Automating Chain of Concepts Prompting for Large Language Models

Master's thesis

Wissenschaftliche Arbeit zur Erlangung des Grades
M.Sc. Data engineering and analytics
an der TUM School of Computation, Information and Technology der Technischen Universität München.

Betreut von	Prof. Thomas Runkler
	Lehrstuhl für Theoretische Informatik
Eingereicht von	Hurile Borjigin
	Ricarda-Huch-Straße 1
	80807 München
	+49 176 6625 9079
Eingereicht am	München, 31.01.2026

Anhang I

Erklärung

Ich versichere hiermit, dass ich die von mir eingereichte Abschlussarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

_____

Ort, Datum, Unterschrift

Contents

List of Figures

List of Tables

Abstract

Large Language Models (LLMs) have recently demonstrated remarkable capabilities across a wide range of natural language processing tasks, such as natural language understanding, text summarization, and machine translation. However, vanilla LLMs' performance often degrades when confronted with domain-specific tasks, conceptual reasoning, particularly in industrial settings where proprietary knowledge, complex concept hierarchies, and subtle inter-concept relationships are essential. Previous work introduced Chain of Concepts (CoC) [1], a knowledge-injection prompting strategy that leverages a domain-specific Directed Acyclic Graph (DAG) to iteratively provide conceptual information to the LLM. However, CoC relies on manually curated DAGs, an approach that is labor-intensive, subjective, and difficult to scale or reproduce.

Inspired by these works, in this paper we propose an automated pipeline for building a concept tree from large industrial documentation, which contains the necessary domain concepts and their hierarchies, followed by retrieving examples to support in-context learning, and finally generating a sequence of prompts by traversing the concept tree.

This removes the dependency on domain experts, enabling systematic, scalable, and reproducible customization of LLMs for assisting with domain-specific tasks.

We evaluate AutoCoC on ontology generation in OWL/Turtle format across 10 diverse domains, comparing against handcrafted and reverse-engineered Chain of Concepts baselines. AutoCoC achieves the highest mean meta-property assignment confidence (0.893) and OntoClean semantic compliance rate (90.59%), while generating substantially richer ontologies with 21.7 classes per file compared to 13.5 (baseline) and 14.8 (reverse-engineered). Self-consistency analysis across 200 runs demonstrates high concept-level stability (0.889) with 69.2% core concepts consistently recovered. The method incurs a 40% increase in evaluation time, which is a favorable trade-off given significant gains in semantic correctness, structural coverage, and complete automation. These results validate prompt-based knowledge injection as a practical approach for enhancing LLM performance on constraint-driven, domain-specific generation tasks.

1. Introduction

1.1. Background & Problems

Large Language Models (LLMs) have revolutionized solving everyday problems such as text summarization, copy-writing, and knowledge acquisition [2]. Their strong performance on many general tasks is primarily enabled by the broad "base knowledge" captured during pre-training on large-scale open-world data. Notably, LLMs can often solve tasks that were never explicitly introduced during training, a phenomenon commonly described as emergent capabilities [3]. These capabilities can be amplified through specialized natural language prompting techniques, most prominently In-Context Learning (ICL) [4] and Chain-of-Thought (CoT) prompting [5]. These prompting strategies altered an LLM's behavior in different ways without altering its parameters. In-Context learning and Chain-of-Thoughts have been successful in improving LLMs' ability to imitate patterns and reason from existing knowledge, respectively.

However, open-world data often lacks up-to-date specialized knowledge, such as in healthcare, chemistry, legal analysis, or engineering guidelines. As a result, LLMs inherit these gaps and perform poorly on tasks that depend on such information. Previous work found that when LLMs tackle tasks and lack the essential conceptual knowledge or relevant concepts and their relations to solve them, they often hallucinate or parrot. While LLMs excel at formal linguistic tasks, they lack essential domain-specific or closed knowledge to solve certain tasks; they often fail to help, even mislead the ones who overtrust them.

Hence the task of filling up the knowledge gap becomes an essential step to utilize LLMs in such domains. Previous research has been conducted on knowledge injection into LLMs. Based on when the knowledge is injected and how it reacts with the model, these methods can fall into the following four categories:

1. Dynamic injection
2. Static embedding

3. Prompt optimisation

4. Modular adapters.

All methods have their own advantages and disadvantages, and the choice of which to use depends heavily on the application scenario and budget. Static knowledge injection and modular knowledge adapters integrate knowledge before inference and involve parameter updates via either full fine-tuning or adapter-based tuning. In contrast, dynamic knowledge injection and prompt optimization inject knowledge at inference time without altering model parameters: the former retrieves external information, while the latter leverages internal knowledge through designed prompts.

Given the constraints of domains with limited resources, scarce documentation, and rapidly evolving knowledge bases, we focus specifically on **prompt optimization** approaches to knowledge injection. Unlike static embedding or modular adapters that require parameter updates and substantial training data, or dynamic injection methods that depend on external retrieval systems and comprehensive knowledge bases, prompt optimization methods offer a practical solution for resource-constrained scenarios. They require no model retraining, can be updated instantly as domain knowledge evolves, and work effectively even when formal knowledge bases are incomplete or unavailable.

Our work is inspired by **Chain of Concepts (CoC)**, a prompt optimization approach introduced in prior research that structures domain knowledge according to hierarchies based on abstraction levels. The original Chain of Concepts framework proposed that explicitly presenting conceptual structures through carefully designed prompts could guide LLM reasoning without parameter updates or external retrieval. While we acknowledge this work as a foundational inspiration for our methodology, our subsequent investigation revealed significant limitations in the original approach's effectiveness and generalizability across different domains and task types.

Building on this foundation while addressing its shortcomings, our method centers on **knowledge grounding**—anchoring the model's reasoning process to a structured representation of domain concepts and their interrelations. By explicitly presenting concepts at different levels of abstraction (from high-level principles to specific instances) and their relationships, we provide the LLM with a conceptual scaffold that guides its reasoning. This approach is motivated by the observation that domain expertise involves not just knowing individual facts, but understanding how concepts relate to one another hierarchically and how abstract principles instantiate in concrete cases.

Our approach aims to encourage the large language model to:

- attend to the relevant concepts,
- follow explicit relations between them, and
- build solutions in a manner consistent with the domain's conceptual structure.

By structuring prompts to reflect the conceptual organization of a domain, our method leverages the model's existing reasoning capabilities while compensating for gaps in specialized knowledge. This approach sits at the intersection of prompt engineering and knowledge representation, offering a practical and adaptive solution for specialized domains where traditional knowledge injection methods are infeasible due to limited resources, incomplete documentation, or rapidly changing knowledge landscapes.

1.2. Motivation

While prompt optimization methods offer significant advantages for knowledge injection in resource-constrained domains, existing approaches face substantial practical challenges that limit their real-world applicability. Current methods typically rely on manually curated domain ontologies[6] or concept hierarchies[1] to structure the prompts that inject domain knowledge. Some approaches directly feed chunks of formal ontologies to LLMs, while others require domain experts to carefully design conceptual structures that reflect the abstraction levels and relationships within a domain.

However, the process of curating these knowledge structures—whether formal ontologies or concept hierarchies—presents significant obstacles that hinder the widespread adoption and effectiveness of prompt-based knowledge injection. These challenges can be categorized into three fundamental dimensions:

- Accuracy

  Manual curation of domain knowledge structures is inherently prone to errors and incompleteness. Domain experts may inadvertently omit important concepts, misrepresent relationships between concepts, or fail to capture the full scope of domain knowledge. The complexity of modern specialized domains means that even experienced experts cannot always maintain a complete and accurate mental model of all relevant concepts and their interrelations. Furthermore, translating expert knowledge into structured formats suitable for LLM prompting introduces additional opportunities for misalignment between the intended knowledge representation and its actual implementation.

- Reproducibility

  The manual curation process is heavily influenced by individual experts' experiences, perspectives, and subjective judgments. Different experts may structure the same domain knowledge in fundamentally different ways, emphasizing different concepts, organizing hierarchies differently, or defining relationships based on their particular viewpoints. This subjectivity makes it difficult to reproduce knowledge structures across different applications or to ensure consistency when updating existing structures. The lack of standardized procedures for knowledge curation means that the effectiveness of prompt-based knowledge injection becomes dependent on the particular expert involved, rather than being a reliable and transferable methodology.

- Scalability

  Perhaps most critically, manual curation does not scale effectively to multiple domains or evolving knowledge bases. Each new domain requires substantial expert time and effort to develop appropriate knowledge structures. As domains evolve, such as with new concepts emerging, relationships changing, and existing knowledge being refined, results in the manually curated structures quickly become outdated. Updating these structures requires the same labor-intensive process as initial creation, creating a maintenance burden that grows with the number of domains and the pace of knowledge evolution. This scalability limitation is particularly problematic in rapidly advancing fields where knowledge can become obsolete within months or even weeks.

These challenges in accuracy, reproducibility, and scalability represent fundamental barriers to the practical deployment of prompt-based knowledge injection methods in real-world specialized domains. They motivate our work toward developing an automated approach that can generate accurate, consistent, and easily updatable knowledge structures without requiring extensive manual expert curation. Our goal is to preserve the advantages of prompt optimization methods (no parameter updates, adaptability, minimal infrastructure requirements) while addressing the practical limitations that currently prevent their widespread adoption.

1.3. Research Objectives & Research Questions

This research aims to develop and validate an automated approach to constructing concept trees from industrial documentation and applying them to domain-specific problem-solving through prompt optimization. Our work addresses the fundamental limitations of manual knowledge curation, including the limitations in accuracy, reproducibility, and scalability, by developing a method that can systematically extract, organize, and apply domain knowledge without extensive expert intervention.

The overarching goal is to bridge the gap between the theoretical promise of prompt-based knowledge injection and its practical deployment in resource-constrained specialized domains. We seek to demonstrate that automated concept tree construction can match or exceed the effectiveness of expert-crafted knowledge structures while offering superior consistency, reproducibility, and scalability.

1.3.1. Research Objectives

- RO1: Automate the construction of concept trees from large industrial documentation and generation of sequences of prompts from the concept tree.

  This objective focuses on developing computational methods that can process unstructured or semi-structured domain documentation to extract relevant concepts, identify relationships between them, and organize them into hierarchical structures suitable for guiding LLM reasoning. The automation must handle the complexity and volume typical of real-world industrial documentation while producing structures that effectively capture the domain's conceptual organization.

- RO2: Ensure the correctness and consistency of the generated concept tree.

  Automated methods must produce reliable and stable outputs. This objective addresses the need for mechanisms to validate the quality of generated concept trees, ensure consistency across multiple runs, and maintain appropriate granularity in the conceptual hierarchy. We aim to develop both intrinsic quality measures and correction mechanisms that can identify and address structural deficiencies.

- RO3: Apply the final generated prompts to assist or solve in a realistic use case.

  To demonstrate practical applicability, we must identify appropriate downstream tasks where concept-grounded prompting provides tangible value. This objective

requires selecting use cases that genuinely benefit from explicit conceptual structure, implementing the application of automated concept trees to these tasks, and documenting the operationalization process for future applications.

- RO4: Evaluate the effectiveness of our automated method.

  Rigorous evaluation is essential to validate our approach. This objective encompasses both intrinsic evaluation of the concept trees themselves and extrinsic evaluation of their impact on downstream task performance. We aim to establish comprehensive evaluation criteria and conduct comparative studies against expert-crafted baselines and alternative approaches.

1.3.2. Research Questions

RQ1: How can concept trees be constructed automatically from large industrial documentation?

This foundational question addresses the core technical challenge of our work: transforming unstructured documentation into structured conceptual hierarchies suitable for prompt-based knowledge injection.

- RQ1.1 (Extraction): How can domain concepts be reliably identified from documentation, and how should candidate relations between concepts be proposed?

  The extraction phase requires methods to distinguish meaningful domain concepts from general language, identify which concepts are relevant for reasoning tasks, and propose plausible relationships between concepts based on their co-occurrence patterns, linguistic cues, and semantic similarity.

- RQ1.2 (Tree building): How can a concept hierarchy be constructed such that the resulting structure is informative for reasoning?

  Given extracted concepts and candidate relations, this sub-question addresses the algorithmic challenge of organizing them into a tree structure that reflects meaningful abstraction levels and supports effective reasoning. It explores criteria for parent-child relationships, methods for resolving conflicting placement options, and strategies for ensuring the tree structure aligns with how domain knowledge is actually applied in problem-solving.

- RQ1.3 (Effectiveness): Which design choices most improve the quality and usefulness of the resulting tree/CoC under practical token and context constraints?

Real-world deployment faces practical constraints on prompt length and the use of context windows. This sub-question investigates how design parameters, such as tree depth, branching factor, concept selection criteria, and relation types, affect both the intrinsic quality of the concept tree and its practical utility when converted to prompts. It aims to identify optimal configurations that balance comprehensiveness with efficiency.

RQ2: How can correctness and consistency of the generated concept tree and prompts be ensured or measured?

Quality assurance is critical for automated methods. This question explores mechanisms to validate outputs and ensure reliability across different runs and documentation sources.

- RQ2.1 (Consistency): How stable are the extracted concepts and relations across repeated runs?

  Automated methods that produce different results each time are unreliable for practical deployment. This sub-question investigates the degree of variation in outputs when the same documentation is processed multiple times, identifies sources of instability (e.g., randomness in LLM-based extraction, order-dependent processing), and explores techniques to improve reproducibility without sacrificing quality.

