Automatic Chain of Concepts: Conceptual Prompting for LLMs by Constructing Concept Trees

Hurile Borjigin[†1,2], Nishtha N. Vaidya[†1,2], Thomas A. Runkler[1,2] and Veronika Haderlein-Høgberg[1]

[1] *Siemens AG, Munich, Germany* [2] *Technical University of Munich*

Abstract

Large Language Models (LLMs) perform well on general NLP tasks but often underperform in domain-specific settings where proprietary knowledge and concept hierarchies are required. Chain of Concepts (CoC) mitigates this by injecting structured conceptual guidance via manually curated graphs, but expert curation is costly and hard to scale or reproduce. We propose Automatic Chain of Concepts (AutoCoC), a fully automated pipeline that (i) induces a rooted concept tree directly from domain documentation, (ii) enriches nodes with representative examples for in-context learning, and (iii) linearizes the tree into a pedagogically ordered prompt sequence via breadth-first traversal. We evaluate AutoCoC on OWL ontology generation (Turtle) across 10 domains, comparing against handcrafted CoC and reverse-engineered baselines. AutoCoC achieves the best mean composite score (0.832) and produces substantially richer ontologies (21.7 classes/file vs. 13.5 and 14.8). Semantic constraint compliance (OntoClean-inspired) is marginally higher (90.6% vs. 89.4%), while confidence is competitive (0.893 vs. 0.864), though per-domain confidence remains higher for the handcrafted baseline in several domains. A self-consistency analysis over 200 runs shows high concept-level stability (0.889) but moderate structural variability (0.625). These results suggest that automated concept-structure induction is a viable way to scale concept-based prompting for structured, domain-specific generation tasks without relying on expert-authored hierarchies. Overall, AutoCoC contributes a reproducible and scalable alternative to expert-authored concept hierarchies by automatically deriving domain-specific prompt structures from documentation, making concept-based prompting more practical for structured knowledge generation.

Keywords

Large Language Models, Knowledge Injection, Knowledge Engineering, Knowledge Graph, Ontology

1. Introduction

Large Language Models (LLMs) achieve strong performance across a wide range of natural language processing tasks, from summarization and translation to knowledge acquisition [1]. Much of this capability derives from the broad world knowledge captured during pre-training on large-scale open-domain corpora, enabling what have been termed emergent capabilities—the ability to solve tasks never explicitly seen during training [2]. Prompt-based techniques such as In-Context Learning (ICL) [3] and Chain-of-Thought (CoT) prompting [4] further amplify these abilities by guiding model behavior at inference time without modifying parameters: ICL enables pattern imitation from demonstrations, while CoT elicits multi-step reasoning.

Despite these strengths, LLMs remain limited by the coverage and currency of their training data. Domains such as healthcare, engineering, chemistry, and legal analysis rely on proprietary or rapidly evolving knowledge that is absent or underrepresented in public corpora. When the conceptual knowledge required to solve a task is missing, such as domain-specific concepts, their relations, and hierarchical organization. LLMs frequently hallucinate plausible but incorrect outputs or reproduce surface patterns without genuine understanding [1]. Bridging this *knowledge gap* is therefore a prerequisite for deploying LLMs reliably in specialized settings.

Joint Workshop on Statistics and Knowledge Integration for Logic, Learning, Ethical Decisions, and LLMs (SKILLED-LLMs 2026), co-located with FLoC 2026, Lisbon, Portugal

[†] Hurile Borjigin and Nishtha N. Vaidya contributed equally to this work. Hurile Borjigin conducted this research during his Master's thesis internship at Siemens AG and the Technical University of Munich.

hurile.borjigin@icloud.com (H. Borjigin[†]); nishtha.vaidya@siemens.com (N. N. Vaidya[†])

Several paradigms have been proposed to inject domain-specific knowledge into LLMs. Following the taxonomy of Song et al. [5], these methods fall into four categories: (i) *static knowledge embedding* through fine-tuning or continued pre-training, (ii) *modular adapters* that attach trainable auxiliary modules, (iii) *dynamic knowledge injection* via retrieval-augmented generation, and (iv) *prompt optimization*, which structures prompts to supply conceptual guidance at inference time. The first two require parameter updates and substantial training data; dynamic injection demands retrieval infrastructure and comprehensive knowledge bases. In contrast, prompt optimization is training-free, compatible with closed-source models, and can be updated instantly as domain knowledge evolves—making it particularly suited to low-resource and rapidly changing domains where the other paradigms are impractical.

Within prompt optimization, the Chain of Concepts (CoC) framework [6] demonstrated that injecting a structured hierarchy of domain concepts into prompts substantially improves LLM performance on constrained, domain-specific generation tasks. However, CoC relies on *manually curated* directed acyclic graphs (DAGs) authored by domain experts. This manual dependency creates three practical barriers: (a) it does not scale to large or evolving documentation corpora, (b) handcrafted structures are difficult to reproduce across domains and studies, and (c) subjective design decisions introduce inconsistencies in abstraction level, hierarchy depth, and relation typing.

To address these limitations, we propose **Automatic Chain of Concepts (AutoCoC)**, a fully automated pipeline that replaces expert-driven hierarchy design with algorithmic concept-tree induction from raw documentation. AutoCoC (i) recursively constructs a rooted concept tree using dual relevance scoring, (ii) enriches each node with retrieved examples for in-context learning, and (iii) serializes the tree into a pedagogically ordered prompt sequence via breadth-first traversal. The resulting prompts are injected at inference time to guide structured generation, requiring no model modification, no external retrieval system, and no domain-expert involvement.

We evaluate AutoCoC on OWL ontology generation in Turtle format across 10 diverse domains—a task that demands accurate concept hierarchies, domain-specific relations, and adherence to formal syntax and semantic constraints, and thus directly exposes LLM limitations that structured knowledge injection is designed to address. Semantic plausibility of the generated taxonomies is assessed using OntoClean-inspired meta-property constraints [7]; we note that OntoClean is used here as an *evaluation framework*, and integrating automated meta-property inference into the construction pipeline itself is left for future work. The main contributions of this paper are:

1. A **fully automated concept-tree construction** algorithm that extracts and hierarchically organizes domain concepts from unstructured documentation, removing the manual curation bottleneck of prior concept-based prompting methods.
2. A **dual-scoring and dynamic reparenting** mechanism that refines concept placement during tree construction based on domain relevance and parent–child coherence scores.
3. An **example-enrichment and BFS serialization** procedure that produces a reusable, ordered prompt sequence for structured generation tasks.
4. An **empirical evaluation** on OWL ontology generation across 10 domains, demonstrating improved structural coverage and competitive semantic quality compared to handcrafted and reverse-engineered baselines, together with a stability analysis over 200 runs quantifying concept-level and structural consistency.

2. Related work

2.1. Large Language Models: Capabilities and Limitations

Large language models (LLMs) have shown strong performance across a broad range of natural language processing tasks, including text classification, question answering, summarization, translation, and dialogue generation [1, 8, 9]. Their effectiveness is commonly attributed to large-scale pre-training, contextual representation learning, and the ability to adapt to new tasks through zero-shot, few-shot,

and instruction-based prompting [3, 10]. Scaling has also been associated with emergent capabilities that are not always observed in smaller models [2].

Despite these capabilities, LLMs remain limited in settings that require reliable reasoning, factual grounding, and specialized domain knowledge. Prior work has shown that LLMs may generate fluent but unsupported content, commonly described as hallucination, and may still fail on seemingly simple reasoning tasks [11, 12]. Other studies further distinguish between linguistic competence and more general functional reasoning, suggesting that fluent language generation does not necessarily imply robust conceptual understanding [13].

A central limitation concerns the representation and use of knowledge inside LLMs. Research distinguishes between *parametric knowledge*, implicitly stored in model weights, and *non-parametric knowledge*, explicitly represented in external res latexources such as documents, knowledge graphs, ontologies, or symbolic systems [14, 15]. While parametric knowledge supports broad generalization, it is difficult to inspect, update, and validate. Recent work further suggests that models may encode knowledge internally without reliably expressing it in their outputs [16]. These limitations motivate hybrid approaches that combine LLMs with structured external knowledge sources [17].

2.2. Knowledge Injection Methods for Large Language Models

Several approaches have been proposed to inject domain-specific knowledge into LLMs beyond general-purpose pre-training. Song et al. [5] categorize knowledge injection methods into four main paradigms: dynamic knowledge injection, static knowledge embedding, modular adapters, and prompt optimization.

Dynamic knowledge injection augments inference with externally retrieved information. This enables flexible updates without retraining, but requires retrieval infrastructure and can introduce additional latency. Static knowledge embedding incorporates domain knowledge into model parameters through fine-tuning or continued pre-training, often improving domain performance but increasing training cost and reducing update flexibility. Modular adapters provide parameter-efficient domain adaptation through trainable auxiliary modules, although they still require model access, architectural integration, and domain-specific training data [18].

Prompt optimization injects knowledge through the input context rather than through model parameters or external retrieval. Prompt-based methods use instructions, demonstrations, reasoning scaffolds, or structured context to guide model behavior at inference time. Chain-of-thought prompting and related approaches show that intermediate reasoning steps can improve performance on complex tasks [4]. Later work has explored automatic construction and selection of reasoning demonstrations, reducing the amount of manual prompt engineering required [19, 20].

