OG-NSD: Neuro-Symbolic Ontology Drafting from Natural-Language Requirements

Efstratios Skaperdas[1,*], Kristi Cami[1] and Nick Bassiliades[1]

[1]Department of Informatics, Aristotle University of Thessaloniki, Greece

Abstract

Natural-language requirements are easy for analysts and domain stakeholders to write, but difficult to validate systematically and to transform into machine-interpretable semantic artifacts. We present OG-NSD, a neuro-symbolic pipeline for ontology drafting from natural-language requirements. The pipeline combines LLM-based drafting with ontology-aware prompting, SHACL validation, OWL/DL reasoning, competency-question evaluation, and an iterative repair loop that converts symbolic violations into corrective prompts. We evaluate the approach through a controlled pilot study spanning neural baselines, a validation-oriented baseline, ontology-aware drafting, and iterative repair policies, with auxiliary observations on portability and iteration-level CQ behavior. The results show a deliberately mixed but informative picture. Low-temperature drafting provides a strong raw baseline ($F1 = 0.4769$, $11/21$ competency questions), ontology-aware prompting improves vocabulary alignment but not necessarily precision, and iterative repair helps only when conservative stopping policies are used. Among the tested repair strategies, the `max_only` configuration yields the best balance within the iterative family ($F1 = 0.3427$, $11/21$ competency questions), whereas aggressive CQ-driven repair leads to uncontrolled schema growth, repeated undeclared class insertions, and unstable reasoning. These findings support the main claim of the paper: symbolic feedback is most useful when treated as an active control mechanism for ontology drafting rather than as a purely post-hoc diagnostic layer. All experiments are configuration-driven and the corresponding artifacts are publicly released to support reproducibility.

Keywords

LLMs, ontology engineering, requirements formalization, neuro-symbolic AI, SHACL, OWL

1. Introduction

Requirements are commonly written in natural language because this format is accessible to analysts, developers, and domain stakeholders. Yet natural-language requirements are often ambiguous, underspecified, and difficult to validate systematically. This limits downstream automation, including traceability, consistency checking, reuse, and formal analysis. In ontology engineering, the problem is particularly important: requirement documents may implicitly define classes, relations, hierarchies, and domain constraints, yet transforming them into formal semantic artifacts remains labor-intensive and expert-driven [1, 2].

Large language models (LLMs) have recently shown strong capabilities for extracting structure from free text and producing draft knowledge representations. This raises a natural question: can LLMs accelerate ontology drafting directly from requirement descriptions? The answer is promising, but not straightforward. In practice, LLM outputs may omit constraints, invent classes or properties, drift away from domain vocabulary, or produce ontologies that are syntactically plausible but semantically weak. For ontology engineering, such defects are critical because quality is not measured only by fluent output, but by structural validity, logical consistency, and the ability to support intended queries [8, 11, 9].

In this paper, we present OG-NSD (Ontology-Guided Neuro-Symbolic Drafting), a pipeline for transforming natural-language requirements into draft OWL ontologies through a closed neuro-symbolic loop. The key idea is simple: an LLM produces a candidate ontology, symbolic validators detect structural and logical problems, and these signals are transformed into targeted corrective prompts that guide

First Workshop on LLM-driven Knowledge Graph and Ontology Engineering (LLMs4KGOE 2026), co-located with the 23rd European Semantic Web Conference (ESWC 2026), May 11, 2026, Dubrovnik, Croatia

*Corresponding author.

eskapera@csd.auth.gr (E. Skaperdas); kcamia@csd.auth.gr (K. Cami); nbassili@csd.auth.gr (N. Bassiliades)

0009-0004-6423-0240 (E. Skaperdas); 0000-0001-6035-1038 (N. Bassiliades)

the next repair step. Rather than treating validation as a final filtering stage, OG-NSD incorporates validation into generation itself.

The contribution of this work is fourfold:

- We define a modular neuro-symbolic pipeline for LLM-driven ontology drafting from requirements.
- We operationalize SHACL violations, reasoning outcomes, and competency-question failures as structured feedback for iterative repair.
- We present a controlled experimental comparison that separates the effects of raw drafting, validation-oriented control, ontology-aware prompting, and iterative repair policies, while also reporting auxiliary portability and diagnostic observations.
- We provide a realistic pilot evaluation over ATM and healthcare requirements, showing both the strengths and the limits of the approach.

The paper therefore makes a deliberately measured claim rather than assuming that the full neuro-symbolic loop uniformly dominates all baselines. We do not claim that the full neuro-symbolic loop uniformly dominates every baseline under every metric. Instead, the experiments show a more nuanced picture: low-temperature drafting is already strong, ontology-aware prompting improves alignment but can still introduce structural noise, and repair helps only when its stopping policy is well controlled. That empirical profile is precisely what makes the problem scientifically interesting.

2. Related Work

The proposed work lies at the intersection of ontology engineering, natural-language requirements formalization, symbolic validation, and LLM-assisted structured generation.

