ULOGIC-MIND Project White Paper

Leopoldo Cano - leo@ulogicmind.ai - January 2026

"A verified neurosymbolic intelligence architecture
that combines LLM intuition with deterministic symbolic verification
through the new ULOGIC-Language and the ULOGIC-Kernel,
enabling reusable, exact, machine-verifiable scientific knowledge"

PROBLEMS AND SOLUTION:

1. **The Problem**: Current LLMs are powerful generators of plausible reasoning, but they do not guarantee exact correctness, reproducibility, or formal validity.
2. **The Solution**: ULOGICMIND separates intuition from verification: neural systems propose "ULOGIC formalizations" of proofs and reasoning, and the ULOGIC Kernel verifies deterministically. The novel and distinctive key to this NeSy architecture is ULOGIC (a new *Universal Logic Language*).
3. **Knowledge Reuse**: Verified reasoning should not disappear after each answer. It should become reusable, composable, inspectable knowledge. That is the role of TekDocs.
4. **Infrastructure Vision**: Science needs repositories not just for text and code, but for exact, machine-verifiable knowledge. TekHub is the repository layer for that future.

KEY IDEAS AND INNOVATIONS:

1. **ULOGICMIND** combines LLM intuition with deterministic symbolic verification through the ULOGIC-Language and the ULOGIC-Kernel, enabling reusable, exact, machine-verifiable scientific knowledge and discoveries.
2. **ULOGIC** is a universal formal language for exact, verifiable reasoning (built from new fundamental principles different from those of current formal logic developed during the 20th century). It bridges LLM intuition with deterministic symbolic verification through the ULOGIC Kernel. ULOGIC is designed so that reasoning steps can be formally represented, deterministically checked, composed, stored, and reused as verified knowledge objects.

3. **The ULOGIC Kernel** is the deterministic symbolic verification engine that checks formal reasoning generated in ULOGIC, enabling exact, reproducible, machine-verifiable intelligence.
4. **TekDocs** are atomic, reusable, machine-verifiable knowledge objects encoded in ULOGIC. They enable exact scientific knowledge to be stored, composed, validated, and reused at scale.
5. **TekHub is the GitHub for Scientific Knowledge**: the repository layer for storing, versioning, sharing, and composing TekDocs, a GitHub-like infrastructure for exact, verified scientific knowledge.
6. **TekHub is to verified scientific knowledge what GitHub is to source code.**
7. **ULOGIC vs Lean | Formal Reasoning for Neurosymbolic AI vs Interactive Theorem Proving**
   - Lean is an established theorem prover and programming language
   - ULOGIC is designed as a broader **LLM-compatible formal reasoning layer + reusable knowledge infrastructure**
   - ULOGIC focuses more explicitly on:
     - human-readable formal knowledge objects
     - AI-generated formalization
     - deterministic verification pipeline
     - repository-native exact knowledge

---

1. ULOGIC is an Abstract Structural Language

1.1. Abstract-Expression versus String-Expression representations

ULOGIC-LANGUAGE is a logical-structural system designed to decouple semantic intent from its representation as text. Unlike traditional systems, ULOGIC treats logical and mathematical expressions (including proofs, algorithms, and all types of complex exposition) as **abstract objects**, allowing for total flexibility in their representation.

(Concept of Representation):

- `Abstract-Expression` –>is represented by (many) –>`String-Expression`
- `String-Expression` –>represents one –>`Abstract-Expression`

(Auto-formalization Process):

- `String-Expression-Original` –>Find the `String-Expression-Canonical` –>That defines the `Abstract-Expression`

