Università di Genova
DIBRIS DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI

Course S&DAI

From natural language sentences to Prolog terms via LLMs

LEFEVRE Titouan (Michel Jean-Claude)

Università di Genova
DIBRIS DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI

Standard Project

Max points: 12.5

Università di Genova
DIBRIS DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI

This project presents a neuro-symbolic pipeline that converts natural language sentences into valid Prolog facts and rules

Context and Motivation

Why?

- LLMs are fluent and creative but probabilistic. They suffer from hallucinations and lack logical consistency.
- Logic Programming (as Prolog) is deterministic and verifiable but requires strict, formal syntax that is hard for non-experts to use.
- **Combine strengths :**
  - use the LLM as a "translator"
  - Use Prolog as the "reasoning engine"

Project Objectives

What does this system do?

- **Extract:** Converts Natural Language (e.g., "John loves Mary") into Prolog formulas (e.g., loves(john, mary).)
- **Validate:** Uses a Python-based syntax validator to ensure code is executable
- **Evaluate:** Benchmarks semantic accuracy using an "LLM-as-a-Judge" approach
- **(bonus) Interface :** Provides a web-based GUI (Gradio) for real-time interaction

State of the Art

ChatBDI Framework

Goal: Transforming strict logical multi-agent systems (MAS) into conversational agents without modifying source code

How?

- **Contextual Retrieval (Grounding):** The system first queries the agent for its *current* beliefs and available actions to restrict the scope
- **Vector Similarity (Embeddings):**
  - Calculates the embeddings of the user's sentence.
  - Compares it to the agent's known concepts using Cosine Similarity to find the best match.
  - *Safety:* If no match is close enough, the request is rejected.
- **Generative Translation:** The LLM constructs the final logical format using the identified concept

System Implementation

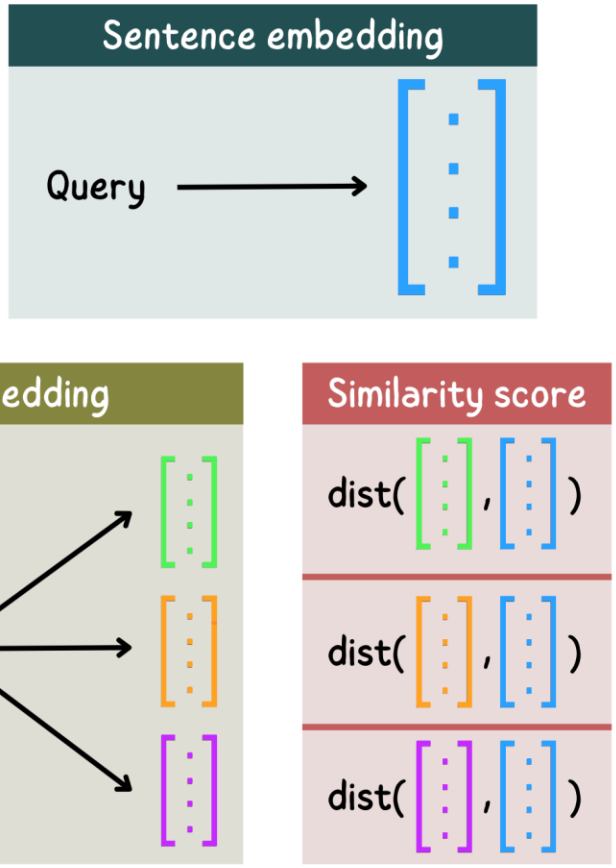
Architecture

Goal: : Translate natural language sentences to Prolog terms via LLMs

- **LLM Engine:** Llama 3.1:8b (4.9GB) running locally via Ollama
- **Logic Engine:** SWI-Prolog interfaced via PySWIP library
- **Embeddings:** all-MiniLM-L6-v2 (SentenceTransformer)
  - derived from BERT-like architectures

System Implementation

Contextual Anchoring



Before generating code, the system searches the Knowledge Base (KB) for relevant predicates using Vector Similarity

System Implementation

Contextual Anchoring issues

```
Input: "John loves Mary."
Predicates: ['parent', 'man', 'loves', 'mortal']

# Semantic Search Results:
# 1. 'loves'       -> Score: 0.274
# 2. 'parent_of'   -> Score: 0.149
# 3. 'mortal'      -> Score: 0.116
```

