CS-F429 Course Project: Neurosymbolic Commonsense Reasoner

November 25, 2025

Abstract

This report outlines our approach to building a NeuroSymbolic AI system capable of commonsense reasoning in both textual and visual domains. We present a hybrid pipeline that leverages the semantic understanding of Large Language Models (LLMs) and Vision-Language Models (VLMs) while grounding reasoning in structured symbolic logic. For the textual task, we utilize Mistral-7B-Instruct to extract predicates from the CLUTRR dataset, employing a local Prolog solver for multi-hop inference and Mistral-7B-Instruct again to generate Chain-of-Thought (CoT) explanations. For the visual task, we implement a multi-stage "Eye-Brain-Judge" pipeline following the ESCA Framework that uses DETR (ResNet-50) for object localization and LLaVA for scene graph generation enforced by geometric verification. Our results demonstrate the efficacy of separating semantic parsing from logical execution.

Problem Statement

The primary goal of this project is to construct a neuro-symbolic pipeline that bridges the gap between the flexibility of neural models and the rigor of symbolic logic. The project is divided into two main tasks:

1. **Textual Reasoning:** Converting natural language stories into symbolic predicates, performing logical deduction using a solver, and generating a natural language Chain-of-Thought (CoT) explanation.
2. **Visual Reasoning:** Extracting structured scene graphs from images and performing reasoning over the identified objects and their relationships using a generated scene-graph.

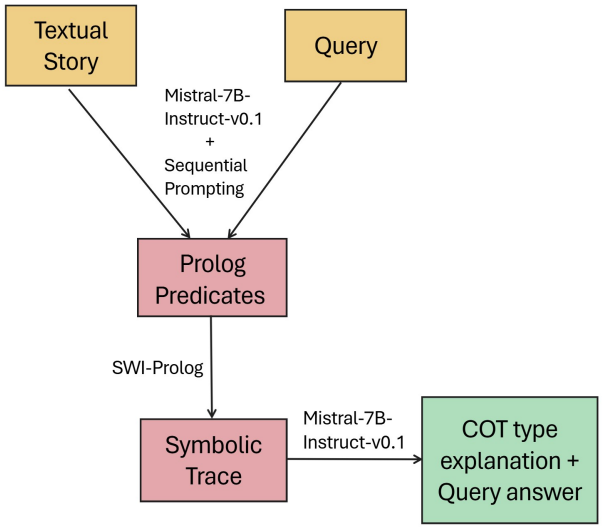

Figure 1: *Architecture and Pipeline*

Task 1: Text-based Neurosymbolic Reasoning

Dataset

We utilized the CLUTRR dataset (facebook/clutrr), which consists of short stories describing family relationships. The dataset requires multi-hop reasoning (ranging from 2 to 4 hops) to determine the relationship between two specific entities.

Architecture and Pipeline

Our approach separates the reasoning process into three distinct stages: Semantic Extraction, Symbolic Inference, and Surface Realization.

Semantic Extraction We employed **Mistral 7B Instruct** as the frozen semantic extractor. A significant portion of our experimentation involved **prompt engineering** to ensure the LLM output was syntactically compatible with our symbolic solver.

We found that standard instructions often led to verbose or hallucinated outputs. To mitigate this, we developed a **few-shot and sequential prompting strategy** with strict constraints:

- **Output Format:** The model was restricted to outputting a single Python-style list of Prolog predicates (e.g., `[mother(Alice, Bob), father(Bob, Charlie)]`).
- **Vocabulary Constraint:** We enforced a strict "closed vocabulary" of allowed predicates (lowercase), for example: `mother, father, parent, daughter, son, sister, brother, spouse`.
- **Syntactic Structure:** The model was explicitly instructed to avoid any conversational text, nar-

ration, or explanations outside the brackets to facilitate easy parsing.
- **Steps:** There are 2 calls to the frozen LLM. The first generates simplified sentences which summarize the main parts of the story. The second

Predicate Processing and Knowledge Base Construction To bridge the gap between the probabilistic output of the LLM and the deterministic nature of Prolog, we implemented a custom Python parsing module. This module performs the following operations:

1. **Regex Parsing:** We utilize regular expressions to extract subject-object pairs from the LLM's text stream, ensuring robustness against minor formatting errors (e.g., whitespace variations).
2. **Filtering:** Extracted predicates are validated against an `ALLOWED_INPUTS` set. Any hallucinated relation types outside the defined schema are discarded.
3. **Dummy Fact Injection:** A critical challenge in Prolog integration is that queries fail if a predicate type is entirely undefined in the Knowledge Base (KB). To prevent this, our script checks for the presence of all required relations. If a relation (e.g., `aunt`) is missing from the story, the system automatically injects a dummy fact (e.g., `aunt(dummy1, dummy2)`) into the `.pl` file. This ensures the solver runs without existence errors.

