Type Theory and Large Language Models: A Framework for Self-Reflective Reasoning

Technical Whitepaper

November 13, 2024

Abstract

We present a theoretical framework for enhancing large language models (LLMs) with type-theoretical reasoning capabilities. Moving beyond the limitations of traditional knowledge representations, we establish a tripartite relationship between linguistic expressions, type-theoretical models, and semantic vector spaces. This framework enables LLMs to engage in self-reflective reasoning through interaction with a type theory compiler, providing a bridge between neural and symbolic approaches to knowledge representation.

1 Introduction

The challenge of representing and reasoning about knowledge in artificial intelligence systems has led to various approaches, each with their own limitations. Traditional knowledge graphs based on triple stores, while intuitive, lack inherent consistency guarantees. Large language models, despite their impressive capabilities, are prone to hallucination. We propose a framework that combines the rigorous foundations of type theory with the flexibility of neural representations to enable verifiable knowledge reasoning.

2 Theoretical Foundations

2.1 From Triple Stores to Type Theory

Traditional knowledge representation often relies on triple stores, expressing knowledge as subject-predicate-object relationships. While intuitive, this approach faces fundamental challenges:

- Lack of inherent consistency guarantees

- Need for additional layers (e.g., OWL) to enforce logical constraints
- Arbitrary nature of relationship definitions

This leads us to seek a more fundamental approach to knowledge representation. Type theory emerges as a natural candidate, offering:

- A foundational system on par with set theory
- Built-in consistency through type checking
- Natural representation of objects and their relationships
- Inherent support for logical inference

2.2 The Curry-Howard Correspondence

The connection between type theory and logic is formalized through the Curry-Howard correspondence [?], which establishes that:

$$\begin{array}{lcl} \text{Types} & \leftrightarrow & \text{Axioms} \\ \text{Programs} & \leftrightarrow & \text{Proofs} \\ \text{Evaluation} & \leftrightarrow & \text{Proof Normalization} \end{array}$$

This isomorphism provides us with a crucial tool: when we construct type-theoretical models of knowledge, we are simultaneously constructing proofs of logical propositions.

This triple correspondence provides us with a complete framework for reasoning about:

- Knowledge representation (through types), by making formal inferences about knowledge
- Verification (through proofs), we can verify logical consistency through type checking
- Computation (through evaluation)

For example, a medical diagnosis system could be expressed as:

$$\text{Diagnosis} : \Pi(s : \text{Symptoms}).\Sigma(d : \text{Disease}).\text{Explains}(d, s)$$

where Π denotes a dependent function type[1] and Σ denotes a dependent pair type[2].

[1] A dependent function type $\Pi(x : A).B(x)$ can be understood as "for all x of type A, there is a B that may depend on x". In traditional logic notation, this corresponds to universal quantification $\forall x : A.B(x)$.

[2] A dependent pair type $\Sigma(x : A).B(x)$ represents a pair where the second component's

2.3 Category Theory and Neural Networks

Following Shiebler et al. [**?**], we can view machine learning models categorically. Given a monoidal category $(C, \otimes, I)$, a neural network can be seen as a morphism:

$$f : P \otimes A \to B$$

where:

- P represents the parameter space
- A represents the input space
- B represents the output space
- $\otimes$ is the monoidal product

Remark 1 (Categorical Structure)**.** *For the categorical framework to be mathematically sound, all components must allow for identity morphisms. While matrix multiplications (via identity matrices) and addition operations (via zero) satisfy this requirement, the traditional ReLU activation function ($max(0, x)$) does not admit any possible identity morphism. This technical issue can be resolved by using leaky ReLU ($max(ax, x)$), which theoretically permits an identity morphism when $a = 1$, though in practice we would use different values of a for the actual neural network implementation.*

2.4 Tripartite Framework

We establish three domains and their relationships in diagram **??**:

$$\begin{tikzcd}
& \text{Expressions} \arrow[dl, "\text{model}"'] \arrow[dr, "\text{embed}"] & \\
\text{Types} \arrow[rr, leftrightarrow, "\phi"] & & \mathbb{R}^d
\end{tikzcd}$$

Figure 1: The tripartite framework showing mappings between expressions (unstructured text), types (formal models), and vector spaces (neural representations).

where:

type may depend on the first component. This corresponds to existential quantification $\exists x : A.B(x)$ in logic.