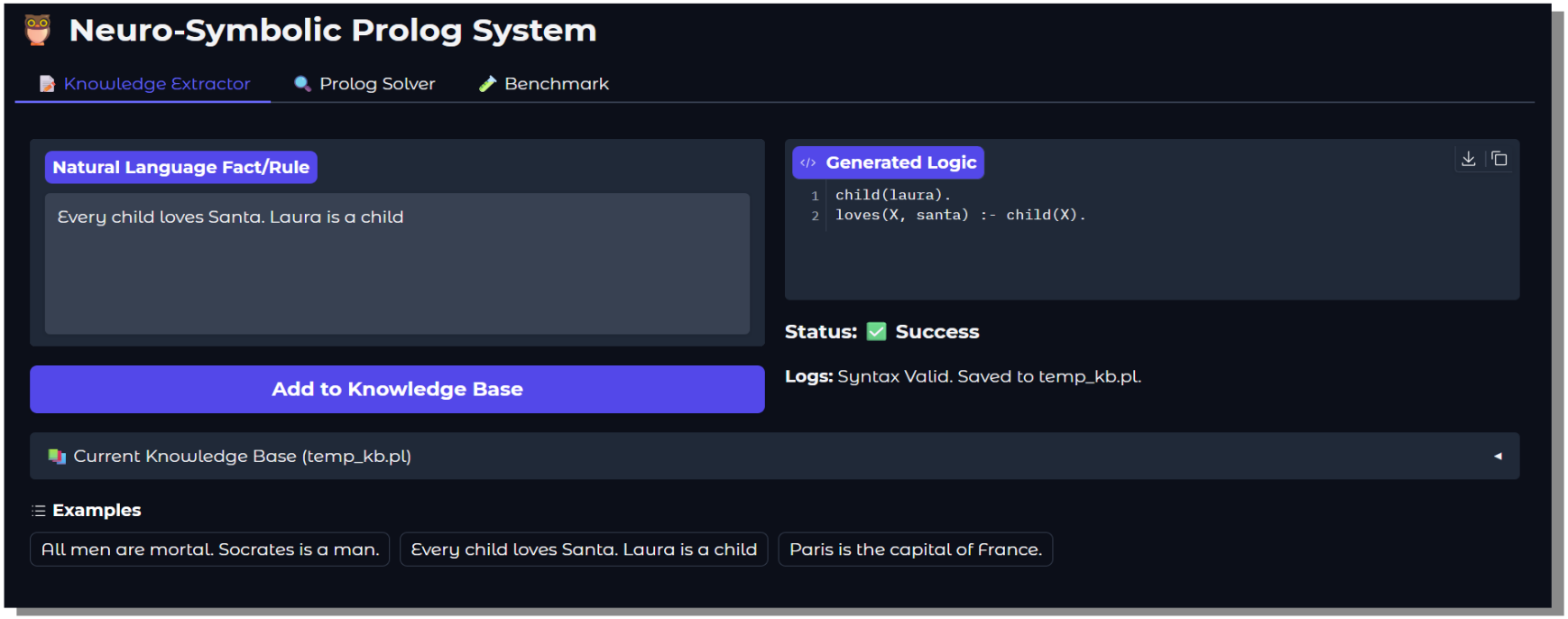
- Input vectors represent full thoughts ("John loves Mary")
- Predicate vectors represent single concepts ("loves")
- **Result:** Low similarity scores (e.g., 0.274 for loves matching the sentence)

Adjusted thresholds to capture correct predicates despite low confidence.

System Implementation

Extraction

Use this tab to populate the knowledge base



System Implementation

Extraction prompt

Base prompt :

```
base_prompt = f"""
You are an expert in Prolog Logic. Your task is to translate Natural Language into syntactically correct Prolog code.
{hint}
"""
```

Hint :

```
hint = f"""
CONTEXT: The Knowledge Base already contains these predicates: {formatted_preds}.
Prioritize using them to maintain consistency.
"""
```

```
Input: "John loves Mary."
Predicates: ['parent', 'man', 'loves', 'mortal']

# Semantic Search Results:
# 1. 'loves'       -> Score: 0.274
# 2. 'parent_of'   -> Score: 0.149
# 3. 'mortal'      -> Score: 0.116
```