1.2. Classification of Expressions (Abstracts: Units & Blocks) (Strings: Representation-Formats)

1. **Abstract-Expressions (AbsExp)** are "abstract objects" of two types (disjoint):
   - **Descriptive Unit (ExpUnit)**: Are lists, bags, sentences, fragments of sentences...
   - **Expositive Block (ExpBlock)**: Are an exposition of definitions, theorems, algorithms...
2. **String-Expressions (StrExp)** are "strings-of-chars" that represent an "abstract-object-expression"
   - There exist different equivalent **Representational-Formats**:
     - Formats **UlogicFormal-Canonical**: `HALU-Canonical`, `XMLU-Canonical`
     - Formats **UlogicFormal-Variants**: `HALU-F1`, `HALU-F2`, `HALU-F3`, `SimpleTree`, `VisualTree`, `XMLU-Total`
     - Formats **SemiFormal**: `Math2k` and `Math2K+` (the language of mathematics, logic, and computation textbooks)
     - Formats **InformalPrecise**: Natural language, perhaps with symbols, but still sufficiently precise to obtain the strict UlogicFormal expression automatically (capability of **auto-formalization**)

Example:: Let us consider an expression of the type *DescriptiveUnit*, defined and represented as follows:

1. Canonical "definitional" representation **HALU-Canonical (HALU-C)**:

```
(DOM:P,x)
```

2. Equivalent **HALU-F3** representation:

```
(DOM:
    P,
    x,
)
```

3. Equivalent **SimpleTree** representation:

```
DOM:
   P
   x
```

4. Equivalent **UXML** representation:

```
<DOM>
    <a>P</a>
    <a>x</a>
</DOM>
```

5. **SemiFormal-Math2K** representations:

```
P(x)
or also the equivalent
x ∈ P
```

6. **Informal-Precise** representations:

```
‘‘Let x be an element of P’’
or equivalent
‘‘Let us have x belonging to P’’
 or similar...
```

---

2. Core Architecture: Expressions (Abstract Objects)

In ULOGIC, the core of the system are the **Abstract-Expressions** or **AbsExp**. It is convenient to view them as abstract entities: `Abstract-Expressions` can be represented by different `String-Expressions`. This allows the separation of “logic” from “writing formats”. The **Abstract-Expressions** or **AbsExp** are divided into two strictly **disjoint** categories:

2.1. DescripUnit (Descriptive Unit):

It is the atomic unit of information. It represents the “raw material” of knowledge.

- **Typical content:** Axioms, hypotheses, lists of objects (ordered or not), bags (multisets) and fundamental factual or mathematical assertions.
- **Role:** It can be seen as the description of a state of affairs.

2.2. ExpoBlock (Expositive Block)

It is the complex and narrative structure. It represents the “flow” or the logical architecture.

- **Typical content:** Demonstrations and proofs, definitions, propositions and theorems, inference sequences, algorithms
- **Role:** An `ExpositiveBlock` expounds how units (`DescriptiveUnits`) are connected to build an argument, a solution, or an algorithm
- **Role:** `ExpositiveBlocks` are the building blocks of TekDocs (Transportable Encapsulated Knowledge Documents).

Example:

- 1. A canonically expressed DescripUnit: `(DOM:==>,(LST:A,B))` (HALUC Format) which is equivalent to `A==>B` (Math2K Format)
- 2. A canonically expressed ExpoBloc: `<proof(n)>...</proof(n)>`, representing a “demonstration or proof.”

3. Interface Layer: Representations using StringExpressions

3.1. String-Expressions: Introduction

StringExpressions or **StrExp** are the "skin" of the expressions; they are their **Representation** or **Serialization** (synonyms). It is what the user writes or what the machine processes at the primary level. They **represent** an AbsExp. In reality, the only way to "define" an abstract expression is to use a canonical string-expression.

3.2. String-Expressions: Definition

- **Definition:** They are strings of characters (`string-of-chars`).
- $N : 1$ **Relationship:** A single **AbsExp** (abstract object) can be **serialized** into multiple **StringExpression** representations.
- **Canonical Representation**: Every abstract object AbsExp has a canonical representation **StrExp-Canonical** that identifies it.
- **Practical Intuition**: We can see the abstract object `AbsExp` as an equivalence class of concrete `StringExp` objects (different equivalent representations). Thus, the abstract object `AbsExp` is created by a convention of equivalences between strings. It offers freedom of formats, separating "logic" from "writing."
- **Independence:** The logical system operates on the abstract object, ignoring syntactic variations as long as they map to the same expression. The practical way to do this is to use the appropriate **StringExpression-Canonical** representation.