- Expressions represent unstructured language (texts, documents, natural language)
- Types represent well-formed formulas in type theory
- $\mathbb{R}^d$ represents vector spaces of neural embeddings

Key relationships:

- model : Expressions $\to$ Types encompasses both automated parsing (eg when a human writes a well-formed formula and it is parsed by a compiler) and human knowledge modeling (where a human represents knowledge conceptually as a collection of objects and relationships)
- embed : Expressions $\to \mathbb{R}^d$ represents neural embedding, which is a specific instance of the general form $f : P \otimes A \to B$ where A is the space of expressions and $B = \mathbb{R}^d$
- ϕ : Types $\to \mathbb{R}^d$ represents the machine-only capability to connect formal models with their neural representations

This framework highlights an important asymmetry: while humans can directly model expressions as types (model), the connection between types and vector spaces (ϕ) is accessible only to machines, forming the basis for automated reasoning and self-reflection.

3 Self-Reflective Learning Process

The framework enables a systematic approach to knowledge acquisition and verification through type theory. We describe the key components and processes that implement this approach.

1. **Knowledge Embedding**: Initial knowledge is embedded in the LLM's vector space
2. **Model Generation**: The LLM generates a type-theoretical model of its understanding
3. **Verification**: The model is verified through type checking and query testing
4. **Refinement**: Results feed back into the LLM's representations

We can express this as a composition of mappings:

$$\text{model} = \phi \circ \text{embed}$$

Where, instead of doing the slower route where a human has to first create a TypeDB schema and then write queries, the human can let the LLM read a text and interact with a consistent model M without worrying about hallucinations.

The process is iterative and self-improving:

$$\text{model}_{i+1} = \text{refine}(\text{model}_i, \text{verify}(\text{query}(\text{model}_i)))$$

where:

- model_i is the current type-theoretical model
- query generates test cases
- verify checks consistency
- refine updates the model based on verification results

3.1 Knowledge Components

At the core of our framework are queries to a type-theoretical model, implemented in TypeDB, which serve as the formal representation of knowledge. These are accompanied by hypotheses:

Definition 1 (Hypothesis)**.** *A hypothesis consists of:*

- *A query q in TypeQL*
- *An expected result r_i (which may be a complex structure)*
- *A confidence measure $c \in [0, 1]$*
- *An explanatory annotation α describing why the system expects this result*

These hypotheses represent the system's beliefs about what queries should return, derived from its vector space understanding. For example, in a medical domain, a hypothesis might expect a query about "coughing blood" to return a set of potential conditions with their urgency levels and recommended actions.

3.2 The Scientific Cycle

The system engages in a continuous cycle of hypothesis formation, testing, and refinement:

1. **Hypothesis Formation**: Given a vector representation $v \in \mathbb{R}^d$, the system generates a set of hypotheses $H = \{h_1, ..., h_n\}$ where each h_i contains a query expressing an expected property of the domain.
2. **Model Construction**: The system maintains a type-theoretical model M in TypeDB expressing its current understanding. Each hypothesis h_i generates both schema elements and test queries.
3. **Verification**: For each hypothesis h_i, the system:
    - Verifies that the query q_i is well-formed TypeQL
    - Evaluates $M \models q_i$ directly, yielding a result r in some type T
    - Compares this with the expected result r_i from the hypothesis
    - When $M \not\models_i$, records both the actual result and the context in which the expectation failed
4. **Refinement**: Based on verification results, the system either:
    - Refines individual hypotheses when results don't match expectations
    - Updates the model M when multiple hypotheses consistently indicate misalignment
    - Extends the schema when necessary to capture newly understood relationships, and retrains itself to reflect the new understanding

3.3 Interaction Modes

The system supports two primary modes of interaction:

3.3.1 Natural Language Interface

For general users, the system processes natural language through the composition:

$$\text{query} \xrightarrow{\text{embed}} \mathbb{R}^d \xrightarrow{\phi} \text{TypeQL}$$

The answer can draw on the inverse, ϕ^{-1}, to generate a natural language response based on the model M.

3.3.2 Formal Query Interface

enabling formal verification of the system's knowledge and access to structured responses.

$$\varphi \xrightarrow{\text{check}} \text{Types} \xrightarrow{\text{evaluate}} \mathbb{B}$$

This dual approach ensures both usability for general interaction and rigorous verifiability where needed. The continuous scientific cycle runs as a background process, gradually improving the system's formal model through hypothesis testing and refinement.

The use of TypeQL as the query language allows for rich, structured responses beyond simple boolean values, enabling the system to express complex relationships and reasoning chains while maintaining formal verifiability.

References

[1] Shiebler, D., Gavranović, B., & Wilson, P. (2021). *Category Theory in Machine Learning.* arXiv preprint arXiv:2106.07032.

[2] Mimram, S. (2020). *Program = Proof.*