- RQ2.2 (Granularity): How can the method avoid overly coarse trees (missing key intermediate concepts) or overly fine-grained trees (unnecessary detail that distracts reasoning)?

  The appropriate level of detail depends on the domain and task. This sub-question addresses the challenge of automatically determining optimal granularity, which ensures the concept tree includes sufficient detail to guide reasoning effectively while avoiding information overload. It explores metrics for assessing granularity and mechanisms for automatic adjustment based on downstream performance.

- RQ2.3 (Correction): Can the algorithm self-correct suboptimal early placements as more context becomes available, and does this improve overall structure quality?

  Initial decisions about concept placement may prove suboptimal as additional concepts are processed. This sub-question investigates whether allowing the algorithm

to revise earlier structural decisions leads to better final trees, explores specific revision strategies (e.g., promoting concepts, restructuring subtrees), and measures the quality improvement from such corrections against their computational cost.

RQ3: How can automated concept tree-based prompts be applied to solve domain-specific problems, and which use case is appropriate?

This question focuses on translating our automated approach into practical problem-solving capability, identifying domains and tasks where explicit conceptual structure provides measurable value.

- RQ3.1 (Use case selection): Which task/use case best reflects the need for explicit concepts, relations, and constraints?

  Not all domain tasks benefit equally from concept-grounded prompting. This sub-question explores criteria for identifying suitable use cases, tasks where conceptual structure genuinely aids reasoning and where gaps in base LLM knowledge create clear opportunities for improvement. It investigates characteristics of promising use cases and develops a framework for assessing use case suitability.

- RQ3.2 (Artifact quality): For a data modeling use case, does the automated concept tree-guided process produce more semantically coherent outputs than hand-crafted CoC?

  Taking data modeling as a concrete instantiation, this sub-question examines whether artifacts produced using automated concept trees exhibit better semantic coherence, consistency with domain constraints, and alignment with expert expectations compared to those produced using manually crafted concept chains. It develops specific quality criteria for data models and conducts comparative assessments.

RQ4: How effective is automated concept tree-based prompting compared to expert-crafted CoC and other baselines?

This evaluative question establishes the comparative performance of our approach across multiple dimensions and against relevant alternatives.

- RQ4.1 (Evaluation): On the downstream task, how do automated concept tree-based prompts and expert-crafted CoC compare using semantic evaluation criteria?

This sub-question requires defining appropriate semantic evaluation criteria for the chosen downstream task, implementing both automated and expert-crafted approaches, and conducting rigorous comparative assessment. It explores both automated metrics (where applicable) and expert judgment-based evaluation to establish whether automated concept trees can match or exceed the effectiveness of manual curation while offering practical advantages in consistency, reproducibility, and scalability.

1.4. Contributions

The primary contribution of this thesis is the automation of prompt construction for explicit knowledge injection in large language models, specifically targeting prompt-based methods that inject structured domain knowledge without parameter updates. Unlike retrieval-augmented generation (RAG) that injects relevant text fragments, or fine-tuning approaches that modify model parameters, our work focuses on concept-grounded prompting where hierarchical domain knowledge structures guide LLM reasoning through carefully designed instruction sequences.

This approach is particularly suited for structured artifact generation tasks in specialized domains, such as data modeling, schema design, and formal specification creation, where outputs must conform to domain-specific constraints, relationships, and conceptual hierarchies that base LLMs lack. Our automated method addresses the scalability and reproducibility challenges that have limited the practical deployment of prompt-based knowledge injection methods like Chain of Concepts (CoC), enabling their application across multiple evolving domains without expert curation bottlenecks.

The main contributions of this thesis can be summarized as follows:

1. Fully automated concept tree construction

   We eliminate the need for manual expert curation by developing a recursive algorithm that automatically extracts domain concepts and constructs hierarchical knowledge structures from large-scale industrial documentation, enabling prompt-based knowledge injection to scale across domains and documentation sources.

2. Dynamic hierarchy refinement mechanism

   We propose a self-correcting construction process that iteratively adjusts concept placements and hierarchical relationships during generation based on multiple

structural and semantic criteria, improving tree quality as more domain context becomes available.

3. Dual scoring mechanisms for quality assurance

   We introduce complementary scoring approaches for concept relevance ranking and hierarchical edge selection that significantly improve both the structural coherence of generated concept trees and their semantic alignment with domain knowledge organization.

4. Automated example retrieval for few-shot prompting

   We design methods for automatically extracting and selecting representative code snippets and examples from industrial documentation, enabling concept-grounded few-shot learning that enhances LLM pattern recognition and domain-specific reasoning capabilities.

5. Empirical validation on structured generation tasks

   We demonstrate the effectiveness of automatically generated knowledge-injection prompts on a downstream data modeling task, where our approach outperforms manually curated Chain of Concepts baselines in domain knowledge coverage, hierarchical coherence, consistency, and task-specific quality metrics, while offering superior reproducibility and scalability.

1.5. Thesis Structure

2. Related work

2.1. Large Language Models: Capabilities and Limitations

Large language models excel at a broad range of natural language tasks, but they remain constrained by data coverage, lack of proprietary/domain knowledge, and fundamental limitations in reasoning and reliability. Carefully designed prompts can unlock "emergent" behaviors that were never explicitly trained, but this does not eliminate those underlying constraints.

2.1.1. LLM Strengths in NLP

LLMs have pushed state-of-the-art performance on many core NLP tasks, including text classification, question answering, summarization, translation, and dialogue. Their strength lies in learning rich contextual representations from large corpora, enabling coherent, fluent text generation and robust generalization even in zero-shot or few-shot settings.

- Studies report that modern LLMs match or surpass specialized models in tasks like fake news detection, hate speech detection, and semantic event extraction when appropriately prompted or fine-tuned[2].
- In domains such as clinical and radiology NLP, LLMs perform competitively on information extraction and report understanding, especially when given well-structured instructions and examples[7].
- Larger models tend to align better with human language structure and even neural activity, indicating that scale improves their ability to capture high-level linguistic and semantic patterns[2][8].

2.1.2. The Knowledge Gap

LLMs are typically trained on large, mostly public text corpora and thus do not natively contain proprietary, confidential, or very niche domain knowledge that was never present in their training data. As a result, they may produce plausible-sounding but incorrect outputs when asked about highly specialized internal procedures, closed-source tools, or up-to-the-minute proprietary data.

- In expert domains such as specialized medicine, finance, or internal enterprise workflows, performance often drops unless models are augmented with external knowledge bases, retrieval systems, or fine-tuning on in-domain data[9][2].
- This "knowledge gap" is compounded by the fact that LLMs cannot verify claims against live proprietary databases by default, which leads to hallucinations and incomplete coverage for organization-specific tasks[10].
- Hybrid architectures that combine LLMs with symbolic systems or knowledge graphs (e.g., Answer Set Programming or semantic web resources) are being explored to inject domain knowledge and improve reliability[11].

2.1.3. Knowledge Mechanisms in Large Language Models

Recent research has begun to systematically investigate how LLMs acquire, store, utilize, and evolve knowledge, moving beyond surface-level performance analysis toward mechanistic understanding. A comprehensive survey by Wang et al.[12] proposes a unifying taxonomy of knowledge mechanisms in LLMs, distinguishing between knowledge utilization at a fixed point in time and knowledge evolution across the model's life cycle.

Knowledge utilization refers to how knowledge encoded in an LLM is accessed and applied during inference. Wang et al. decompose this process into three progressively complex levels: memorization[13], comprehension and application[14][15], and creation[16]. While modern LLMs demonstrate strong memorization of factual knowledge, multiple studies show that this knowledge is often brittle, inconsistently retrieved, and sensitive to surface-level variations in prompts. This fragility manifests as hallucinations, contradictory outputs, and failures in multi-step reasoning, especially in structured or domain-specific tasks.

Importantly, the survey highlights a distinction between parametric knowledge[17][18], which is implicitly stored in model weights, and non-parametric knowledge, which is explicitly represented in external structures such as knowledge graphs or symbolic systems.

Parametric knowledge enables strong generalization and compression but suffers from opacity, outdated information, and limited controllability. These limitations motivate hybrid approaches that introduce structured external knowledge at inference time.

Beyond static utilization, knowledge in LLMs evolves throughout pre-training, fine-tuning, and post-deployment interaction[19]. The survey identifies persistent conflicts during this evolution, including interference between high-frequency and low-frequency facts, contradictions in training data, and misalignment between internal memory and newly introduced information. Fine-tuning and instruction tuning primarily improve knowledge usage rather than introducing fundamentally new knowledge, further underscoring the difficulty of reliably updating parametric memory.

These observations suggest that while LLMs possess substantial latent knowledge, they lack stable internal mechanisms for organizing, validating, and systematically applying domain-specific conceptual structures.

From a knowledge mechanism perspective, prompt-based knowledge injection can be interpreted as a lightweight, inference-time method for externalizing structure that is missing or unstable in parametric memory. Rather than attempting to overwrite or retrieve isolated facts, structured prompts impose an explicit organization over concepts, relations, and dependencies. This aligns with the survey's conclusion that combining parametric knowledge with structured non-parametric representations is a promising direction for improving reliability and reasoning in LLMs.

2.1.4. Emergent Abilities Through Prompt Engineering

Prompt engineering demonstrates that new knowledge and skills can be injected into large language models at inference time without updating model parameters. As shown by Wei et al.[3], emergent abilities arise when specialized prompting strategies, such as few-shot prompting, chain-of-thought[5], instruction-style prompts, and scratchpads[20], provide structured in-context signals that define task knowledge, reasoning patterns, and procedural strategies. These prompts function as lightweight training data, enabling models to acquire new reasoning and algorithmic capabilities on the fly. This provides empirical evidence that prompt engineering serves as a mechanism for dynamically injecting knowledge and skills, rather than merely conditioning input, motivating structured prompting as a form of inference-time learning.

2.2. Knowledge Injection Methods for Large Language Models

A growing body of research has investigated how domain-specific knowledge can be injected into LLMs to overcome the limitations of purely general-purpose pre-training. A recent comprehensive survey by Song et al.[21] categorizes existing approaches into four major paradigms: dynamic knowledge injection, static knowledge embedding, modular adapters, and prompt optimization. These paradigms differ primarily in when knowledge is introduced (training time vs. inference time) and how it interacts with the model (external retrieval, parameter updates, auxiliary modules, or prompts).

In the following, we briefly summarize these four categories and motivate our focus on prompt-based knowledge injection.

2.2.1. Dynamic Knowledge Injection

Dynamic knowledge injection introduces external knowledge at inference time by retrieving relevant information from an external knowledge base or retrieval system and concatenating it with the model input. Formally, the model output depends on both the input and the retrieved knowledge, while the model parameters remain unchanged. This paradigm offers high flexibility and allows knowledge to be updated without retraining. However, it relies heavily on retrieval quality, introduces additional latency, and requires maintaining a reliable external knowledge infrastructure. As a result, system complexity and operational cost increase with scale.

2.2.2. Static Knowledge Embedding

Static knowledge embedding integrates domain knowledge directly into the model parameters through continued pre-training or fine-tuning on domain-specific corpora. This approach typically yields strong performance and fast inference, since no external retrieval is required at runtime. However, it incurs high training cost, requires large domain-specific datasets, and is difficult to update when knowledge evolves. Moreover, repeated fine-tuning risks catastrophic forgetting and limits scalability across multiple domains.

2.2.3. Modular Adapters

Modular adapter methods introduce small, trainable modules into a frozen backbone model to store domain knowledge in a parameter-efficient manner. Only a small subset of parameters is trained, which reduces computational cost and mitigates catastrophic forgetting. While adapters offer a compromise between flexibility and efficiency, they require additional architectural design choices, hyperparameter tuning, and domain-specific training data. This increases engineering complexity and limits their applicability in rapidly changing or low-resource settings.

2.2.4. Prompt Optimization

Prompt optimization injects knowledge purely through carefully designed prompts, without modifying model parameters or relying on external retrieval. By providing structured instructions, demonstrations, or reasoning scaffolds, prompts activate and reorganize the model's internal knowledge to perform domain-specific tasks. This paradigm has several distinctive advantages: it requires no training, incurs no additional inference cost, and is immediately applicable to closed-source models. It has therefore been widely adopted for low-resource domain adaptation and rapid prototyping.

More importantly, prompt optimization is particularly well-suited for low-resource, fast-evolving, and data-scarce domains, where pre-training, fine-tuning, or building a full retrieval-augmented system is infeasible. In such settings, domain knowledge is often incomplete, unstable, or too small to justify heavy-weight training pipelines. Prompt-based methods provide a low-cost, lightweight, and rapidly adaptable mechanism for injecting task knowledge at inference time.

In this work, we deliberately focus on prompt optimization. This choice is not motivated by simplicity alone, but by a methodological principle: the simpler the injection mechanism, the fewer design decisions are required, and the more generalizable the method becomes. Prompt-based knowledge injection minimizes architectural and training assumptions, avoids dependence on external infrastructure, and offers a unified interface across tasks and domains. As a result, it provides a robust foundation for long-term, general-purpose structured prompting frameworks in complex conceptual and reasoning tasks.

