Hierarchy Guided Extraction with Answer Set Programming and Large Language Models

Mario Alviano[1,†], Massimo Busiello[1,†] and Lorenzo Grillo[1,2,*]

[1]University of Calabria, 87036 Rende (CS), Italy
[2]University of Campania "Luigi Vanvitelli", 81100 Caserta (CE), Italy

Abstract

Large Language Models (LLMs) are increasingly used for semantic data extraction from unstructured text. However, they struggle with complex combinatorial reasoning and global consistency constraints. We propose a logic-guided framework combining LLM-based extraction with Answer Set Programming (ASP), delegating extraction to the LLM and rigorous validation to ASP. Prior logic-based pipelines evaluate flat logical guards for every target predicate, creating a significant computational bottleneck. To overcome this, we introduce Hierarchical Guided eXtraction (HGX), which natively embeds domain dependencies into an expressive parent-child hierarchy. By leveraging structural pruning, through macro-level routing, lexical pre-filtering, and recursive gated extraction, HGX immediately short-circuits dependent sub-trees if an ancestor concept fails validation, systematically bypassing redundant LLM and ASP queries. We formalize the semantics of HGX and evaluate it on complex, multi-domain benchmarks. Results demonstrate that HGX achieves up to a 92% reduction in ASP solver invocations compared to flat evaluation strategies, while strictly preserving extraction accuracy.

Keywords

Answer Set Programming, Semantic Data Extraction, Logic Programming, Large Language Models, Neural–Symbolic Integration, Hierarchy based logic

1. Introduction

Large Language Models (LLMs; [1]) have recently been adopted for semantic data extraction and parsing from unstructured text [2, 3, 4, 5], enabling the automatic construction of structured representations such as relational facts or knowledge bases. In standard logic-based pipelines, such as [LLM]+ASP [6] and LLMASP [7], extraction is commonly performed independently for each target predicate, with logical validation and inference applied only post-hoc.

LGX (Logic Guided eXtraction [8]) improved upon this by executing an ASP query prior to extracting a predicate. This approach ensures that a predicate is extracted only if its associated logic guard is satisfied, given the current state of the partial model. However, this design necessitates an ASP solver invocation for every potential extraction. While optimizations can be implemented, the continuous reliance on solver calls remains a significant computational bottleneck, particularly as the complexity of the domain grows. Furthermore, a misconfiguration in a predicate's guard can potentially compromise the entire extraction pipeline. To overcome these limitations, we propose HGX (Hierarchical Guided eXtraction), which natively embeds these logic constraints into a hierarchical structure, bypassing redundant evaluations through structural pruning.

Like LGX, this work lies at the intersection of Neuro-Symbolic AI and Information Extraction. Unlike probabilistic neuro-symbolic frameworks such as DeepProbLog [9] and NeurASP [10] that integrate neural outputs via probabilistic semantics, we focus on utilizing pre-trained LLMs as black-box oracles. In this modular setting, existing approaches typically employ logic only for post-hoc validation [11] or iterative refinement [12, 13]. Answer Set Programming (ASP [14, 15]) is particularly well-suited for

CILC 2026: The 41st Italian Conference on Computational Logic, June 23–25, 2026, Ferrara, Italy
*Corresponding author.
†These authors contributed equally.
mario.alviano@unical.it (M. Alviano); bslmsm01r27d086k@studenti.unical.it (M. Busiello); lorenzo.grillo@unicampania.it (L. Grillo)
0000-0002-2052-2063 (M. Alviano); 0009-0002-6099-3895 (L. Grillo)

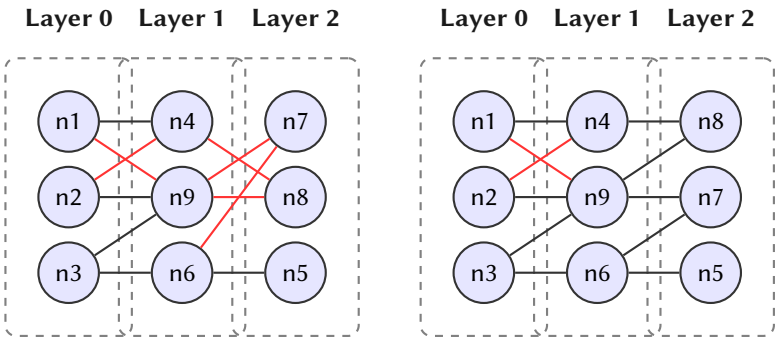

Figure 1: Comparison of a three-layer graph before (left) and after (right) applying a crossing minimization strategy. Intersecting edges are highlighted in red. Reordering the nodes, specifically in Layer 2, significantly reduces the total number of crossings and improves the readability of the layout.

this task due to its ability to represent complex relational structures and non-monotonic dependencies [16, 17].

