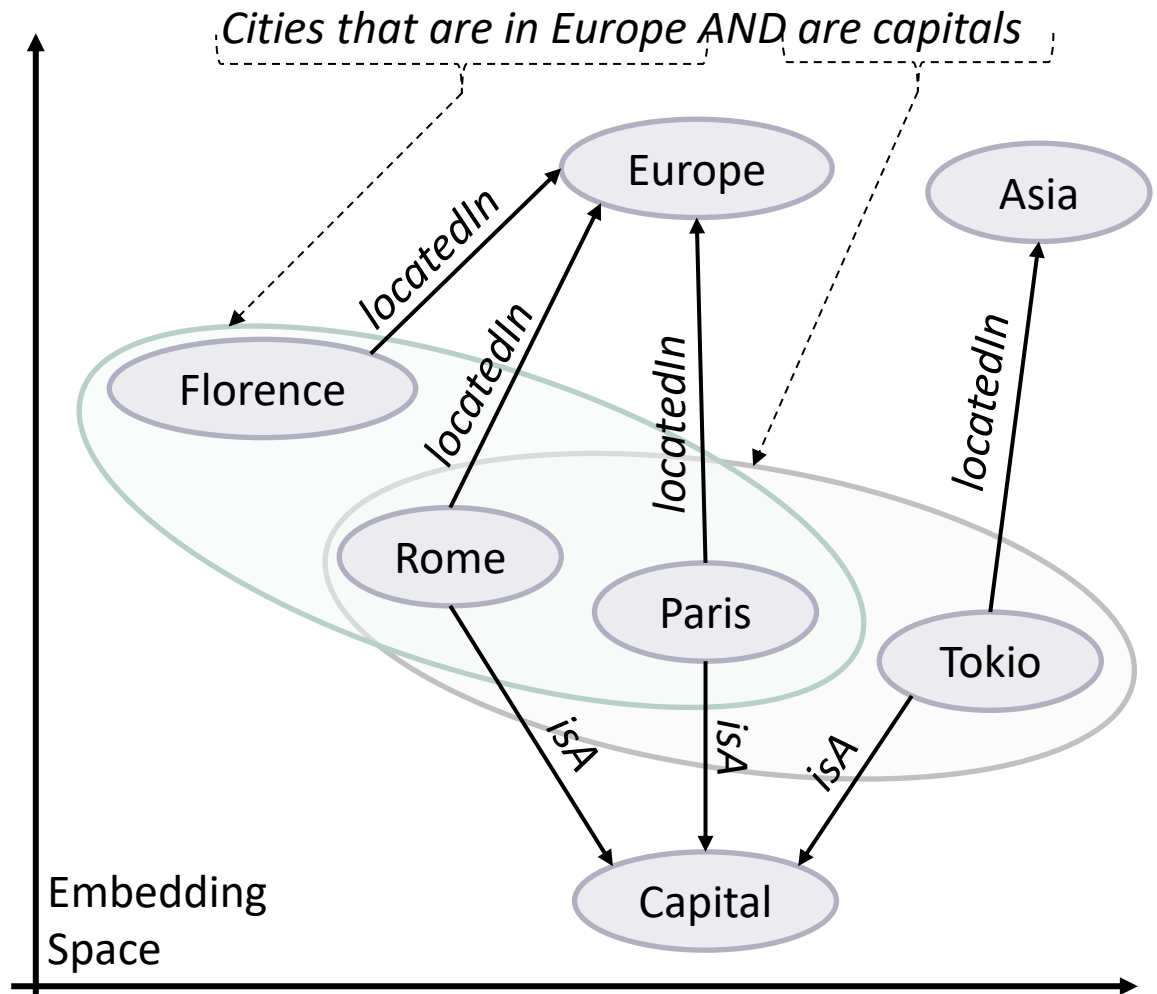
1. Compute similarity between *q* and all memory slots *m*<sub>*i*</sub>
2. Get probability distribution  $\alpha$  using softmax
3. Update the state computing the weighted sum of memory slot according to the  $\alpha$  weights:

$$s_1 = \sum_i \alpha_i m_i$$

4. repeat using the updated state

# Learning Logical Operators in Embedding Space

- Geometric query embedding methods extend KGE-style representations to **full logical queries**, modelling them as geometric regions in vector space.
- Instead of representing only entities and relations entire **queries are embedded as vectors or sets/regions**
- Logical operations are approximated geometrically:
  - Projection (relation traversal)  $\rightarrow$  vector transformation
  - Intersection (AND)  $\rightarrow$  geometric overlap of regions
  - Union (OR)  $\rightarrow$  combination of regions



# Interpretability in IR-based Systems

- A major criticism of IR-based KGQA is its **lack of interpretability**.
- Unlike semantic parsing, these models **do not produce explicit reasoning steps**
- Recent work focuses on extracting **reasoning paths** or **subgraphs** that justify the final answer by identifying which nodes and edges (triples) contributed most to the prediction, e.g.,  $\alpha$  in attention-based model
- IR-based KGQA explanations are **similar to XAI**: post-hoc and identify which parts of the graph contributed most to the prediction, rather than providing explicit symbolic reasoning steps

## Dealing with Noise and Ambiguity

- Real-world questions are often **noisy, ambiguous, or incomplete**
- KGQA systems must handle **spelling variations, implicit constraints, and missing links** in the graph
- IR-based methods are generally **more robust** to incomplete graphs
- SP **struggles** when exact logical mappings are unclear.