2.3. Knowledge Representation: From Graphs to Tree Structures

2.3.1. Knowledge Graphs and Directed Acyclic Graphs

Knowledge graphs and directed acyclic graphs (DAGs) are widely adopted formalisms for representing structured domain knowledge[22]. Knowledge graphs encode entities and their relations as general graphs, enabling rich relational reasoning and flexible traversal. DAGs further impose acyclicity, supporting hierarchical abstraction while still allowing multiple parent–child dependencies. These representations are particularly suitable for modeling complex, interconnected domains and have been extensively used in ontology engineering, reasoning systems, and retrieval-augmented generation.

However, while graph-based representations offer high expressiveness, they also introduce significant challenges in terms of construction cost, algorithmic complexity, and prompt design. Traversing and linearizing general graphs or DAGs for LLM consumption requires non-trivial design choices, increases the number of structural decisions, and complicates both automation and evaluation.

2.3.2. Concept Tree Structure: A Simplified Approach

To balance expressiveness with practical constraints, this thesis adopts a concept tree as a simplified knowledge representation.

This design choice is motivated by three considerations:

1. Cost management: Tree structures significantly reduce construction, traversal, and serialization costs compared to general graphs or DAGs.

2. Complexity management: A tree imposes a strict hierarchical organization, simplifying both automated generation and structured prompting.

3. Thesis time constraints: Given the limited time for development and evaluation, a tree-based abstraction provides a tractable, reproducible experimental setting.

This approach relies on two explicit modeling assumptions:

- Tree representability: Domain knowledge can be approximated as a hierarchical structure, where abstract concepts dominate concrete sub-concepts.

- Single-tree assumption: Each domain is represented by a single primary concept tree, serving as a canonical conceptual backbone.

While these assumptions restrict representational generality, they intentionally reduce the design space and the number of modeling decisions. This simplification is not a limitation by convenience, but a methodological choice: the simpler the representation, the fewer structural degrees of freedom, and the more generalizable the framework becomes in long-term, cross-domain settings.

Node Attributes in the Concept Tree

Each node in the concept tree is represented as a structured record with a fixed set of semantic and relational attributes, as summarized in Table 2.1. This design preserves essential semantic information while maintaining a strictly hierarchical and machine-interpretable structure.

Table 2.1.: 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.

Motivation and design rationale The selection of node attributes in the concept tree is deliberate and function-driven, reflecting two core objectives: constructing a seman-

tically coherent hierarchical representation of domain knowledge, and enabling deterministic traversal and controlled prompt generation for downstream reasoning tasks. Rather than modeling concepts as unstructured text nodes, each node is represented as a semantically enriched record that explicitly encodes concept identity, semantic role, relational context, relevance signals, and pedagogical guidance.

This design avoids the complexity of maintaining a separate graph-level edge schema while retaining sufficient expressive power to support recursive expansion, relevance-aware pruning, and structured prompt synthesis. The motivations behind the individual attributes are explained below.

Node identity and type The `id` attribute uniquely identifies each node and directly corresponds to the concept name, enabling unambiguous referencing during traversal and downstream prompt construction. The `type` attribute explicitly distinguishes the semantic role of a node, such as an object, abstract concept, organizational unit, tool, or contextual label. This differentiation prevents semantic ambiguity during recursive expansion and allows the language model to apply type-aware reasoning when generating child concepts and prompts.

Semantic grounding through description The `description` attribute provides a concise but expressive semantic definition of the concept. It serves two purposes. First, it acts as a grounding signal during child node retrieval, guiding the language model toward semantically compatible expansions. Second, it supplies controlled explanatory content during prompt generation. By limiting the scope and length of descriptions at the node level, the concept tree maintains semantic precision while avoiding prompt overloading, which improves both tree quality and downstream reasoning performance.

Implicit relational encoding Instead of representing relations as a separate edge data structure, parent-child relations are embedded directly within nodes using the attributes `relation_from_parent` and `relation_description`. This design simplifies the overall representation and traversal logic while preserving explicit relational semantics. Storing relational information locally within each node ensures that contextual relationships are always available during traversal and prompt generation, which improves conceptual coherence across the generated prompt sequence.

Relevance-aware control signals The `score_parent` and `score_domain` attributes provide quantitative relevance signals that guide the recursive construction of the con-

cept tree. The parent relevance score captures local semantic compatibility with the immediate parent, while the domain relevance score reflects global alignment with the target domain context. These scores enable relevance-based filtering, adaptive stopping criteria, and controlled recursion depth, directly contributing to the accuracy and domain focus of both the concept tree and the resulting prompts.

Support for in-context learning The `example` attribute stores illustrative examples or code snippets associated with each concept. These examples are later injected into prompts to support in-context learning, allowing the language model to infer syntactic patterns, formatting conventions, and usage contexts. Associating examples directly with concepts ensures that demonstrations remain contextually relevant and pedagogically aligned with the concept being explained or applied.

Hierarchical organization The `children` attribute encodes the hierarchical structure of the concept tree by maintaining explicit references to subordinate concepts. This organization enables deterministic traversal strategies, such as breadth first search, and supports systematic prompt sequencing. The hierarchical structure formed by the `children` attribute provides the structural foundation for the proposed Chain of Concepts mechanism.

This attribute schema enables each node to encode both conceptual content and structural context, supporting deterministic traversal, structured prompting, and fine-grained relevance control within the concept tree.

2.3.3. Core Differences Between Graph, DAG, and Tree Representations

Although knowledge graphs, DAGs, and trees are closely related formalisms, they differ fundamentally in their structural constraints and the implications these constraints have for automation and reasoning.

Knowledge graphs impose minimal structural restrictions: nodes may have arbitrary numbers of parents and children, and cycles are permitted. This expressiveness enables rich relational modeling but leads to ambiguity in traversal order, non-unique linearizations, and increased algorithmic complexity. As a result, both construction and downstream consumption by LLMs require extensive heuristics and domain-specific design choices.

Directed acyclic graphs restrict cycles while retaining the ability for nodes to have multiple parents. DAGs support partial ordering and hierarchical abstraction, making them well suited for ontology engineering and dependency modeling. However, multiple inheritance introduces structural ambiguity: a concept may appear in several semantic contexts, complicating prompt serialization, traversal strategies, and evaluation. Linearizing a DAG for sequential prompting therefore requires additional disambiguation policies.

Trees impose the strongest constraint: each node has exactly one parent, except for the root. This enforces a single, unambiguous hierarchical path from the most abstract concept to any concrete concept. While this limits representational expressiveness, it guarantees deterministic traversal, unique serialization, and a fixed conceptual ordering. These properties are particularly advantageous for automated concept extraction, recursive expansion, and breadth-first prompt generation.

In summary, graphs and DAGs prioritize expressiveness and relational richness, whereas trees prioritize determinism, simplicity, and controllability. This thesis deliberately adopts a tree-based representation to minimize structural degrees of freedom, reduce algorithmic and prompting complexity, and enable systematic evaluation of concept-driven knowledge injection.

2.4. Ontology Engineering as a Downstream Task

The Web Ontology Language (OWL) provides a formal and expressive framework for representing structured domain knowledge in a machine-interpretable form. OWL supports the definition of classes, properties, individuals, and logical constraints, enabling automated reasoning, consistency checking, and semantic interoperability across systems.

In practice, OWL ontologies are commonly serialized using the Turtle (Terse RDF Triple Language) format. Turtle is a concrete syntax for RDF graphs, designed to be both compact and human-readable. Rather than introducing new semantic constructs, Turtle provides syntactic conveniences such as namespace prefixes, abbreviated triple expressions, and statement grouping, which make OWL axioms easier to author, inspect, and debug. This balance between formal rigor and readability makes Turtle a practical choice for ontology engineering workflows that involve both automated generation and manual validation. In this thesis, Turtle is adopted as the target representation for evaluating structured ontology generation.

From a structural perspective, Turtle represents knowledge as a set of RDF triples of the form

$$\langle \text{subject}, \text{predicate}, \text{object} \rangle,$$

where subjects and objects denote resources or literals, and predicates denote relationships or properties. OWL ontological constructs are expressed by composing such triples using standardized vocabularies, including `owl:`, `rdfs:`, and `rdf:`. Consequently, an OWL ontology serialized in Turtle can be viewed as a labeled, directed graph augmented with formal semantic constraints.

At a conceptual level, OWL ontologies consist of:

- **Classes**, which represent abstract concepts or categories in a domain and are declared using `owl:Class`;
- **Object properties**, which define typed relationships between classes;
- **Data properties**, which associate classes with literal values such as numbers or strings;
- **Individuals**, which represent concrete instances of classes.

In Turtle, class hierarchies are commonly expressed using the `rdfs:subClassOf` predicate, forming directed acyclic structures that encode different levels of conceptual abstraction. Property semantics are constrained through domain and range declarations, which specify the classes to which a property may be applied and the types of values it may take. These axioms implicitly define reasoning rules: for example, if an individual participates as the subject of a property whose domain is a given class, that individual can be inferred to belong to the domain class.

Listing 2.1 illustrates a simplified OWL ontology expressed in Turtle format. The example models a basic domain in which devices, sensors, and physical quantities are represented using class hierarchies and semantic relationships.

Listing 2.1: OWL ontology with concept hierarchy and relations in Turtle format

```
@prefix ex: <http://example.org/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

ex:Device a owl:Class .
```

```
ex:Sensor a owl:Class ;
    rdfs:subClassOf ex:Device .

ex:Actuator a owl:Class ;
    rdfs:subClassOf ex:Device .

ex:PhysicalQuantity a owl:Class .

ex:measures a owl:ObjectProperty ;
    rdfs:domain ex:Sensor ;
    rdfs:range ex:PhysicalQuantity .
```

In this example, `ex:Sensor` and `ex:Actuator` inherit from the more abstract concept `ex:Device`, forming a simple conceptual hierarchy. The object property `ex:measures` further constrains the ontology by specifying that only sensors may participate as subjects of this relation and that their measurements must correspond to physical quantities. These constraints are semantically meaningful and enable logical inference and validation by OWL reasoners.

Such explicit and formally grounded representations make OWL ontologies well suited for modeling complex domains in which correct reasoning depends on understanding both conceptual structure and inter-concept relationships. At the same time, the strict syntactic and semantic requirements imposed by Turtle make ontology generation a challenging downstream task for large language models. This characteristic makes Turtle-based OWL ontologies a suitable evaluation target for assessing whether injected conceptual knowledge improves structural correctness, semantic consistency, and constraint compliance in generated outputs.

2.4.1. LLM Limitation in Ontology Generation

Despite their strong general-purpose language capabilities, large language models face fundamental limitations when applied to ontology generation tasks[23].

First, pre-training corpora rarely contain explicit and well-structured concept hierarchies, resulting in a lack of detailed hierarchical knowledge required for precise ontology modeling. Second, domain-specific ontological knowledge is often sparsely represented in general training data, leading to incomplete or incorrect class and relation definitions in specialized domains. Third, ontology construction requires strict structural preci-

sion, including correct use of OWL constructs, well-formed class axioms, and logically consistent relationships. Such formal requirements are not naturally aligned with the probabilistic text generation paradigm of LLMs.

As a consequence, LLMs tend to produce syntactically invalid Turtle files, inconsistent hierarchies, or semantically incorrect relations when asked to generate ontologies directly.

2.4.2. Ontology Generation as a Knowledge Injection Use Case

Ontology generation in Turtle format constitutes a particularly suitable downstream task for evaluating structured knowledge injection.

This task simultaneously requires:

- Precise modeling of concept hierarchies,
- Accurate representation of domain-specific relations, and
- Strict adherence to formal syntax and semantics.

These requirements expose the core weaknesses of vanilla LLM prompting while directly benefiting from structured conceptual guidance. By framing ontology generation as a knowledge injection problem, this thesis aims to enable or significantly improve LLMs' data modeling capabilities through structured, prompt-based knowledge injection.

In this setting, the concept tree provides an explicit intermediate representation of domain knowledge, which is injected into the prompt to guide the model toward generating syntactically valid and semantically consistent OWL ontologies. This approach addresses the fundamental limitation of LLMs in proprietary and low-resource knowledge domains, while leveraging their strong natural language understanding and generation abilities.

2.5. Ontology Evaluation: OntoClean

Ontology evaluation is a critical aspect of ontology engineering, as syntactically valid ontologies may still exhibit semantic inconsistencies, conceptual misclassifications, or violations of modeling principles. While structural and logical checks can identify certain errors, they are often insufficient to assess whether concepts are correctly defined

and organized at the semantic level. OntoClean is a well-established ontology evaluation framework that addresses this gap by providing formal criteria for assessing the ontological soundness of hierarchical structures.

OntoClean[24] is based on the notion of meta-properties, which characterize the ontological nature of concepts independently of their domain-specific meaning. The meta-properties used in OntoClean include:

- Rigidity, which specifies whether a property is essential to all its instances;
- Identity, which indicates whether a concept supplies a principle for distinguishing and identifying its instances;
- Unity, which concerns whether instances of a concept form a coherent whole;
- Dependence, which captures whether the existence of an instance depends on another entity.

These meta-properties impose constraints on how concepts may be organized within a taxonomy. In particular, OntoClean defines a set of inheritance constraints that restrict subclass relationships. For example, an anti-rigid concept (such as a role) must not have a rigid concept as a subclass, and a concept that does not supply an identity criterion should not subsume one that does. Violations of these constraints indicate ontological modeling errors, even if the ontology is logically consistent.