Example 1 (Hierarchical Guided eXtraction)**.** *Consider the* Crossing Minimization *problem, where one seeks an ordering of nodes within layers of a graph that minimizes edge crossings (see Figure 1). Solving such combinatorial reasoning tasks directly with a standard LLM is challenging, as it requires enforcing global consistency constraints and exploring complex structural dependencies. Rather than expecting the underlying LLM to solve reasoning tasks implicitly, better results are obtained through logic-guided frameworks, which ensure a clear division of labor: the LLM-based oracle extracts candidate facts from text, while ASP performs rigorous reasoning and consistency checks.*

Here, the extraction task consists in building a structured graph representation from natural language descriptions (e.g., `edge(n1,n9)`*,* `layer_size(0,3)`*, or* `in_layer(0,n1)`*). Issuing extraction requests for all target predicates independently, as in standard LLMASP [7], is inefficient and error-prone, potentially leading to spurious extractions like layer assignments when no layering information is present. While LGX addresses this by evaluating logic guards before extracting each predicate, HGX natively optimizes the process: if a parent condition (e.g., the presence of layers) evaluates to false, the entire extraction path for its sub-predicates is structurally pruned. We will use this layered-graph scenario as a running example throughout the paper (see Example 2).* ■

Parallel to the reasoning-acting decomposition in ReAct [18] and conceptually akin to goal-directed ASP evaluation [19], our frameworks interleave extraction with logic derivation. This allows logically implied facts to be inferred early, ensuring extraction resources are spent only on facts consistent with the current partial model. In HGX, this principle is operationalized specifically through structural pruning, bypassing explicit logic evaluations and state-storage for descendant nodes when ancestors fail. Evaluations conducted on benchmarks from the same domains as LGX demonstrate that HGX substantially reduces the number of ASP solver invocations while maintaining identical performance scores, trading off only the structural configuration complexity required to map the hierarchy.

2. Preliminaries and Baseline Setting

This section introduces a formal abstraction of LLM-based extraction pipelines. The presentation is intentionally minimal and focuses only on concepts required to formalize hierarchy-guided extraction.

Text, Predicates, and Facts. Let T denote a finite unstructured text document, and $\mathcal{T}$ be the domain of text documents. Let $\mathcal{P}$ be a fixed finite set of predicates, each with a fixed arity. A *ground fact* is an atom of the form $p(c_1, \ldots, c_k)$, where $p \in \mathcal{P}$ has arity $k \geq 0$ and each c_i is a constant from a fixed finite domain of constants. Let $\mathcal{F}_p$ denote the set of all ground facts over predicate p, for all $p \in \mathcal{P}$, and

$\mathcal{F}$ be the set of all ground facts, i.e., $\mathcal{F} = \bigcup_{p \in \mathcal{P}} \mathcal{F}_p$. A *database* D is a finite set of ground facts, that is, $D \subseteq \mathcal{F}$. Let $\mathcal{D}$ denote the domain of databases, defined as $2^{\mathcal{F}}$. In the following, we assume databases grow monotonically during an extraction process, that is, along a finite sequence $D_0 \subseteq \cdots \subseteq D_n$.

Logic Programs. Let $\mathcal{LP}$ denote the set of logic programs over predicates in $\mathcal{P}$. Specifically, we consider normal logic programs under the stable model semantics. A *normal rule* is an expression *"head if body"* of the form

$$a \leftarrow b_1, \ldots, b_m,\ not\ c_1, \ldots, not\ c_n,$$

where a, b_i, and c_j are atoms. A *normal program* Π is a finite set of normal rules over predicates in $\mathcal{P}$. Programs are interpreted under their standard ground instantiation with respect to the fixed finite domain of constants. Given a ground program Π and a set of ground atoms I, the *Gelfond–Lifschitz reduct* Π^I is obtained by removing all rules whose body contains a negative literal $not\ c$ with $c \in I$, and by deleting all remaining negative literals from the bodies of the remaining rules. A set of atoms I is a *stable model* (or *answer set*) of Π if I is the least model of the reduct Π^I. In the intended extraction setting, the programs are constructed so that $\Pi \cup D$ has a unique intended answer set. We denote it by $Ans_\Pi(D)$.