  Equivalence Example:

  - `String A`: "For all x such that x is even..."
  - `String B`: "$\forall x \in 2\mathbb{Z}$..."
  - **Identity:** Both REPRESENT the same abstract **DescripUnit** (which in turn has a canonical HALUC representation as we will see)

4. Interface Layer: Representational-Formats and Autoformalization

4.1. Representation (=Serialization) and Autoformalization

(Concept of Representation):

- `Abstract-Expression` –>is represented (=serialized) by (many) –>`String-Expression`
- `String-Expression` –>represents one –>`Abstract-Expression`

(Autoformalization Process):

- It is the process by which an agent (such as LEOX) obtains the Abstract-Expression from a String-Expression.
- In practice, it is the process of finding the **Associated-Canonical-String-Expression** from the starting String-Expression (which may be semi-formal, informal-precise, or written in any other form).
- `String-Expression-Original` –>Find the `String-Expression-Canonical` –>That defines the `Abstract-Expression`

(canonical-example): 1. Let us take for example the "Informal-Precise" expression `Let x be an element of P`. 2. We could first write it as a bridge in SemiFormal-Math2k mode as `P(x)` 3. And thus find the canonical **strict-formalization** UlogicFormal (HALUC format) `(DOM:P,x)` which is definitional. 4. The HALUC canonical format defines the abstract object, and then we can represent it in many other equivalent ways.

4.2. Representational-Formats:

- StringExpressions can be written following different formats (writing-serialization conventions).
- Each **Representational-Format** is a specification that determines how to write (serialize) the expression.
- Any serialization convention is possible (one can write as one wishes), allowing for the creation of future formats (extension of the ULOGIC standard).

In ULOGIC, the following **Representational-Formats** exist.

4.2.1. CASE ExpUnit:

a. The **UlogicFormal-Canonical** format for ExpUnits:

- Format **HALU-Canonical (HALU-C)**: Utilizes the format `(HEAD:exp1,exp2...)`
- Characteristics:
  - These formats are strictly defined with a strict EBNF grammar

- ExpUnits are represented (serialized) canonically using the HALU-C format by default
- This format is the **deep grammar** hidden in everyday semi-formal and informal logical and mathematical expressions.

b. **UlogicFormal-Variants** formats for ExpUnit:

- These are absolutely strict, and totally equivalent to the canonical ones:
  - HALU Variants: Format **HALU-F1**, Format **HALU-F2**, Format **HALU-F3** similar to HALUC, but adding spaces and line breaks for formatting.
  - XMLU Variants:
    - Format **XMLU-Total (XMLU-T)**: Utilizes the format `<HEAD>...</HEAD>` wrapping the atoms with `<a>atom</a>`
    - Format **XMLU** in general: Like XMLU-C but adding spaces, indentations, and line breaks for readability
  - TXT-Format Variants:
    - Format **TreeSimple**: Removing parentheses and `,` and writing in an indented form similar to YAML (described below)
    - Format **TreeVisual**: Analogous to TreeSimple adding visual marks `|` that give a tree-like appearance
- Characteristics:
  - These formats are strictly defined with a strict EBNF grammar
  - All are **absolutely exact** formats equivalent to the canonical forms (they are different representations of the same thing).
  - It is 100 % deterministic (algorithmic) to pass from one representation to another.