The key contribution of OntoClean lies in its emphasis on semantic correctness rather than syntactic validity alone. By evaluating whether hierarchical relationships respect ontological principles, OntoClean enables the detection of subtle modeling flaws that may otherwise remain unnoticed. This is especially important in complex domains, where incorrect abstraction levels or improper concept generalizations can propagate errors throughout the ontology.

2.6. Gap Summary

The literature reviewed in this chapter reveals a fundamental tension in contemporary LLM-based knowledge work: while concept-driven prompting strategies such as chain of thoughts, in-context learning, and possibly Chain of Concepts demonstrate clear benefits for complex reasoning tasks, and while industrial applications increasingly demand

structured knowledge injection from proprietary documentation, existing approaches suffer from a critical limitation, the construction and validation of conceptual structures remain largely manual processes.

Current concept-driven approaches, including Chain of Concepts or Onto, rely on manually curated concept hierarchies. This manual dependency creates three interrelated problems:

- Scalability

  Manual curation does not scale to large or continuously evolving documentation corpora, limiting applicability in real-world industrial settings.

- Reproducibility

  Hand-crafted concept structures are difficult to reproduce across domains, experimental settings, or research teams, hindering comparative evaluation and methodological validation.

- Semantic consistency

  Manual construction is prone to modeling errors, including incorrect hierarchies, missing or redundant relations, and inconsistent abstraction levels.

This thesis aims to address both gaps by developing an automated pipeline to construct concept trees from proprietary documentation and apply them to structured knowledge injection tasks. Specifically, it transitions from manual concept design to automated, reproducible, and semantically validated construction of hierarchical knowledge representations, enabling practical deployment of concept-driven prompting in industrial and low-resource domain settings.

3. Methodology: Automated Chain of Concepts

3.1. Overview of Automated Chain of Concepts (AutoCoC)

This chapter presents the Automated Chain of Concepts (AutoCoC) method, a novel approach to structured knowledge injection for large language models. AutoCoC is inspired by the Chain of Concepts framework, but introduces a fundamental shift from manual to fully automated construction of conceptual structures. The process can be formalized as:

$$\mathbf{p}^* = \arg\min_{\mathbf{p}} \mathcal{L}(M([\mathbf{p}, \mathbf{x}]; \theta), \mathbf{y}^*), \tag{3.1}$$

where

- $\mathbf{p}^*$: The optimal prompt found through optimization.
- $\mathbf{x}$: The task-specific input data.
- θ: The static, pre-trained parameters of the language model.
- $M([\mathbf{p}, \mathbf{x}]; \theta)$: The model's prediction given the prompt and input.
- $\mathbf{y}^*$: The gold-standard or target output.
- $\mathcal{L}$: The loss function measuring prediction error.

The original Chain of Concepts approach demonstrates that providing models with well-designed concept structures and generating sequences of prompts that explain concepts and their relationships in order of abstraction level can effectively inject domain knowledge and enable the ability to assist with domain-specific tasks. This is achieved by filling knowledge gaps with curated relations and hierarchies between concepts, while

constraining the model to reason over the provided conceptual information rather than relying solely on its pre-training data.

However, manual construction of concept hierarchies poses significant limitations in scalability, reproducibility, and semantic consistency. AutoCoC addresses these limitations by automating the entire pipeline of concept structure construction and prompt generation.

The AutoCoC methodology structures knowledge from documentation using the concept tree representation. The pipeline consists of two main stages:

1. Automated Concept Tree Construction: This stage automates the construction of concept hierarchies and relations to build the concept tree representation. Using LLM-based extraction and structuring techniques, the system processes domain documentation to identify concepts, establish hierarchical relationships, and populate node attributes. This stage is followed by retrieval of examples or code snippets to support in-context learning, enriching each concept node with concrete instantiations that enable pattern induction and syntax imitation.

2. Structured Prompt Generation: This stage generates the corresponding sequence of prompts by traversing the concept tree. The traversal strategy determines the order in which concepts are presented to the model, ensuring that abstract concepts are introduced before concrete ones, and that conceptual dependencies are respected throughout the prompting sequence.

By automating both the construction of conceptual structures and their translation into structured prompts, AutoCoC enables scalable, reproducible, and semantically grounded knowledge injection for downstream tasks in proprietary and low-resource domains.

The following sections detail each component of the AutoCoC pipeline, including concept extraction, tree construction, example retrieval, prompt sequence generation, and evaluation methodology.

3.2. Workflow

Figure 3.1 illustrates the high-level workflow of the AutoCoC pipeline. The process begins with text documentation as input and proceeds through three main stages. First, the recursive top-down algorithm constructs the concept tree by extracting concepts and their hierarchical relationships from the documentation. Second, the system retrieves

relevant examples and code snippets to enrich each concept node with concrete instantiations. Finally, a breadth-first traversal algorithm generates the AutoCoC output: a structured sequence of prompts that presents concepts in order of increasing specificity, ensuring that abstract foundational concepts are introduced before their more concrete specializations.

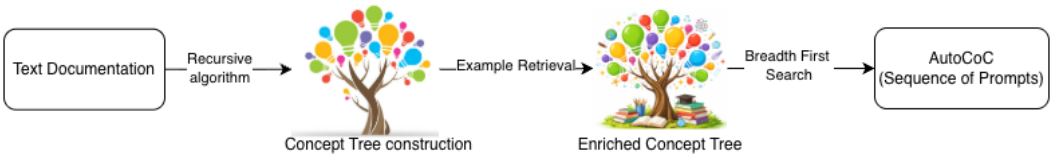

Figure 3.1.: High-Level Workflow of Automated Chain of Concepts

3.3. Recursive Top-Down Concept Tree Construction Algorithm Overview

Algorithm 1 illustrates the overall procedure for constructing a rooted concept tree in our approach. The algorithm operates on several key inputs and control parameters that jointly define the scope, depth, and computational budget of the tree construction process.

The primary input is the documentation D, which contains the domain-specific conceptual knowledge required to solve downstream conceptual or modeling tasks. This documentation may consist of engineering guidelines, official tool documentation, standards, or technical manuals. In addition, the algorithm takes a domain context Δ, provided as a concise textual description of the target domain. The domain context serves to constrain and focus the concept extraction process, guiding the language model toward domain-relevant interpretations and relationships.

The root concept r specifies the most abstract concept in the hierarchy and serves as the entry point of the recursive expansion procedure. Starting from this root node, the algorithm incrementally builds the concept tree by discovering and attaching more specialized concepts, as shown in Figure 3.2. To ensure tractability and prevent uncontrolled expansion, the algorithm is governed by several hyperparameters. These include a maximum depth limit K, which bounds the hierarchical depth of the resulting tree, and a call budget M, which restricts the total number of language model invocations. In addition, policy thresholds are used to decide whether candidate concepts should be

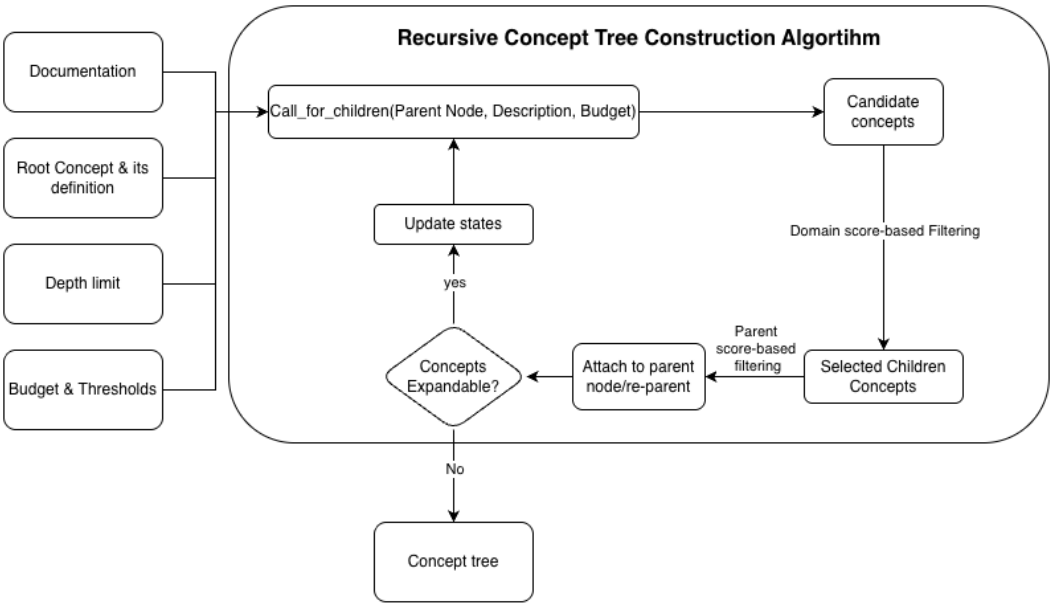

Figure 3.2.: Concept Tree Construction Workflow

attached or further expanded, based on edge strength, domain relevance scores, and cycle-prevention constraints.

The construction process is implemented as a recursive procedure *Expand*, which, for a given parent concept and current depth, queries the language model for candidate child concepts along with relationship types and confidence scores. Weak or cyclic relationships are filtered out, while stronger relationships may trigger attachment or reattachment operations to maintain a coherent and non-redundant tree structure. Among the attached children, only those satisfying the expansion policy, typically those with maximal domain relevance scores, are selected for further recursive expansion. The algorithm terminates when any stopping condition is met, and returns the root node of the constructed concept tree.

Algorithm 1 Automated Recursive Concept Tree Builder

Require: Documentation D, domain context Δ, root concept r, depth limit K, call budget M, policy thresholds (attach/expand)

Ensure: Rooted concept tree T

Initialize state (node index, parent map, expanded set, remaining budget M)
procedure EXPAND(p, d)
    if $d \geq K$ **or** $p \in$ expanded **or** $M = 0$ **then**
        return
    end if
    $M \leftarrow M - 1$
    $C \leftarrow$ LLMCALLFORCHILDREN($p,\ D,\ \Delta$) ▷ Each item: child u, edge score s_p, domain score s_d, relation rel
    for all $(u, s_p, s_d, rel) \in C$ **do**
        if TOOWEAK(s_p) **or** CREATESCYCLE(p, u) **then**
            continue
        end if
        if u unattached **then**
            ATTACH($p \to u,\ rel,\ s_p$)
        else if STRONGEREDGE(s_p) **then**
            REATTACH($p \to u,\ rel,\ s_p$)
        end if
        Record s_d for u
    end for
    $W \leftarrow$ children of p passing expand policy, with *maximal* s_d (ties allowed)
    for all $u \in W$ **do**
        EXPAND($u,\ d + 1$)
    end for
end procedure
EXPAND($r, 0$); **return** r

3.4. Dual Scoring System

AutoCoC employs a dual scoring system to decouple two fundamentally different decisions in concept tree construction: (i) whether a candidate concept should be attached as a child of a given parent, and (ii) whether that concept should be further expanded

during recursive traversal.

For each candidate child concept proposed during tree construction, the language model assigns two independent scores:

- *score_parent*:

  Measures the semantic strength of the hierarchical relationship between the candidate concept and the proposed parent. This score reflects whether the candidate is a natural taxonomic, definitional, or conceptual specialization of the parent concept, and thus whether the parent–child attachment is structurally appropriate.

- *score_domain*:

  Measures the relevance and centrality of the candidate concept within the target domain context Δ. This score reflects how essential the concept is to the domain as a whole, independent of its local position in the hierarchy, and indicates whether it is a strong candidate for further recursive expansion.

Both scores are generated by the language model during the LLMCallForChildren operation. To enable consistent interpretation and deterministic filtering, AutoCoC maps raw score values to fine-grained semantic categories using predefined threshold ranges, as summarized in Table 3.1.

Table 3.1.: Two-score scheme with granular semantic interpretations

Range	*score_parent*	*score_domain*
0.90–1.00	Definitional or taxonomic direct child; attachment is almost certain.	Core domain concept; highest priority for recursive expansion.
0.70–0.89	Standard and widely accepted direct child.	Important and frequently referenced within the domain.
0.40–0.69	Semantically related but secondary.	Supportive or auxiliary concept.
< 0.40	Marginal or weak hierarchical relation.	Peripheral to the domain; rarely expanded.

Table 3.1 illustrates how the same numeric range has distinct semantic interpretations depending on the scoring dimension. For *score_parent*, higher values indicate stronger hierarchical validity, while low values signal weak or questionable parent–child relationships. For *score_domain*, higher values identify concepts that are central to the domain and thus valuable for deeper exploration, whereas low values indicate peripheral concepts that contribute limited domain coverage.

The dual scoring system directly supports multiple algorithmic mechanisms. The TooWeak filter relies on *score_parent* to exclude candidate edges that fail to meet minimum attachment criteria. The StrongerEdge policy compares *score_parent* values across alternative parents and enables dynamic reattachment when a more semantically appropriate parent is discovered. In contrast, the expansion policy operates exclusively on *score_domain*, prioritizing concepts with high domain relevance for recursive expansion and budget allocation.

By separating hierarchical correctness from domain importance, the dual scoring system allows AutoCoC to balance structural coherence with domain focus. Strong parent–child relationships ensure a well-formed conceptual hierarchy, while selective expansion guided by domain relevance steers the tree toward the most informative and representative regions of the conceptual space.