*Improving robustness remains a key challenge, especially for deployment in open-domain settings*

## LLM in KGQA

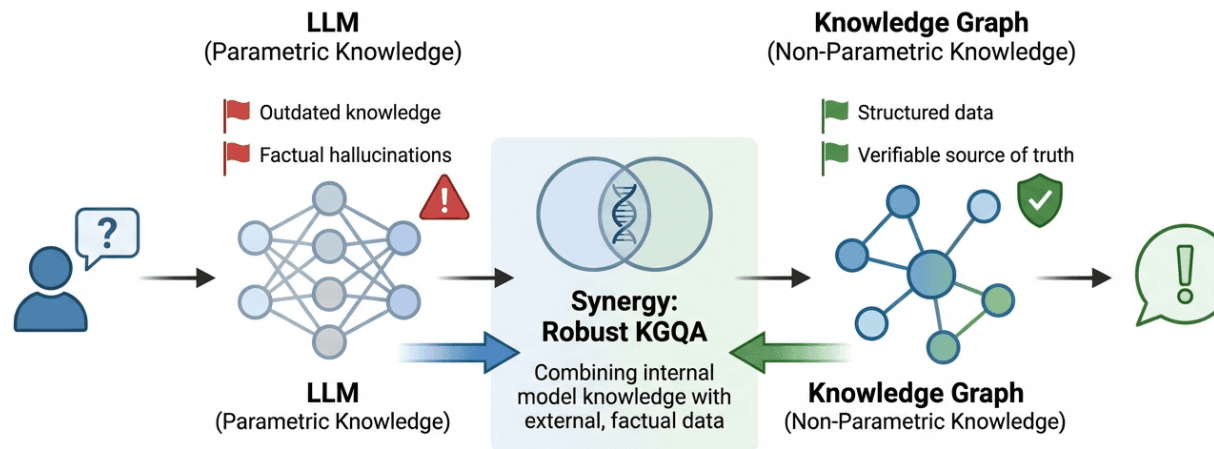
- Large Language Models are reshaping KGQA by shifting the paradigm from rigid query generation to **flexible reasoning systems**
- LLMs can interpret, decompose, and adapt their reasoning dynamically
- This transition enables systems to handle ambiguity, incomplete information, and complex multi-step questions more effectively

*Rather than acting as standalone components, LLMs become **central orchestrators** that coordinate multiple reasoning strategies*

- KGQA is no longer just about querying a graph—it is about **integrating structured knowledge into a broader reasoning process**

# LLM in KGQA

- LLMs suffer from **outdated knowledge** and **hallucinations**
- KG provide a **structured** and **verifiable** source of truth
- The synergy between **parametric knowledge** (in the model) and **non-parametric knowledge** (in the KG) is a key design principle



**Key Design Principle:**  
Combining parametric and non-parametric knowledge enables reliable, up-to-date KGQA.

# Constraining LLM Reasoning with Graph Structure

- Knowledge Graphs can be used not only as a source of facts but also as a guide for reasoning
- By exposing graph paths or subgraphs, systems can **constrain the reasoning space of the LLM**
- Instead of exploring all possible interpretations, the model focuses on **paths that are valid within the graph structure**. This reduces hallucinations and improves logical consistency

# Constraining LLM Reasoning with Graph Structure

*Who is the president of the country where Rome is located?*

## LLM without KG

LLM explore several interpretation:

- Rome → (guess) Italy → president → Mattarella
- Rome → Europe → president → ???
- Rome → Catholic Church → Pope

## LLM with KG

1. Relevant subgraph extraction
  - Rome --locatedIn--> Italy
  - Italy --president--> Mattarella
  - Italy --capital--> Rome
  - France --capital--> Paris
2. Constraints on reasoning
3. Guided reasoning:
  - Rome → locatedIn → Italy → president → Mattarella

# Constraining LLM Reasoning with Graph Structure

*Who is the president of the country where Rome is located?*

## LLM with KG

### 1. Relevant subgraph extraction

Rome --locatedIn--> Italy  
Italy --president--> Mattarella  
Italy --capital--> Rome  
France --capital--> Paris

### 2. Constraints on reasoning

### 3. Guided reasoning:

- Rome → locatedIn → Italy  
→ president → Mattarella

- This can be obtained for example by subgraph retrieval + prompting
  - Entity linking: *Rome* → KG node
  - Get neighbour nodes (k-hops)
  - Prompt:

Question: ...

Facts:

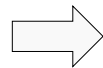
- Rome is located in Italy
- Italy has president Mattarella

Answer: ...

## Ensuring Valid Outputs

- One of the main challenges in using LLMs for SPARQL generation is **ensuring syntactic and semantic correctness**
- Graph-constrained decoding addresses this issue by **restricting the generation process**
- Using structures such as trees or grammars derived from the KG schema, the model is allowed to **generate only valid tokens and relations**
- This dramatically reduces invalid queries and improves execution success

dbo:birthplace  
dbo:birthdate  
dbo:country



```
(root)
├── dbo:
│   ├── birthPlace
│   ├── birthDate
│   └── country
```

During SPARQL generation

```
SELECT ?x WHERE {
  ?x dbo:bi...
```

The model may want to generate:

✗ dbo:bird (non esiste)

✓ dbo:birthPlace

# Constrained Decoding for SPARQL

- Constrained decoding can be implemented through different mechanisms:

- **Grammar-based decoding:** enforces SPARQL syntax (e.g., SELECT–WHERE structure)

```
QUERY → SELECT VAR WHERE CLAUSE  
CLAUSE → TRIPLE+  
TRIPLE → SUBJECT PREDICATE OBJECT
```

- **Trie-based constraints:** restrict tokens to valid entities and relations from the KG