Symbolic Reasoning (Local Prolog) The sanitized predicates are written to a Prolog file alongside a static ruleset (`rules.pl`). We utilized a local Prolog engine to infer the target relationship.

We initially attempted to use other solvers suggested in the problem statement, such as **Z3** and **MiniZinc**. However, we encountered significant implementation challenges:

1. **Lack of Proof Traces:** The primary requirement for the CoT generation step is a step-by-step derivation. Z3 and MiniZinc primarily act as satisfiability solvers and do not readily expose human-readable inference chains (SLD resolution traces) needed for verbalization.
2. **Resource Constraints:** These heavy-weight solvers proved difficult to run efficiently within the memory and compute constraints of a Kaggle notebook environment.

Consequently, we opted for a lightweight, local swi-Prolog implementation, which provided both efficient inference and transparent derivation paths.

CoT Generation The solver trace, representing the rigorous logical proof, was passed back to **Mistral 7B Instruct**. The model was tasked with "verbalizing" this trace, converting the sequence of rule applications into a natural language Chain-of-Thought explanation and the final answer.

Prolog Reasoning Rules

The symbolic reasoning module is organized into three distinct processing levels:

1. **The Facts Layer:** Accepts raw input from the LLM. This layer handles specific relationship predicates extracted directly from the text, such as `father(X,Y)`, `son(Y,X)`, or `husband(X,Y)`.
2. **The Core Inference Layer:** Acts as the central logic engine, converting mixed inputs into standardized primitive predicates:
   - `gender(Person, Type)`: Infers gender from role semantics (e.g., being a `father` implies `male`).
   - `parent(Parent, Child)`: Unifies relationships into parent-child pairs (e.g., `son(C,P)` is converted to `parent(P,C)`).
   - `spouses(X, Y)`: Enforces symmetry for marriage relations.
3. **The Query Layer:** The final reasoning interface used to determine the answer. It uses the inferred primitives to validate specific family roles (e.g., `is_uncle`, `is_sister`).

Examples of Rules:

Example 1: Sibling-Based Parent Inference The system includes logic to propagate parenthood across siblings.

- **Given Facts:** `father(homer, lisa)` and `brother(bart, lisa)`.
- **Internal Inference:**
  1. The system identifies that Homer is the father of Lisa.
  2. It identifies that Bart is the brother of Lisa.
  3. The rule `parent(A,B) :- father(A,C), brother(B,C)` activates.
- **Query Result:** The system successfully deduces `parent(homer, bart)`.

Example 2: Complex Transitive Inference (Uncles) The system combines sibling logic and parenthood to identify extended family.

- **Given Facts:** `father(abe, homer)`, `brother(herb, homer)`, `father(homer, bart)`.
- **Target Query:** `?- is_uncle(herb, bart).`
- **Execution Trace:**

1. `is_uncle` requests a brother of the parent.
2. `parent(homer, bart)` is established from facts.
3. `is_brother(herb, homer)` is checked. The system verifies they share a parent (Abe) and their genders are male.

- **Result:** `true`.

Advanced Task: Visual Neurosymbolic Reasoning

Dataset

For the visual task, we selected a subset of images from the **GQA dataset**. The objective was to answer queries regarding spatial relationships and object attributes.

Architecture: Two-Stage Hybrid Pipeline

Instead of relying on a single end-to-end model to detect objects and deduce relationships simultaneously, we implemented a **Two-Stage Hybrid Pipeline** for Scene Graph Generation (SGG). This architecture decouples localization (finding objects) from reasoning (describing attributes and relationships).

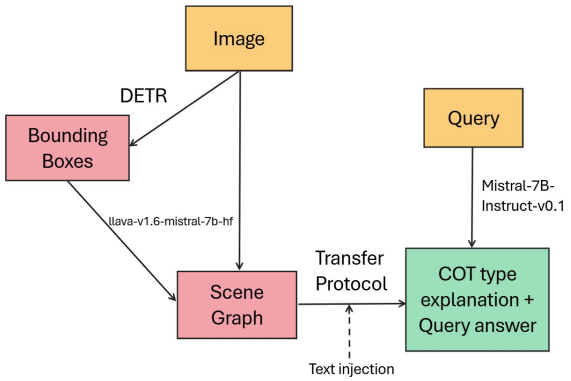

Figure 2: *Architecture and Pipeline*

The data flow operates in three distinct phases, often referred to as the "Eye," the "Brain," and the "Judge."

Phase 1: The "Eye" (Localization with DETR)
We utilized **DETR (ResNet-50)** (Facebook's Detection Transformer) for this phase.