4.2.2. CASE ExpBlock:

a. The **UlogicFormal-Canonical** format for ExpBlocks:

- Format **XMLU-Canonical (HALU-C)**: Utilizes the format `<tag>...<tag>` with different nested `<tag>`, as specified in the syntax of TekDocs (a TekDoc is an ExpBlock composed of other Blocks)
- Characteristics:
  - These are descriptive blocks: they state a definition, a theorem, a proof-demonstration, the definition of an algorithm, the execution of an algorithm...
  - XML-type tagging is the most convenient, although we will use tags with numbering such as `<section(n)>` or `<proposition(n)>` to improve ordering, structure, and cross-references.

b. **UlogicFormal-Variants** formats for ExpBlocks:

- These are absolutely strict, and totally equivalent to the canonical ones:
  - XMLU Variants:
    - Format **XML-Std (XMLS)**: Special XMLU tags `<tags(n)>` are written normalized as `<tag num="n">` as correct standard XML.
    - Format **XMLU-Readable (XMLU-R)**: Same as XMLU-Canonical, but with indentations and line breaks (indifferent) to improve readability.
    - Format **XMLU-Total (XMLU-T)**: Everything as `<tags>`, including the internal ExpUnits that the block contains. For example, if one of the steps of the proof is `(DOM:P,x)`, it is also expressed as `<DOM><a>P</a><a>x</a></P>`
- Characteristics:
  - These formats are strictly defined with a strict EBNF grammar
  - All are **absolutely exact** formats equivalent to the canonical forms (they are different representations of the same thing).
  - It is 100 % deterministic (algorithmic) to pass from one representation to another.

4.2.3. Limit Cases "Pseudo-Formats":

In ULOGIC, we see expressions and reasonings of logic, mathematics, and natural language as limit-imprecise-defective cases of the ULOGIC grammar. In other words: Wherever there is "precise reasoning," it will be possible to find the "ULOGIC deep grammar." That is auto-formalization. BUT the vagueness and imprecision can be high,

1. **SemiFormal** Format: –>[AUTOMATIC FORMALIZATION USING CONVENTIONS]

- Format Variants:
  - Format **Math2k**: The language of mathematics and logic books updated from the years 2000-2025 written in a "rigorous" form, which have evolved by integrating notations invented in 20th-century formal languages and logic systems.
  - Format **Math2k+**: This is like the Math2k language but with some additions inherited from ULOGIC conventions
- Characteristic:
  - It is not strictly a "format" insofar as there is no strict specification, but sufficiently precise conventions can be established to achieve "auto-formalization" and find the deep-structure of habitual mathematical expressions.
  - CASE EXPUNIT:
    - **Auto-formalization** of ExpUnit: From the SemiFormal language, it is possible to automatically find the **underlying UlogicFormal** structure as long as clear and well-defined **Auto-formalization Conventions** are used.

    - The **Auto-formalization at Scale** of ExpUnit: is one of the characteristics of the ULOGICMIND system and its **disruptive advantage** over languages like LEAN or similar.
- CASE EXPBLOCK:
    - The exposition of Definitions, Propositions, Algorithms, and Demonstrations in Math2K is poorly standardized and even worse expressed. In particular, mathematicians are not even aware that "algorithms" are also mathematics.
    - The "Auto-formalization" of propositions or definitions is quite direct, but to formalize standard semi-formal demonstrations, it is necessary to fill multiple gaps.

2. **Informal-Precise** Pseudo-Format: –>[FORMALIZATION POSSIBLE WITH ASSISTANCE AND DECISIONS]

- Format Variants:
    - Natural language, perhaps with certain symbology, that attempts to be precise in concepts and arguments.
    - We find it in technical manuals, precise philosophy manuals, law, and similar.
    - For example `Let x be an element belonging to P`
- Characteristic:
    - It is not strictly a format, insofar as there is no writing specification, only a heuristic guide on how to formalize.
    - "Auto-formalization" is not always 100 % automatable, but can be achieved in "assisted-auto-formalization" mode.
    - The typical scenario is that a human collaborates with the AI to fill the detected gaps, until a UlogicFormal representation is achieved that "in the judgment of the human" (or collective of humans) represents the underlying meaning.