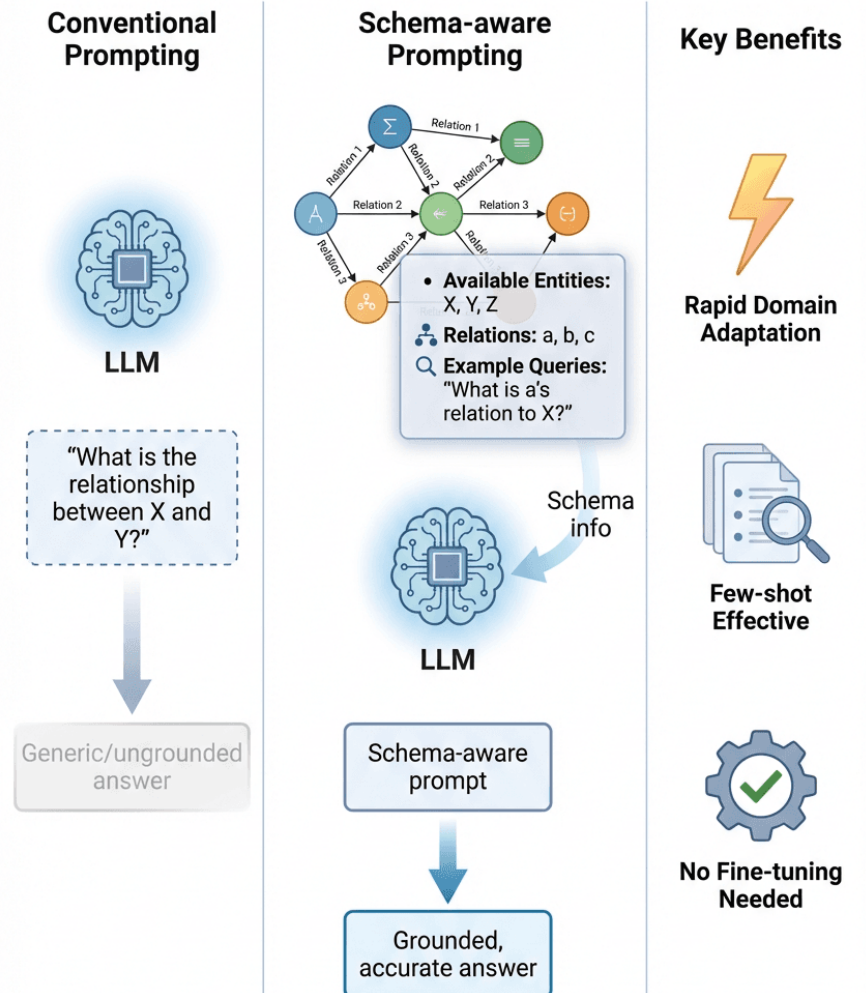
```
(root)  
├── dbo:  
│   ├── birthPlace  
│   ├── birthDate  
│   └── country
```

- **Dynamic masking:** filters candidate tokens at each step based on context and schema using a mask defined on the KG schema

Method	Constr. Level	Control
Grammar-based	Syntax	SPARQL Structure
Trie-based	Schema	Entities & raltions
Dyn. Masking	Token-level	Step-wise validity

# Schema-Aware Prompting

- Schema-aware prompting provides the LLM with explicit information about the **structure of the knowledge graph**:
  - entities,
  - relations,
  - example queries
- By incorporating this information into the prompt, the **model can generate more accurate and grounded outputs** without the need for fine-tuning
- This technique is particularly effective in few-shot settings and enables rapid adaptation to new domains



## In-Context Learning for KGQA

- In-context learning allows LLMs to generalize from a small number of examples **provided in the prompt**
- For KGQA, this typically involves showing pairs of **natural language questions** and corresponding **SPARQL queries**
- The model tries to learn the mapping implicitly and apply it to new questions, reducing the need for large annotated datasets and enabling flexible deployment

## Feedback Loop

- A powerful strategy for improving LLM outputs is to incorporate **execution feedback**
- After generating a SPARQL query, the system executes it and **observes the result**:
  - If the query fails or returns incorrect results, the **model can refine its output** based on error messages or empty results
  - This iterative loop allows the system to **self-correct** and converge toward valid queries.
- Execution feedback transforms KGQA into an **interactive reasoning process** rather than a one-shot prediction task.

# ChatKBQA

- ChatKBQA introduces a new perspective on Knowledge Graph Question Answering by rethinking how logical forms are **generated** and **grounded**
- Traditional systems follow a retrieve-then-generate paradigm:
  - Entities and relations are first retrieved from the graph
  - Use entities and relations to construct a query
- ChatKBQA reverses this process:
  - Generate a logical representation of the question with a fine-tuned LLM
  - Performs entity and relation grounding
- This design separates the problem of **understanding the structure of the question** from the problem of **mapping it to specific graph elements**

# LLM fine-tuning in ChatKBQA

- The training process is supervised and relies on *question–logical form* pairs derived from the target KG
- LLM learns **how to express questions in a formal query representation**
- This means the model is not trained to store facts, but to learn the **mapping between language and graph query structures**, including operations such as joins, filters, and multi-hop relations
  - training on KG facts is **not scalable** and leads to **overfitting** to specific triples, resulting in poor generalization and limit in handling dynamic or unseen knowledge
- LLM becomes specialized in the **schema and query patterns of a specific Knowledge Graph**, improving semantic parsing accuracy but reducing portability across different KGs

# Decoupling Logical Structure and Grounding

- A key innovation in ChatKBQA is the use of **abstract logical forms** (S-expressions) that represent the semantic structure of the question independently of specific entities
- These logical forms:
  - encode compositional operations (e.g., joins, filters, constraints)
  - define the reasoning structure before any grounding decision is made
  - allow multiple candidate entities and relations to be matched later
- This design reduces error propagation by separating:
  - **structural ambiguity** (how the question is composed)
  - **grounding ambiguity** (which entities/relations are correct)
- As a result, the system can explore multiple grounding candidates while preserving a stable logical structure.