This work focuses on prompt optimization because it is suitable for low-resource and rapidly evolving domains where fine-tuning or retrieval-based systems may be impractical. Prompt-based knowledge injection is lightweight, training-free, compatible with closed-source models, and does not require modification of model parameters. However, it also depends heavily on how domain knowledge is represented and serialized in the prompt. This makes the design of compact, structured, and reproducible knowledge representations an important problem.

2.3. Chain of Concepts for Prompt-Based Knowledge Injection

The Chain of Concepts (CoC) framework proposes the use of an explicit concept hierarchy as a prompt-level knowledge structure for domain-specific generation tasks [6]. Instead of relying only on task instructions or isolated examples, CoC organizes domain concepts and their relations into a structured representation that can be injected into the prompt. In this setting, the concept structure acts as an intermediate layer between domain knowledge and the LLM, guiding generation toward concepts, relations, and constraints that are relevant to the target domain.

CoC is closely related to prompt optimization because it does not require model fine-tuning or modification of model parameters. It also differs from retrieval-based approaches because the injected knowledge is not selected dynamically from an external corpus at inference time, but is instead provided

through a pre-defined concept structure. This makes CoC a useful reference point for studying structured prompt-level knowledge injection.

However, the original CoC formulation relies on manually constructed concept structures, typically authored by domain experts. This introduces practical limitations for broader use: manual construction is difficult to scale to large or frequently updated documentation sources, the resulting structures may be hard to reproduce across domains or studies, and design choices such as abstraction level, hierarchy depth, and relation typing can vary between annotators. These limitations motivate automated ap

2.4. Knowledge Representation: From Graphs to Tree Structures

Knowledge graphs, ontologies, and directed acyclic graphs (DAGs) are widely used for structured knowledge representation because they can model entities, concepts, relations, and constraints in an explicit form [21, 22]. These representations support ontology engineering, semantic reasoning, information retrieval, and knowledge-intensive LLM applications. In the context of LLMs, structured knowledge can help constrain generation, provide domain grounding, and improve semantic consistency [23, 17].

Recent work has explored the use of LLMs for ontology learning, taxonomy construction, and concept hierarchy induction. LLMs have been applied to ontology learning tasks such as term typing, taxonomy discovery, and relation extraction [24]. Other studies investigate whether LLMs can directly construct concept hierarchies or taxonomies through prompting [25, 26]. Related work on taxonomy induction also studies zero-shot hypernymy prediction, layer-wise taxonomy construction, and tree reconcili latexation from language model predictions [27, 28, 29]. These studies show that LLMs can support hierarchy construction, but they also highlight recurring issues such as structural inconsistency, hallucinated relations, and the need for post-processing.

Graph-based representations remain highly expressive, but their expressiveness introduces practical challenges for prompt-based knowledge injection. Traversing, linearizing, and selecting relevant subgraphs for LLM prompting requires design decisions that can complicate automation and evaluation. Graphs and DAGs may also introduce multiple parent–child paths, making prompt serialization less direct.

To balance expressiveness and practicality, this work adopts a concept tree as a simplified knowledge representation. Tree structures reduce construction and traversal complexity, simplify serialization for prompting, and provide a tractable abstraction for automated generation and evaluation. This choice assumes that domain knowledge can be approximated hierarchically through a primary conceptual backbone. While this reduces representational flexibility compared with full graph structures, it intentionally constrains the design space and improves reproducibility across domains.

Ontology Generation as a Knowledge Injection Use Case Ontology generation provides a suitable downstream task for evaluating structured knowledge injection because it requires both conceptual correctness and formal representation. In particular, OWL ontology generation in Turtle format requires accurate concept hierarchies, domain-specific relations, and adherence to syntactic and semantic constraints. Prior work has studied LLM-assisted ontology engineering, including ontology generation from requirements, ontology learning from text, and LLM-supported modelling suggestions [30, 31, 32]. These studies suggest that LLMs can assist ontology construction, but also show that generated ontologies require careful evaluation and structural guidance.

In this work, the concept tree serves as an intermediate representation of domain knowledge that is injected into prompts to guide the generation of syntactically valid and semantically consistent OWL ontologies. This use case addresses LLM limitations in proprietary and low-resource domains while leveraging their natural language generation capabilities.

3. Methodology: Automatic Chain of Concepts

3.1. Overview of Automatic Chain of Concepts (AutoCoC)

AutoCoC is an automated extension of the Chain-of-Concepts framework for structured knowledge injection into large language models. It replaces manual hierarchy design with a fully automated pipeline that extracts conceptual structures from domain documentation and converts them into pedagogically ordered prompt sequences. The objective can be formalized as finding the optimal prompt sequence

$$\mathbf{p}^* = \arg\min_{\mathbf{p}} \mathcal{L}(M([\mathbf{p}, \mathbf{x}]; \theta), \mathbf{y}^*), \tag{1}$$

where $\mathbf{p}^*$ is the optimized prompt, $\mathbf{x}$ is the task input, θ are the fixed model parameters, and $\mathcal{L}$ measures prediction error against the target $\mathbf{y}^*$.

The pipeline consists of two stages:

1. **Automated Concept Tree Construction**: Recursively extracts concepts, hierarchical relations, and node attributes from documentation using LLM calls, then enriches nodes with retrieved examples.
2. **Structured Prompt Generation**: Traverses the tree to produce a sequence of prompts that introduce concepts in order of increasing specificity, respecting dependencies.

By automating both stages, AutoCoC achieves scalable, reproducible, and semantically consistent knowledge injection for proprietary or low-resource domains.

3.2. Recursive Top-Down Concept Tree Construction

The core of AutoCoC is the recursive algorithm (Algorithm 1) that builds a rooted, acyclic concept tree from documentation D and a domain context Δ. Starting from a user-specified root concept r, the algorithm incrementally discovers child concepts while respecting a maximum depth K and a total LLM call budget M.

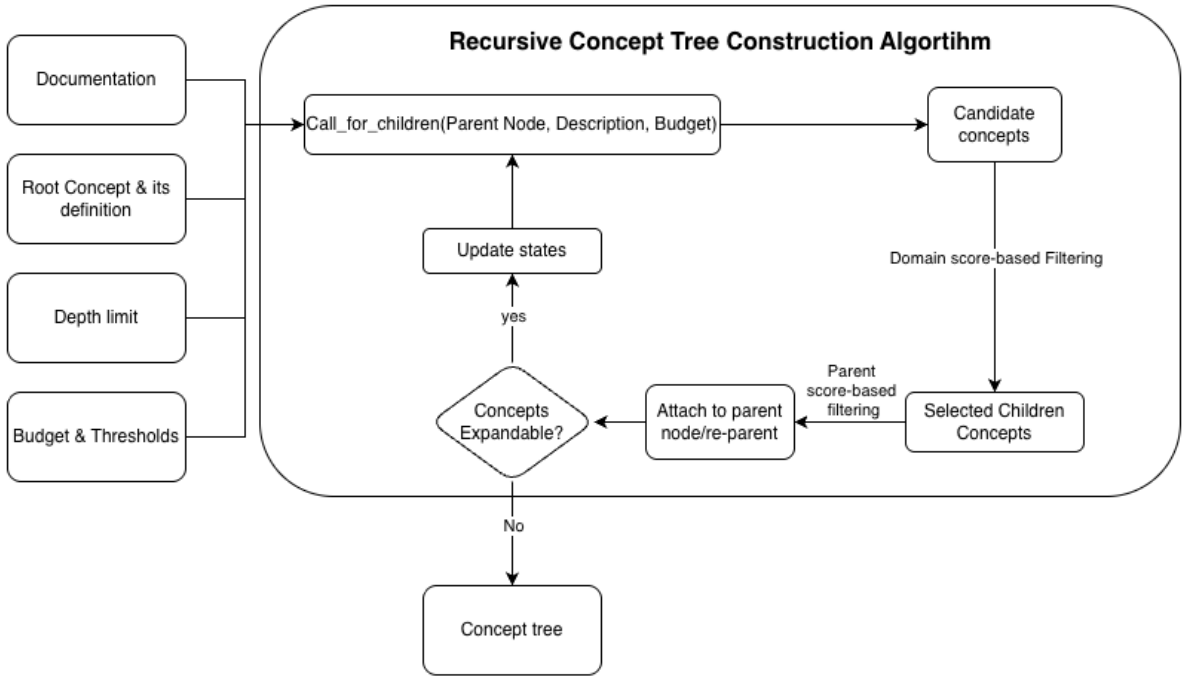


Figure 1: Concept Tree Construction Workflow

Algorithm 1 Automated Recursive Concept Tree Builder

Require: Documentation D, domain context Δ, root r, depth limit K, call budget M, policy thresholds
Ensure: Rooted concept tree T

1: Initialize state (node index, parent map, expanded set, remaining budget M)
2: **procedure** EXPAND(p, d)
3: **if** $d \geq K$ **or** $p \in$ expanded **or** $M = 0$ **then**
4: **return**
5: **end if**
6: $M \leftarrow M - 1$
7: $C \leftarrow$ LLMCALLFORCHILDREN($p,\ D,\ \Delta$) ▷ child u, edge score s_p, domain score s_d, relation rel
8: **for all** $(u, s_p, s_d, rel) \in C$ **do**
9: **if** TOOWEAK(s_p) **or** CREATESCYCLE(p, u) **then**
10: **continue**
11: **end if**
12: **if** u unattached **then**
13: ATTACH($p \rightarrow u,\ rel,\ s_p$)
14: **else if** STRONGEREDGE(s_p) **then**
15: REATTACH($p \rightarrow u,\ rel,\ s_p$)
16: **end if**
17: Record s_d for u
18: **end for**
19: $W \leftarrow$ children of p passing expand policy (maximal s_d, ties allowed)
20: **for all** $u \in W$ **do**
21: EXPAND($u,\ d + 1$)
22: **end for**
23: **end procedure**
24: EXPAND($r, 0$); **return** r

For each parent, the LLM proposes candidate children together with relation types and two independent scores (detailed below). Weak or cyclic edges are discarded; stronger edges trigger dynamic reattachment. Only the most domain-relevant children (selected by s_d) are expanded further.

