Leveraging Declarative Frameworks for Explainable Neuro-Symbolic AI and AI agents

Preliminary draft for Independent Study (Rohith)

Leveraging Declarative Frameworks for Explainable Neuro-Symbolic AI and AI agents

The growing need for transparency and accountability in artificial intelligence has made Explainable AI (XAI) a critical area of research, especially as AI systems increasingly influence high-stakes decisions. While deep learning excels at pattern recognition, it often operates as a "black box," limiting interpretability. Declarative frameworks like Logica, which leverage rule-based logical reasoning, offer a promising solution for enhancing transparency. Similarly, Neuro-Symbolic AI combines neural networks' learning capabilities with symbolic reasoning's interpretability, creating hybrid systems that balance accuracy and explainability. When integrated with AI agents autonomous systems become capable of perceiving, reasoning, and acting in dynamic environments these technologies can enable intelligent systems that provide clear, human-understandable explanations for their decisions and actions.

This study proposes to explore how declarative frameworks such as Logica can enhance explainability in neuro-symbolic AI systems and AI agents. By embedding declarative logic into hybrid architectures, the research will aim at developing a prototype system that integrates rule-based reasoning with neural networks to address real-world use cases like healthcare diagnostics or autonomous traffic management. The study will also investigate how declarative reasoning can improve the transparency of AI agents' decision-making processes in dynamic environments, enabling them to explain not only *what* decisions were made but also *why*. The system's performance will be evaluated using XAI metrics such as interpretability, transparency, and user trust. This research aims to advance responsible AI by creating systems that are both powerful and inherently explainable, bridging the gap between technical innovation and ethical accountability.

Part 1: Formal Verification of LLM Output Consistency

Summary

The goal of this part of the study is to develop a comprehensive framework for formally verifying the logical consistency of Large Language Model (LLM) outputs through neuro-symbolic integration, automated theorem proving, and counterexample-guided refinement. Building on recent breakthroughs in formal methods and LLM interpretability, this study aims to present a novel architecture that achieves a large verification coverage while maintaining low latency overhead. The research addresses critical gaps in AI safety for high-stakes domains like healthcare, legal analysis, and autonomous systems.

Problem Statement and Motivation

The LLM Consistency Challenge

Modern LLMs exhibit three critical failure modes:

1. **Logical Contradictions**: 38% of GPT-4 outputs contain conflicting statements in multi-step reasoning tasks (*according to FVEL, 2024) [1]*

2. **Factual Drift**: 24% accuracy drop in clinical QA systems after 3 months of continuous fine-tuning (*according to - CheckEmbed, 2024) [2]*

3. **Contextual Fragility**: 61% variance in legal document analysis based on prompt phrasing *(according to PVG, 2023) [3]*

Traditional statistical validation methods fail to provide mathematical guarantees of correctness, necessitating formal verification approaches.

System Architecture

The proposed system architecture integrates neural network-based language understanding with formal symbolic reasoning to ensure logical consistency in the outputs of large language models. This hybrid architecture is designed to validate, correct, and refine LLM outputs in real time.

1. Neural-Symbolic Interface Layer

The neural-symbolic interface serves as the bridge between the continuous, high-dimensional representations generated by LLMs and the discrete, rule-based frameworks used in formal verification. This layer operates in two stages:

1. **Attention-Based Predicate Extraction**
    - The system analyzes attention weights from the transformer layers of the LLM to identify relationships between tokens. For example, in a medical diagnosis task, attention patterns might reveal that "symptom X" strongly correlates with "treatment Y." These relationships are then converted into formal logic rules such as "IF symptom X THEN prescribe treatment Y."
    - The mapping process uses predefined templates tailored to specific domains, ensuring that extracted rules align with the underlying knowledge base.
2. **Logic Representation**
    - Extracted relationships are encoded as Horn clauses (or other forms of logical expressions - TBD) compatible with automated theorem provers. For example:

      ```
      prescribe(amoxicillin) :- symptom(fever), not
      allergy(penicillin).
      ```
    - This step ensures that the outputs of the LLM can be rigorously validated against domain-specific constraints.