Extraction Oracle. We abstract the behavior of a Large Language Model (LLM) by means of an *extraction oracle*. Let $\mathcal{P}_E = \{p_1, \ldots, p_n\} \subseteq \mathcal{P}$ be a finite set of *target predicates*, i.e., predicates whose facts are to be extracted from text. When needed, we assume that the set of target predicates $\mathcal{P}_E$ is equipped with a fixed ordering. Formally, an extraction oracle is a function

$$\begin{array}{rcl} \mathcal{O} : \mathcal{T} \times \mathcal{P}_E & \rightarrow & \mathcal{D} \\ (T, p) & \mapsto & D \end{array}$$

such that $D \subseteq \mathcal{F}_p$. Intuitively, $\mathcal{O}(T, p)$ returns a (possibly empty) set of candidate facts for predicate p extracted from the input text T. The oracle may be incomplete or return spurious facts, and no assumptions are made regarding determinism, completeness, or monotonicity of its behavior. Since at most one oracle call is performed per target predicate, the length of the extraction sequence is bounded by $|\mathcal{P}_E|$.

Prompt Transformations. We model prompt construction abstractly as text transformations. Let $\Gamma : \mathcal{T} \rightarrow \mathcal{T}$ be a global transformation encoding macro-task or domain-level instructions, and let $\gamma_p : \mathcal{T} \rightarrow \mathcal{T}$ be predicate-specific transformation encoding instructions for extracting predicate $p \in \mathcal{P}_E$. The extraction oracle is always applied to transformed texts of the form $\gamma_p(\Gamma(T))$, that is, when we write $\mathcal{O}(T, p)$ we actually mean $\mathcal{O}(\gamma_p(\Gamma(T)), p)$. Stated differently, prompt transformations are modeled as text-to-text functions applied to the input text before oracle invocation; for clarity of presentation, they are omitted from the notation.

Example 2 (Running example)**.** *Let Γ be the macro-task function mapping a text $\underline{t}$ to*

```
Follow my instructions on how to map the INPUT to the given OUTPUT format. The
INPUT text describes a (possibly layered) undirected graph.
INPUT: t     OUTPUT:
```

Let $\gamma_{\texttt{in\_layer/2}}$ be the transformation associated with `in_layer/2`, mapping a text $\underline{t}$ to

```
t in_layer(layer, node).
For each layer, list every node that belongs to that layer.
```

Let T be the following unstructured text document:

```
The graph has three layers (0 to 2), each containing three nodes. Layer zero
contains nodes n1, n2, and n3, while layer 2 contains nodes n7, and n9. There
is an edge between n1 and n9, and node n4 is connected to n2 and n8.
```

So, if we call $\mathcal{O}(T, p)$, we are actually calling $\mathcal{O}(T', p)$, where $T' = \gamma_p(\Gamma(T))$ is

```
Follow my instructions [···] a (possibly layered) undirected graph.
INPUT: The graph has [···] n2 and n8.      OUTPUT: in_layer(layer, node).
For each layer, list every node that belongs to that layer.
```

The expected output are facts `in_layer(0,n1)`, `in_layer(0,n2)`, `in_layer(0,n3)`, `in_layer(2,n7)`, *and* `in_layer(2,n9)`, *while facts like* `in_layer(2,n8)`, *and* `in_layer(1,n5)` *must not be produced in output as they are not explicitly mentioned in T. Specifically, we expect that the extraction process terminates reporting only facts that are actually present in the given input, avoiding any speculative interpolation of missing data.* ■