# Decoupling Logical Structure and Grounding

*“Which actors starred in movies directed by Christopher Nolan?”*

- Step 1 — Logical Form (Structure Only)
  - The model first generates an abstract representation of the query
  - This structure captures the reasoning pattern:
    - find movies directed by X
    - find actors starring in those movies
- Step 2 — Grounding (Entities and Relations)
  - the system maps labels of the logical form to the KG
  - it uses pretrained text retrieval models

```
JOIN(  
  actor,  
  starred_in(  
    JOIN(  
      movie,  
      directed_by(Christopher Nolan)  
    )  
  )  
)
```

- Christopher Nolan → dbr:Christopher\_Nolan
- directed\_by → dbo:director
- starred\_in → dbo:starring

# Decoupling Logical Structure and Grounding

- Grounding is often ambiguous but not fatal

directed\_by → dbo:director / dbp:director  
starred\_in → dbo:starring / dbo:actor

- Incorrect grounding leads to wrong KG mappings. However, it can be recovered by trying alternative entity or relation matches
  - Execution feedback (empty or incorrect results) can guide correction
- Unlike structural errors, **grounding errors are recoverable**, because the system can iteratively refine entity and relation mappings through multiple attempts

# Strengths and Limitations of ChatKBQA

- ChatKBQA significantly improves the robustness of semantic parsing by **isolating the generation of logical structure from the grounding phase**
- However, the approach still depends heavily on the **accuracy of the generated logical form**.
- This makes ChatKBQA particularly strong when:
  - the question structure closely matches known KG patterns
  - entity and relation mappings are relatively unambiguous
  - the KG is sufficiently complete
- In these cases, decoupling structure from grounding significantly improves robustness.
- However, limitations remain:
  - errors in logical form generation are not recoverable
  - performance degrades when schema coverage is incomplete
  - the approach is still constrained by template-like reasoning structures

# Think-on-Graph

- Think-on-Graph reframes KGQA as a **search problem over the graph**, rather than a query generation task
- Instead of producing a complete logical form, the system **incrementally explores the KG using an LLM** starting from seed entities extracted from the question
- This paradigm shifts the focus from *generating queries* to *navigating the graph in a goal-directed way*

**Question:**  
What is the majority party now in the country where **Canberra** is located?

LLM-only

(Chain-of-Thought Prompt): Let's think step by step.

Response: **Canberra** is the capital of **Australia**. According to my knowledge up to September 2021 the prime minister of Australia is **Scott Morrison**, who is a member of the **Liberal Party**. So the answer should be **Liberal Party**. ❌

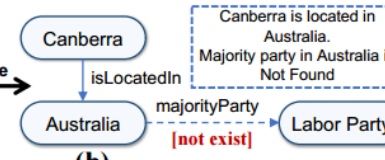
(a)

LLM ⊕ KG

(Prompt): Please generate a SPARQL query for this question.

Response: SELECT ?country ?party WHERE {  
?canberra dbprop:isLocatedIn ?country .  
?country dbprop:majorityParty ?party .

Retrieve



Prompt

Response: Sorry, based on my query result from the knowledge base, I **cannot answer your question** since I do not have enough information. ❌

(b)

LLM ⊗ KG

"Think"

Looking for triples related to **Canberra**

On graph

"Think"

The most relevant one is (**Canberra, capital of, Australia**). Information not enough for answering the question. Looking for triples related to **Australia**

On graph

"Think"

The most relevant one is (**Australia, prime minister, Anthony Albanese**).

LLM-inherent knowledge

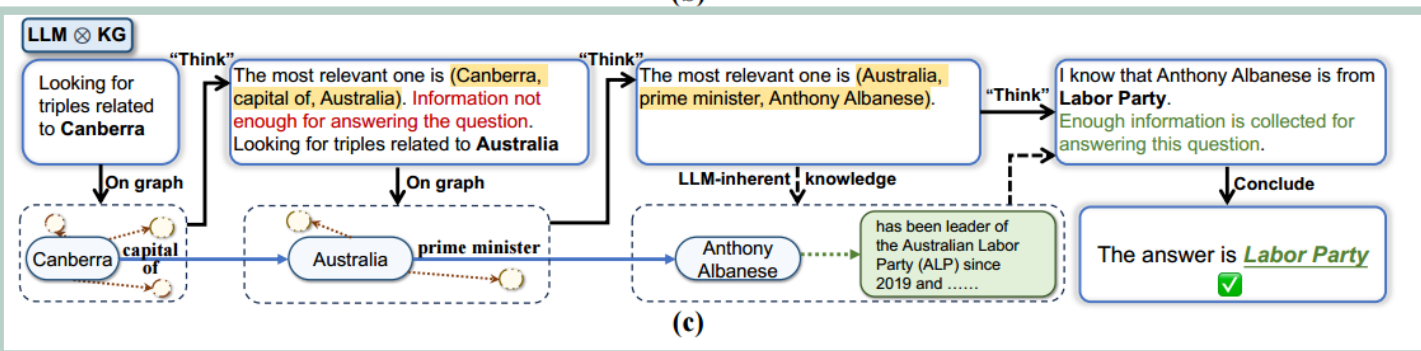
"Think"

I know that Anthony Albanese is from **Labor Party**. Enough information is collected for answering this question.