A first line of work concerns the role of ontologies as formal, reusable representations of domain knowledge. Foundational work by Gruber framed ontologies as explicit specifications of conceptualizations and emphasized portability and reuse [1]. Uschold and Grüninger further systematized ontology development by discussing principles, methods, and application contexts [2]. In ontology engineering practice, competency questions (CQs) became a standard way to delimit ontology scope and evaluate adequacy with respect to intended tasks [3, 4].

A second line of work concerns the formal languages and validation mechanisms used to specify and check ontologies. OWL 2 provides the standard semantic framework for ontology representation on the Semantic Web [6, 7]. SHACL provides a graph-based validation language for checking RDF data against structural and typing constraints [5]. These technologies offer strong guarantees at the symbolic level, but they are usually applied after ontology construction rather than integrated into generation loops.

A third line of work explores LLMs for ontology engineering and ontology learning. Recent studies report that LLMs can assist with relation extraction, glossary induction, ontology extension, schema suggestion, and direct ontology generation from textual descriptions [8, 9, 10, 11]. At the same time, these works also highlight recurring weaknesses: hallucinated schema elements, inconsistent use of vocabulary, weak traceability to source text, and difficulty maintaining structural discipline across domains.

Finally, recent neuro-symbolic perspectives argue that symbolic constraints should not be treated merely as post-hoc diagnostics, but as mechanisms that actively shape neural generation and reasoning [12]. OG-NSD aligns with this direction, but focuses specifically on ontology drafting from requirement sets under a controlled validation-and-repair workflow. Existing LLM-for-ontology-engineering studies mainly emphasize generation assistance, prompting strategies, or broad task landscapes. In contrast, OG-NSD operationally integrates drafting, symbolic validation, competency-question testing, and iterative repair under explicitly logged configurations. The main novelty is therefore not a new ontology-learning algorithm, but a controlled neuro-symbolic workflow and an evaluation framing that exposes when repair helps and when it harms.

More closely related to our setting are recent LLM-based approaches that incorporate constraints, validation signals, or iterative refinement during ontology or schema construction. Compared to these directions, OG-NSD differs in two key aspects. First, it treats SHACL validation, DL reasoning, and competency-question evaluation as first-class, explicitly logged control signals within a unified loop. Second, it exposes repair behavior through configuration-driven experiments that isolate stopping policies and their impact on structural stability. This shifts the focus from improving generation quality alone to understanding and controlling the interaction between drafting, validation, and repair.

3. OG-NSD Method

3.1. Problem Setting

Given a set of natural-language requirements $R = \{r_1, \ldots, r_n\}$ and optional domain assets A (e.g., base ontology, SHACL shapes, competency questions, glossary terms, and exemplar mappings), the goal is to generate a draft ontology O in OWL/Turtle that captures the key classes, properties, hierarchies, and constraints needed to support downstream validation and querying.

We assume a practical setting in which exact gold formalizations may be incomplete or domain-dependent. Therefore, across the full experimental suite, ontology quality is not reduced to exact overlap with a reference ontology, but is assessed more broadly through structural validation, reasoning-based checks, and competency-question performance where applicable.

3.2. Inputs and Assets

The implementation relies on a clearly defined set of inputs and supporting assets. Requirement files are stored in JSONL or JSON form, with each requirement processed as a standalone natural-language statement. The pipeline also accepts base ontologies, SHACL shapes, competency-question files, configuration files, and optional gold ontologies for evaluation. This design supports reproducibility and allows experiments to be replayed without modifications to the core code.

The main asset categories are:

- **Requirements:** JSONL/JSON files containing requirement statements, such as `atm_requirements.jsonl` and `health_requirements.jsonl`.
- **Domain ontology:** optional base ontology used for grounding and graph merging.
- **SHACL shapes:** domain-specific structural constraints encoded in Turtle.
- **Competency Questions:** SPARQL ASK queries used to assess conceptual adequacy.
- **Ontology context and exemplars:** optional vocabulary, prefixes, and small Turtle exemplars injected into prompts during ontology-aware drafting.
- **Configuration files:** experiment-specific parameter snapshots controlling LLM mode, reasoning, repair iterations, and paths.
- **Outputs and reports:** TTL ontologies, JSON metrics, and intermediate artifacts stored under `build/` or `runs/`.

Figure 1 summarizes the overall OG-NSD workflow and the role of the main assets used throughout the pipeline. The figure makes explicit that ontology drafting is not treated as a one-shot generation task: requirements, domain vocabulary, SHACL shapes, and competency questions jointly support a closed validation-and-repair process.

As Figure 1 suggests, the method is organized around a separation of concerns: drafting expands coverage, validation enforces discipline, and repair controls the interaction between the two. This decomposition matters analytically because it lets us study where errors originate and which part of the pipeline is actually responsible for improvements.