----- > retained predicate for the prompt

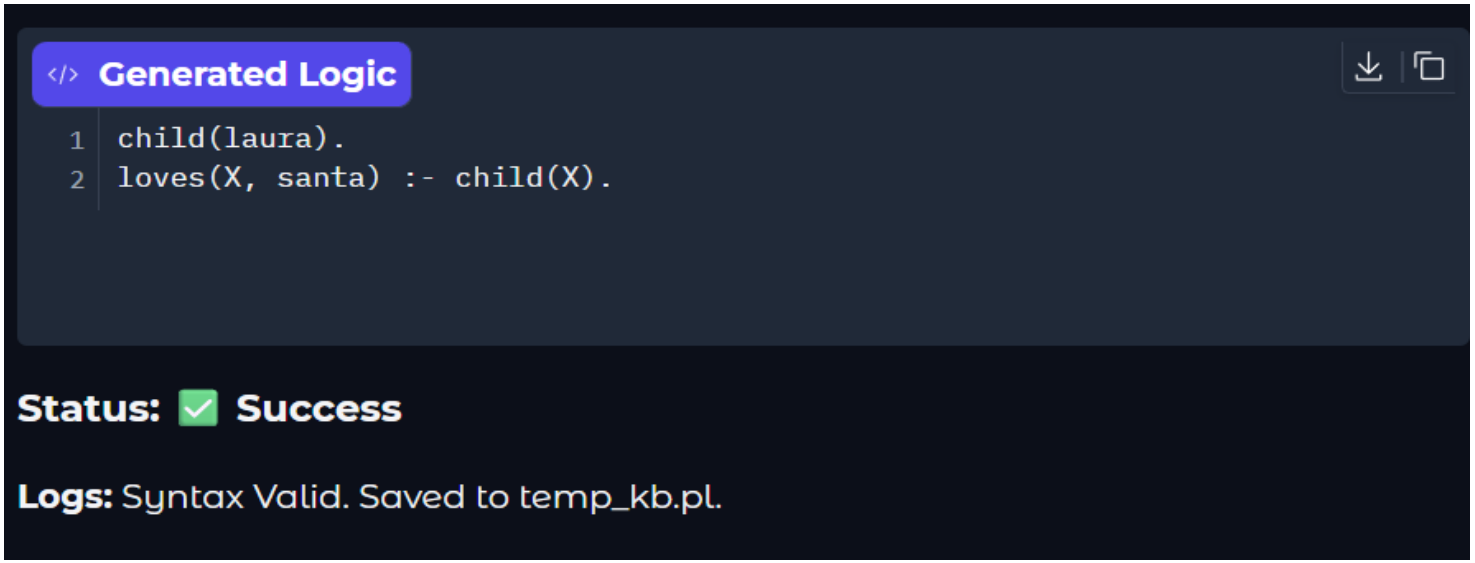
To ensure valid syntax, the prompt also enforces strict Prolog rules.

System Implementation

Validator

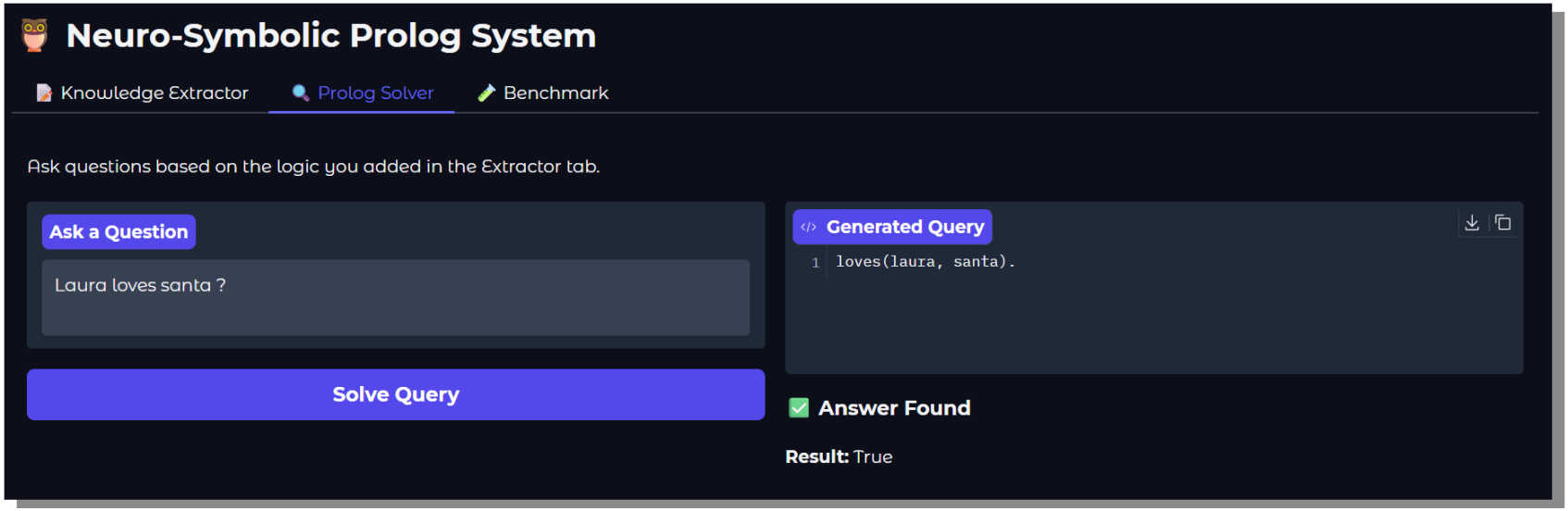
Intercepts LLM code before saving. It attempts to load it into a temporary Prolog session.