3.5. Dynamic Reparenting

During the recursive construction of the concept tree, the algorithm may encounter the same concept multiple times in different contexts or at different levels of the hierarchy. The initial placement of a concept under a particular parent is based on the information available at that point in the traversal. However, as the tree grows and more concepts are discovered, a previously attached concept may be found to have a stronger or more natural relationship with a newly discovered parent node.

Dynamic reparenting addresses this challenge by allowing the algorithm to reassign concepts to more appropriate parents when stronger hierarchical relationships are identified. This mechanism is essential for achieving an optimal concept tree structure, as it enables the system to refine the hierarchy incrementally rather than committing to potentially suboptimal placements.

Conditions for Reparenting Reparenting is triggered when a concept u that already exists in the tree is proposed as a child of a new parent p' during a subsequent $LLMCallForChildren$ operation. The algorithm compares the *score_parent* of the existing edge (connecting u to its current parent p) with the *score_parent* of the proposed new edge (connecting u to p').

If the new edge exhibits a higher *score_parent*, indicating a stronger or more semantically appropriate hierarchical relationship, the algorithm performs a reparenting operation. Specifically, the system:

1. Detaches u and all its descendant nodes from the current parent p
2. Reattaches u to the new parent p' with the updated edge score and relationship type
3. Updates the parent mapping to reflect the new hierarchical position
4. Preserves the entire subtree rooted at u, maintaining all previously established child relationships

This operation is subject to cycle prevention checks: reparenting is only permitted if it does not introduce a cycle in the tree structure. The *CreatesCycle* function ensures that p' is not a descendant of u, thereby maintaining the acyclic property of the concept tree.

Impact and Benefits Dynamic reparenting yields several important benefits for concept tree construction:

- Optimized Hierarchical Structure:

  By allowing concepts to migrate to more appropriate parents as the tree evolves, the algorithm converges toward a more semantically coherent and domain-aligned hierarchy. Concepts are ultimately positioned under parents with which they share the strongest taxonomic or definitional relationships.

- Prevention of Suboptimal Early Decisions:

  Without reparenting, the algorithm would be constrained by the order in which concepts are discovered. Early placements might be based on incomplete information, leading to a suboptimal final structure. Reparenting mitigates this issue by enabling retrospective refinement.

- Enhanced Stability and Consistency:

  By continuously selecting the strongest available parent–child relationships, the algorithm produces a more stable tree structure that better reflects the true conceptual organization of the domain. This stability improves the downstream effectiveness of the structured prompting sequence.

- Cycle Prevention and Tree Integrity:

The reparenting mechanism, combined with explicit cycle checks, ensures that the concept tree remains acyclic and well-formed throughout the construction process, even as concepts are dynamically repositioned.

Through dynamic reparenting, the AutoCoC algorithm achieves a level of structural optimization that would be difficult to obtain through a single-pass, fixed-placement strategy. This iterative refinement process is a key innovation that enables the system to construct concept trees that are both hierarchically coherent and domain-optimized.

3.6. CoC Generation from the Concept Tree

Once the concept tree has been constructed, the AutoCoC system generates a sequence of structured prompts through a level-by-level breadth-first traversal. This process transforms the hierarchical concept tree into a pedagogically ordered sequence of formal syntax specifications.

The generation algorithm proceeds as follows:

1. Extract tree levels using BFS traversal
2. Generate a foundation prompt for the root concept, establishing overall framework syntax and rules
3. For each subsequent level, sort concepts by *score_domain* and generate individual concept prompts using LLM calls
4. Each prompt includes the concept's definition, relationship context, canonical examples, and prerequisite concepts

The output is available in multiple formats: human-readable documents, structured JSON, and chat history formats designed for LLM training. Each generated prompt uses low temperature (0.2) to ensure consistent, precise syntax specifications rather than creative variation.

4. Experiment

4.1. Datasets

This section describes the three datasets used to generate the concept tree and further prompts in our AutoCoC method.

4.1.1. Handcrafted CoC(Baseline)

The first dataset consists of a handcrafted Chain of Concepts curated by domain experts. This dataset serves as our baseline and is used without any preprocessing. It contains 3,495 tokens of carefully structured conceptual knowledge.

4.1.2. Reverse-Engineered CoC

The second dataset is generated through reverse engineering, where we apply our AutoCoC method to the handcrafted CoC baseline. The goal is to refine, enrich, or reconstruct the expert-curated content, demonstrating AutoCoC's ability to work with existing structured knowledge. This dataset also contains 3,495 tokens.

4.1.3. OWL Official Documentation

The third dataset comprises the official OWL (Web Ontology Language) specification documentation. This dataset contains 43,757 tokens from the main documentation and 2,596 tokens of Turtle serialization syntax, which support example retrieval for concept nodes during tree construction.

4.2. OWL Generation

The primary downstream task chosen for this experiment is the generation of OWL (Web Ontology Language) ontologies serialized in the Turtle format. This task requires

the system to transform natural language domain descriptions into formal, machine-interpretable knowledge structures consisting of classes, properties, and logical axioms.

We selected ontology generation because it rigorously evaluates the model's ability to maintain structural and semantic consistency. Unlike open-ended generation tasks, ontologies have strict validity criteria:

- Syntactic Correctness:

  The output must be valid Turtle syntax.

- Hierarchical Logic:

  The class structures must be logically sound (e.g., correct $subClassOf$ relations).

- Domain Accuracy:

  The defined concepts must accurately reflect the domain expertise.

Using LLMs as assistants in this task represents a significant shift from traditional manual ontology engineering. Manual creation is time-consuming, requires specialized expertise, and is difficult to scale. LLMs can process vast amounts of documentation to suggest concepts and relations rapidly. However, without structured guidance, they often hallucinate or produce invalid syntax. The impact of this experiment lies in demonstrating how structured knowledge injection (AutoCoC) can harness the semantic power of LLMs while ensuring the structural rigor required for valid ontology generation.

4.3. Experimental Protocol

This section describes the protocols defined to ensure fair, scientific evaluation and comparison across different methods.

4.3.1. Concept Tree Generation Protocol

To evaluate the self-consistency of concept tree generation, we repeated the generation process 200 times for each domain using identical input parameters. Statistical analysis of these repeated generations allows us to measure the consistency and stability of the generated concept trees across multiple runs.

4.3.2. Downstream Task Protocol

The downstream ontology generation task is evaluated across 10 diverse domains, as shown in Table 4.1, to assess the true performance gap between different methods and prompting strategies. For each domain, we perform 30 independent generations using identical inputs and calculate the average evaluation results to account for variability.

Domain	Description
Academic	Models academic institutions, programs, personnel, and educational processes
Automotive	Models vehicles, components, transportation infrastructure, and manufacturing
Biomedical	Models medical conditions, treatments, healthcare professionals, and clinical workflows
Coffee	Models coffee ingredients, equipment, brewing methods, and quality characteristics
Motors	Models industrial electric motors, components, and operational specifications
Pizza Making	Models pizza ingredients, equipment, preparation techniques, and styles
Real Estate	Models buildings, property transactions, and geographic aspects
Robotic Control	Models robots, sensors, control systems, and operational modes
Smart Home	Models IoT devices, automation systems, and environmental monitoring
Supply Chain	Models supply chain entities, logistics, inventory, and business relationships

Table 4.1.: Target domains for downstream OWL ontology generation.

4.3.3. Controlled Variables

Throughout the entire experiment, we use GPT-4o from OpenAI as the constant LLM to isolate the effect of different prompting strategies and knowledge injection methods.

4.4. Evaluation Metrics

4.4.1. Concept tree self-consistency

To evaluate the self-consistency of concept tree generation, you generate 200 trees using identical input parameters and analyze the stability and consistency across these runs.

- Concept Stability Score:

  This metric measures how consistently the same concepts are extracted across multiple runs.

$$\text{Concept Stability Score} = \frac{|\text{concepts appearing in } \geq 90\% \text{ of runs}|}{|\text{union of all concepts} - \text{rare concepts}|} \tag{4.1}$$

  Higher scores indicate more consistent concept extraction. A score close to 1.0 means the model reliably identifies the same core concepts.

- Concept Frequency Distribution

  This tracks how often each concept appears across all runs.

$$\text{Frequency}(\text{concept}_i) = \frac{\text{appearance count of concept}_i}{\text{total number of runs}} \tag{4.2}$$

  This helps identify "core" concepts (high frequency) versus "peripheral" concepts (low frequency), revealing which concepts are central to the domain versus which are inconsistently extracted.

- Concept Variance

  This measures the variability in the total number of concepts extracted across runs.

$$\text{Concept Variance} = \text{std}([\text{len}(\text{concepts}_{\text{run 1}}), \text{len}(\text{concepts}_{\text{run 2}}), \ldots, \text{len}(\text{concepts}_{\text{run 200}})]) \tag{4.3}$$

  Lower variance indicates more stable extraction size. If the standard deviation is low, the model consistently extracts approximately the same number of concepts.

- Core vs. Peripheral Ratio

This ratio characterizes the stability pattern of concept extraction.

$$\text{Core vs. Peripheral Ratio} = \frac{|\text{concepts with frequency} > 0.5|}{|\text{concepts with frequency} \leq 0.5|} \tag{4.4}$$

A higher ratio indicates that most extracted concepts are stable "core" concepts, suggesting reliable knowledge extraction. A lower ratio means many concepts are peripherally or inconsistently identified.

- Edge Stability Score

  This measures how consistently the relationships (edges) between concepts are extracted.

$$\text{Edge Stability Score} = \frac{|\text{edges appearing in } \geq 95\% \text{ of runs}|}{|\text{union of all edges} - \text{rare edges}|} \tag{4.5}$$

  Higher scores indicate stable relation extraction. This is crucial because the structure of the concept tree depends on consistent relationships between concepts.

- Parent-Child Stability

  This specifically measures the consistency of hierarchical relationships in the concept tree.

$$\text{Parent-Child Stability} = \frac{|\text{parent-child pairs appearing in } \geq 95\% \text{ of runs}|}{|\text{total unique parent-child pairs}|} \tag{4.6}$$

  This reveals how often the same hierarchical structure emerges. High stability means the model consistently recognizes the same conceptual hierarchies, which is essential for reliable knowledge representation.

4.4.2. Downstream Ontology quality

The generated OWL ontologies are evaluated using two complementary approaches: semantic correctness, based on the OntoClean methodology as described in Section 2.5, and syntactic validity, based on structural and quantitative metrics.

As discussed in the Section 2.5, OntoClean provides a rigorous framework for assessing ontological soundness through meta-properties (rigidity, identity, unity, dependence)

and their inheritance constraints. Our evaluation applies OntoClean principles to automatically detect violations in the generated ontologies using an LLM-based evaluator.

Each class in the generated ontology is automatically evaluated by an LLM that assigns meta-property labels based on the class's semantic definition, hierarchical position, and relationships within the ontology. The evaluator analyzes:

- Identity ($\pm I$):

  Whether instances can be uniquely identified

- Unity ($\pm U / \sim U$):

  Whether instances are perceived as wholes, collections, or lack cohesion

- Rigidity ($\pm R / \sim R$):

  Whether the property is essential, non-rigid, or anti-rigid

- Dependence ($\pm D$):

  Whether instances depend on entities from other classes

For each class, the system considers contextual information including parent classes, subclasses, and sibling classes to produce more accurate meta-property assignments. The assignment process produces a confidence score (0.0–1.0) reflecting the certainty of the label assignment, along with detailed reasoning explaining the choices.

Once meta-properties are assigned, the evaluation system checks whether all subclass relationships satisfy OntoClean inheritance constraints. Four primary violation types are detected:

- Identity inheritance violations:

  A parent class with $+I$ (supplies identity) has a subclass without $+I$

- Unity inheritance violations:

  A parent class with $+U$ (has unity) has a subclass without $+U$

- Rigidity inheritance violations:

  A parent class with $+R$ (rigid) has a subclass with $\sim R$ (anti-rigid)

- Dependence inheritance violations:

  A parent class with $+D$ (dependent) has a subclass without $+D$

Semantic Evaluation Metrics

Computation of Semantic Metrics. Let $\mathcal{O} = \{O_1, \ldots, O_N\}$ denote the set of generated ontology files, and let $\mathcal{C}_i$ be the set of OWL classes contained in ontology O_i. For each class $c \in \mathcal{C}_i$, an LLM-based evaluator assigns OntoClean meta-property labels for identity, unity, rigidity, and dependence, together with an associated confidence score $\gamma(c) \in [0, 1]$ that reflects the model's certainty in its classification.

Let $\mathcal{R}_i = \{(p, s)\}$ denote the set of all direct subclass relations in ontology O_i, where p is the parent class and s is the subclass. For each relation $(p, s) \in \mathcal{R}_i$, OntoClean inheritance constraints are checked. A constraint violation is recorded whenever a subclass s fails to inherit a required meta-property from its parent class p according to OntoClean rules, such as when a rigid parent class subsumes an anti-rigid subclass.