Logic Guided eXtraction (LGX). LGX proceeds over a fixed ordering $\mathcal{P}_E = \langle p_1, \ldots, p_n \rangle$ of target predicates. Each predicate p_i is associated with a logic program Π_{p_i}, which is applied after processing p_i, and with a possibly empty set $\mathcal{G}_{p_i}$ of *guards*, where each guard is a boolean conjunctive query (essentially, the body of a normal rule). A guard $g \in \mathcal{G}_{p_i}$ is said to be *monotone* if it does not contain negated atoms; in this case, if g is true in D (denoted $D \models g$), then $D' \models g'$ for every D' such that $D \subseteq D'$. We collectively refer to the ordering $\mathcal{P}_E$, the guards $\{\mathcal{G}_{p_i}\}_{i=1}^n$, and the programs $\{\Pi_{p_i}\}_{i=1}^n$ as a *LGX configuration*. Let $\mathcal{C}$ denote the domain of all such configurations, and let $C \in \mathcal{C}$ be a fixed configuration. LGX is the function

$$\begin{array}{rcl} \mathsf{LGX} : \mathcal{T} \times \mathcal{D} \times \mathcal{C} & \to & \mathcal{D} \\ (T, D_0, C) & \mapsto & D_n \end{array}$$

which maps an input text T, an initial database D_0, and a configuration $C \in \mathcal{C}$ to the final database D_n by computing, for all $i = 1, \ldots, n$, the following database:

$$D_i = \begin{cases} Ans_{\Pi_{p_i}}(D_{i-1} \cup \mathcal{O}(T, p_i)) & \text{if } D_{i-1} \models g \text{ for all } g \in \mathcal{G}_{p_i}, \\ D_{i-1} & \text{otherwise.} \end{cases} \tag{1}$$

Example 3 (Continuing Example 2)**.** *To illustrate the condition-based execution of LGX, consider the target predicates* $\mathcal{P}_E = \langle$`layer_size/2`, `in_layer/2`, `edge/2`$\rangle$. *Assume that, when invoked, the extraction oracle yields the following facts from the text:* $\mathcal{O}(T,$ `layer_size/2`$) =$ $\{$`layer_size(0,3)`, `layer_size(1,3)`, `layer_size(2,3)`$\}$, *and* $\mathcal{O}(T,$ `in_layer/2`$) =$ $\{$`in_layer(0,n1)`, `in_layer(0,n2)`, `in_layer(0,n3)`, `in_layer(2,n7)`, `in_layer(2,n9)`$\}$.

Let Π be the following sub-program:

```
r :  warning(unmatched_layer_size, (Layer,Size,Cnt)) :-  layer_size(Layer,Size),
        Cnt = #count{Node : in_layer(Layer,Node)}, Size != Cnt.
```

where r detects size discrepancy of the extracted layers. In LGX, we define explicit logic guards to control the flow. Let g be the guard `layer_size(_,_)` *(requiring layer information to exist), and g' be* `not warning(_,_)` *(requiring no layer anomalies). We define the LGX configuration C as:*

$$C = \big(\langle \texttt{layer\_size/2}, \texttt{in\_layer/2}, \texttt{edge/2} \rangle, \langle \emptyset, \{g\}, \{g'\} \rangle, \langle \emptyset, \{r\}, \emptyset \rangle\big)$$

Under this configuration, $\mathsf{LGX}(T, \emptyset, C)$ *evaluates the extraction dynamically. The guard g ensures that* `in_layer/2` *is extracted only if some layer size information was previously found. After invoking the oracle for* `in_layer/2`*, the framework applies rule r. Because there are missing nodes in the input text for layers 1 and 2, the solver derives* `warning(unmatched_layer_size, (1,3,0))` *and* `warning(unmatched_layer_size, (2,3,2))`.

Consequently, the guard g' evaluates to false. This short-circuits the admissibility check for `edge/2`*, entirely preventing the LLM invocation for this predicate and avoiding the generation of spurious edges on an inconsistent graph topology.* ■

3. Hierarchical Guided Extraction Framework

In condition-based frameworks like LGX, introduced in Section 2, extraction is guided by logical guards evaluated prior to each oracle call. While this successfully interleaves extraction and reasoning, preventing the blind extraction of all target predicates, it introduces a significant computational bottleneck. Specifically, the ASP solver must be invoked to evaluate the admissibility condition for every single candidate predicate, failing to exploit shared structural dependencies.