3.3. Dual Scoring System (generated in line 7 in Algorithm 1)

To separate attachment decisions from expansion decisions, AutoCoC uses a dual-scoring mechanism produced by the LLM in every child-generation call (in line 7 in Algorithm 1):

- $score\_parent$: Strength of the hierarchical (taxonomic/definitional) relationship between parent and candidate child.
- $score\_domain$: Overall relevance and centrality of the candidate within the target domain Δ.

Raw scores are mapped to semantic categories via the thresholds in Table 1. The attachment policy uses only $score\_parent$ (via TOOWEAK and STRONGEREDGE), while the expansion policy uses only $score\_domain$.

This decoupling ensures structural coherence (strong hierarchies) while focusing computational budget on domain-central concepts.

3.4. Dynamic Reparenting (corresponding to line $8 - 16$ in Algorithm 1)

During recursion the same concept may be proposed under multiple parents. When a stronger $score\_parent$ edge is discovered for an already-attached node u, the algorithm performs dynamic reparenting: it detaches the entire subtree rooted at u from its current parent and reattaches it to

Table 1
Dual-score scheme with semantic interpretations

Range	$score\_parent$	$score\_domain$
0.90–1.00	Definitional/taxonomic direct child; attachment almost certain.	Core domain concept; highest expansion priority.
0.70–0.89	Standard direct child.	Important and frequently referenced.
0.40–0.69	Semantically related but secondary.	Supportive/auxiliary concept.
< 0.40	Marginal/weak relation.	Peripheral; rarely expanded.

the new parent (subject to cycle prevention). This mechanism continuously optimizes the hierarchy, correcting early suboptimal placements and converging toward the most semantically coherent tree without violating acyclicity.

3.5. CoC Generation from the Concept Tree

Once the tree is complete, a breadth-first traversal converts it into the final AutoCoC prompt sequence:

1. Extract levels via BFS.
2. Generate a foundation prompt for the root that defines overall syntax and rules.
3. For each subsequent level, sort concepts by $score\_domain$ and produce one prompt per concept containing its definition, relations, canonical examples, and prerequisites.

All LLM calls for prompt generation use temperature 0.2 to ensure precise, consistent syntax specifications. The resulting sequence is output in human-readable, JSON, and chat-history formats suitable for downstream use.

4. Experiments

4.1. Datasets

Three datasets were used to construct the concept tree and generate prompts in AutoCoC. The datasets cover both expert-structured knowledge and source documentation, allowing the method to be evaluated under refinement and bottom-up induction settings. The processed datasets and generated concept-tree artifacts can be made available upon acceptance or provided by the authors upon reasonable request.

- **Handcrafted CoC (Baseline)**: An expert-curated Chain of Concepts containing 3,495 tokens. This dataset serves as the reference structure and is used without additional preprocessing, providing a manually organized baseline for comparison.
- **Reverse-Engineered CoC**: The handcrafted baseline was also used as input to AutoCoC, with the same 3,495-token content, to evaluate whether the method can refine or reconstruct an existing structured knowledge source. This setting tests AutoCoC's compatibility with previously curated concept hierarchies.
- **OWL Official Documentation**: The full OWL specification, containing 43,757 tokens, was used together with Turtle serialization examples containing 2,596 tokens. This dataset represents a larger unstructured technical source and supports concept-node example retrieval during tree construction.

4.2. OWL Generation

The downstream task is the generation of OWL ontologies serialized in Turtle format from natural-language domain descriptions. The model must produce machine-interpretable structures containing classes, properties, and logical axioms.

This task was selected because it demands strict structural and semantic fidelity, unlike open-ended generation. Validity criteria include:

- Syntactic correctness (valid Turtle).
- Logically sound hierarchies (e.g., correct `subClassOf` relations).
- Domain accuracy reflecting expert knowledge.

Ontology generation tests whether structured knowledge injection via AutoCoC enables LLMs to produce valid, scalable ontologies while minimizing hallucinations and syntax errors common in unstructured prompting.

4.3. Experimental Protocol

All experiments used GPT-4o (OpenAI) as the fixed LLM to isolate the effect of prompting strategies.

Concept-Tree Generation Protocol For each domain, the concept-tree generation process was repeated 200 times with identical inputs. Statistical analysis across runs quantifies self-consistency and stability.

Downstream Task Protocol Ontology generation was evaluated on 10 diverse domains (Table 4). For each domain, 30 independent generations were performed and averaged.

4.4. Evaluation Metrics

4.4.1. Concept-Tree Self-Consistency

Self-consistency is measured across the 200 repeated generations per domain using:

- **Concept Stability Score**: fraction of concepts appearing in $\geq 90\%$ of runs,

$$\text{Stability} = \frac{|\{\text{concepts in } \geq 90\% \text{ runs}\}|}{|\text{union of concepts} - \text{rare concepts}|}.$$

- **Concept Frequency**: $\text{Freq}(c_i) = \frac{\text{count}(c_i)}{200 \text{ runs}}$.
- **Concept Variance**: standard deviation of concept counts across runs.
- **Core vs. Peripheral Ratio**: $\frac{|\{c:\text{Freq}(c)>0.5\}|}{|\{c:\text{Freq}(c)\leq 0.5\}|}$.
- **Edge Stability Score**: fraction of edges appearing in $\geq 95\%$ of runs (analogous formula).
- **Parent-Child Stability**: fraction of hierarchical pairs appearing in $\geq 95\%$ of runs.

Higher stability scores and lower variance indicate reliable, reproducible knowledge extraction.

4.4.2. Downstream Ontology Quality

Ontology quality is assessed along two axes: semantic correctness (OntoClean) and syntactic validity. The full OntoClean background and evaluation criteria are provided in Appendix A.5.

Semantic Correctness (OntoClean) An LLM evaluator assigns OntoClean meta-properties to each class ($+I/\sim I$, $+U/\sim U$, $+R/\sim R$, $+D$) with confidence $\gamma(c) \in [0, 1]$, using class definitions, hierarchy, and siblings. Inheritance constraints are then checked for four violation types (identity, unity, rigidity, dependence).

Key metrics over N generated ontologies:

$$V_{\text{total}} = \sum_{i=1}^{N} \sum_{(p,s)} \mathbb{I}_{\text{viol}}(p, s), \quad V_{\text{avg}} = \frac{V_{\text{total}}}{N},$$

$$\bar{\gamma} = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \gamma(c),$$

plus minimum/maximum confidence, standard deviation σ_γ, label distributions $P_{m,\ell}$, and per-violation-type frequencies f_t.

Lower V_{avg} and higher $\bar{\gamma}$ indicate stronger ontological soundness.

Syntactic Validity and Structural Metrics Each ontology is parsed with a standards-compliant Turtle parser. Success rate is

$$R_{\text{succ}} = \frac{F_{\text{succ}}}{N},$$

where F_{succ} is the number of parseable files.

Additional structural metrics include total/average number of classes (C_{total}, C_{avg}) and the fraction of classes successfully evaluated by the OntoClean pipeline. Evaluation runtime per file (T_{avg}) is also recorded.

Together, these metrics provide a comprehensive, multi-dimensional comparison of prompting strategies and knowledge-injection methods.

5. Results

This section reports empirical results of the AutoCoC pipeline. We evaluate (i) concept trees generated from distinct knowledge sources and (ii) the self-consistency of recursive tree construction. Downstream ontology generation results compare AutoCoC against handcrafted and reverse-engineered baselines across ten domains.

5.1. Concept Tree Generation from Distinct Knowledge Sources

Concept trees were induced from two data regimes. First, the reverse-engineered CoC was used as a handcrafted baseline to test knowledge refinement. This input contained 3,495 tokens and produced a 9-node concept tree with a 5,488-token AutoCoC. The generated tree preserves the core abstractions of the baseline while introducing refined intermediate concepts.