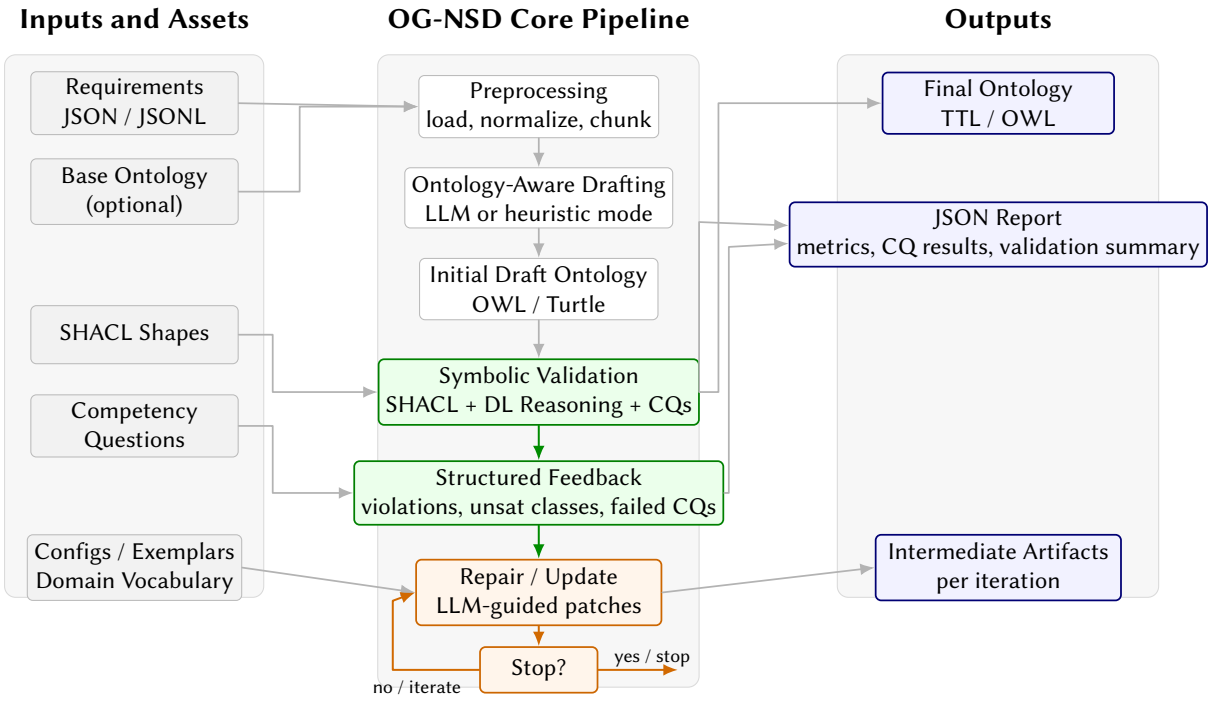

Figure 1: Overview of the OG-NSD pipeline. Natural-language requirements and domain assets are transformed into a draft ontology through ontology-aware drafting. The candidate ontology is then symbolically validated using SHACL, DL reasoning, and competency questions, and iteratively refined through structured feedback until the stopping criteria are satisfied.

3.3. Preprocessing

Preprocessing in OG-NSD is intentionally lightweight. The system performs syntactic loading, whitespace normalization, filtering of empty or malformed entries, and optional chunking of requirement sets. The design avoids aggressive linguistic transformations such as stemming or lemmatization, because the original requirement formulations are treated as the semantic ground truth.

Requirement loading is implemented so that the rest of the pipeline remains format-agnostic. The requirement loader supports both JSONL, where each line is an independent requirement, and JSON lists. In both cases the output is normalized into a uniform list of strings. Large requirement sets can be chunked into batches in order to respect LLM context limits and to keep execution cost under control. This chunking step functions as an operational split mechanism rather than as a train/validation/test split. It allows the same requirement subset to be used across multiple runs, thus improving reproducibility and comparability.

The same stage also supports the injection of ontology vocabulary and exemplars into prompts. These exemplars are small Turtle fragments that illustrate how requirement phrases should be translated into ontology constructs. They function as a style guide for namespace usage, subclass modeling, and property declarations.

3.4. Configuration, Model Settings, and Reproducibility

The behavior of the pipeline is controlled through a unified configuration layer. A configuration object specifies paths to requirements, gold ontologies, SHACL shapes, competency questions, and optional base ontologies, as well as operational parameters such as LLM mode, maximum number of repair iterations, maximum number of requirements to process, whether reasoning is enabled, and whether the run should stop after drafting.

This design supports two desirable properties: reproducibility, because each experiment corresponds to a fixed configuration snapshot, and comparability, because differences between runs can be attributed

to explicit parameter changes rather than to hidden code modifications.