The article introducing LGX [8] addresses such a bottleneck by means of caching strategies for guard evaluation. Intuitively, in LGX (consecutive) predicates can be grouped by sharing monotone guards, so to evaluated them only for the first predicate in the group. Here we address the problem differently and introduce the Hierarchical Guided eXtraction (HGX) framework [1], in which the structural hierarchy of the domain actively controls and optimizes the extraction process. The key idea is to organize target predicates into a parent-child structure that acts as a macro-level router and structural filter [2]. By embedding broad logical dependencies, schema-level routing, and lexical pre-filtering directly into this structure, extraction decisions for sub-predicates are strictly contingent on the successful validation of their ancestors. As a result, entire branches of target predicates, or even entire domain schemas, can be structurally bypassed (pruned) without requiring additional guard evaluations.

A forest (over the target predicates $\mathcal{P}_E$) is a set of trees. Let $\mathcal{H}$ be the set of forests. For a forest $H \in \mathcal{H}$ and a predicate $p \in \mathcal{P}_E$, let $parent(p, H)$ denote the set comprising the parent of p in H (if any); note that $parent(p, H)$ is the empty set if p is a root, and a singleton otherwise.
A *HGX configuration* extends a LGX configuration $C \in \mathcal{C}$ with a forest $H \in \mathcal{H}$. In the following, we assume that $\mathcal{P}_E = \langle p_1, \ldots, p_n \rangle$ is actually derived by a depth-first traversal of H. To control the extraction flow, we define a predicate p_i as *admissible* at step i if both its structural dependency is met ($parent(p_i, H) \subseteq A_{i-1}$) and its local guards are satisfied ($D_{i-1} \models g$ for all $g \in \mathcal{G}_{p_i}$).
We define HGX as the function

$$\begin{array}{rcl} \mathsf{HGX} : \mathcal{T} \times \mathcal{D} \times \mathcal{C} \times \mathcal{H} & \rightarrow & \mathcal{D} \\ (T, D_0, C, H) & \mapsto & D_n \end{array}$$

which maps an input text T, an initial database D_0, and a configuration $(C, H) \in \mathcal{C} \times \mathcal{H}$ to the final database D_n. Starting with an empty set of processed predicates $A_0 = \emptyset$, the output is obtained by computing the sequence of pairs (A_i, D_i) for all $i = 1, \ldots, n$ via the following update rule:

$$(A_i, D_i) = \begin{cases} (A_{i-1} \cup \{p_i\}, Ans_{\Pi_{p_i}}(D_{i-1} \cup \mathcal{O}(T, p_i))) & \text{if } p_i \text{ is admissible,} \\ (A_{i-1}, D_{i-1}) & \text{otherwise.} \end{cases} \tag{2}$$

4. Implementation and Experiment

Algorithm 1 extends the LGX algorithm in [8] by introducing an additional set A to store admissible predicates. In the main loop of the algorithm, such a set is used to check if the parent of a predicate (if any) was found admissible in previous iterations. If not, the predicate is skipped (because it belongs to a subtree whose root was skipped). In more details, the set $A \subseteq \{p_1, \ldots, p_n\}$ stores the predicates that have been found admissible in previous iterations of the topological loop. When the loop reaches predicate p_i, the check $parent(p_i) \subseteq A$ (line 4) checks whether every ancestor of p_i in the hierarchy H has been marked admissible. If the parent is absent from A, then the entire subtree rooted at that parent has been pruned, so p_i cannot succeed and is skipped without evaluating its local guards $\mathcal{G}_{p_i}$ or invoking the ASP solver. When the parent is present, the algorithm proceeds exactly as in the original LGX procedure: it evaluates the guards (using monotonicity-based caching), short-circuits on the first false guard, and, if all guards hold, extracts facts via the oracle $\mathcal{O}$ and the program Π_{p_i}. Any newly derived

[1] The complete source code, along with the prompts and datasets used for our evaluation, is publicly available at https://github.com/massimob27/HGX/tree/main

[2] An example application file can be found at https://github.com/massimob27/HGX/blob/main/APPLICATION/G%2BLNRS_Hierarchical_Test.yml.