Second, the OWL official documentation was used to test bottom-up induction from unstructured technical text. The input consisted of the full OWL specification with 43,757 tokens, together with Turtle examples containing 2,596 tokens. This setting produced a 19-node concept tree and a 9,541-token AutoCoC. Compared with the reverse-engineered CoC, the resulting tree is deeper and richer, capturing major OWL abstractions from the documentation.

Together, these experiments show that AutoCoC can support both refinement of expert-structured knowledge and coherent hierarchy induction from large technical documentation.

5.2. Self-Consistency of Concept Tree Construction

Self-consistency was quantified over 200 independent runs with identical inputs.

The concept frequency distribution is strongly bimodal. Using a 0.5 threshold:

- Core concepts ($> 50\%$ frequency) comprise 69.2% of unique concepts.
- Peripheral concepts ($\leq 50\%$) account for the remaining 30.8%.

Stability metrics reveal:

- Concept Stability Score: 0.889 (high semantic reproducibility).
- Edge Stability and Parent–Child Stability: both 0.625 (moderate structural variability).
- Concept count: tightly clustered around a mode of 9 ($\sigma = 0.55$).

AutoCoC thus achieves robust core semantics with controlled flexibility in hierarchical organization.

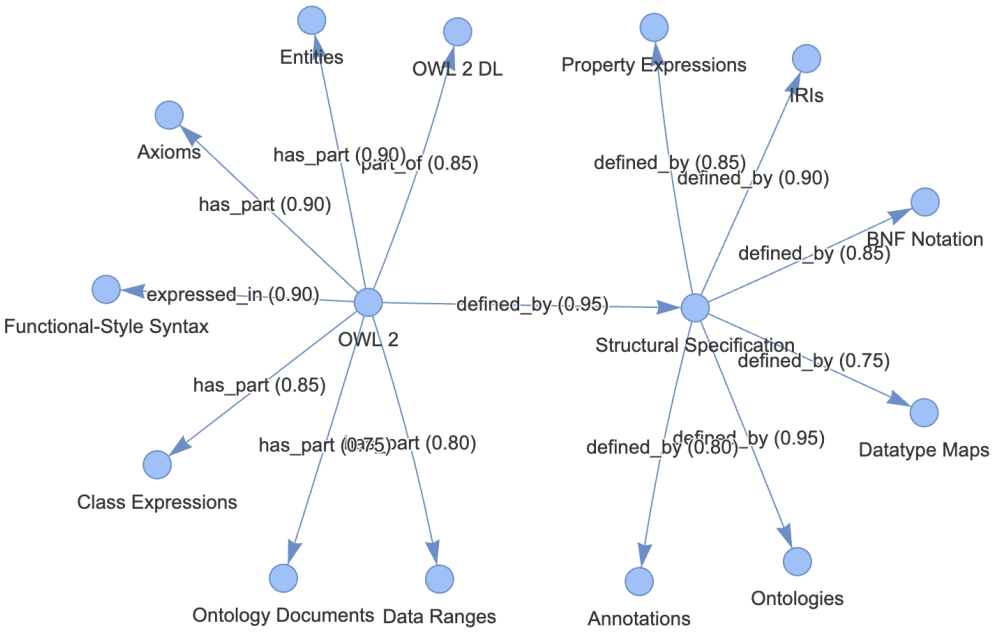


Figure 2: Concept tree generated from the full OWL specification documentation.

5.3. Downstream Ontology Generation Results on 10 Domains

Figure 4 compares AutoCoC, handcrafted CoC, and reverse-engineered CoC across ten domains on three metrics: OntoClean compliance rate, meta-property confidence score, and average number of OWL classes per file.

AutoCoC leads in structural richness (9/10 domains) and OntoClean compliance (4/10 domains), while the handcrafted baseline excels in confidence (5/10 domains).

Aggregated results are summarized in Table 2. AutoCoC achieves the highest mean confidence score (0.893) and the highest OntoClean compliance rate (90.59%) across ten domains. It also produces the richest ontology structure, with 21.7 classes per file on average, compared with 13.5 for the handcrafted CoC baseline and 14.8 for the reverse-engineered CoC setting. This structural gain comes with a higher mean evaluation time of 43.78 seconds, compared with 25.15 seconds for the baseline and 20.28 seconds for the reverse-engineered setting. The original bar plots are provided in Appendix B.

Table 2
Average performance comparison across ten domains.

Method	Confidence	OntoClean	Classes/file	Eval. time (s)
AutoCoC	**0.893**	**90.59%**	**21.7**	43.78
Handcrafted CoC	0.864	88.62%	13.5	25.15
Reverse-engineered CoC	0.831	89.41%	14.8	**20.28**

Overall, AutoCoC provides the strongest balance between semantic correctness and representational capacity in downstream ontology generation.

6. Conclusions

This paper addressed the question of whether concept-based prompting for domain-specific LLM tasks can be fully automated without sacrificing semantic quality. We introduced Automatic Chain of Concepts (AutoCoC), a pipeline that induces a rooted concept tree from unstructured documentation

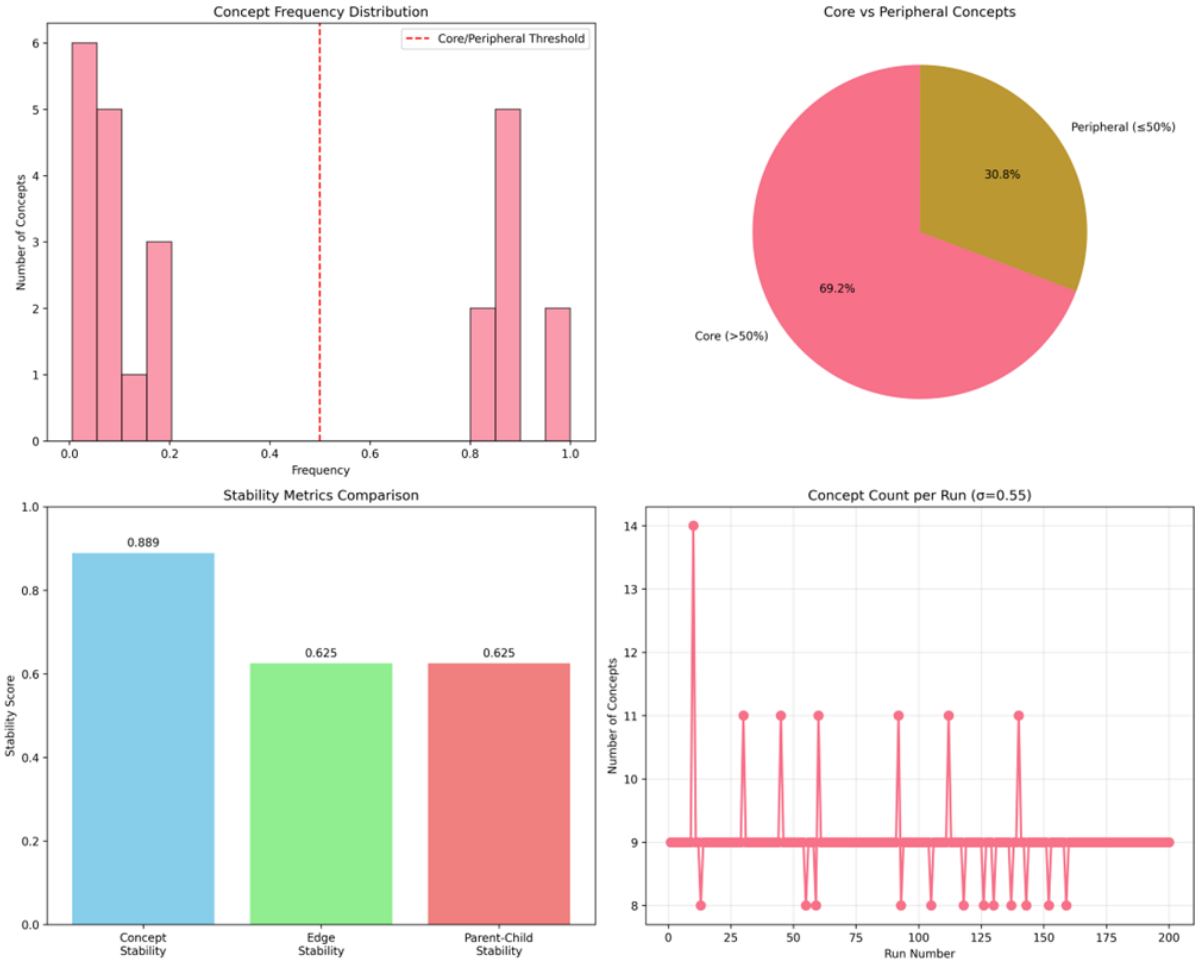


Figure 3: Self-consistency analysis of concept tree construction across 200 runs. The top row shows concept recurrence statistics, while the bottom row reports stability metrics and tree size variability.

through recursive expansion with dual relevance scoring, enriches nodes with retrieved examples, and serializes the result into a structured prompt sequence via breadth-first traversal. AutoCoC removes the dependency on manually authored concept hierarchies that has limited the scalability and reproducibility of prior concept-based prompting methods.