Conclude

The answer is **Labor Party** ✓

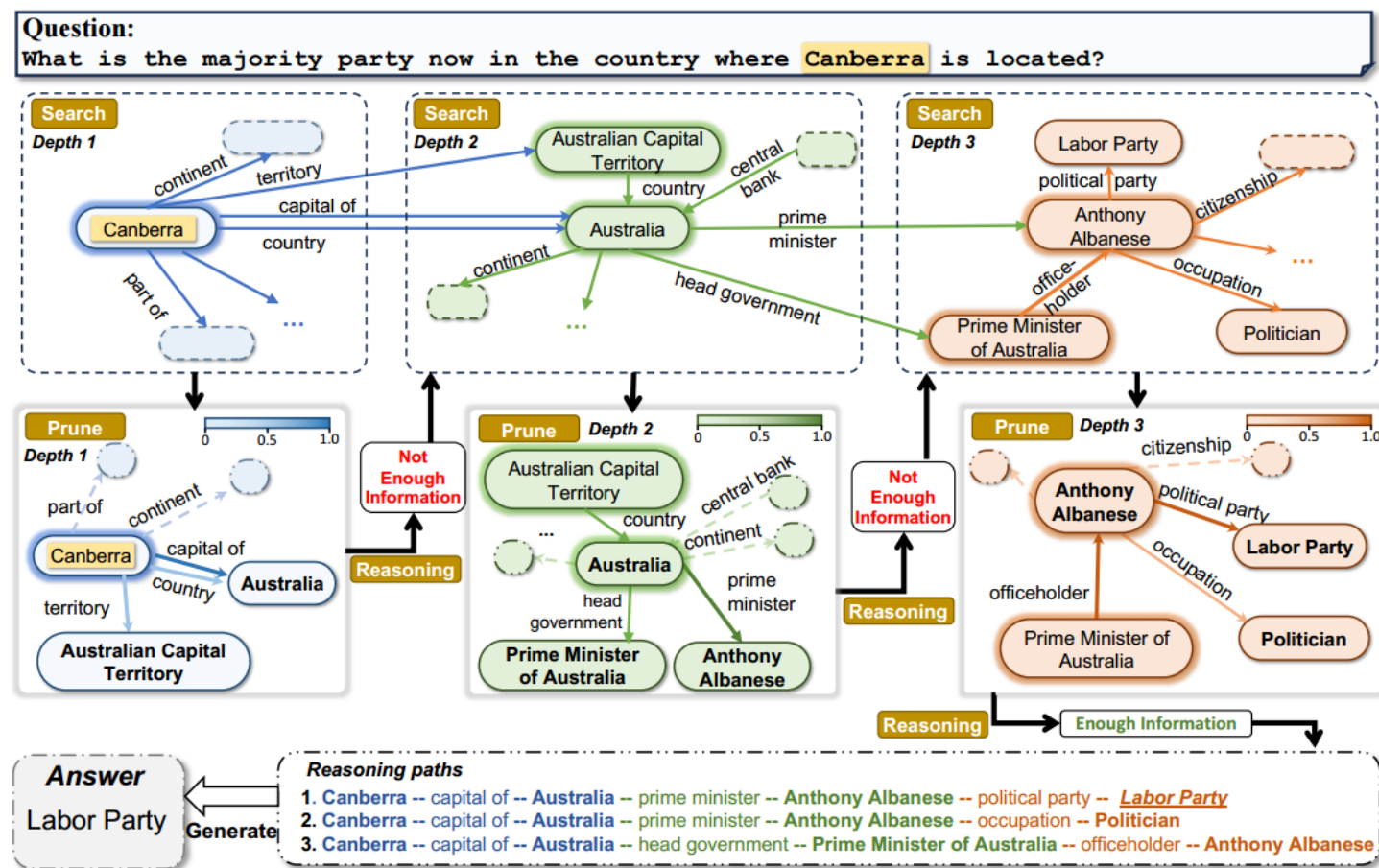
(c)



Sun, Jiashuo, et al. "Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph." *arXiv preprint arXiv:2307.07697* (2023).

# LLM-Guided Graph Traversal

- Starting from seed entities, the system iteratively expands neighbouring nodes and relations to generate multiple candidate paths
- LLM is then used to:
  - evaluate the relevance of each path with respect to the question
  - assign scores to guide the search
  - prune less promising branches
- In this setting, the LLM does not generate answers or queries, but acts as a **reasoning controller that shapes the search process**.



# LLM-Guided Graph Traversal

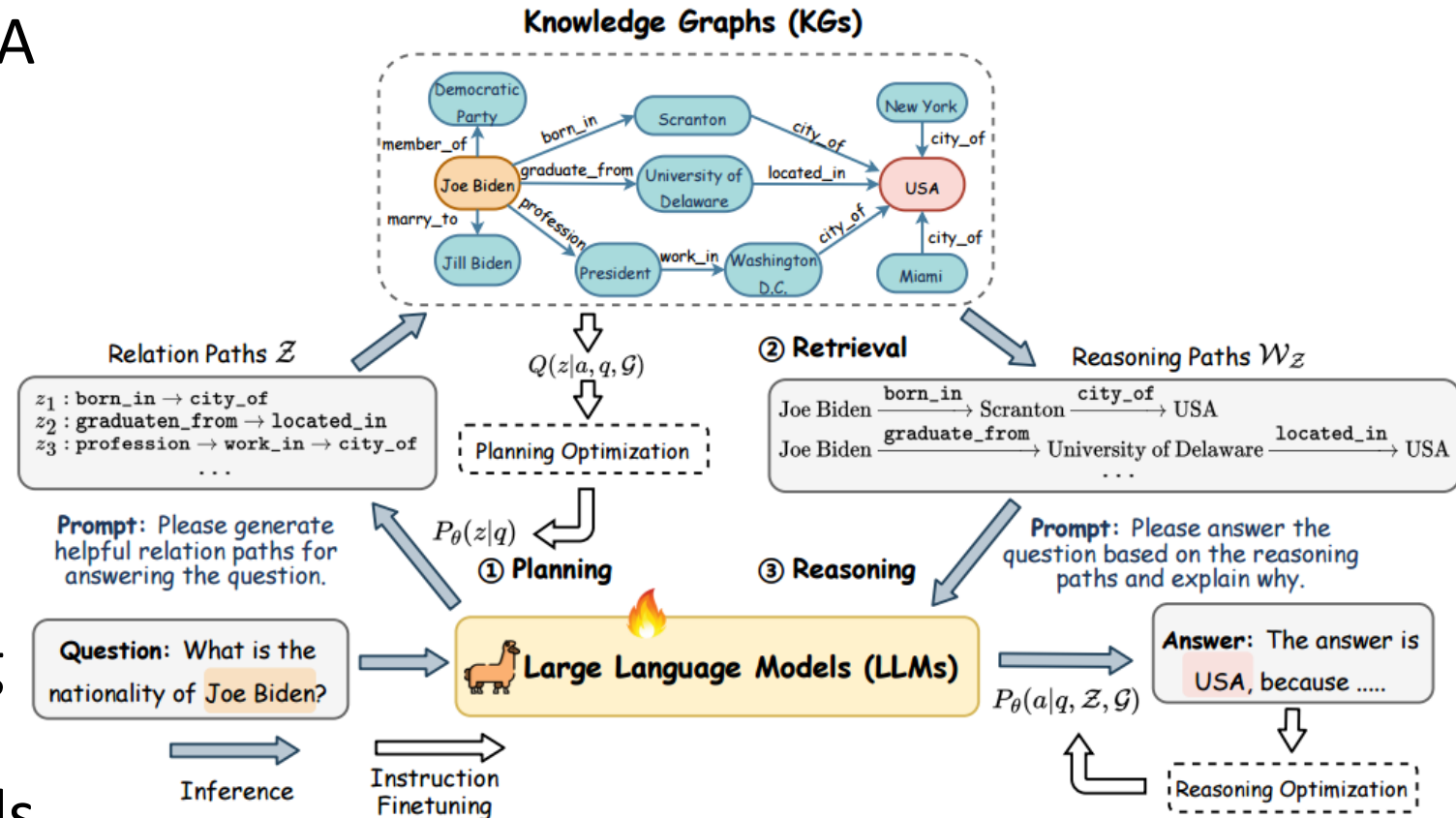
- LLM does not require any **task-specific fine-tuning**
- ToG relies on **iterative prompting** to guide reasoning over the graph
- This makes the framework **training-free and plug-and-play**, allowing it to work with off-the-shelf models such as GPT-4, ChatGPT, or LLaMA-2 without modifying their internal parameters
- KG interaction are handled with **simple external tools** that implements basic formal queries to get neighbours of given entity