The implementation consists of modules for requirement loading and chunking, LLM drafting, ontology parsing and serialization, SHACL validation, DL reasoning, CQ execution, and JSON reporting. The drafting stage supports both a live LLM backend and a heuristic fallback mode. In the reported experiments, the neural baseline explicitly studies two temperatures ($T = 0.1$ and $T = 0.7$), allowing us to quantify the effect of generation stochasticity. Ontology-aware runs use domain vocabulary and exemplar-guided prompting. Full repair experiments are executed under fixed stopping policies, with maximum iteration budgets defined in the corresponding configuration files. All runs produce serialized ontologies together with JSON summaries, which makes the evolution from initial draft to final output auditable and directly reproducible.

The public implementation and experimental artifacts are available in the project repository at https://github.com/KristiCami/Ontology-Guided. The released experiment scripts support both OpenAI-backed execution and a deterministic heuristic fallback. In the public code, the effective backend, temperature, ontology-context usage, reasoning flag, and iteration budget are selected through configuration files, while the generated run artifacts record the conditions of each execution. We therefore interpret the reported runs through their configuration snapshots and saved outputs rather than assuming a single fixed backend across all experiments. In the OpenAI-backed runs, ontology drafting and repair were performed using the GPT-4o-mini model, with temperature controlled per experiment (e.g., $T = 0.1$ and $T = 0.7$ in E1). The exact backend identifiers and per-run settings are recorded in the released configuration snapshots and artifacts.

3.5. LLM Drafting

The drafting stage is the neural core of OG-NSD. Its input is a set of normalized requirements, and its output is an initial draft ontology in Turtle. The system supports two execution modes: an OpenAI-backed mode and a heuristic fallback mode. In the public implementation, the effective LLM mode and temperature are selected through configuration, while the heuristic mode remains useful for offline runs and deterministic reproduction.

Ontology-aware prompting is a key feature of the system. When enabled, the pipeline extracts vocabulary from a gold or base ontology—classes, properties, domains, ranges, and prefixes—and injects this information into the prompt as structured context. This does not amount to copying ontology axioms; rather, it acts as a lexical and structural contract that reduces drift and promotes consistent namespace usage. The resulting draft is serialized and stored as a temporary artifact so that its contribution can be studied independently before symbolic correction is applied.

3.6. Validation and Repair Loop

After drafting, the candidate ontology is validated at multiple levels.

SHACL validation. SHACL captures structural expectations such as missing properties, invalid typing, incomplete declarations, and domain-specific modeling rules. The resulting violations are fine-grained and interpretable, making them suitable as repair signals [5].

DL reasoning. Reasoning checks whether the ontology remains logically coherent under OWL semantics. It can reveal inconsistencies and unsatisfiable classes that are not visible through purely local shape checks [6, 7].

Competency-question testing. Competency questions are implemented as SPARQL ASK queries. They provide an independent signal of domain adequacy by checking whether the ontology supports intended domain questions [3, 4].

The outputs of these validators are then transformed into repair-oriented feedback. Instead of exposing raw tool logs to the model, OG-NSD produces compact structured prompts that specify the

problem, its local context, and the expected correction. Let $O^{(t)}$ denote the ontology at iteration t. The repair step is written as:

$$O^{(t+1)} = \text{RepairLLM}(O^{(t)}, V^{(t)}_{\text{shacl}}, V^{(t)}_{\text{reason}}, C^{(t)}), \tag{1}$$

where $V^{(t)}_{\text{shacl}}$ and $V^{(t)}_{\text{reason}}$ denote validation signals and $C^{(t)}$ denotes optional competency-question feedback.

Each repair iteration includes four substeps: validation, violation aggregation, repair-prompt synthesis, and patch integration into the current graph. The process is therefore closer to counterexample-guided repair than to naive prompt repetition. The entire iteration history is stored in JSON form, making it possible to inspect what changed and why.

All symbolic validators in OG-NSD are fully automated. SHACL validation, DL reasoning, and competency-question execution are performed programmatically within the pipeline; no human validator is involved in the reported experiments. Human input is limited to the preparation of domain assets (e.g., SHACL shapes and competency questions), not to the validation or repair process itself.

In the current implementation, competency questions are provided as SPARQL ASK queries. As a result, CQ authoring remains an external preparation step rather than an automatically generated component of the pipeline. OG-NSD automates the use of CQs in evaluation and repair, but not their initial construction.

From validation diagnostics to repair prompts. A key design choice in OG-NSD is that raw validation outputs are not directly passed to the LLM. Instead, validator signals are transformed into compact structured diagnostic records and then into repair prompts.

Each diagnostic record aggregates information from the validation stage and contains the following fields:

- **Validator source:** SHACL, DL reasoning, or CQ evaluation.
- **Violation type:** the category of the issue (e.g., missing declaration, invalid restriction, failed constraint).
- **Affected entity or axiom:** the class, property, or triple involved.
- **Local context:** relevant triples or modeling fragments surrounding the issue.
- **Expected repair action:** a concise description of the intended correction.

Rather than forwarding verbose tool logs, the system synthesizes short corrective instructions that are easier for the LLM to interpret and apply consistently.

Example. The following example illustrates the transition from an aggregated diagnostic record to the resulting repair prompt.