- **Success:** Code is saved to temp_kb.pl.
- **Failure:** Syntax error returned to UI (prevents corrupting the KB).



System Implementation

Querying



- The solver cleans input and formats NL query in Prolog.
- The solver runs the query using the knowledge from the **temp_kb.pl** file.

Benchmark

LLM-as-a-Judge

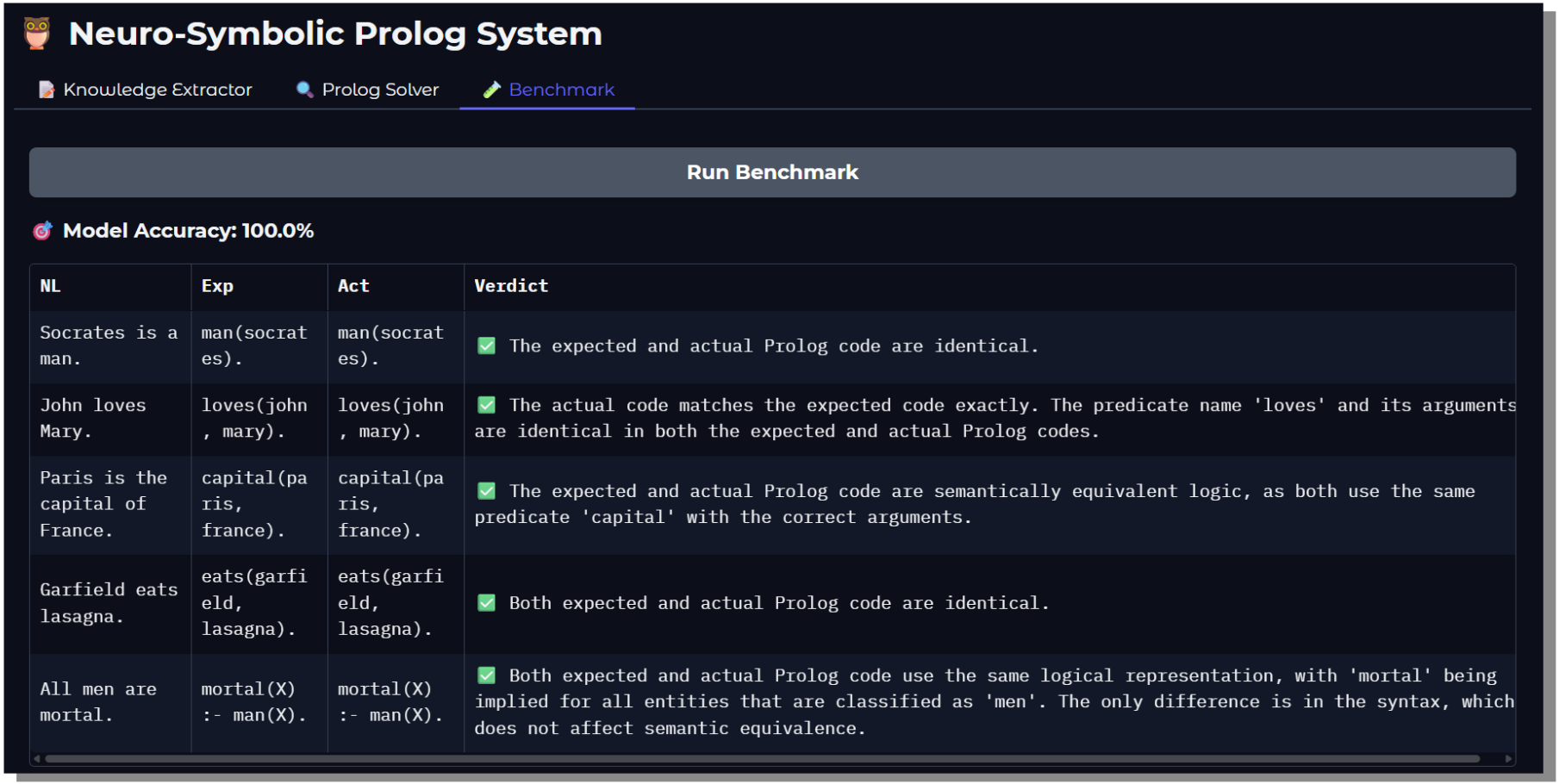
The benchmark engine compares the **Generated Code** against **Expected Code** using an LLM to judge semantic equivalence.