## Strengths and Limitations of ToG

- Think-on-Graph is particularly effective for **complex multi-hop questions**, where predefined query structures are difficult to generate
- Its exploratory nature allows it to discover **non-obvious reasoning paths**
- However, this flexibility comes at a cost. The search space can grow rapidly, making the approach **computationally expensive**
- Additionally, ToG provides **partial interpretability** through its exploration traces, but lacks a global structured representation of reasoning, making its outputs harder to summarize and verify

# Reasoning on Graphs

- Reasoning on Graphs introduces a structured approach to KGQA by explicitly **separating reasoning into three stages**:
  - planning: specifies which relations should be followed
  - retrieval: on a search space constrained by the plan and
  - answer generation
- By planning before interacting with the graph, the system reduces uncertainty and avoids unnecessary exploration



# Graph-aware Planning and Sub-graph Retrieval

- In RoG, the reasoning plan is generated by an **LLM**
- The model **internalizes valid graph traversal** patterns through instruction tuning (**fine tuning**) on graph-structured data
  - examples are *question-sequence of KG* relations (i.e. plans)
  - examples are obtained from *question-answer* pairs:
    1. Entity linking with KG
    2. Look for shortest path between question entity and answer entity
- Once a plan is generated, a KG **subgraph** is obtained using a Breadth-First Search (BFS) to find candidate path instances
  - start from the initial entity (e.g., Obama)
  - follow the plan
- Use BFS to support multiple paths

Who is the father of the wife of Barack Obama?



Barack Obama → hasWife → Michelle Obama  
Michelle Obama → hasFather → F.C. Robinson



hasWife → hasFather

Question: *Which actors starred in movies directed by Nolan*

Plan: directed → starring

1-hop:

- Nolan → directed → Inception
- Nolan → directed → Interstellar
- Nolan → directed → Dunkirk
- ...

2-hop

- Inception → starring → DiCaprio
- Inception → starring → Gordon-Levitt
- Inception → starring → Page
- Interstellar → starring → McConaughey
- Interstellar → starring → Hathaway
- ...

# Reasoning

- KG retrieval can often include **noisy or irrelevant paths**
- **RoG reasoner** is carried out by LLM that is fine-tuned on examples including *question*, the *relation path*, and the *correct answer*
- RoG's reasoning steps:
  1. Paths linearization: (Nolan → directed → Inception) → "*Nolan directed Inception*"
  2. LLM prompting passing question and paths
  3. LLM does (a) selection of relevant paths; (b) aggregation of information; (c) answer generation

# Used prompt

## Planning Prompt Template

Please generate a valid relation path that can be helpful for answering the following question:  
<Question>

## Reasoning Prompt Template

Based on the reasoning paths, please answer the given question. Please keep the answer as simple as possible and return all the possible answers as a list.

Reasoning Paths:

<Reasoning Paths>

Question:

<Question>

# Interpretability and Plug-and-Play Flexibility

- RoG is interpretable because it produces explicit relation plans and grounded KG paths that can be inspected and verified as reasoning evidence
- The relation plans generated by RoG are **model-agnostic representations** (yet tied to specific KG schema) that can be used by different reasoning modules at inference time
- While training is resource-intensive (taking 38 hours on 2 A100-80G GPUs), it significantly boosts performance on complex 3-hop and 4-hop questions where unconstrained zero-shot models often fail

## MindMap

- MindMap takes a different approach by focusing on how graph information is presented to the language model
- Instead of navigating the graph or generating queries, it **encodes graph structure directly into the prompt**, enabling the model to reason over it internally
- This approach leverages the LLM's ability to process structured context and perform multi-step reasoning within its own latent space

Wen, Y., Wang, Z., & Sun, J. (2024, August). Mindmap: Knowledge graph prompting sparks graph of thoughts in large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*

# From Chain-of-Thought to Graph-of-Thought

- MindMap extends the idea of Chain-of-Thought prompting by introducing a **graph-structured reasoning process**
  - Several subgraphs are extracted from the KG
  - LLM is used to consolidate them in a reasoning graph
- At the same time, the model can integrate this structured information with its implicit knowledge
  - robust against KG errors or incompleteness
- The system produce a mental map of the inference process:
  - show logic path (e.g., involved triples)
  - presented as decision tree

# From Chain-of-Thought to Graph-of-Thought

- Evidence Graph Mining
  - Question entity identification and KG grounding (semantic similarity)
  - Subgraph exploration: path-based (path linking identified entities) and neighbour-based (including neighbour nodes)
  - Pruning of redundant information (clustering and sampling)
- Evidence Graph Aggregation
  - Natural language conversion of each subgraph
  - Disambiguation to merge semantically similar entities

