Enhancing Large Language Models with Neurosymbolic Reasoning for Multilingual Tasks

Sina Bagheri Nezhad, Ameeta Agrawal
Portland State University, USA

NeSy 2025

Introduction

- LLMs excel in several NLP tasks but struggle with long-context, cross-lingual reasoning.
- Information is often scattered across languages and lengthy documents.
- Retrieval-Augmented Generation (RAG) helps but fails in multi-target reasoning.

Key Contributions:

- Propose NSAR framework – a neurosymbolic method that merges symbolic reasoning with neural inference for verifiable, interpretable pathways.
- Develop NSAR prompt for fact extraction and Python code generation.
- Combine with CoT, ReAct, Self-Reflection for additional gains.
- Experiments show NSAR outperforms retrieval-based and neural-only methods in cross-lingual, long-context question answering tasks.

Experiments

Dataset

- Extended mLongRR dataset: Up to 512k words (really long context).
- 7 languages (English, Vietnamese, Swahili, Persian, Russian, Hindi, Arabic).
- Query in English; context in target language.
- Contexts: News articles of 2k-512k words; needles in haystack language.
- 3 "needles" (e.g., "The special magic {city} number is: {number}")

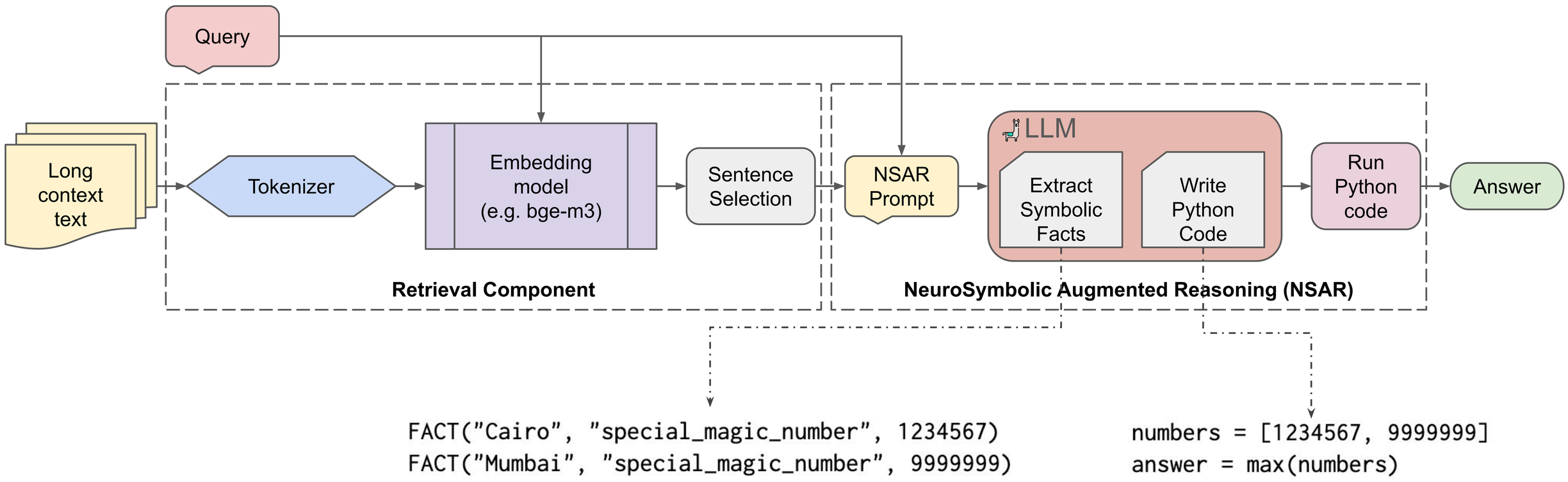
Evaluation

- Query "What is the largest special magic number?"
- Retrieve and reason over multiple needles.

Models & Baselines

- Models: GPT-4o-mini, Llama 3.2 (90B).
- Baselines: RAG-Vanilla, CoT, ReAct, Self-Reflection.
- Hybrid: NSAR+3 (NSAR + CoT + ReAct + Self-Reflection).

NeuroSymbolic Augmented Reasoning (NSAR)



Retrieval Component

- Tokenize long, multilingual context into sentences using Punkt tokenizer.
- Embed with *bge-m3 embedding* (multilingual, 1024-dim).
- Compute semantic distance to query; select top-k sentences (k=3,5,10,20,50).

NeuroSymbolic Augmented Reasoning (NSAR)

- Addresses LLM limitations in reliability and interpretability for multi-target tasks.
- NSAR Prompt guides 3 stages:
  i. Symbolic Fact Extraction: e.g., FACT("Cairo", "special_magic_number", 1234567)
  ii. Python Code Generation: e.g., numbers = [1234567, 9999999]; answer = max(numbers)
  iii. Final Answer Extraction: Execute code for deterministic output.