Algorithm 1: HGX-Exec(T, D_0, C, H): Hierarchical-guided extraction with structural pruning

Input : Text T, initial database D_0, configuration $(C, H) = (\langle p_1, \ldots, p_n \rangle, \{\mathcal{G}_{p_i}\}, \{\Pi_{p_i}\}, H)$
Output: Final database D

1 $(D, \mathcal{K}_m^\top, \mathcal{K}_m^\bot, \mathcal{K}_n^\top, \mathcal{K}_n^\bot) := (D_0, \emptyset, \emptyset, \emptyset, \emptyset)$ // result and cache initialization
2 $A := \emptyset$ // admissible predicates
3 **for** $i := 1$ **to** n **do** // process target predicates topologically
4 **if** $parent(p_i) \subseteq A$ **then** // structural pruning check
5 $admissible := \top$
6 **foreach** *guard* $g \in \mathcal{G}_{p_i}$ **do** // evaluate fine-grained local guards
7 **if** $g \in \mathcal{K}^\top_{mono(g)} \cup \mathcal{K}^\bot_{mono(g)}$ **then** // cache hit
8 $hold := (g \in \mathcal{K}^\top_{mono(g)})$ // retrieve answer from cache
9 **else** // cache miss
10 $hold := (q \in Ans_{\{q \leftarrow g\}}(D))$ // answer query
11 $\mathcal{K}^{hold}_{mono(g)} := \mathcal{K}^{hold}_{mono(g)} \cup \{g\}$ // cache answer
12 **if** $hold = \bot$ **then** // short-circuit on first false guard
13 $admissible := \bot$ // p_i is not admissible
14 **break** // skip remaining guards
15 **if** $admissible$ **then**
16 $D' := Ans_{\Pi_{p_i}}(D \cup \mathcal{O}(T, p_i))$ // extract and derive facts
17 **if** $D' \neq D$ **then** // if there are new facts
18 $(D, \mathcal{K}_m^\bot, \mathcal{K}_n^\top, \mathcal{K}_n^\bot) := (D', \emptyset, \emptyset, \emptyset)$ // update DB, invalidate cache
19 $A := A \cup \{p_i\}$ // update admissible predicates
20 **return** D

facts update the database D and invalidate cache entries that may have become stale; the predicate p_i is then added to A (line 19) so that its children can benefit from structural pruning in later iterations. Consequently, the only syntactic differences with respect to the algorithm in [8] are the initialization of A (line 2), the parent-check guard (line 4), and the update $A := A \cup \{p_i\}$ (line 19), which together enable aggressive hierarchical pruning while preserving the fine-grained, guard-based expressiveness of LGX.

We empirically evaluate HGX to assess its effectiveness in reducing ASP solver invocations while maintaining the exact same extraction accuracy as the LGX baseline. HGX is powered by clingo 5.8 [20] as the underlying ASP solver, and Meta Llama 3.1 [21] as the extraction oracle, deployed in two configurations: a lightweight version (8B, hereafter referred to as *Small*) and a medium-capacity version (70B, referred to as *Medium*). To minimize stochastic variance, we set the LLM sampling temperature to 0 and employ a persistent cache for oracle calls. This guarantees that identical queries yield the same candidate facts across different system runs, effectively isolating the impact of the logic-guided control flow on both extraction efficiency and quality.

Our evaluation relies on two primary benchmarks designed to assess different structural aspects of logic-guided extraction: broad inter-domain routing (**G + LNRS**) and deep intra-domain branching (**E**). **G + LNRS** is adapted from [7], and aggregates four distinct ASP domains (Labyrinth, Nomystery, Ricochet Robots, and Sokoban) along with the graph domain introduced for LGX [8]. This benchmark serves as a rigorous testbed for multi-domain scenarios. In this setting, an initial *macro-level routing* condition acts as a hierarchical guard, restricting all subsequent oracle calls to the specific puzzle schema and immediately pruning disjoint domains to prevent cross-domain noise. **E** is a single-domain use case about electrical grid topologies, and its designed to explicitly test the impact of HGX on a single, uniform domain. Unlike the macro-level splits in G + LNRS, this setting pushes the limits of the hierarchical approach by leveraging deep, shared predicate branches that unify common sub-components within a single context. This complex intra-domain structure heavily penalizes flat evaluation strategies, thereby highlighting the short-circuiting advantages of HGX when navigating deep hierarchies.