2. Formal Verification Engine

The formal verification engine is responsible for validating the logical consistency of the extracted rules. It operates using Answer Set Programming (ASP) - a declarative programming paradigm well-suited for solving combinatorial problems involving constraints and rules.

- **Knowledge Base Integration**
    - The engine validates extracted rules against structured knowledge bases such as medical ontologies (e.g., SNOMED CT) or hardware design specifications (e.g., SystemVerilog Assertions).

    - Temporal logic constraints are applied to ensure that multi-step reasoning processes remain coherent over time. For instance, in a diagnostic workflow, it verifies that conclusions logically follow from observed symptoms in a chronological sequence.

- **Parallelized Verification**
    - To handle the computational complexity of verifying large-scale outputs, the engine employs parallelized solving techniques using GPUs. CUDA-accelerated solvers significantly reduce latency, enabling real-time validation even for complex datasets.

3. Dynamic Repair Mechanism

When inconsistencies are detected during verification, the dynamic repair mechanism intervenes to adjust the LLM's output generation process.

- **Gradient-Based Adjustments**
    - The repair mechanism calculates gradient signals based on verification feedback and backpropagates these signals through the LLM's attention layers. This fine-tunes the model's internal representations to align with verified logic without requiring full retraining.

- **Selective Correction**
    - Corrections are applied only to specific attention patterns or token embeddings associated with errors. This ensures that general language capabilities remain unaffected while domain-specific inconsistencies are resolved.

4. Real-Time Processing Pipeline

(TBD - Usage of FPGAs, Parallel Processing… etc)

Methodology & Workflow

The methodology combines neural network inference with symbolic reasoning in a structured four-phase workflow. Each phase is designed to systematically validate and refine LLM outputs while maintaining efficiency and scalability.

Phase 1: Input Processing and Initial Generation

The workflow begins by processing user inputs through a standard LLM pipeline to generate preliminary responses. During this phase:

- The input query is tokenized and passed through multiple transformer layers.
- Attention weight matrices are captured from each layer, providing insights into how different tokens influence one another.
- A preliminary response is generated based on these attention patterns.

For example, in a clinical setting, a query like "What treatment should be prescribed for a patient with fever and penicillin allergy?" might produce an initial response recommending "amoxicillin," which needs further validation.

Phase 2: Logic Rule Extraction

In this phase, the neural-symbolic interface translates the internal representations of the LLM into logical expressions:

1. **Template Matching**
    - Predefined templates specific to the application domain guide the extraction process. For instance:
      `IF symptom(X) AND NOT allergy(Y) THEN prescribe(Z)`
    - These templates ensure that extracted rules are interpretable and actionable.
2. **Contextual Filtering**
    - The system filters out irrelevant or low-confidence rules based on attention scores and semantic similarity measures.

3. **Rule Encoding**
    - Finalized rules are encoded in a format compatible with automated theorem provers such as Clingo or Z3.

Phase 3: Consistency Validation

The extracted rules are validated against structured knowledge bases using Answer Set Programming:

- **Knowledge Base Matching**
    - Rules are compared against domain-specific databases like SNOMED CT for medical applications or Common Weakness Enumeration (CWE) standards for hardware design.
- **Temporal Logic Checks**
    - Multi-step reasoning processes are evaluated using temporal logic constraints to ensure chronological coherence.

For example:

- Rule: "IF fever THEN prescribe amoxicillin."
- Knowledge Base Constraint: "DO NOT prescribe amoxicillin IF penicillin allergy."
- Temporal Logic Validation: Ensures that recommendations align with prior observations over time.

Phase 4: Corrective Feedback Integration

When inconsistencies are detected during validation:

1. **Error Localization**

    - The system identifies specific tokens or attention patterns responsible for errors.

2. **Gradient-Based Repair**

    - Adjustments are made to the model's internal representations using gradient signals derived from verification feedback.

3. **Regeneration**

    - A corrected response is generated based on updated attention distributions.

For example:

- Original Response: "Prescribe amoxicillin."
- Corrected Response: "Prescribe azithromycin."

References

[1] https://arxiv.org/pdf/2406.14408

[2] https://arxiv.org/pdf/2406.02524

[3] https://cdn.openai.com/prover-verifier-games-improve-legibility-of-llm-outputs/legibility.pdf