Model used to judge : Llama 3.1:8b (4.9GB) running locally via Ollama

```
TEST_DATASET = [
    {"nl": "Socrates is a man.", "expected": "man(socrates)."},
    {"nl": "John loves Mary.", "expected": "loves(john, mary)."},
    {"nl": "Paris is the capital of France.", "expected": "capital(paris, france)."},
    {"nl": "Garfield eats lasagna.", "expected": "eats(garfield, lasagna)."},
    {"nl": "All men are mortal.", "expected": "mortal(X) :- man(X)."},
    {"nl": "Every child loves Santa.", "expected": "loves(X, santa) :- child(X)."},
    {"nl": "Whales are mammals.", "expected": "mammal(X) :- whale(X)."},
    {"nl": "X is grandparent of Z if X is parent of Y and Y is parent of Z.",
     "expected": "grandparent(X, Z) :- parent(X, Y), parent(Y, Z)."},
    {"nl": "X is a mother of Y if X is parent of Y and X is female.",
     "expected": "mother(X, Y) :- parent(X, Y), female(X)."},
    {"nl": "X is a sibling of Y if Z is parent of X and Z is parent of Y.",
     "expected": "sibling(X, Y) :- parent(Z, X), parent(Z, Y)."}
]
```

Benchmark

Results



NL	Exp	Act	Verdict
Socrates is a man.	man(socrates).	man(socrates).	✅ The expected and actual Prolog code are identical.
John loves Mary.	loves(john, mary).	loves(john, mary).	✅ The actual code matches the expected code exactly. The predicate name 'loves' and its arguments are identical in both the expected and actual Prolog codes.
Paris is the capital of France.	capital(paris, france).	capital(paris, france).	✅ The expected and actual Prolog code are semantically equivalent logic, as both use the same predicate 'capital' with the correct arguments.
Garfield eats lasagna.	eats(garfield, lasagna).	eats(garfield, lasagna).	✅ Both expected and actual Prolog code are identical.
All men are mortal.	mortal(X) :- man(X).	mortal(X) :- man(X).	✅ Both expected and actual Prolog code use the same logical representation, with 'mortal' being implied for all entities that are classified as 'men'. The only difference is in the syntax, which does not affect semantic equivalence.

Issue : I update my prompt to enhance results

Limitations & Future Work

Moving toward Agentic Workflows

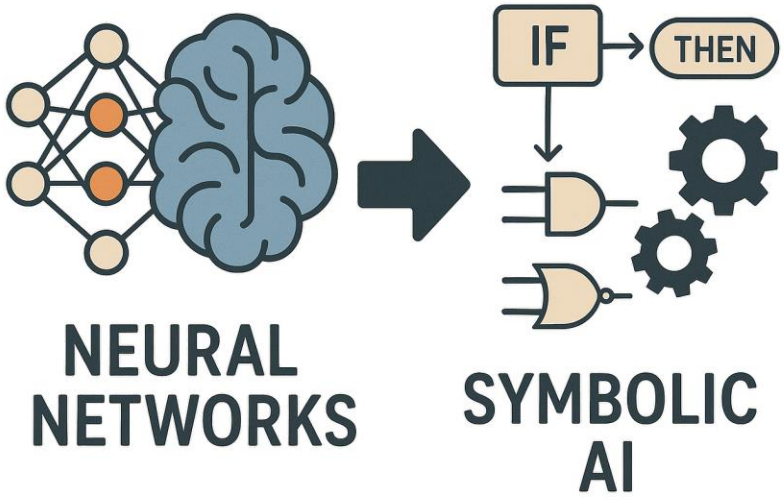
First Limitation: Embedding mismatch (Sentence vs. Word)

- **Proposed Solution:** Use an LLM to analyze the sentence and extract the relevant words

Second Limitation: LLM fail to translate

- **Proposed Solution:** Use an bigger model

Conclusion



- Demonstrated that **Hybrid Guidance** (LLM + Embeddings) is required to keep predicates consistent
- The system allows users to interact with a logic engine using natural language

References

[1] Andrea Gatti, Viviana Mascardi, and Angelo Ferrando. Let me talk to you! natural language interaction between humans and bdi agents via chatbdi. 2025.

[2] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bertnetworks. In Proceedings of the 2019 C

UniGe

DIBRIS