Table 1
Experimental results on Graph + Logic Puzzles (G + LNRS) and Electrical Grid (E) benchmarks. We report the number of ASP solver invocations ($\mathcal{S}$ Calls) alongside accuracy metrics (F1-Score and Perfect Rate).

Bench.	$\mathcal{O}$	Pipeline	$\mathcal{S}$ Calls	F1-Score	Perfect Rate
G + LNRS	Small (8B)	LGX	5133	0.820	0.212
		HGX	**377**	0.820	0.212
	Medium (70B)	LGX	5153	0.903	0.654
		HGX	**385**	0.903	0.654
E	Small (8B)	LGX	663	0.870	0.090
		HGX	**171**	0.870	0.090
	Medium (70B)	LGX	657	0.989	0.787
		HGX	**165**	0.989	0.787

Table 1 summarizes the experimental results on the multi-domain (G + LNRS) and single-domain (E) benchmarks. The empirical data supports two key observations regarding the impact of the HGX architecture:

- **Drastic Reduction in Solver Invocations.** Across both benchmark types and all oracle capacities, HGX achieves a massive reduction in the number of calls to the ASP solver. In the multi-domain G + LNRS benchmark, solver calls drop from over 5,100 in LGX to under 400 in HGX, a reduction of approximately 92%. Furthermore, even within the single, uniform domain of the Electrical Grid benchmark, structural pruning successfully reduces solver calls by nearly 74%. This confirms that HGX aggressively bypasses redundant guard evaluations at both the macro-routing and micro-branching levels.
- **Accuracy Equivalence.** The accuracy metrics (F1-Score and Perfect Rate) of HGX identically match those of LGX. This empirically verifies that our pruning strategy is strictly sound: it dramatically optimizes computational efficiency without compromising the completeness or quality of the extracted solution.

5. Conclusion

In this paper, we introduced the Hierarchical Guided eXtraction (HGX) framework, a novel neuro-symbolic architecture that optimizes the interaction between Large Language Models and Answer Set Programming (ASP) for structured data extraction. While previous condition-based pipelines, such as LGX, successfully interleaved reasoning and extraction, they suffered from significant computational overhead due to the exhaustive evaluation of flat logical guards. HGX addresses this bottleneck by natively embedding domain dependencies into a highly expressive parent-child hierarchy. By organizing target predicates into a forest-like structure, our framework actively leverages structural pruning. Crucially, if an ancestor node fails to be extracted, its entire dependent sub-tree is immediately bypassed, short-circuiting redundant queries to both the LLM oracle and the ASP solver. We empirically validated its effectiveness. Our experimental evaluation on complex, multi-domain benchmarks demonstrates that HGX achieves a massive reduction in ASP solver invocations, by up to 92%, while maintaining the exact same extraction accuracy and perfect rate as the baseline. This proves that structural pruning maximizes computational scalability without compromising the quality or completeness of the solution.

Declaration on Generative AI

During the preparation of this work, the author(s) used Google Gemini and GPT-5 in order to: Grammar and spelling check. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, A. Mian, A comprehensive overview of large language models, 2025. URL: https://doi.org/10.1145/3744746. doi:10.1145/3744746.

[2] D. Xu, W. Chen, W. Peng, C. Zhang, T. Xu, X. Zhao, X. Wu, Y. Zheng, E. Chen, Large language models for generative information extraction: A survey, CoRR abs/2312.17617 (2023).

[3] Y. Zhu, H. Yuan, S. Wang, J. Liu, W. Liu, C. Deng, H. Chen, Z. Liu, Z. Dou, J.-R. Wen, Large language models for information retrieval: A survey, 2025. URL: https://doi.org/10.1145/3748304. doi:10.1145/3748304.

[4] H. Jin, Y. Zhang, D. Meng, J. Wang, J. Tan, A comprehensive survey on process-oriented automatic text summarization with exploration of llm-based methods, 2024. doi:10.48550/ARXIV.2403.02901. arXiv:2403.02901.

[5] W. Zhang, X. Li, Y. Deng, L. Bing, W. Lam, A survey on aspect-based sentiment analysis: Tasks, methods, and challenges, 2023. doi:10.1109/TKDE.2022.3230975.