- **Role:** Grounding and Localization.
- **Function:** DETR employs a transformer-based encoder-decoder architecture to detect objects and generate precise bounding boxes. We initially considered using **Grounding DINO** (an Open-Set Zero-Shot detector) to allow for text-prompted detection, but the additional model weights required more VRAM than the Kaggle environment allowed when used in addition with a VLLM prompted to extract object information from the image. We also conducted comparative tests with **YOLO**; however, DETR demonstrated superior consistency and detection accuracy for the objects in our dataset.
- **Necessity:** Large Multimodal Models (LMMs) like LLaVA are capable of rich description but are notorious for poor spatial precision and coordinate hallucination. DETR provides precise "hard" mathematical coordinates that act as the ground truth for the pipeline, effectively solving the "Object Hallucination" problem common in pure VLM approaches.

Phase 2: The "Brain" (Semantic Reasoning with LLaVA)
We utilized **llava-v1.6-mistral-7b-hf** for this phase.

- **Role:** Contextualization, Attribute Extraction, and Relation Prediction.
- **Function:** This model takes the image and the list of detected objects (from Phase 1) to reason about attributes (e.g., color, material) and semantic relationships (e.g., "holding", "looking at").
- **Prompt Strategy - Ground Truth Injection:** A key innovation in our approach was **Context-Aware Prompting**. Instead of asking the model "What is in this image?", we explicitly injected the DETR detections into the prompt: *"Here are the detected objects... with bounding boxes... Use this information as ground truth."* This constrains the LLM to focus only on physically verified objects and allows us to assign unique IDs (e.g., `obj1`, `obj2`) that map back to the coordinate system.

Phase 3: The "Judge" (Geometric Verification)
A critical limitation of LLMs is **Relationship Bias** based on statistical co-occurrence (e.g., assuming a "keyboard" is always "in front of" a "monitor" because that is common in training data, even if the image shows otherwise).

To combat this, we implemented a deterministic **Geometric Post-Processing** stage.

- **Role:** Verification and Filtering.
- **Function:** We calculate a `spatial_relationship_score` by comparing the LLaVA text prediction against the physical coordinates from DETR. If LLaVA predicts ("couch", "behind", "laptop") but the bounding box heuristics indicate the couch is below the laptop, the confidence score is

penalized. This effectively filters out "common sense" hallucinations that defy physics.

Comparison to ESCA

Our approach shares conceptual similarities with the **ESCA (Embodied Scene-Graph Generation)** framework but operates under significantly tighter computational constraints. The full ESCA architecture employs a sophisticated three-stage pipeline:

1. **Concept Extraction:** Using heavy VLLMs (e.g., InternVL, Gemini, or ChatGPT) to identify relevant object concepts.
2. **Entity Identification:** Using **Grounding DINO** combined with **SAM 2** (Segment Anything Model 2) to localize and segment these entities.
3. **Scene Graph Generation:** Using **SGClip**, a fine-tuned CLIP model, to deduce relationships between the identified entities.

While the authors provided the **ESCA-Video-87K** dataset, fine-tuning a CLIP-based model (SGClip) on such a large-scale dataset requires computational resources far exceeding a Kaggle environment. Consequently, we implemented a streamlined approximation: we utilized DETR for the entity identification phase and substituted the specialized SGClip model with **LLaVA**. This allowed us to perform relation inference in a zero-shot manner without the need for extensive pre-training.

Limitations and Deviations

Compute Constraints and Model Size

A primary limitation of this project was the restricted computational budget (single P100/2x T4 GPUs on Kaggle). Neurosymbolic reasoning and fine-grained visual grounding are tasks that typically benefit from the emergent properties of very large models. We observed that smaller open-source models (7B parameters) often struggle with the strict syntactic requirements of symbolic logic and the "common sense" physics required for visual reasoning, necessitating extensive prompt engineering and regex patching to function correctly.

Absence of Visual Prolog Generation

In Task 1 (Textual Reasoning), we successfully converted stories into Prolog code. However, for Task 2 (Visual Reasoning), we **did not** convert the generated scene graph into Prolog predicates for a symbolic solver.

We found that while textual stories have a closed logic world (finite family relations), real-world images present an **Open-World Problem**. There are effectively infinite possible predicates required to describe visual scenes (e.g., spatial relations, attributes, affordances), making it impossible to hard-code a comprehensive `rules.pl` file. Instead of symbolic execution, we used the scene graph for **Geometric Verification**, where the "reasoning" was performed by algorithmically comparing the semantic output of the VLM against the hard coordinate constraints of the object detector.

Future Work

To bridge the gap between our current implementation and the state-of-the-art ESCA framework, future iterations should incorporate:

- **VLLM-Based Concept Extraction (InternVL):** Following Phase 1 of ESCA, we would replace the generic prompting of LLaVA with a dedicated high-resolution VLLM like **InternVL** or **Gemini**. This would significantly improve the system's ability to identify abstract concepts before attempting localization.
- **Grounding DINO + SAM 2 Integration:** Following Phase 2 of ESCA, we would replace the fixed-set DETR detector with the Grounding DINO and SAM 2 pipeline. This would allow for **Open-Set** detection followed by precise segmentation masks, enabling the system to reason over fine-grained object properties (area, shape, occlusion) rather than just bounding boxes.