Evaluated on OWL ontology generation in Turtle format across ten domains, AutoCoC achieves the highest mean composite score (0.832 vs. 0.729 for the handcrafted baseline and 0.726 for the reverse-engineered baseline), driven primarily by a substantial gain in structural richness: 21.7 classes per ontology file compared to 13.5 and 14.8, respectively. Mean OntoClean-inspired compliance is marginally higher (90.6% vs. 89.4%), and mean meta-property assignment confidence is competitive (0.893 vs. 0.864), though the handcrafted baseline retains higher per-domain confidence in five of ten domains. These results indicate that AutoCoC's principal advantage lies in representational coverage and overall composite quality rather than uniform semantic superiority across all domains and metrics.

Self-consistency experiments over 200 runs reveal a clear separation between concept-level stability (0.889) and structural stability (0.625). Core domain concepts are highly reproducible across runs, but the hierarchical arrangement of those concepts exhibits moderate variability. This suggests that while the pipeline reliably identifies *what* to include, the *how* of hierarchical organization remains sensitive to stochastic LLM behavior—an important target for future improvement.

Limitations. Several factors temper the generality of these findings. First, the evaluation is restricted to a single downstream task (OWL ontology generation) and a single LLM; whether AutoCoC's benefits transfer to other structured-generation tasks or model families is an open question. Second, no statistical

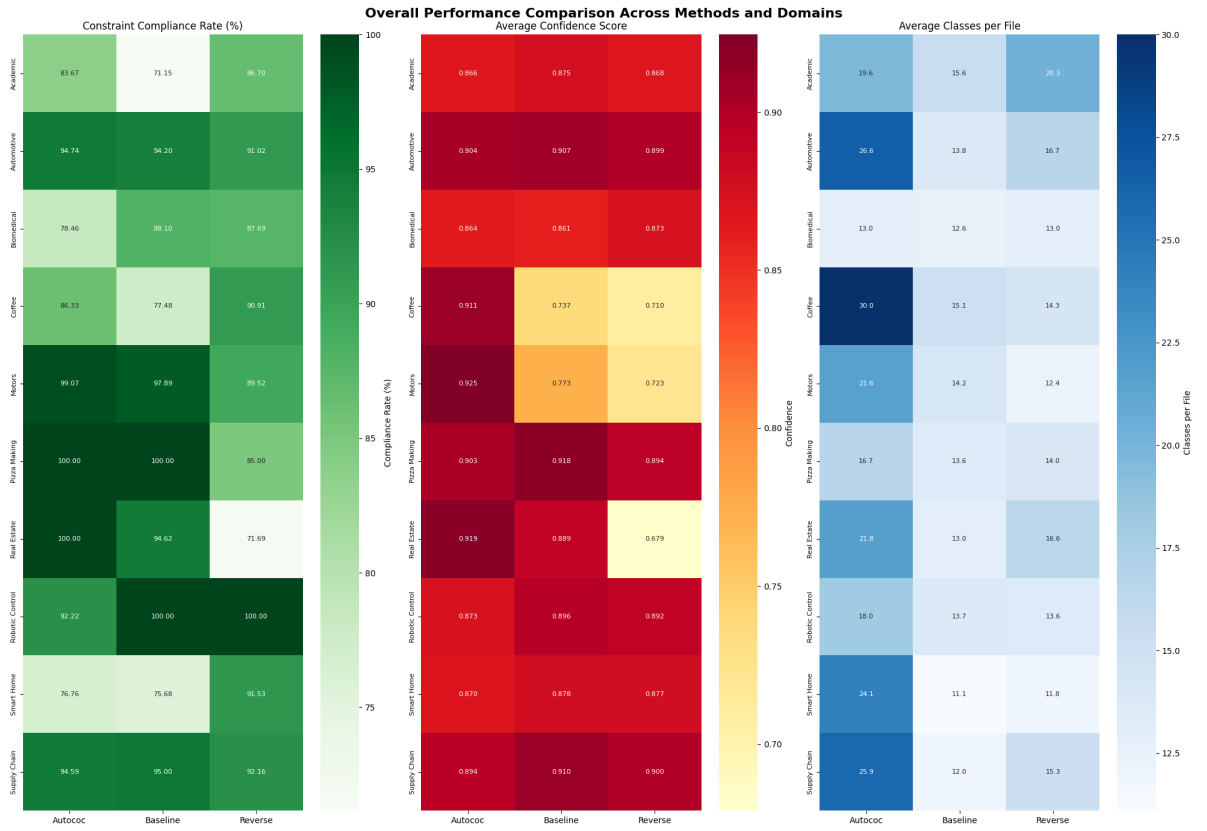


Figure 4: Overall performance comparison across methods and domains. Left: OntoClean constraint compliance rate. Middle: average confidence score of meta-property assignments. Right: average number of OWL classes per generated ontology file.

significance tests were applied to cross-domain metric comparisons, and the compliance margin over the handcrafted baseline is narrow ($\sim$1.2 percentage points). Third, the pipeline incurs an approximately 40% increase in evaluation runtime relative to using pre-built prompts—a manageable but non-trivial cost. Finally, the current evaluation uses OntoClean meta-properties as a post-hoc quality lens; integrating automated meta-property inference directly into the tree-construction loop could strengthen semantic guarantees but remains unimplemented.

Future work. Three directions appear most promising. (1) *Structural stability*: consensus-tree aggregation or constrained decoding methods could reduce hierarchical variability across runs without sacrificing concept diversity. (2) *Task and model generalization*: evaluating AutoCoC on additional structured-generation tasks (e.g., knowledge-graph completion, database schema design) and across multiple LLM families would clarify the scope of its advantages. (3) *Semantic validation integration*: incorporating OntoClean-style meta-property checking into the construction pipeline itself, rather than using it only for evaluation, could enable the tree-building process to self-correct semantic constraint violations during concept placement. In summary, the main contribution and novelty of AutoCoC lie in replacing manual concept-hierarchy design with an automated, documentation-driven construction process that preserves competitive semantic quality while improving representational coverage and reproducibility for domain-specific structured generation.

Declaration on Generative AI

During the preparation of this work, the author(s) used GPT-5 and Grammarly in order to: Grammar and spelling check. After using these tool(s)/service(s), the authors reviewed and edited the content as

needed and take 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, 2024. URL: http://arxiv.org/abs/2307.06435. doi:10.48550/arXiv.2307.06435. arXiv:2307.06435 [cs].

[2] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, W. Fedus, Emergent Abilities of Large Language Models, 2022. URL: http://arxiv.org/abs/2206.07682. doi:10.48550/arXiv.2206.07682. arXiv:2206.07682 [cs].

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language Models are Few-Shot Learners, 2020. URL: http://arxiv.org/abs/2005.14165. doi:10.48550/arXiv.2005.14165. arXiv:2005.14165 [cs].

[4] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, D. Zhou, Chain-of-Thought Prompting Elicits Reasoning in Large Language Models, 2023. URL: http://arxiv.org/abs/2201.11903. doi:10.48550/arXiv.2201.11903. arXiv:2201.11903 [cs].

[5] Z. Song, B. Yan, Y. Liu, M. Fang, M. Li, R. Yan, X. Chen, Injecting Domain-Specific Knowledge into Large Language Models: A Comprehensive Survey, 2025. URL: http://arxiv.org/abs/2502.10708. doi:10.48550/arXiv.2502.10708. arXiv:2502.10708 [cs].

[6] N. N. Vaidya, T. Runkler, T. Hubauer, V. Haderlein-Hoegberg, M. M. Brandt, Conceptual In-Context Learning and Chain of Concepts: Solving Complex Conceptual Problems Using Large Language Models, 2024. URL: http://arxiv.org/abs/2412.15309. doi:10.48550/arXiv.2412.15309. arXiv:2412.15309 [cs].

[7] Y. Zhao, N. Vetter, K. Aryan, Using Large Language Models for OntoClean-based Ontology Refinement, 2024. URL: http://arxiv.org/abs/2403.15864. doi:10.48550/arXiv.2403.15864. arXiv:2403.15864 [cs].

[8] J. Yang, H. Jin, R. Tang, X. Han, Q. Feng, H. Jiang, B. Yin, X. Hu, Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond, 2023. URL: http://arxiv.org/abs/2304.13712. doi:10.48550/arXiv.2304.13712. arXiv:2304.13712 [cs].

[9] W. Hariri, Unlocking the Potential of ChatGPT: A Comprehensive Exploration of its Applications, Advantages, Limitations, and Future Directions in Natural Language Processing, 2025. URL: http://arxiv.org/abs/2304.02017. doi:10.48550/arXiv.2304.02017. arXiv:2304.02017 [cs].

[10] S. Sivarajkumar, M. Kelley, A. Samolyk-Mazzanti, S. Visweswaran, Y. Wang, An Empirical Evaluation of Prompting Strategies for Large Language Models in Zero-Shot Clinical Natural Language Processing, 2023. URL: http://arxiv.org/abs/2309.08008. doi:10.48550/arXiv.2309.08008. arXiv:2309.08008 [cs].

[11] A. Alansari, H. Luqman, Large Language Models Hallucination: A Comprehensive Survey, 2025. URL: http://arxiv.org/abs/2510.06265. doi:10.48550/arXiv.2510.06265. arXiv:2510.06265 [cs].