*“Who is the president of the country where Tesla was founded?”*

Tesla → founded\_in → USA  
USA → president → Biden

Tesla → headquarters → Texas  
Texas → governor → Abbott

Tesla → CEO → Elon Musk

Subgraphs

Tesla:

- founded\_in → USA
- headquarters → Texas
- CEO → Elon Musk

USA:

- president → Biden

Texas:

- governor → Abbott

Aggregation

# From Chain-of-Thought to Graph-of-Thought

- LLM Reasoning on the Mind Map
  - Integration of structured knowledge (from KG) with LLM internal knowledge
  - Mind map generation, including final answer, inference process, and decision tree

```
SystemMessage(content= """You are an excellent AI doctor, and you can diagnose diseases and recommend medications based on the symptoms in the conversation."""),
HumanMessage(content="""Patient input: """ + Question),
AIMessage(content=f """Combine the knowledge you already have, you have some extra medical knowledge information in the following:\n\n ### """ + path_reasoning_graph + """\n\n### """ + neighbor_reasoning_path),
HumanMessage(content= """What disease does the patient have? What tests should patient take to confirm the diagnosis? What recommended medications can cure the disease? Think step by step.\n\n\n
Output1: The answer includes disease and tests and recommended medications.\n\n
Output2: Show me inference process as a string about extract what knowledge from which Path-based Evidence or Neighbor-based Evidence, and in the end infer what result. \n Transport the inference process into the
following format:\n Path-based Evidence number('entity name'-'>'relation name'-'>...)->Path-based Evidence number('entity name'-'>'relation name'-'>...)->Neighbor-based Evidence number('entity name'-'>'relation name'-'>...)-
>Neighbor-based Evidence number('entity name'-'>'relation name'-'>...)->result number('entity name')->Path-based Evidence number('entity name'-'>'relation name'-'>...)->Neighbor-based Evidence number('entity name'-'>'relation
name'-'>...). \n\n
Output3: Draw a decision tree. The entity or relation in single quotes in the inference process is added as a node with the source of evidence, which is followed by the entity in parentheses.\n\n
There is a sample:\n ... """)
```

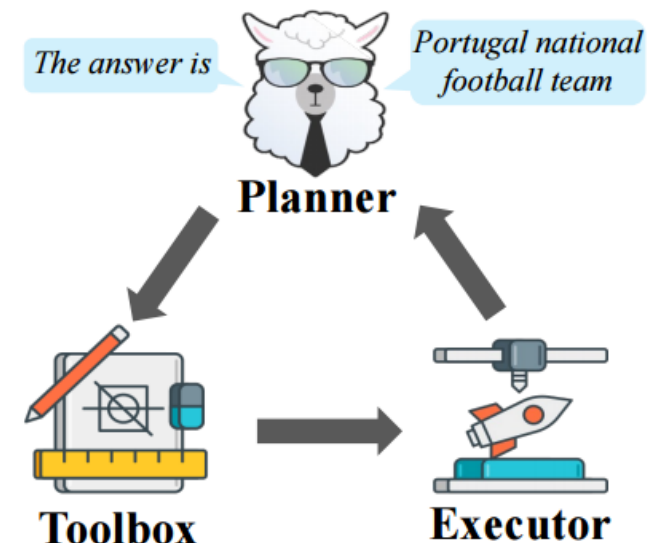
# Strengths and Limitations of MindMap

- MindMap is particularly effective in scenarios where a **global understanding of the graph is required**, such as in complex or specialized domains.
- It also provides natural language explanations that improve interpretability.
- However, its performance depends heavily on **prompt design**, and it **does not guarantee strict adherence** to the underlying graph structure

# KG-Agent

- KG-Agent is a recent framework that reframes KGQA as an **agentic decision-making process** rather than static query generation
- Instead of producing a full query in a single step, a relatively small language model is **trained** to act as an **autonomous agent** that incrementally interacts with the KG
- The key idea is to move from rigid pipelines (e.g., SPARQL generation) to **dynamic reasoning through tool use and iterative decisions**
- This allows the model to:
  - actively explore the graph
  - adapt its strategy during reasoning
  - handle complex multi-hop questions without predefined templates

Which sports team for which **Cristiano Ronaldo** played in 2011 was founded last ?



# System Architecture

- KG-Agent is built around four tightly coupled components

## LLM-based Planner

A fine-tuned language model that decides:

- which action to take next
- which tool to invoke based on the current reasoning state

## Multifunctional Toolbox

A set of structured operations that define how the agent can interact with the KG

## KG-based Executor

A backend module that executes tool calls on the KG and returns results

## Knowledge Memory

A dynamic memory that stores:

- intermediate results
- retrieved entities
- reasoning history

- This modular design separates **reasoning, execution, and memory**, enabling flexible and interpretable workflows.

# Tool-Based Interface for Graph Interaction

- The agent does not directly generate queries.
- It interacts with the KG through a **set of tools**, which define the action space.

## Retrieval Tools

Used to explore the KG structure (e.g., retrieve relations or neighbouring entities)

## Logical Tools

Used to manipulate intermediate results

- filtering
- counting
- intersection

## Semantic Tools

Used to handle ambiguity and lexical gaps

- entity disambiguation
- relation matching via pretrained models

# Instruction Tuning with Reasoning Programs

- KG-Agent is trained using **instruction tuning over structured reasoning traces**, rather than plain text outputs
- From existing KGQA datasets, the authors generate:
  - sequences of **tool calls**
  - corresponding to valid reasoning paths in the KG
- The model learns to:
  - map a question to a sequence of actions
  - interact with the KG step by step
- ~10K training examples are sufficient to achieve strong performance allowing the model to learn how to use the KG, not to memorize its content

Question: Who is the father of the wife of Barack Obama?