[6] Z. Yang, A. Ishay, J. Lee, Coupling large language models with logic programming for robust and general reasoning from text, in: ACL (Findings), Association for Computational Linguistics, 2023, pp. 5186–5219. URL: https://doi.org/10.18653/v1/2023.findings-acl.321. doi:10.18653/V1/2023.FINDINGS-ACL.321.

[7] M. Alviano, L. Grillo, F. Lo Scudo, L. A. R. Reiners, Integrating answer set programming and large language models for enhanced structured representation of complex knowledge in natural language, in: Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2025, Montreal, Canada, August 16-22, 2025, ijcai.org, 2025, pp. 4330–4338. URL: https://doi.org/10.24963/ijcai.2025/482. doi:10.24963/IJCAI.2025/482.

[8] M. Alviano, L. Grillo, N. Leone, F. Lo Scudo, Logic-guided data extraction with answer set programming and large language models, Theory Pract. Log. Program. 26 (2026). URL: https://www.dropbox.com/scl/fi/67tvxu84caen4z6djsk3f/ICLP26_LGX.pdf?rlkey=zb2fmv7tjakx5s05drxhyw3zn&dl=1, accepted at (ICLP 2026).

[9] R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, L. D. Raedt, Deepproblog: Neural probabilistic logic programming, in: NeurIPS, 2018, pp. 3753–3763.

[10] Z. Yang, A. Ishay, J. Lee, Neurasp: Embracing neural networks into answer set programming, in: C. Bessiere (Ed.), Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, ijcai.org, 2020, pp. 1755–1762. URL: https://doi.org/10.24963/ijcai.2020/243. doi:10.24963/IJCAI.2020/243.

[11] T. Eiter, T. Geibinger, N. Higuera, J. Oetsch, A logic-based approach to contrastive explainability for neurosymbolic visual question answering, in: IJCAI, ijcai.org, 2023, pp. 3668–3676.

[12] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, S. Gupta, B. P. Majumder, K. Hermann, S. Welleck, A. Yazdanbakhsh, P. Clark, Self-refine: Iterative refinement with self-feedback, in: A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, S. Levine (Eds.), Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023, 2023. URL: http://papers.nips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

[13] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, S. Yao, Reflexion: language agents with verbal reinforcement learning, in: A. Oh, T. Naumann, A. Globerson, K. Saenko,

M. Hardt, S. Levine (Eds.), Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023, 2023. URL: http://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

[14] V. Marek, M. Truszczyński, Stable models and an alternative logic programming paradigm, in: The Logic Programming Paradigm: a 25-year Perspective, 1999, pp. 375–398. doi:10.1007/978-3-642-60085-2_17.

[15] I. Niemelä, Logic programming with stable model semantics as a constraint programming paradigm, 1999. doi:10.1023/A:1018930122475.

[16] A. Calì, D. Calvanese, G. D. Giacomo, M. Lenzerini, On the role of integrity constraints in data integration, IEEE Data Engineering Bulletin 25 (2002) 39–45.

[17] N. Leone, T. Eiter, W. Faber, M. Fink, G. Gottlob, L. Granata, G. Greco, E. Kalka, G. Ianni, D. Lembo, M. Lenzerini, V. Lio, B. Nowicki, R. Rosati, M. Ruzzi, W. Staniszkis, G. Terracina, Data integration: a challenging ASP application, in: LPNMR, volume 3662 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 379–383.

[18] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, Y. Cao, React: Synergizing reasoning and acting in language models, in: The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023, OpenReview.net, 2023. URL: https://openreview.net/forum?id=WE_vluYUL-X.

[19] J. Arias, M. Carro, Z. Chen, G. Gupta, Constraint answer set programming without grounding and its applications, in: M. Alviano, A. Pieris (Eds.), Datalog 2.0 2019 - 3rd International Workshop on the Resurgence of Datalog in Academia and Industry co-located with the 15th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2019) at the Philadelphia Logic Week 2019, Philadelphia, PA (USA), June 4-5, 2019, volume 2368 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2019, pp. 22–26. URL: https://ceur-ws.org/Vol-2368/paper2.pdf.

[20] M. Gebser, R. Kaminski, T. Schaub, Complex optimization in answer set programming, 2011.

[21] L. Team, The llama 3 herd of models, CoRR abs/2407.21783 (2024).