We define an indicator function $\mathbb{I}_{\text{viol}}(p, s)$ that takes value 1 if the relation (p, s) violates any OntoClean inheritance constraint and 0 otherwise. The total number of constraint violations across all ontologies is then computed as

$$V_{\text{total}} = \sum_{i=1}^{N} \sum_{(p,s)\in\mathcal{R}_i} \mathbb{I}_{\text{viol}}(p, s). \tag{4.7}$$

The average number of constraint violations per ontology file is defined as

$$V_{\text{avg}} = \frac{1}{N} \sum_{i=1}^{N} \sum_{(p,s)\in\mathcal{R}_i} \mathbb{I}_{\text{viol}}(p, s) = \frac{V_{\text{total}}}{N}. \tag{4.8}$$

To quantify the reliability of the LLM's meta-property assignments, we compute the average confidence score over all evaluated classes. Let $\mathcal{C} = \bigcup_{i=1}^{N} \mathcal{C}_i$ denote the set of all classes across all ontologies. The mean confidence is given by

$$\bar{\gamma} = \frac{1}{|\mathcal{C}|} \sum_{c\in\mathcal{C}} \gamma(c). \tag{4.9}$$

In addition to the mean, we summarize the empirical distribution of confidence scores using standard descriptive statistics, including the minimum, maximum, median, and standard deviation:

$$\gamma_{\min} = \min_{c\in\mathcal{C}} \gamma(c), \quad \gamma_{\max} = \max_{c\in\mathcal{C}} \gamma(c), \quad \sigma_\gamma = \sqrt{\frac{1}{|\mathcal{C}|} \sum_{c\in\mathcal{C}} (\gamma(c) - \bar{\gamma})^2}. \tag{4.10}$$

To analyze how frequently each OntoClean meta-property label occurs, we compute a normalized label distribution. Let $m \in \{\text{identity}, \text{unity}, \text{rigidity}, \text{dependence}\}$ index the meta-properties, and let $\ell \in \{+, -, \sim\}$ denote the positive, negative, and anti labels, respectively. We define

$$P_{m,\ell} = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \mathbb{I}\big(\text{label}_m(c) = \ell\big), \tag{4.11}$$

where $\text{label}_m(c)$ is the assigned label for meta-property m on class c, and $\mathbb{I}(\cdot)$ is the indicator function. This yields, for each meta-property, an empirical probability distribution over its possible labels.

Finally, to characterize systematic modeling errors, all detected constraint violations are grouped by violation type $t \in \{\text{identity}, \text{unity}, \text{rigidity}, \text{dependence}\}$. Let $\mathcal{V}_t$ denote the set of violations of type t. The frequency of each violation type is computed as

$$f_t = |\mathcal{V}_t|. \tag{4.12}$$

For each group, we additionally record the affected ontology files and representative example relations (p, s) that illustrate the nature of the violation.

Lower values of V_{total} and V_{avg}, together with higher mean confidence $\bar{\gamma}$ and well-formed label distributions, indicate stronger semantic correctness and better alignment with OntoClean's ontological modeling principles.

Syntactic Validity: Structural Metrics

In addition to semantic correctness, we evaluate the syntactic validity and structural characteristics of the generated ontologies. These metrics assess whether the ontologies are well-formed, machine-parseable, and contain the expected structural elements required for downstream semantic analysis and OntoClean evaluation.

Computation of File-Level Syntactic Metrics. Let $\mathcal{O} = \{O_1, \ldots, O_N\}$ denote the set of generated ontology files. Each ontology file is parsed using a standards-compliant OWL/Turtle parser. We define a binary indicator function

$$\mathbb{I}_{\text{parse}}(O_i) = \begin{cases} 1, & \text{if } O_i \text{ is successfully parsed without syntax errors,} \\ 0, & \text{otherwise.} \end{cases} \tag{4.13}$$

The total number of generated ontology files is

$$F_{\text{total}} = N. \tag{4.14}$$

The number of successfully parsed files is computed as

$$F_{\text{succ}} = \sum_{i=1}^{N} \mathbb{I}_{\text{parse}}(O_i), \tag{4.15}$$

and the number of files that fail to parse due to syntax or serialization errors is

$$F_{\text{fail}} = F_{\text{total}} - F_{\text{succ}}. \tag{4.16}$$

The average evaluation success rate, which quantifies the syntactic validity of the generated ontologies, is defined as the proportion of successfully parsed files:

$$R_{\text{succ}} = \frac{F_{\text{succ}}}{F_{\text{total}}}. \tag{4.17}$$

Computation of Class-Level Structural Metrics. For each successfully parsed ontology file O_i, all declared OWL classes are extracted by traversing the RDF graph representation of the ontology. Let $\mathcal{C}_i$ denote the set of extracted classes from ontology O_i, and let $\mathcal{C} = \bigcup_{i=1}^{N} \mathcal{C}_i$ be the set of all classes across all ontologies.

The total number of OWL classes defined across all generated ontologies is then

$$C_{\text{total}} = \sum_{i=1}^{N} |\mathcal{C}_i|. \tag{4.18}$$

The average number of classes per ontology file is computed as

$$C_{\text{avg}} = \frac{1}{F_{\text{total}}} \sum_{i=1}^{N} |\mathcal{C}_i|. \tag{4.19}$$

Not all extracted classes can necessarily be processed successfully by the OntoClean evaluator. Let $\mathcal{C}_i^{\text{eval}} \subseteq \mathcal{C}_i$ denote the subset of classes in O_i that receive a complete set of meta-property labels together with an associated confidence score. The total number

of evaluated classes is defined as

$$C_{\text{eval}} = \sum_{i=1}^{N} |\mathcal{C}_i^{\text{eval}}|. \tag{4.20}$$

The ratio $C_{\text{eval}}/C_{\text{total}}$ provides an additional indicator of structural well-formedness and evaluator compatibility.

Computation of Runtime Metrics. The evaluation runtime is measured at file granularity. For each ontology file O_i, the total time required to perform parsing, class extraction, meta-property assignment, and OntoClean constraint checking is recorded and denoted by t_i.

The total evaluation time across all ontology files is computed as

$$T_{\text{total}} = \sum_{i=1}^{N} t_i, \tag{4.21}$$

and the average evaluation time per ontology file is defined as

$$T_{\text{avg}} = \frac{1}{F_{\text{total}}} \sum_{i=1}^{N} t_i = \frac{T_{\text{total}}}{F_{\text{total}}}. \tag{4.22}$$

High values of R_{succ}, together with appropriate class counts C_{total} and C_{avg}, and moderate evaluation times T_{total} and T_{avg}, indicate that the generation pipeline produces syntactically valid and structurally sound ontologies at scale.

Combined Role of Syntactic and Semantic Metrics. By combining the syntactic metrics defined above with the semantic metrics derived from OntoClean analysis, we obtain a comprehensive and multi-dimensional assessment of ontology quality. Syntactic metrics ensure that generated ontologies are well-formed, parseable, and structurally complete, while semantic metrics verify that the conceptual modeling adheres to established ontological principles. Together, these metrics enable systematic comparison of different prompting strategies and knowledge injection methods across diverse domains.

5. Results

This chapter reports the empirical results of applying the proposed AutoCoC pipeline to multiple knowledge sources. We evaluate (i) the structure and content of the generated concept trees under different data regimes and (ii) the self-consistency of the recursive tree construction process under repeated stochastic sampling. Together, these analyses assess both the descriptive adequacy and the robustness of the proposed method.

5.1. Concept Tree Generation from Distinct Knowledge Sources

We construct concept trees from two qualitatively different data sources, each corresponding to a distinct knowledge-extraction scenario. The first scenario evaluates whether AutoCoC can reconstruct and enrich an existing expert-curated concept structure. The second scenario evaluates whether AutoCoC can induce a coherent and structured conceptual hierarchy directly from large-scale unstructured technical documentation.

5.1.1. Reverse-Engineered Chain of Concepts

The first dataset is generated through reverse engineering of a handcrafted Chain of Concepts (CoC) baseline. In this setting, the expert-curated CoC is treated as input documentation, and AutoCoC is applied to reconstruct, refine, and enrich the original conceptual structure. This experiment evaluates whether the method can recover stable high-level abstractions from an already structured representation and produce a coherent pedagogical prompt sequence from it.
Input. Handcrafted CoC processed through reverse engineering (3,495 tokens)
Output characteristics.

- Number of concept nodes: 9

- Generated AutoCoC length: 5,488 tokens

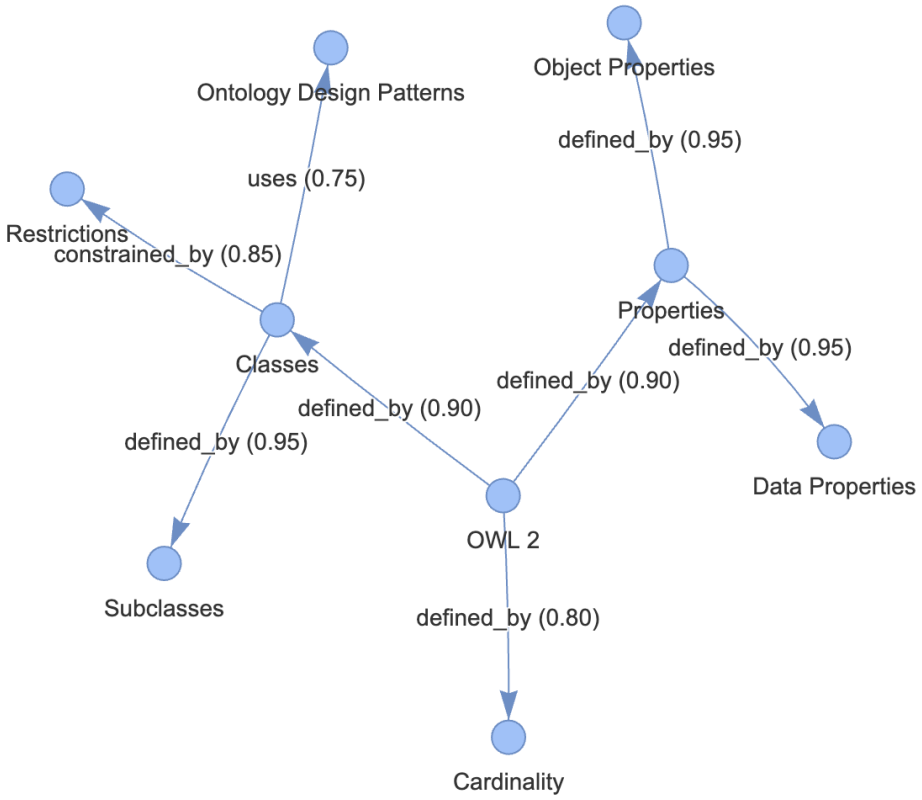

Figure 5.1.: Concept tree generated from the reverse-engineered handcrafted CoC.

The resulting tree preserves the high-level abstractions present in the original CoC while introducing refined intermediate concepts and clearer hierarchical groupings. This demonstrates that AutoCoC can operate in a knowledge-refinement regime, where the input already encodes structured expert knowledge.

5.1.2. OWL Official Documentation

The second dataset consists of the official OWL (Web Ontology Language) specification. This setting evaluates AutoCoC in a fully bottom-up knowledge-induction regime, where the method must extract and organize conceptual structure directly from unstructured technical text.

The documentation corpus comprises the main OWL specification and a supplementary Turtle serialization reference, which is used during example retrieval for concept nodes in the second stage of tree expansion.

Input.

- OWL specification documentation: 43,757 tokens
- Turtle syntax documentation: 2,596 tokens

Output characteristics.

- Number of concept nodes: 19
- Generated AutoCoC length: 9,541 tokens

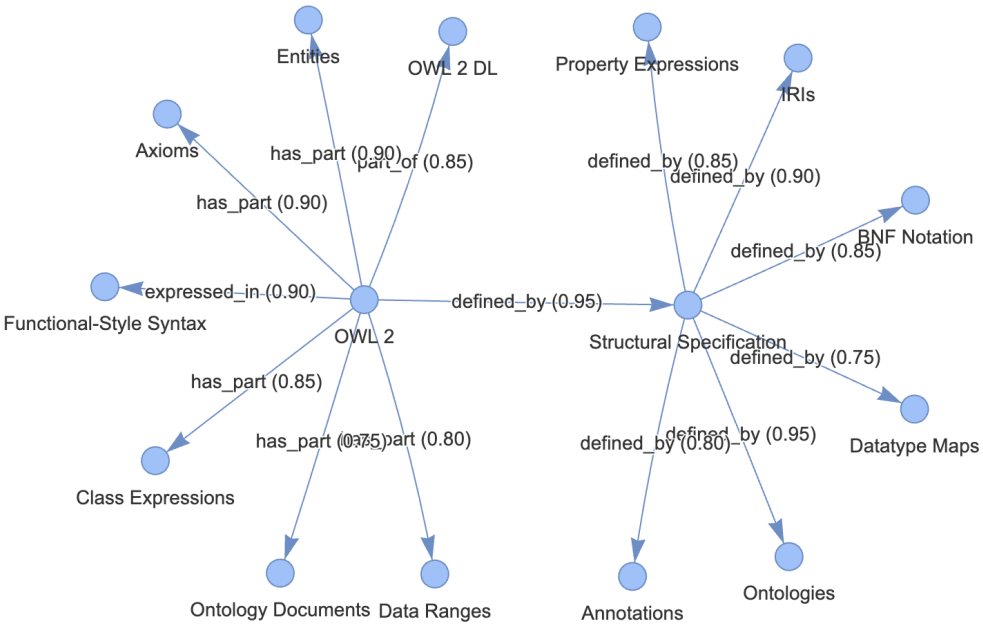

Figure 5.2.: Concept tree generated from the full OWL specification documentation.