Thought: I need to find the spouse of Barack Obama

Action: `get_tail_entity(Barack Obama, spouse)`

Observation: Michelle Obama

Thought: Now I need to find the father of Michelle Obama

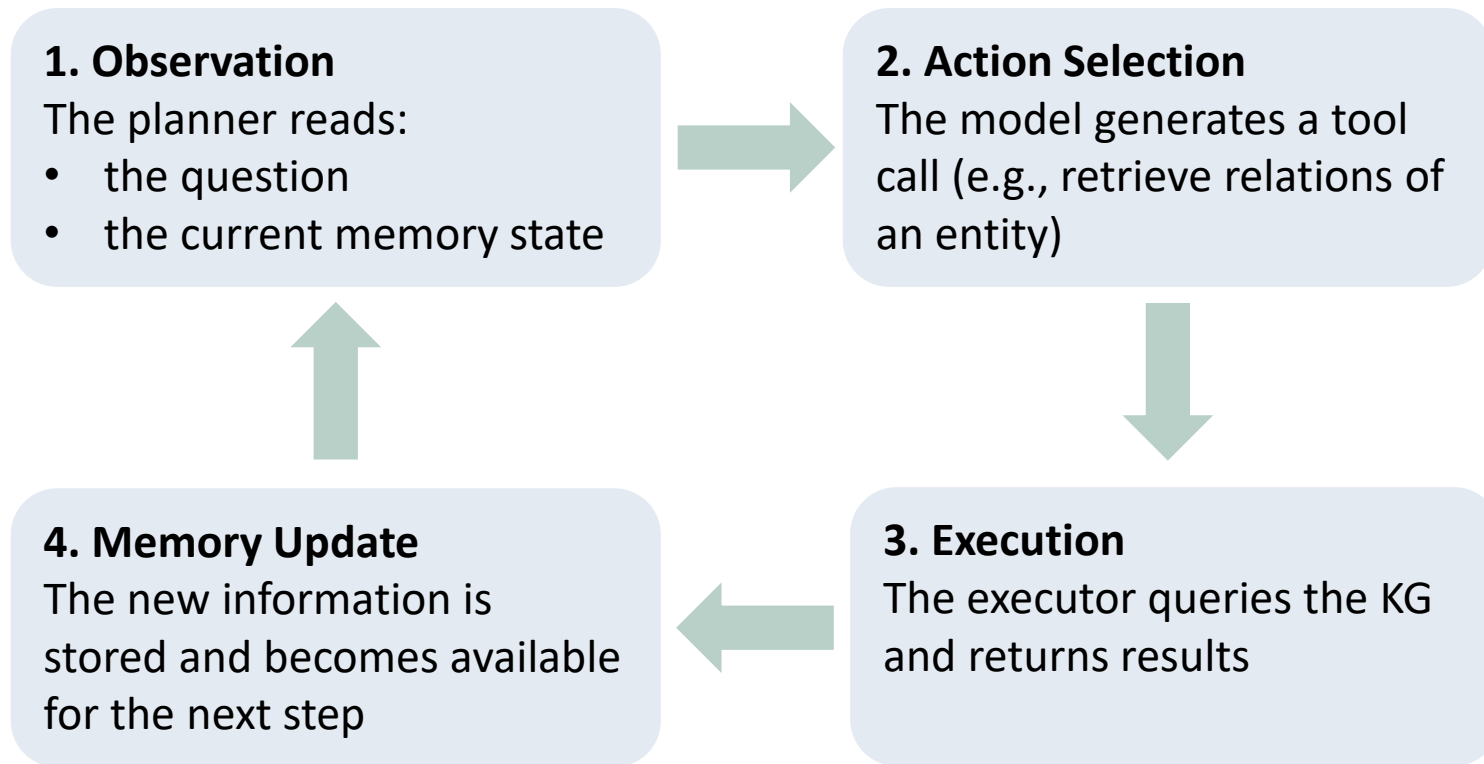
Action: `get_tail_entity(Michelle Obama, father)`

Observation: Fraser Robinson

Final Answer: Fraser Robinson

# Reasoning as an Action–Observation Cycle

- KG-Agent operates through an iterative loop resembling agentic frameworks:



- This process enables:
- adaptive multi-hop reasoning
  - recovery from intermediate errors
  - dynamic exploration of the KG

# Efficiency and Scalability of Agentic KGQA

- KG-Agent demonstrates strong performance on complex benchmarks and achieves results comparable to or better than larger models
- **Efficiency advantages:**
  - faster inference compared to multi-call prompting frameworks
  - reduced memory footprint by retrieving only local subgraphs
- **Scalability:**
  - supports very large KGs (e.g., Freebase-scale)
  - avoids loading the full graph into memory
- **Generalization:**
  - strong transfer across domains
  - enabled by tool-based reasoning rather than schema memorization

# Summary

Framework	Core Paradigm	Learning Setting	Key Innovation / Feature	Backbone Examples
<b>ChatKBQA</b>	Generate-then-Retrieve	Fine-tuning (logical form supervision)	Generates a structured logical form (e.g., S-expression) first, then grounds entities and relations via retrieval	LLaMA-2, ChatGLM, Baichuan
<b>Think-on-Graph (ToG)</b>	Iterative Graph Exploration	Training-free (prompt-based reasoning)	Uses LLM-guided beam search over KG paths with step-by-step pruning and expansion decisions	GPT-4, ChatGPT, LLaMA-2-70B
<b>Reasoning on Graphs (RoG)</b>	Planning → Retrieval → Reasoning	Fine-tuning (relation plan + reasoning supervision)	Predicts schema-level relation plans first, then retrieves KG paths consistent with the plan for faithful reasoning	LLaMA2-Chat-7B
<b>MindMap</b>	Graph-of-Thought Prompting	Training-free (prompt engineering)	Merges multiple retrieved paths and neighbourhoods into a structured “graph prompt” for LLM-based synthesis	GPT-3.5, GPT-4
<b>KG-Agent</b>	Autonomous Tool-Using Agent	Fine-tuning (tool-use / function-call trajectories)	LLM learns to interact with KG via tools (retrieve, filter, reason) in an iterative action–observation loop with memory	LLaMA2-7B