Aggregated diagnostic

```
source: SHACL
focus_node: :Withdrawal
issue: missing property constraint satisfaction
message: instances of :Withdrawal must be linked to exactly one
:BankAccount
local_triples: :Withdrawal rdf:type owl:Class .
suggested_action: add or correct property modeling involving
:hasAccount
```

Repair prompt
You are repairing an OWL/Turtle ontology. Problem: the current ontology violates a structural constraint related to `:Withdrawal` and account linkage. Required action: revise the ontology so that `:Withdrawal` is properly linked to `:BankAccount` through the intended property. Preserve existing prefixes and avoid introducing unrelated classes or properties. Return only Turtle triples to add or modify.

In practice, multiple validator outputs may be merged into a single aggregated diagnostic before prompt synthesis, depending on the selected repair policy. This aggregation balances specificity with stability: overly fine-grained prompts may lead to fragmented local fixes, while overly coarse prompts may lack actionable guidance. The current implementation therefore uses compact, targeted prompts derived from structured diagnostics in order to encourage localized corrections while preserving global ontology structure.

Not all violations are guaranteed to be repairable within the current loop. In practice, persistent failures may arise from malformed ontology fragments, repeated invalid restrictions, missing domain knowledge, or conflicting repair suggestions produced across iterations. When no valid patch is produced, when the same failure pattern persists, or when the maximum iteration budget is reached, the run terminates and the remaining unresolved issues are explicitly recorded in the final artifacts.

3.7. Termination, Serialization, and Reporting

The repair loop terminates according to the selected stopping policy, using criteria such as the absence of hard SHACL violations, CQ stabilization, unchanged or unavailable patches, or the maximum number of iterations. These criteria balance quality improvement against runtime and LLM cost.

The final ontology is serialized to Turtle via an ontology abstraction layer built on top of RDFLib. The serialized graph may combine the initial draft, an optional base ontology, and the patches introduced during repair. Intermediate graphs and JSON reports are preserved so that the evolution from draft to final ontology remains fully traceable. As a result, the output of a run is not a single ontology file, but a pair consisting of the final TTL graph and an analytical report containing SHACL summaries, reasoner results, CQ traces, and repair metadata.

3.8. Robustness and Auditability

A further implementation detail that is important for interpreting the experiments is that OG-NSD is designed to fail transparently rather than silently. Requirement files are normalized before drafting, Turtle outputs are parsed before downstream validation, and malformed or invalid restrictions are explicitly recorded in the run artifacts rather than being hidden inside later stages. Likewise, intermediate outputs are preserved per iteration so that one can distinguish defects introduced during initial drafting from defects created or amplified during repair.

This matters experimentally for two reasons. First, it improves auditability: when a repair policy behaves poorly, the analyst can inspect the exact draft, validation output, patch set, and post-reasoning graph associated with that run. Second, it improves comparability: because execution is configuration-driven and artifact-producing, differences between methods can be tied to concrete choices such as stopping policy, threshold setting, or prompting mode. In practice, this logging discipline is especially useful in failure cases such as graph explosion, repeated invalid restrictions, or reasoner instability, where a pure black-box evaluation would reveal that something went wrong but not where it went wrong.

3.9. Evaluation Methodology

The evaluation of generated ontologies in OG-NSD is explicitly multi-layered. It combines four complementary perspectives.

Table 1
Structural summary of the two domain ontologies and evaluation assets.

Domain	Reqs.	Classes	Obj. props	Data props	Subclass axioms	Domain/range	CQs
ATM	30	12	12	6	15	36	21
Healthcare	30	18	19	25	9	44	8

First, **exact** matching compares predicted axioms against the reference ontology at strict IRI level after duplicate removal and namespace normalization. Second, we report a **relaxed reasoning-aware overlap** metric. This metric compares a normalized schema-triple representation of the predicted ontology against the corresponding normalized representation of the reference ontology. When reasoning is enabled, the predicted graph is first expanded through reasoning; otherwise, the asserted graph is used. The normalized schema space includes schema-level triples involving classes, properties, subclass relations, and selected domain/range-style constraints after duplicate removal, namespace normalization, and canonical handling of selected ontology constructs. The metric should therefore be interpreted as a schema-oriented approximate semantic overlap measure rather than as full logical equivalence. Third, structural and logical quality are assessed through SHACL validation and DL reasoning outcomes. Fourth, competency questions assess whether the ontology captures the domain knowledge needed to support intended queries.

This layered design is necessary because no single metric captures ontology quality in full. An ontology may be syntactically close to the gold reference while remaining logically weak, or it may be logically valid but still fail to support the intended domain questions.

4. Experimental Setup

4.1. Domains, Assets, and Research Questions

The experiments use two domains: ATM and healthcare. Each domain provides requirement files, SHACL shapes, competency questions, and, when available, gold ontologies and base ontologies.

Table 1 summarizes the two domains through a compact structural profile of the corresponding reference ontologies together with the main evaluation assets. The goal is to make the experimental setting more concrete by showing not only the number of requirements and competency questions, but also the approximate ontology construction burden in each domain.