The induced concept tree captures the major abstractions of OWL, including ontology entities, axioms, class expressions, properties, and serialization mechanisms. Compared to the reverse-engineered case, the tree is deeper and contains a richer set of intermediate abstractions, reflecting the increased conceptual breadth and heterogeneity of the source documentation.

5.2. Self-Consistency of Concept Tree Construction

AutoCoC constructs concept trees through recursive LLM calls with stochastic decoding. As a result, repeated executions with identical inputs may produce structurally different

yet semantically valid trees. To assess the robustness of this process, we perform a self-consistency analysis over 200 independent runs, evaluating the stability of concept retrieval, hierarchical relations, and output scale.

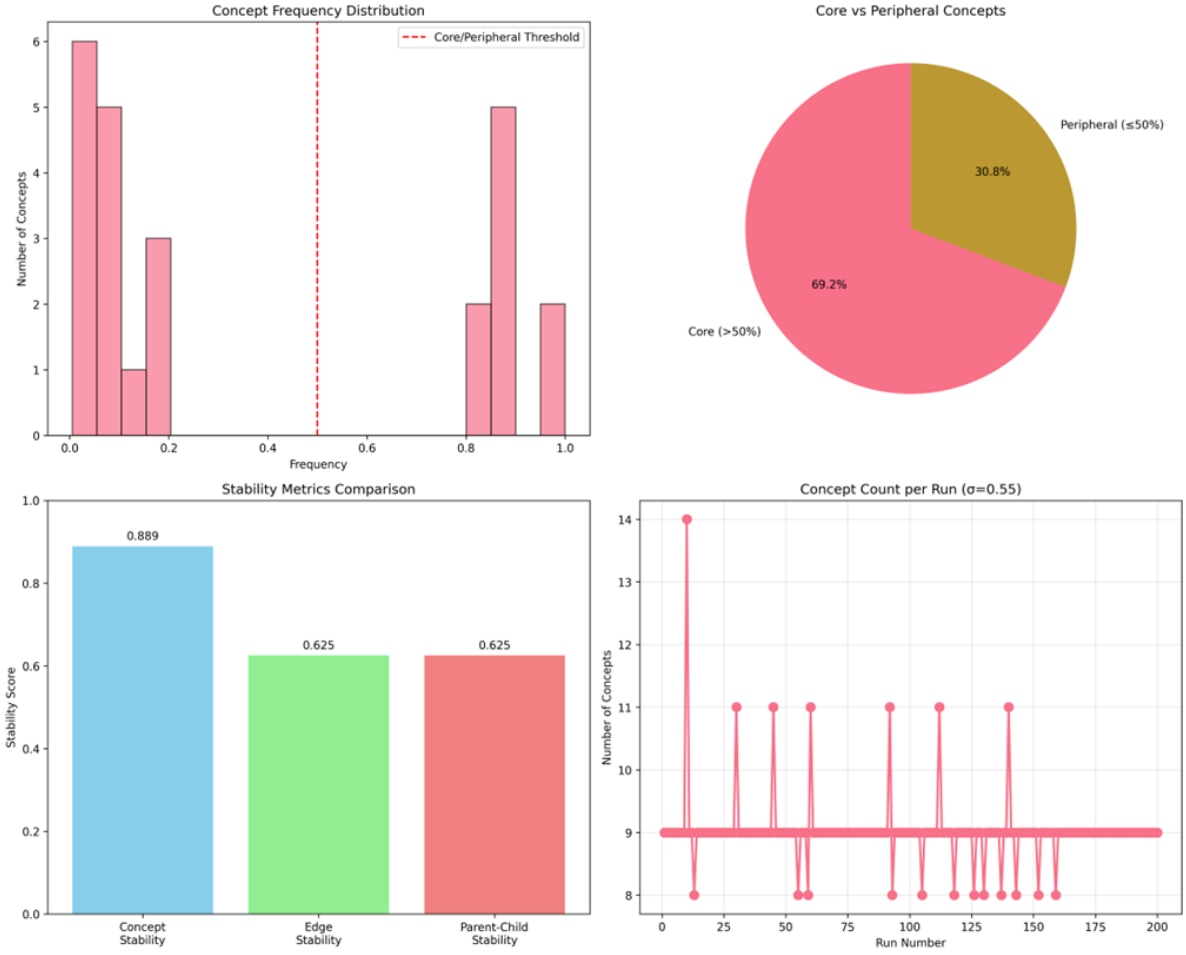

Figure 5.3.: Self-consistency analysis of concept tree construction across 200 runs. The top row shows concept recurrence statistics, while the bottom row reports stability metrics and tree size variability.

Concept Frequency and Core–Peripheral Structure The *concept frequency distribution* (top left) exhibits a strongly bimodal pattern. Concepts tend to appear either in nearly all runs or in only a small fraction of runs, indicating a clear separation between stable and transient elements.

Using a frequency threshold of 0.5, we categorize concepts into two groups (top right):

- **Core Concepts (> 50% frequency):** Accounting for 69.2% of all unique con-

cepts, these consistently recur across runs and form the stable semantic backbone of the concept tree.

- **Peripheral Concepts ($\leq 50\%$ frequency):** Representing the remaining 30.8%, these concepts reflect optional refinements, alternative phrasings, or finer-grained distinctions that are selected less consistently.

Semantic versus Structural Stability The stability metrics (bottom left) reveal a clear distinction between semantic content and structural organization. The Concept Stability score reaches 0.889, indicating that the model reliably identifies the same core concepts across repeated executions.

In contrast, the topological metrics, namely Edge Stability and Parent–Child Stability, are both lower at 0.625. This suggests that while the set of concepts is largely invariant, their hierarchical arrangement is more sensitive to stochastic variation. In a recursive generation process, small differences in early branching decisions can propagate through subsequent expansions, resulting in alternative yet semantically coherent tree structures.

Output Scale Consistency The run-to-run variability of tree size (bottom right) demonstrates strong consistency in output scale. The number of extracted concepts is tightly clustered around a mode of 9, with a low standard deviation of $\sigma = 0.55$. Although occasional outliers occur, with trees containing up to 11 or 14 nodes, the generation process converges to the same scale in the vast majority of runs.

Overall, the self-consistency analysis shows that the AutoCoC pipeline achieves a favorable balance between robustness and flexibility:

- **High Semantic Reproducibility:** The core concept set is highly stable, with a concept stability score of 0.889 and 69.2% of concepts appearing in more than half of the runs.
- **Moderate Structural Variability:** Hierarchical relations exhibit controlled variation (0.625 stability), reflecting the existence of multiple valid conceptual organizations.
- **Scale Stability:** The overall size of the generated trees remains consistent across runs, with minimal variance ($\sigma = 0.55$).

5.3. Downstream Ontology Generation Results on 10 Domains

Figure 5.4 reports a domain-wise comparison of three knowledge-injection methods, AutoCoC, the handcrafted CoC baseline, and reverse-engineered CoC, on downstream OWL ontology generation across ten domains. We evaluate each method along three complementary dimensions: (i) semantic correctness, measured by the OntoClean constraint compliance rate; (ii) reliability of meta-property assignment, measured by the average confidence score; and (iii) structural richness, measured by the average number of OWL classes per generated file.

Across domains, the leading method differs by metric. In terms of **average confidence score**, the handcrafted CoC baseline achieves the best result in **five** domains, AutoCoC in **three** domains, and reverse engineering in **two** domains. For **OntoClean constraint compliance**, AutoCoC leads in **four** domains, while the handcrafted baseline and reverse engineering each lead in **three** domains. Finally, for **average classes per file**, AutoCoC dominates in **nine** domains, with reverse engineering leading in **one** domain and the handcrafted baseline in **zero** domains. This pattern indicates that AutoCoC most consistently produces richer ontologies, while expert-curated CoCs often provide the clearest signal for confident meta-property labeling.

To complement the per-domain analysis, Figure 5.5 summarizes method-level averages aggregated over all ten domains. AutoCoC attains the highest mean confidence score (0.893), indicating the most reliable meta-property assignments on average, whereas reverse engineering yields the lowest mean confidence (0.831). A similar trend holds for semantic correctness: AutoCoC achieves the highest mean OntoClean compliance rate (90.59%), compared to 88.62% for reverse engineering.

Structural richness exhibits the strongest separation between methods (Figure 5.6). AutoCoC generates substantially larger ontologies, with an average of 21.7 classes per file, compared to 13.5 for the handcrafted baseline and 14.8 for reverse engineering. This suggests that AutoCoC extracts a broader and more fine-grained concept inventory from documentation, resulting in ontologies with higher representational capacity. The increased ontology size also manifests in computational cost: as expected, AutoCoC requires longer evaluation time, exceeding the runtime of the other two methods by more than 40% on average.

Overall, the results suggest that AutoCoC provides the strongest balance between semantic correctness and structural coverage in downstream ontology generation. While

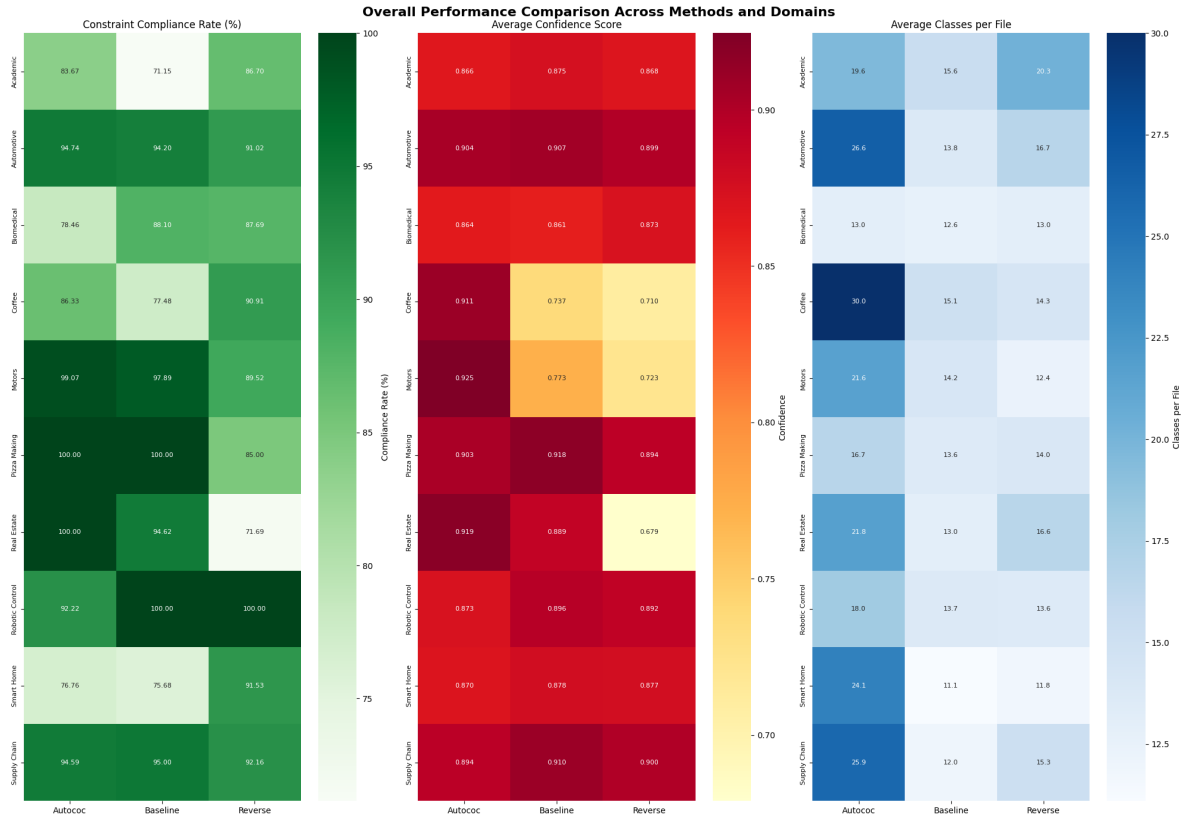

Figure 5.4.: Overall performance comparison across methods and domains. Left: OntoClean constraint compliance rate. Middle: average confidence score of meta-property assignments. Right: average number of OWL classes per generated ontology file.

handcrafted CoCs remain competitive, particularly in achieving the highest confidence in several domains, AutoCoC most consistently produces ontologies that are both structurally richer and more compliant with OntoClean inheritance constraints.

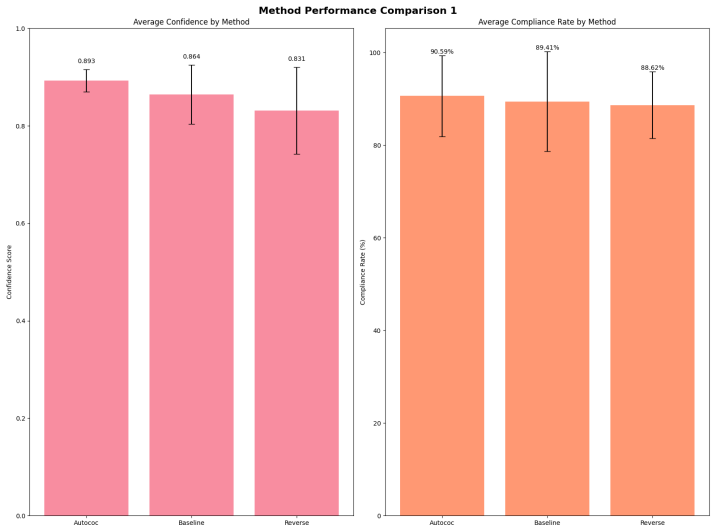

Figure 5.5.: Average performance per method: mean confidence score and mean OntoClean compliance rate across ten domains.