[12] A. Malek, J. Ge, N. Lazic, C. Jin, A. György, C. Szepesvári, Frontier LLMs Still Struggle with Simple Reasoning Tasks, 2025. URL: http://arxiv.org/abs/2507.07313. doi:10.48550/arXiv.2507.07313. arXiv:2507.07313 [cs].

[13] K. Mahowald, A. A. Ivanova, I. A. Blank, N. Kanwisher, J. B. Tenenbaum, E. Fedorenko, Dissociating language and thought in large language models, 2024. URL: http://arxiv.org/abs/2301.06627. doi:10.48550/arXiv.2301.06627. arXiv:2301.06627 [cs].

[14] M. Wang, Y. Yao, Z. Xu, S. Qiao, S. Deng, P. Wang, X. Chen, J.-C. Gu, Y. Jiang, P. Xie, F. Huang, H. Chen, N. Zhang, Knowledge Mechanisms in Large Language Models: A Survey and Perspec-

tive, 2024. URL: https://arxiv.org/abs/2407.15017. doi:10.48550/ARXIV.2407.15017, version Number: 4.

[15] A. Roberts, C. Raffel, N. Shazeer, How Much Knowledge Can You Pack Into the Parameters of a Language Model?, 2020. URL: http://arxiv.org/abs/2002.08910. doi:10.48550/arXiv.2002.08910. arXiv:2002.08910 [cs].

[16] Z. Gekhman, E. B. David, H. Orgad, E. Ofek, Y. Belinkov, I. Szpektor, J. Herzig, R. Reichart, Inside-Out: Hidden Factual Knowledge in LLMs, 2025. URL: http://arxiv.org/abs/2503.15299. doi:10.48550/arXiv.2503.15299. arXiv:2503.15299 [cs].

[17] T. Kuculo, S. Abdollahi, S. Gottschalk, Transformer-Based Architectures Versus Large Language Models in Semantic Event Extraction: Evaluating Strengths and Limitations 16 (2025) 22104968251363759. URL: https://journals.sagepub.com/doi/10.1177/22104968251363759. doi:10.1177/22104968251363759.

[18] Z. Liu, C. Gan, J. Wang, Y. Zhang, Z. Bo, M. Sun, H. Chen, W. Zhang, OntoTune: Ontology-Driven Self-training for Aligning Large Language Models, 2025. URL: http://arxiv.org/abs/2502.05478. doi:10.48550/arXiv.2502.05478. arXiv:2502.05478 [cs].

[19] Z. Zhang, A. Zhang, M. Li, A. Smola, Automatic Chain of Thought Prompting in Large Language Models, 2022. URL: http://arxiv.org/abs/2210.03493. doi:10.48550/arXiv.2210.03493. arXiv:2210.03493 [cs].

[20] K. Shum, S. Diao, T. Zhang, Automatic Prompt Augmentation and Selection with Chain-of-Thought from Labeled Data, 2024. URL: http://arxiv.org/abs/2302.12822. doi:10.48550/arXiv.2302.12822. arXiv:2302.12822 [cs].

[21] G. Nayak, S. Dutta, D. Ajwani, P. Nicholson, A. Sala, Automated assessment of knowledge hierarchy evolution: comparing directed acyclic graphs 22 (2019) 256–284. URL: https://link.springer.com/10.1007/s10791-018-9345-y. doi:10.1007/s10791-018-9345-y.

[22] R. Du, H. An, K. Wang, W. Liu, A Short Review for Ontology Learning: Stride to Large Language Models Trend, 2024. URL: http://arxiv.org/abs/2404.14991. doi:10.48550/arXiv.2404.14991. arXiv:2404.14991 [cs].

[23] M. Alviano, L. Grillo, F. Lo Scudo, L. A. Rodriguez 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, International Joint Conferences on Artificial Intelligence Organization, 2025, pp. 4330–4338. URL: https://www.ijcai.org/proceedings/2025/482. doi:10.24963/ijcai.2025/482.

[24] H. B. Giglou, J. D'Souza, S. Auer, LLMs4OL: Large Language Models for Ontology Learning, 2023. URL: http://arxiv.org/abs/2307.16648. doi:10.48550/arXiv.2307.16648. arXiv:2307.16648 [cs].

[25] M. Funk, S. Hosemann, J. C. Jung, C. Lutz, Towards Ontology Construction with Language Models, 2023. URL: http://arxiv.org/abs/2309.09898. doi:10.48550/arXiv.2309.09898. arXiv:2309.09898 [cs].

[26] B. Chen, F. Yi, D. Varró, Prompting or Fine-tuning? A Comparative Study of Large Language Models for Taxonomy Construction, 2023. URL: http://arxiv.org/abs/2309.01715. doi:10.48550/arXiv.2309.01715. arXiv:2309.01715 [cs].

[27] D. Jain, L. E. Anke, Distilling Hypernymy Relations from Language Models: On the Effectiveness of Zero-Shot Taxonomy Induction, 2022. URL: http://arxiv.org/abs/2202.04876. doi:10.48550/arXiv.2202.04876. arXiv:2202.04876 [cs].

[28] Q. Zeng, Y. Bai, Z. Tan, S. Feng, Z. Liang, Z. Zhang, M. Jiang, Chain-of-Layer: Iteratively Prompting Large Language Models for Taxonomy Induction from Limited Examples, 2024. URL: http://arxiv.org/abs/2402.07386. doi:10.48550/arXiv.2402.07386. arXiv:2402.07386 [cs].

[29] C. Chen, K. Lin, D. Klein, Constructing Taxonomies from Pretrained Language Models, in: Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2021, pp. 4687–4700. URL: http://arxiv.org/abs/2010.12813. doi:10.18653/v1/2021.naacl-main.373. arXiv:2010.12813 [cs].

[30] T. Aggarwal, A. Salatino, F. Osborne, E. Motta, Large Language Models for Scholarly Ontology Generation: An Extensive Analysis in the Engineering Field, 2025. URL: http://arxiv.org/abs/2412.08258. doi:10.48550/arXiv.2412.08258. arXiv:2412.08258 [cs].
[31] A. S. Lippolis, M. J. Saeedizade, R. Keskisärkkä, S. Zuppiroli, M. Ceriani, A. Gangemi, E. Blomqvist, A. G. Nuzzolese, Ontology Generation using Large Language Models, 2025. URL: http://arxiv.org/abs/2503.05388. doi:10.48550/arXiv.2503.05388. arXiv:2503.05388 [cs].
[32] M. J. Saeedizade, E. Blomqvist, Navigating Ontology Development with Large Language Models, in: A. Meroño Peñuela, A. Dimou, R. Troncy, O. Hartig, M. Acosta, M. Alam, H. Paulheim, P. Lisena (Eds.), The Semantic Web, volume 14664, Springer Nature Switzerland, 2024, pp. 143–161. URL: https://link.springer.com/10.1007/978-3-031-60626-7_8. doi:10.1007/978-3-031-60626-7_8, series Title: Lecture Notes in Computer Science.
[33] M. Poveda-Villalón, M. C. Suárez-Figueroa, M. Á. García-Delgado, A. Gómez-Pérez, OOPS! (OntOlogy Pitfall Scanner!): supporting ontology evaluation on-line (2023).
[34] A. Zaitoun, T. Sagi, K. Hose, OntoEval: an Automated Ontology Evaluation System, in: Companion Proceedings of the ACM Web Conference 2023, ACM, 2023, pp. 82–85. URL: https://dl.acm.org/doi/10.1145/3543873.3587318. doi:10.1145/3543873.3587318.
[35] J. Völker, D. Vrandečić, Y. Sure, A. Hotho, AEON – An approach to the automatic evaluation of ontologies 3 (2008) 41–62. URL: https://journals.sagepub.com/doi/full/10.3233/AO-2008-0048. doi:10.3233/AO-2008-0048.
[36] A. S. Lippolis, M. J. Saeedizade, R. Keskisärkkä, A. Gangemi, E. Blomqvist, A. G. Nuzzolese, Large Language Models Assisting Ontology Evaluation, 2025. URL: http://arxiv.org/abs/2507.14552. doi:10.48550/arXiv.2507.14552. arXiv:2507.14552 [cs].

A. Appendix

Node Attributes in the Concept Tree Each node is represented as a structured record with fixed semantic and relational attributes, summarized in Table 3. This representation preserves the information needed for prompt construction while maintaining a hierarchical and machine-interpretable structure.

Table 3
Node attributes in the concept tree representation

Attribute	Description
`id`	Identifier of the node, corresponding to the concept name.
`type`	Node type, indicating whether the node represents a concept, a relation, a label, or a specific type in the context.
`description`	Detailed semantic description of the node.
`relation_from_parent`	The relation linking this node to its parent node.
`relation_description`	Natural language explanation of the parent–child relation.
`score_parent`	Relevance score between this node and its parent node.
`score_domain`	Relevance score between this node and the target domain.
`example`	Illustrative examples used for pattern induction and syntax imitation.
`children`	List of child nodes representing subordinate concepts.