The reported structural counts were extracted automatically from the reference Turtle ontologies at schema level. The Domain/range column reports the total number of `rdfs:domain` and `rdfs:range` axioms.

As Table 1 shows, the two domains share the same evaluation logic but differ in ontology structure and CQ coverage. This makes them suitable for testing the method under a shared pipeline while still exposing differences in schema complexity and domain-specific modeling requirements.

The evaluation is organized around the following research questions:

1. Does validation-oriented control improve the quality of LLM-produced ontology drafts over raw generation?
2. Does ontology-aware prompting improve vocabulary alignment and structural quality relative to less grounded baselines?
3. Under which stopping policies does iterative repair help or hurt ontology quality?

We study four main configurations:

- **E1: Neural baseline.** Pure drafting without SHACL validation, reasoning, or iterative repair.

Table 2
E1 neural baseline results.

Run	Precision	Recall	F1	Pred. triples	Overlap	CQ pass
E1 ($T = 0.1$)	0.4295	0.5360	0.4769	156	67	11/21
E1 ($T = 0.7$)	0.0895	0.1360	0.1079	190	17	2/21

Table 3
Additional E1 characteristics.

Run	Tokens	Reasoning	Notes
E1 ($T = 0.1$)	1559	disabled	conservative generation
E1 ($T = 0.7$)	1416	disabled	more triples, lower quality

- **E2: Validation-oriented baseline.** A baseline run under the standard pipeline configuration, with SHACL validation and optional reasoning enabled.
- **E3: Ontology-aware drafting.** A dedicated non-swept ontology-aware run used to study schema-grounded drafting behavior.
- **E4: Full OG-NSD.** Ontology-aware drafting with SHACL, optional reasoning, and iterative repair under multiple stopping policies.

In addition, we use two auxiliary analyses in a supporting role: a brief portability check across domains and a CQ-oriented instrumentation view over iterative execution. These are included as diagnostic evidence rather than as central experimental claims.

The distinction among E1–E4 is important for interpreting the results. E1 is raw LLM drafting only. E2 enables validation-oriented control within the standard pipeline configuration. E3 isolates ontology-aware prompting without a repair-policy sweep. E4 is the only setting in which stopping policies are explicitly varied and analyzed as part of the full drafting–validation–repair loop.

4.2. Metrics

The evaluation combines exact overlap, relaxed reasoning-aware overlap, SHACL compliance, consistency and reasoner behavior, CQ pass rate, number of predicted triples, overlap count, removed invalid restrictions, missing classes, and graph growth across repair iterations.

5. Results

5.1. E1: Neural Baseline

E1 evaluates the system under pure drafting conditions, without SHACL, reasoning, or repair. Two temperature settings are compared.

Tables 2 and 3 summarize the main quantitative outcomes of the neural baseline under two temperature settings. Together, they show that increasing generation diversity does not improve ontology quality in this setting, but instead leads to weaker overlap and lower competency-question coverage.

All reported results correspond to single configuration runs recorded through the released pipeline artifacts. We chose this pilot setup to prioritize breadth of controlled pipeline comparison and artifact release over repeated-measures statistical analysis. The reported results should therefore be interpreted as configuration-level case studies rather than as variance-estimated performance claims. Because the pipeline is configuration-driven and fully logged, the exact conditions of each experiment can be reproduced from the stored run metadata.

The low-temperature run provides a strong baseline, with $F1 = 0.4769$ and $11/21$ passed competency questions. In contrast, the higher-temperature run produces more triples but sharply degrades both

Table 4
Comparison of E2 and E3.

Run	Precision	Recall	F1	Pred. triples	Overlap	CQ pass
E2 (relaxed)	0.2388	0.5520	0.3333	289	69	12/21
E2 (exact)	0.2749	0.5520	0.3670	251	69	12/21
E3	0.1575	0.5520	0.2451	438	69	12/21

Table 5
E4 repair-policy comparison for the full OG-NSD pipeline.

Policy	Precision	Recall	F1	Pred. triples	CQ pass	Stop / notes
Default	0.2134	0.5360	0.3052	314	11/21	no_hard_violations
Max_only	0.2519	0.5360	0.3427	266	11/21	max_iterations
Hard_and_cq	0.0253	0.1760	0.0442	871	4/21	Pellet runtime failure
Ignore_no_hard	0.0181	0.3440	0.0343	2380	16/21	graph explosion

relaxed overlap and CQ performance. The result is a clear instance of semantic drift: more generation does not translate into better ontology quality. Table 3 reinforces the same interpretation from a different angle: the higher-temperature run is not failing because it is too short or too small, but because the additional generation is less disciplined.

5.2. E2 and E3: Validation-Oriented Baseline vs. Ontology-Aware Drafting

E2 functions as a validation-oriented baseline under the standard pipeline configuration, whereas E3 highlights ontology-aware drafting in a non-swept setting. E2 achieves better balance between exactness and functional stability; E3 increases graph size and retains the same CQ pass rate, but at lower precision.