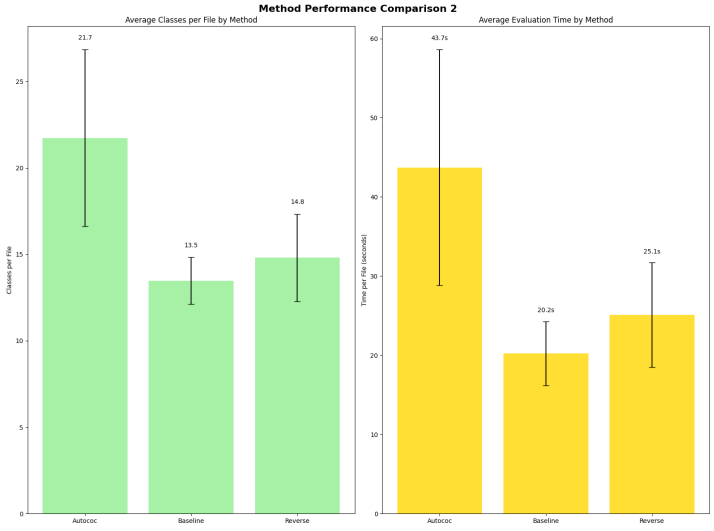

Figure 5.6.: Average structural and runtime performance per method: mean classes per file and mean evaluation time across ten domains.

6. Conclusion & future work

6.1. Conclusion

This thesis investigated whether concept-based prompting, previously limited by manual design effort and subjectivity, can be automated without sacrificing semantic quality or downstream effectiveness. The central finding is that structured concept hierarchies can indeed be constructed automatically from documentation and used to reliably guide large language models in domain-specific tasks.

We introduced AutoCoC, an automated pipeline that transforms unstructured documentation into concept trees and structured prompt sequences. Empirical evaluation across ten domains shows that AutoCoC produces concept structures that are semantically stable at the concept level and effective as guidance mechanisms for structured generation. When applied to ontology generation in OWL/Turtle, AutoCoC not only matches but often exceeds expert-crafted baselines in semantic correctness and constraint compliance, while producing richer and more expressive outputs. These results demonstrate that the benefits of concept-based prompting are not inherently tied to manual curation.

Beyond performance, the key contribution of this work lies in reframing concept structuring as an algorithmic problem rather than a human bottleneck. By removing the dependency on expert-authored concept hierarchies, AutoCoC enables scalable, reproducible, and domain-adaptive concept-based prompting. This substantially lowers the barrier to applying structured reasoning techniques in real-world settings where manual ontology or concept design is impractical.

More broadly, the findings suggest that explicit intermediate concept representations can serve as an effective interface between unstructured documentation and large language models. Rather than relying solely on implicit knowledge stored in model parameters, AutoCoC demonstrates the value of external, dynamically constructed conceptual structure for improving reliability and semantic control in downstream generation tasks. This positions automated concept extraction as a promising direction for future research

in knowledge injection, structured prompting, and hybrid symbolic–neural systems.

6.2. Future Work

The limitations identified above suggest several promising directions for extending and improving the AutoCoC methodology:

- Improve relation extraction and edge stability:

  The current dual-scoring mechanism achieves high concept-level stability but exhibits lower structural stability (0.625). Future work should explore enhanced relation extraction techniques, such as constraint-guided reparenting that incorporates semantic consistency checks, multi-pass refinement strategies, or ensemble-based edge scoring that aggregates multiple LLM judgments to reduce stochastic variation in hierarchical attachment decisions.

- Explore graph structures and multiple trees per domain:

  The single-tree constraint simplifies construction and traversal but limits representational flexibility. Future research should investigate more expressive knowledge structures, including directed acyclic graphs (DAGs) that allow multiple inheritance paths, or multi-tree frameworks where different conceptual hierarchies capture complementary perspectives on the same domain. This would enable richer knowledge representation for heterogeneous domains with multiple overlapping conceptual organizations.

- Broaden evaluation to diverse downstream tasks:

  While ontology generation effectively demonstrates the value of concept-grounded prompting for structured artifact creation, the generalizability of AutoCoC to other task types remains unexplored. Future work should evaluate the approach on diverse downstream applications, such as code generation, API documentation synthesis, technical specification writing, and domain-specific question answering, to establish broader applicability and identify task-specific optimization strategies.

- Comparative evaluation with diverse knowledge injection methods:

  The current evaluation focuses primarily on prompt-based methods (AutoCoC, handcrafted CoC, reverse-engineered CoC). Future research should conduct comprehensive comparisons across all four knowledge injection paradigms, such as

dynamic injection (RAG), static embedding (fine-tuning), modular adapters, and prompt optimization, in order to systematically characterize the trade-offs in accuracy, computational cost, scalability, and domain adaptability across different resource constraints and application scenarios.

Acknowledgements

This thesis marks not only the completion of my master's degree, but also the end of an important chapter of my life. It would not have been possible without the support, guidance, and love of many people who stood by me throughout this journey.

First and foremost, I dedicate this work to my grandmother, Erdentsetseg. She was the kindest and strongest woman I have ever known, and she shaped who I am in ways I am still discovering. During my studies, she would always ask me when I was coming home, and I always promised her that I would visit soon, with the last promise being after finishing my thesis. I did not make it in time. I am deeply sorry for not spending enough time with you. Your strength, warmth, and unconditional love will always remain with me, and this work carries your memory.

I am profoundly grateful to my parents. Your unwavering belief in me, your love, and your constant support have always been the source of my courage to face life's challenges. No matter how uncertain the path was, knowing that you stood behind me gave me the strength to move forward. This achievement belongs to you as much as it does to me.

I would like to sincerely thank Professor Runkler for his guidance and inspiration since the beginning of my master's studies at TUM and throughout this thesis. His mentorship taught me to look beyond details and see the bigger picture, to focus on what truly matters, and to reflect on the meaning and purpose of research. I am equally grateful to my advisor, Nishtha, for her patience, constant guidance, and trust. She played a key role in helping me transition from an academic perspective to an industrial research mindset, generously sharing her experience across both technical and professional dimensions. Through her mentorship, I gained valuable insight into how technology is applied at scale in industry and how it actively shapes the world. Her support in addressing every challenge I encountered during this thesis, both big and small, was invaluable and deeply appreciated.

I would like to thank my friends, those who love me deeply, and those who were once part of my life. You gave me courage, companionship, and moments of happiness when life was difficult. Your presence made these three years a period not only of academic

growth, but also of personal development, resilience, and love. For that, I am truly grateful.

I would also like to express my sincere gratitude to the colleagues at the Siemens Data and AI Research department. Throughout my thesis, they consistently introduced me to inspiring ideas, state-of-the-art technologies, and practical perspectives at the forefront of industrial AI research. Their willingness to share knowledge and provide guidance whenever I faced difficulties played a crucial role in my progress. Beyond technical support, they demonstrated what it means to work as a professional scientist and engineer in a collaborative environment, setting a standard for teamwork, rigor, and responsibility that will strongly influence my future career.

Bibliography

[1] N. N. Vaidya, T. Runkler, T. Hubauer, V. Haderlein-Hoegberg, and M. M. Brandt, "Conceptual in-context learning and chain of concepts: Solving complex conceptual problems using large language models." [Online]. Available: http://arxiv.org/abs/2412.15309

[2] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, "A comprehensive overview of large language models." [Online]. Available: http://arxiv.org/abs/2307.06435

[3] 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, and W. Fedus, "Emergent abilities of large language models." [Online]. Available: http://arxiv.org/abs/2206.07682

[4] 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, and D. Amodei, "Language models are few-shot learners." [Online]. Available: http://arxiv.org/abs/2005.14165

[5] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models." [Online]. Available: http://arxiv.org/abs/2201.11903

[6] Z. Liu, C. Gan, J. Wang, Y. Zhang, Z. Bo, M. Sun, H. Chen, and W. Zhang, "OntoTune: Ontology-driven self-training for aligning large language models." [Online]. Available: http://arxiv.org/abs/2502.05478

[7] S. Sivarajkumar, M. Kelley, A. Samolyk-Mazzanti, S. Visweswaran, and Y. Wang, "An empirical evaluation of prompting strategies for large language

models in zero-shot clinical natural language processing." [Online]. Available: http://arxiv.org/abs/2309.08008

[8] Z. Hong, H. Wang, Z. Zada, H. Gazula, D. Turner, B. Aubrey, L. Niekerken, W. Doyle, S. Devore, P. Dugan, D. Friedman, O. Devinsky, A. Flinker, U. Hasson, S. A. Nastase, and A. Goldstein, "Scale matters: Large language models with billions (rather than millions) of parameters better match neural representations of natural language." [Online]. Available: http://biorxiv.org/lookup/doi/10.1101/2024.06.12.598513

[9] M. Alviano, L. Grillo, F. Lo Scudo, and 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, pp. 4330–4338. [Online]. Available: https://www.ijcai.org/proceedings/2025/482

[10] W. Hariri, "Unlocking the potential of ChatGPT: A comprehensive exploration of its applications, advantages, limitations, and future directions in natural language processing." [Online]. Available: http://arxiv.org/abs/2304.02017

[11] T. Kuculo, S. Abdollahi, and S. Gottschalk, "Transformer-based architectures versus large language models in semantic event extraction: Evaluating strengths and limitations," vol. 16, no. 5, p. 22104968251363759. [Online]. Available: https://journals.sagepub.com/doi/10.1177/22104968251363759

[12] 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, and N. Zhang, "Knowledge mechanisms in large language models: A survey and perspective," version Number: 4. [Online]. Available: https://arxiv.org/abs/2407.15017

[13] A. Schwarzschild, Z. Feng, P. Maini, Z. C. Lipton, and J. Z. Kolter, "Rethinking LLM memorization through the lens of adversarial compression." [Online]. Available: http://arxiv.org/abs/2404.15146

[14] W. Gurnee, N. Nanda, M. Pauly, K. Harvey, D. Troitskii, and D. Bertsimas, "Finding neurons in a haystack: Case studies with sparse probing." [Online]. Available: http://arxiv.org/abs/2305.01610

[15] Y. Hou, J. Li, Y. Fei, A. Stolfo, W. Zhou, G. Zeng, A. Bosselut, and M. Sachan, "Towards a mechanistic interpretation of multi-step reasoning capabilities of language models," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pp. 4902–4919. [Online]. Available: https://aclanthology.org/2023.emnlp-main.299

[16] M. A. Runco and G. J. Jaeger, "The standard definition of creativity," vol. 24, no. 1, pp. 92–96. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/10400419.2012.650092

[17] A. Roberts, C. Raffel, and N. Shazeer, "How much knowledge can you pack into the parameters of a language model?" [Online]. Available: http://arxiv.org/abs/2002.08910

[18] P. J. Phillips, C. A. Hahn, P. C. Fontana, D. A. Broniatowski, and M. A. Przybocki, "Four principles of explainable artificial intelligence." [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ir/2020/NIST.IR.8312-draft.pdf

[19] T. Räuker, A. Ho, S. Casper, and D. Hadfield-Menell, "Toward transparent AI: A survey on interpreting the inner structures of deep neural networks," in *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*. IEEE, pp. 464–483. [Online]. Available: https://ieeexplore.ieee.org/document/10136140/

[20] M. Nye, A. J. Andreassen, G. Gur-Ari, H. Michalewski, J. Austin, D. Bieber, D. Dohan, A. Lewkowycz, M. Bosma, D. Luan, C. Sutton, and A. Odena, "Show your work: Scratchpads for intermediate computation with language models." [Online]. Available: http://arxiv.org/abs/2112.00114

[21] Z. Song, B. Yan, Y. Liu, M. Fang, M. Li, R. Yan, and X. Chen, "Injecting domain-specific knowledge into large language models: A comprehensive survey." [Online]. Available: http://arxiv.org/abs/2502.10708

[22] G. Nayak, S. Dutta, D. Ajwani, P. Nicholson, and A. Sala, "Automated assessment of knowledge hierarchy evolution: comparing directed acyclic graphs," vol. 22, no. 3, pp. 256–284. [Online]. Available: https://link.springer.com/10.1007/s10791-018-9345-y

[23] T. Aggarwal, A. Salatino, F. Osborne, and E. Motta, "Large language models for scholarly ontology generation: An extensive analysis in the engineering field." [Online]. Available: http://arxiv.org/abs/2412.08258

[24] Y. Zhao, N. Vetter, and K. Aryan, "Using large language models for OntoClean-based ontology refinement." [Online]. Available: http://arxiv.org/abs/2403.15864

A. Appendix

A.1. 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.

A.1.1. 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.1.2. 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.1.3. 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.1.4. 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.1.5. 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.1.6. 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.1.7. 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.1.8. 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.1.9. 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.1.10. 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.2. Ontology Evaluation Complete Results

Table A.1.: 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**

Table A.2.: 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 A.3.: 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**