3. **Informal-Vague** Pseudo-Format: –>[NON-UNIVOCAL FORMALIZATION: MULTIPLE OPTIONS AND INTERPRETATIONS POSSIBLE]

- Format Variants:
    - Discourses in natural language where concepts are not defined and reasonings are by analogy, poetic, etc.
- Characteristic:
    - It is not strictly a format because it is not well-defined, but it is convenient to see it as an extreme degenerate form of format.
    - "Formalization" is not possible in practice because the alternatives to specify concepts are endless, and each "formalization" is a possible alternative of meaning (from among an infinite number far from each other). No "formalization" can be considered as "the" formalization (they are rather re-interpretations).

4.3. Key Ideas on ULOGIC Representations:

1. The **canonical formats** are used to express the rules of ULOGIC in a precise manner.
2. The canonical formats are ultimately the **deep structure** of reasoning (which thus becomes independent of syntax).
3. This vision of **deep structure** separated from syntax allows for auto-formalization at scale using ULOGIC (a disruptive leap).
4. To express the rules of the ULOGIC language, it is convenient to differentiate between **ExpUnit** and **ExpBlock**.
5. To highlight the difference, it is better to canonically represent **ExpUnit in HALU format** and **ExpBlock in XMLU format**.
6. The XMLU-T format expresses everything as uniform XML (possible) but worsens the distinction between ExpBlock and ExpUnit.
7. It is easy to define a "HALU" format for ExpBlocks [`<tag>...</tag>` –> `(tag:...)`] but it is less readable than the XML format and worsens the distinction between the two cases.
8. The **key idea** of ULOGIC is that any syntactic format (pictorial, two-dimensional, or however desired) is perfectly usable, provided the **UlogicFormal deep structure can be found using the appropriate conventions and specifications.**
9. This "way of seeing and understanding" expressions (separating syntax from deep grammatical structure) **explains relevant historical facts**: The drawings of Euclidean geometry are actually irrelevant, and "rigor" can only be achieved when those drawings are used merely as "indicators" of underlying formal expressions. This is an essential discovery and a historical milestone highlighted by Hilbert in "Grundlagen der Geometrie" of 1903. ULOGIC allows for the visualization of this fact from a deeper conceptual framework.

---

5. Knowledge Encapsulation: TekDocs & TekHub

For knowledge to be persistent and reusable, ULOGIC uses TekDoc documents and the TekHub repository:

1. **The TekDoc (Transportable Encapsulated Knowledge)**
   - It is the digital container of an **Expositive Block** `<tekdoc>` which is composed of ExpositiveBlocks and DescriptiveUnits.
   - **It is NOT a static PDF**: it is an executable object that contains the demonstration or the algorithm along with its verification trace.

2. **The TekHub** The global repository of verified knowledge. It is the **GitHub of science**, where TekDocs are stored, versioned, and cryptographically linked.

- **Interconnectivity**: A TekDoc can import other remote TekDocs, allowing for the reuse of content and creating a network of interconnected knowledge.

---

6. The NeSy Engine (Neuro-Symbolic): The ULOGIC-MIND Architecture

The NeSy architecture consists of the following four components:

1. **ULOGIC Language (THE KEY)**: This is the central key piece. It is the first formal logical language with sufficient capacity to express logical, mathematical, and scientific reasoning of any level of complexity. It solves set-theoretic paradoxes without the use of types, unifies logic and computation, and possesses auto-metalinguistic capabilities. ULOGIC is the "deep grammar" hidden in mathematics, logical reasoning, and computation. The ULOGIC language allows LLMs to "see the exact reasoning" for the first time.

2. **Neuro Component (LLM-LEOX):** Acts as the creative translator. It reads natural language, proposes hypotheses, and generates initial `Representations` towards the ULOGIC language. It understands ULOGIC and attempts to construct correct TekDocs using the ULOGIC-Language.

3. **Symbolic Component (ULOGIC-KERNEL):** Acts as the deterministic judge. It analyzes the content of a TekDoc (which contains definitions, theorems, demonstrations, algorithms, and executions of algorithms) and determines with 100 % precision if the TekDoc is well-constructed, which means that all reasoning, definitions, theorems, algorithms, and executions of algorithms are absolutely correct and follow the rules of the ULOGIC language.