A.1. Workflow

Figure 5 shows the high-level AutoCoC workflow. Given domain documentation, a recursive top-down algorithm first builds a rooted concept tree by extracting concepts and relations. Each node is then enriched with concrete examples retrieved from the documentation. Finally, a breadth-first traversal generates the AutoCoC prompt sequence, presenting abstract concepts before their specializations.

Figure 5: High-Level Workflow of Automated Chain of Concepts

A.2. Generated Concept Tree from Reverse-Engineered CoC

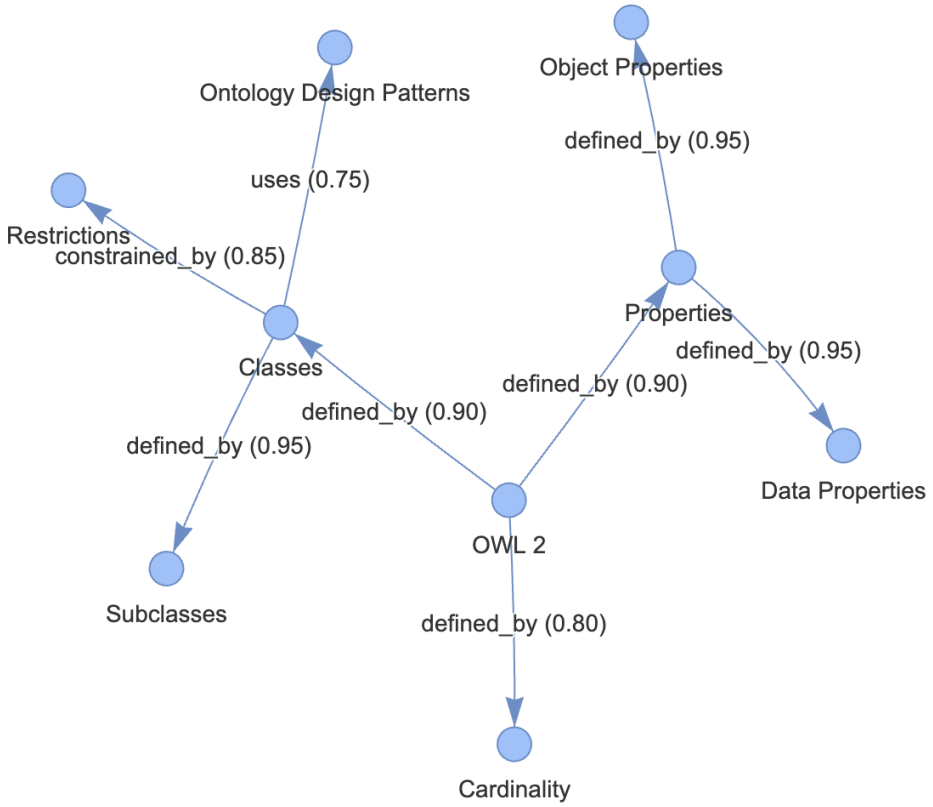


Figure 6: Concept tree generated from the reverse-engineered handcrafted CoC.

A.3. Domain Prompts for Ontology Generation

This appendix presents the ten domain-specific prompts used to evaluate ontology generation across all three methods (Baseline, Reverse-Engineered, and AutoCoC). Each prompt specifies the structural requirements for OWL ontology generation in Turtle format.

Table 4
Target domains for downstream OWL ontology generation.

Domain	Description
Academic	Academic institutions, programs, personnel, and processes
Automotive	Vehicles, components, infrastructure, and manufacturing
Biomedical	Medical conditions, treatments, professionals, and workflows
Coffee	Ingredients, equipment, brewing methods, and quality
Motors	Industrial electric motors and operational specifications
Pizza Making	Ingredients, equipment, techniques, and styles
Real Estate	Buildings, transactions, and geographic aspects
Robotic Control	Robots, sensors, control systems, and modes
Smart Home	IoT devices, automation, and environmental monitoring
Supply Chain	Entities, logistics, inventory, and relationships

A.3.1. Industrial Motors

Create an OWL ontology in Turtle format that models the domain of industrial electric motors.
Your ontology should cover:

- Different types of motors and their classifications
- Motor components and parts
- Operational aspects and processes
- Measurements and specifications
- Relationships between entities

Include:

- A class hierarchy with multiple levels
- Both object properties and datatype properties (consider if some properties are specializations of others)
- Domain and range constraints where appropriate
- Some concrete instances (individuals) with property values
- Labels and comments for key entities

The ontology should be comprehensive enough to represent real-world industrial motor systems.
Output valid Turtle syntax.

A.3.2. Coffee

Create an OWL ontology in Turtle format that models the coffee making process from bean to cup.
Your ontology should cover:

- Coffee ingredients and their varieties
- Equipment and tools used in coffee preparation
- Different brewing methods and techniques
- People involved in the process
- Quality characteristics of coffee

Include:

- A class hierarchy with multiple levels
- Both object properties and datatype properties (consider if some properties are specializations of others)
- Domain and range constraints where appropriate
- Some concrete instances (individuals) with property values

- Labels and comments for key entities

The ontology should be comprehensive enough to represent a specialty coffee shop operation.
Output valid Turtle syntax.

A.3.3. Real Estate

Create an OWL ontology in Turtle format that models real estate buildings and property transactions.
Your ontology should cover:

- Different types of buildings and properties
- Building components and systems
- Property transactions and agreements
- People and organizations involved
- Location and geographic aspects

Include:

- A class hierarchy with multiple levels
- Both object properties and datatype properties (consider if some properties are specializations of others)
- Domain and range constraints where appropriate
- Some concrete instances (individuals) with property values
- Labels and comments for key entities

The ontology should be comprehensive enough to represent a real estate market.
Output valid Turtle syntax.

A.3.4. Robotic Control

Create an OWL ontology in Turtle format that models robotic control systems and processes.
Your ontology should cover:

- Different types of robots and their classifications
- Sensors, actuators, and control components
- Control algorithms and decision-making processes
- Robot tasks and operational modes
- Environmental interactions and feedback systems

Include:

- A class hierarchy with multiple levels
- Both object properties and datatype properties (consider if some properties are specializations of others)
- Domain and range constraints where appropriate
- Some concrete instances (individuals) with property values
- Labels and comments for key entities

The ontology should be comprehensive enough to represent industrial and service robot control systems.
Output valid Turtle syntax.

A.3.5. Pizza Making

Create an OWL ontology in Turtle format that models the pizza making process from ingredients to finished product.
Your ontology should cover:

- Pizza ingredients and their varieties (dough, toppings, sauces)
- Kitchen equipment and tools used in pizza preparation
- Cooking methods and preparation techniques
- Pizza types and styles
- Quality characteristics and nutritional aspects

Include:

- A class hierarchy with multiple levels
- Both object properties and datatype properties (consider if some properties are specializations of others)
- Domain and range constraints where appropriate
- Some concrete instances (individuals) with property values
- Labels and comments for key entities

The ontology should be comprehensive enough to represent a pizzeria operation from preparation to serving.
Output valid Turtle syntax.

A.3.6. Biomedical

Create an OWL ontology in Turtle format that models biomedical and healthcare systems.
Your ontology should cover:

- Medical conditions, diseases, and symptoms
- Treatments, medications, and therapeutic procedures
- Healthcare professionals and their roles
- Medical devices and diagnostic equipment
- Patient care processes and clinical workflows

Include:

- A class hierarchy with multiple levels
- Both object properties and datatype properties (consider if some properties are specializations of others)
- Domain and range constraints where appropriate
- Some concrete instances (individuals) with property values
- Labels and comments for key entities

The ontology should be comprehensive enough to represent clinical healthcare delivery and medical knowledge.
Output valid Turtle syntax.

A.3.7. Academic

Create an OWL ontology in Turtle format that models academic institutions and educational processes.
Your ontology should cover:

- Academic programs, courses, and curricula
- University personnel (faculty, students, staff) and their roles
- Academic facilities and resources
- Educational activities and assessment methods

- Academic achievements and credentials

Include:

- A class hierarchy with multiple levels
- Both object properties and datatype properties (consider if some properties are specializations of others)
- Domain and range constraints where appropriate
- Some concrete instances (individuals) with property values
- Labels and comments for key entities

The ontology should be comprehensive enough to represent university operations and academic processes.
Output valid Turtle syntax.

A.3.8. Automotive

Create an OWL ontology in Turtle format that models automotive systems and transportation processes.
Your ontology should cover:

- Different types of vehicles and their classifications
- Vehicle components, systems, and subsystems
- Transportation infrastructure and traffic management
- Automotive manufacturing and maintenance processes
- Safety systems and performance characteristics

Include:

- A class hierarchy with multiple levels
- Both object properties and datatype properties (consider if some properties are specializations of others)
- Domain and range constraints where appropriate
- Some concrete instances (individuals) with property values
- Labels and comments for key entities

The ontology should be comprehensive enough to represent automotive industry operations and intelligent transportation systems.
Output valid Turtle syntax.