Sample output

1. Extracting relevant facts from the context:

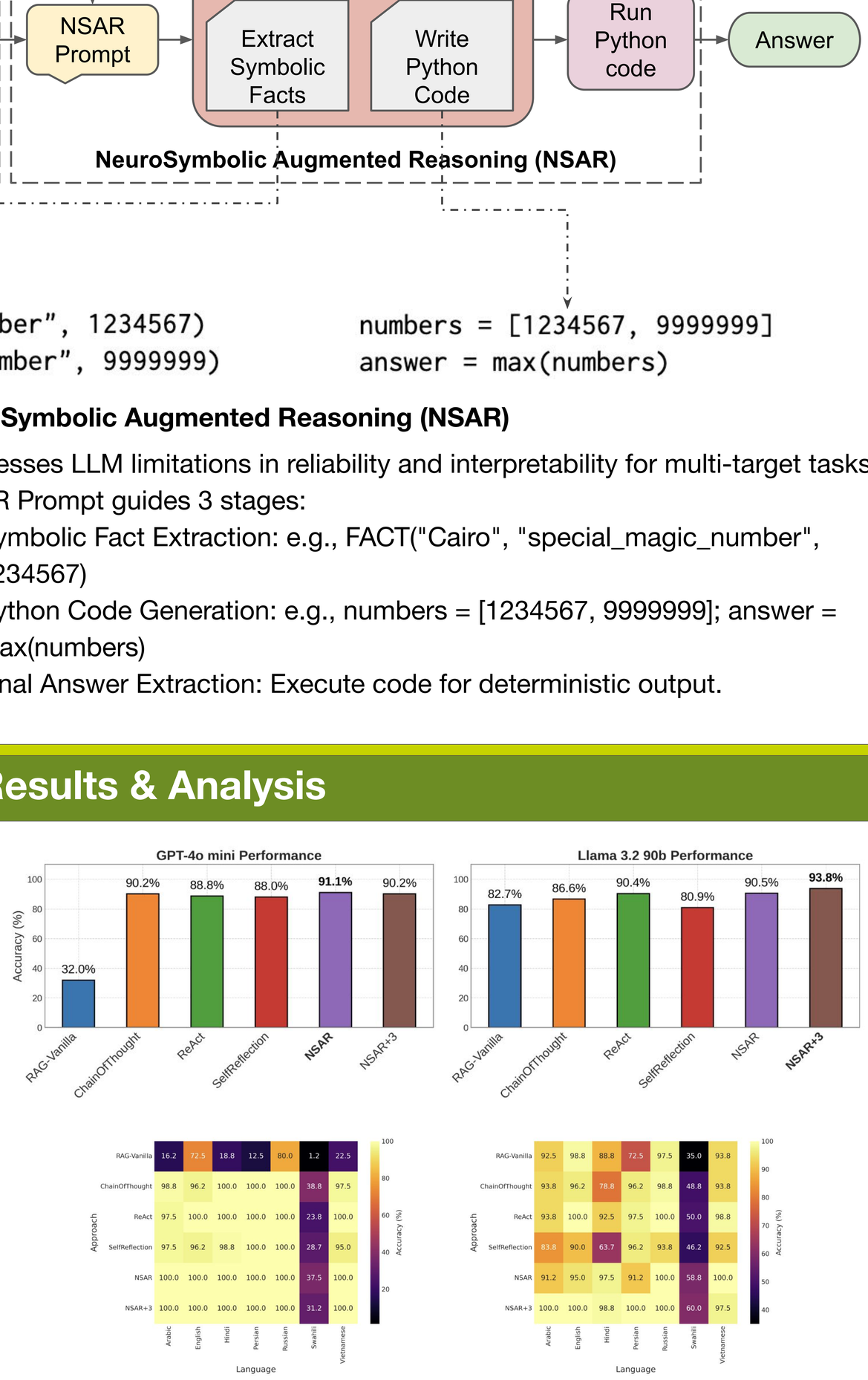
```
FACT(Barcelona, magic_number, 3261366)
FACT(Damascus, magic_number, 3745066)
FACT(Sofia, magic_number, 9396103)
```

2. Generating executable Python code:

```python
# Define the magic numbers as a dictionary
magic_numbers = { "Barcelona": 3261366,
                  "Damascus": 3745066,
                  "Sofia": 9396103 }
# Find the largest magic number
largest_magic_number = max(magic_numbers.values())
largest_magic_number
```

Python Interpreter → 9396103

Results & Analysis



(a) GPT-4o mini. (b) Llama 3.2 90b.