Table 4 compares the validation-oriented setup with ontology-aware drafting. The comparison is useful because it contrasts the validation-enabled baseline with a dedicated ontology-aware drafting setting, making it easier to interpret the effect of the dedicated schema-grounded configuration relative to the broader baseline setup.

These results suggest that ontology-aware prompting improves lexical alignment and coverage, but not necessarily overall precision. Without repair, the extra expressivity of E3 turns into structural noise rather than consistent gains. At the same time, E3 is diagnostically useful because it isolates the effect of grounding from the effect of iterative correction. Put differently, Table 4 shows that more ontology-like output is not automatically better output: controlled structure still matters.

5.3. E4: Full OG-NSD with Repair Policies

E4 evaluates the full neuro-symbolic loop under four repair policies: `default`, `max_only`, `hard_and_cq`, and `ignore_no_hard`. The results reveal that iterative repair is beneficial only when guided by conservative stopping logic.

Table 5 reports the main comparison among the four repair policies considered in the full OG-NSD setting. The goal of this comparison is not only to identify the strongest policy, but also to show how different stopping logics affect the balance between graph size, overlap quality, and competency-question performance.

Figure 2 visualizes the contrast between overlap-oriented quality and competency-question success across repair policies. It highlights the central trade-off of the paper: aggressive CQ-driven repair can raise functional coverage, but often at the cost of precision and structural stability.

The `max_only` policy yields the best balance among the iterative configurations, improving F1 relative to the default policy while keeping the graph compact after reasoning. In contrast, aggressive

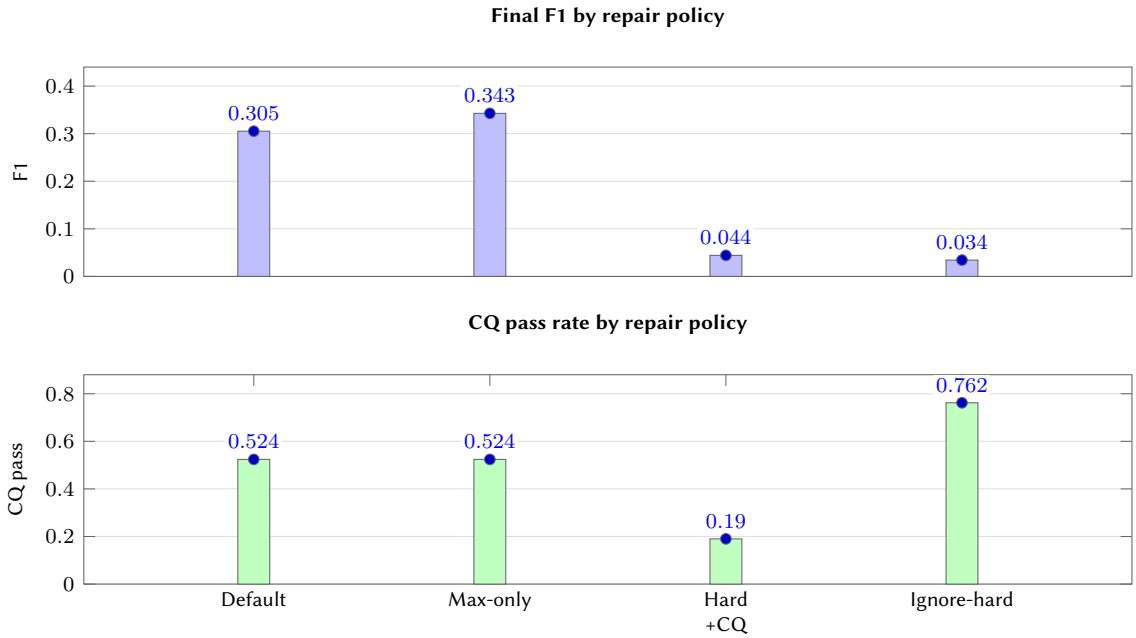

Figure 2: Comparison of E4 repair policies. Conservative policies preserve a better balance between ontology quality and functional adequacy, whereas aggressive CQ-driven repair increases CQ pass rate at the cost of precision and overall graph quality.

Table 6
Iteration-level evidence for the `max_only` policy. Hard/soft SHACL violations remain at zero, while the asserted graph grows across iterations but the reasoned graph remains stable.

Iter.	CQ pass	SHACL (hard/soft)	Asserted triples	Reasoned triples	Missing class decls.
0	11/21	0/0	219	266	37
1	11/21	0/0	624	266	88
2	11/21	0/0	1022	266	162
3	11/21	0/0	1378	266	304

CQ-driven repair destabilizes the ontology: `hard_and_cq` produces reasoner failures, including a runtime failure in the Pellet reasoner (NullPointerException), while `ignore_no_hard` achieves the highest CQ pass rate but at the cost of severe graph explosion and near-collapse of precision. In the reported failing run, the Pellet NPE occurred in the presence of malformed or unstable ontology constructs generated during aggressive repair. We therefore treat it as evidence of repair-induced ontology instability at the tooling boundary, rather than as a validated claim about a specific reasoner bug.