A.3.9. Smart Home

Create an OWL ontology in Turtle format that models smart home systems and IoT devices.
Your ontology should cover:

- IoT devices, sensors, and smart appliances
- Home automation systems and control protocols
- Environmental monitoring and energy management
- User interactions and behavioral patterns
- Security systems and access control

Include:

- A class hierarchy with multiple levels
- Both object properties and datatype properties (consider if some properties are specializations of others)
- Domain and range constraints where appropriate
- Some concrete instances (individuals) with property values
- Labels and comments for key entities

The ontology should be comprehensive enough to represent connected home ecosystems and intelligent building management.
Output valid Turtle syntax.

A.3.10. Supply Chain

Create an OWL ontology in Turtle format that models supply chain and logistics operations.
Your ontology should cover:

- Supply chain entities (suppliers, manufacturers, distributors, retailers)
- Products, materials, and inventory management
- Transportation and logistics processes
- Warehousing and storage systems
- Business relationships and contractual agreements

Include:

- A class hierarchy with multiple levels
- Both object properties and datatype properties (consider if some properties are specializations of others)
- Domain and range constraints where appropriate
- Some concrete instances (individuals) with property values
- Labels and comments for key entities

The ontology should be comprehensive enough to represent end-to-end supply chain operations and logistics networks.
Output valid Turtle syntax.

A.4. Ontology Evaluation Complete Results

Table 5
Baseline method results across 10 domains

Domain	Classes	Avg/File	Conf.	Viol. %	Comp. Score
Academic	156	15.6	0.875	28.85	0.682
Automotive	138	13.8	0.907	5.80	0.768
Biomedical	126	12.6	0.861	11.90	0.713
Coffee	151	15.1	0.737	22.52	0.647
Motors	142	14.2	0.773	2.11	0.733
Pizza Making	136	13.6	0.918	0.00	0.794
Real Estate	130	13.0	0.889	5.38	0.754
Robotic Control	137	13.7	0.896	0.00	0.786
Smart Home	111	11.1	0.878	24.32	0.654
Supply Chain	120	12.0	0.910	5.00	0.754
Mean	**134.7**	**13.5**	**0.864**	**10.59**	**0.729**

A.5. Ontology Evaluation: OntoClean

Ontology evaluation is essential because syntactically valid ontologies may still contain semantic inconsistencies, conceptual misclassifications, or modelling pitfalls. Ontology evaluation methods have therefore been developed to assess structural, logical, and semantic quality [33, 34]. Tool-supported approaches such as OOPS! and OntoEval aim to detect ontology pitfalls and support quality assessment during ontology development [33, 34].

Table 6
Reverse-engineered method results across 10 domains

Domain	Classes	Avg/File	Conf.	Viol. %	Comp. Score
Academic	203	20.3	0.868	13.30	0.792
Automotive	167	16.7	0.899	8.98	0.783
Biomedical	130	13.0	0.873	12.31	0.720
Coffee	143	14.3	0.710	9.09	0.682
Motors	124	12.4	0.723	10.48	0.661
Pizza Making	140	14.0	0.894	15.00	0.728
Real Estate	166	16.6	0.679	28.31	0.617
Robotic Control	136	13.6	0.892	0.00	0.783
Smart Home	118	11.8	0.877	8.47	0.724
Supply Chain	153	15.3	0.900	7.84	0.773
Mean	**148.0**	**14.8**	**0.831**	**11.38**	**0.726**

Table 7
AutoCoC method results across 10 domains

Domain	Classes	Avg/File	Conf.	Viol. %	Comp. Score
Academic	196	19.6	0.866	16.33	0.771
Automotive	266	26.6	0.904	5.26	0.905
Biomedical	130	13.0	0.864	21.54	0.679
Coffee	300	30.0	0.911	13.67	0.910
Motors	216	21.6	0.925	0.93	0.877
Pizza Making	167	16.7	0.903	0.00	0.820
Real Estate	218	21.8	0.919	0.00	0.881
Robotic Control	180	18.0	0.873	7.78	0.791
Smart Home	241	24.1	0.870	23.24	0.792
Supply Chain	259	25.9	0.894	5.41	0.893
Mean	**217.3**	**21.7**	**0.893**	**9.42**	**0.832**

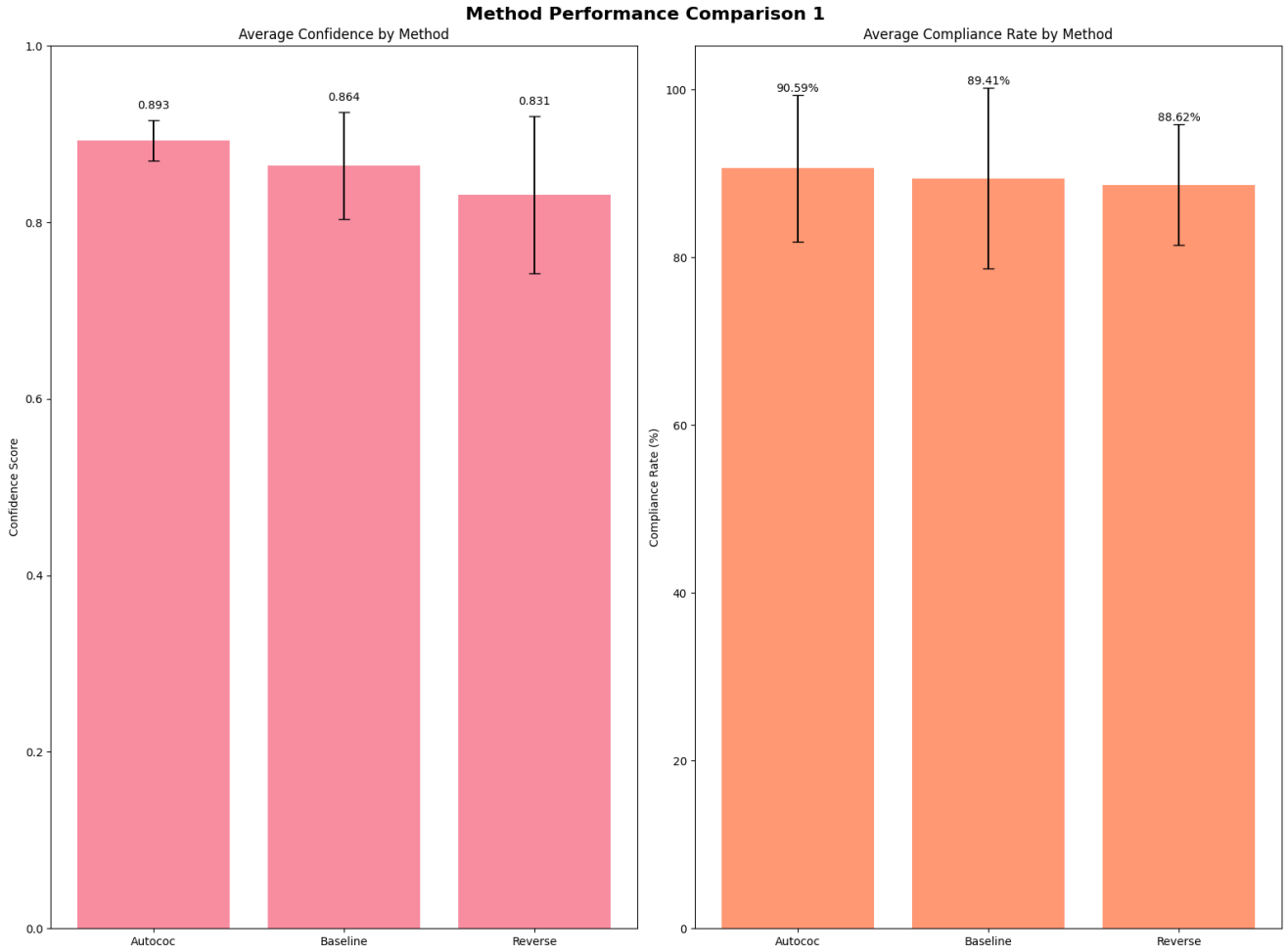
OntoClean provides a formal framework for assessing the ontological soundness of hierarchical structures through meta-properties such as rigidity, identity, unity, and dependence [7]. These meta-properties define inheritance constraints for taxonomy construction. For example, anti-rigid concepts must not subsume rigid concepts, and concepts lacking identity criteria should not subsume concepts that provide identity criteria. Violations indicate semantic modelling errors even when logical consistency is preserved.

Because OntoClean requires expert judgement, several studies have explored automation or LLM assistance for ontology evaluation and refinement. AEON, for example, investigates automatic tagging of OntoClean meta-properties [35], while recent work studies the use of LLMs for OntoClean-based ontology refinement and competency-question-based ontology evaluation [7, 36]. These works motivate the use of structured evaluation criteria for assessing generated ontologies beyond syntax alone.

Existing concept-driven prompting and ontology generation methods remain limited by manual hierarchy construction, weak reproducibility, or insufficient control over the injected knowledge structure. This motivates the fully automated AutoCoC pipeline proposed in this work, which generates a concept latextree from domain knowledge sources and uses it as a structured prompt-level knowledge representation.

B. Additional Result Visualizations

Figure 7 shows the original aggregated bar plots corresponding to the summarized values reported in Table 2.



(a) Mean confidence score and OntoClean compliance rate across ten domains.