4. **UMNIND Orchestrator**: These two components are orchestrated by **ULOGIC-MIND**, the framework that interconnects the LLMs with the Kernel:

- Sends proposals from the LLM to the Kernel
- Returns Kernel messages to the LLM to correct errors
- Allows the LLM to search for knowledge in the TekHub network
- Stores and retrieves correct TekDocs from the TekHub network

7. ULOGIC vs. Conventional Logic (FOL/SOL) and (HOL/LEAN)

Limitations of FOL (First Order Logic) or SOL (Second Order Logic):

- **Rigidity**: Expressions are "chains of strings" constructed in a rigid manner.
- **Incapability**: It does not have sufficient expressive power to articulate advanced mathematics.
- **Insufficiency**: It is totally impossible to express proofs or algorithms WITHIN the language itself.
- **Scale-Auto-formalization**: Impossible; it lacks even sufficient expressive capacity.
- **Practical Utility**: None. One cannot do "advanced mathematics" by writing in FOL or SOL.

Limitations of HOL/LEAN:

- **Rigidity**: Expressions are "chains of strings" constructed in a rigid manner.
- **Obfuscated-Capacity**: Expressing advanced mathematics in HOL or Lean is possible, but in an "obfuscated-cryptic" way, because it requires an encoding so distant from the original mathematical language that it even "means something else" because it is a totally different language.
- **Practical-Insufficiency**: Although demonstrations can be expressed in LEAN, the straitjacket of types makes them almost incomprehensible to a mathematician. They are "something else." Even if the formalization is successful and the compiler says it is "OK," how are we sure that what has been formalized is REALLY the same as what the original theorem stated?
- **Radical-Insufficiency**: Lean cannot express advanced computation, let alone express algorithm execution, and even less "speak about itself."
- **Scale-Auto-formalization**: Impossible. It requires a "translation" from "mathematical language" to "Lean programming language." It is a distant encoding, a translation between TWO ABSOLUTELY DIFFERENT LANGUAGES. Libraries of theorems written in LEAN can only be created using "expert humans" who in turn "decide" if the created LEAN code "really" expresses the same as the original mathematics.
- **Practical-Utility**: None. A mathematician cannot create mathematics using LEAN. Lean is a language for embalming the dead, not for creating new living organisms.

ULOGIC solves all those limitations:

- **Flexibility**: Representation as a string is indifferent. Book mathematics are already in ULOGIC!
- **Scale-Auto-formalization**: Achieved. Finding the "UlogicFormal" deep structure of a standard mathematical expression is the great strength of ULOGIC, since the language is created to be a 1:1 mirror of standard expressions (rather than a distant, incomprehensible encoding like Lean).
- **Sufficiency**: It can express advanced mathematics and reasoning, algorithms, the execution of algorithms and their results, and can speak about itself without leaving the language. Logical, Computational, and Auto-metalinguistic capabilities in THE SAME language.
- **Mathematical Advances**: The solution to mathematical paradoxes (without the use of types) in turn creates a new mathematical set theory and a new philosophical foundation of mathematics.
- **Creative Tool**: The ULOGIC Language is a tool for innovation and the creation of new knowledge. It even allows for the expression of theories and results that ARE impossible to express in the standard Math2K language.

---

8. ULOGIC-MIND: Systemic Advantages for Automated Reasoning

1. **Semantic Decoupling:**
   - Allows changing the syntax (serialization) without altering the underlying logical engine.
   - Allows evolving the way of writing without breaking the logic of the stored knowledge.
2. **Interoperability:** Facilitates automatic translation and modeling of complex knowledge.
3. **Accumulation of Truth:** Thanks to TekHub, science does not have to "reinvent the wheel" (it is built on previously verified blocks).
4. **Zero Human Friction:** The user does not need to learn formal logic. The system translates their intuition to UlogicFormal automatically.