Table 6 provides iteration-level evidence for the `max_only` policy, which is the most balanced iterative configuration in our study. The table is useful because it shows that repeated repair rounds increase the pre-reasoning graph substantially, while the post-reasoning graph remains stable.

Table 7 provides illustrative rather than exhaustive cost estimates derived from pipeline execution traces. The token values in Table 3 correspond to recorded run-level metadata, whereas Table 7 reports reconstructed approximate full-run totals for comparative cost analysis. LLM call counts are reconstructed from the batching and repair-loop control flow, where 30 requirements processed in batches of five yield six drafting calls, and each repair iteration introduces an additional patch call. Token usage is approximated from per-call metadata and scaled to the full run.

Because cumulative token usage is not fully logged, the reported values should be interpreted as approximate and comparative. The table therefore highlights relative differences between draft-only, validation-enabled, ontology-aware, and iterative-repair configurations, rather than exact API-level

Table 7
Illustrative operational cost across the main experiment configurations. Values are reconstructed from pipeline control flow and run artifacts, and should be interpreted as approximate and comparative rather than exact measurements.

Run	LLM calls	Tokens (approx.)	Repair evals (patch calls)	Notes
E1 ($T = 0.1$)	6 (draft)	∼9.4k	0 (0)	draft-only baseline
E2	6 (draft)	∼10.2k	0 (0)	validation enabled
E3	6 (draft)	∼17.5k	0 (0)	ontology-aware prompt
E4 (default)	7 (6 + 1 patch)	∼12k–15k	2 (1)	stop: no hard violations
E4 (max_only)	9 (6 + 3 patch)	∼18k–24k	4 (3)	bounded repair loop
E4 (hard_and_cq)	8 (6 + 2 patch)	∼15k–21k	3 (2)	unstable repair
E4 (ignore_no_hard)	8 (6 + 2 patch)	∼15k–22k	3 (2)	graph growth

accounting.

In Table 6, *Asserted triples* refers to the pre-reasoning graph maintained by the pipeline, while *Reasoned triples* denotes the post-reasoning graph used for evaluation. *Missing class decls.* counts referenced classes that required explicit declaration.

These values correspond to different stages of the pipeline rather than to removal of triples by the reasoner. Reporting both views allows us to separate raw graph growth from the stabilized post-reasoning structure. In this run, the pattern indicates increasing asserted-graph complexity without corresponding gains in post-reasoning size or CQ coverage.

A closer look at the iterative traces further clarifies the behavior. Under the default policy, the system converges to a structurally stable graph without improving the CQ pass rate. Under `max_only`, additional iterations improve F1 while the reasoner keeps the post-reasoning graph at a stable size. The two aggressive policies show the opposite behavior: more patches and more triples, but weaker quality. Table 6 therefore supports an important interpretation of the method: the reasoner is not merely a checker, but also an indirect stabilizer of graph growth.

5.4. Overall Findings

Taken together, the experiments support four main findings. First, low-temperature drafting is already a strong baseline. Second, ontology-aware prompting improves lexical alignment but not necessarily precision unless supported by controlled repair. Third, iterative repair can be beneficial, but only under conservative stopping policies. Fourth, functionality-oriented signals such as competency-question success do not by themselves guarantee structural ontology quality.

6. Discussion

The main value of OG-NSD is methodological. It reframes ontology drafting as a controlled neuro-symbolic process rather than a one-shot generation problem. This is important for LLM-driven ontology engineering because quality cannot be reduced to fluent output alone; it requires explicit structural, logical, and task-oriented criteria.

The experiments also clarify the limits of the approach. Symbolic validation is helpful, but it is not a universal optimizer. Its benefit depends on how validator feedback is translated into prompts and how termination is controlled. Similarly, competency questions are valuable as adequacy signals, but when treated as the sole optimization target they can encourage overgeneration.

One plausible explanation for the negative effect of some repair policies is that highly localized repair prompts may overconstrain the model and encourage patchwise fixes that satisfy individual diagnostics while damaging global ontology coherence. This suggests that future work should compare fine-grained repair prompts against more global "problem-list" repair formulations, in which the model is given a compact set of issues to resolve more holistically rather than a sequence of narrowly scoped local fixes.

A closer inspection of the failed or degraded repair runs reveals that these effects are not arbitrary, but correspond to a small set of recurring error patterns. Across runs, we repeatedly observed four concrete failure modes:

- **Repeated insertion of undeclared classes:** the repair process introduces new class IRIs without consistent declaration, leading to accumulation of missing class declarations across iterations.
- **Malformed or semantically weak restrictions:** repair prompts generate restrictions that are syntactically valid but poorly grounded (e.g., incomplete domain/range usage or incoherent class expressions), which degrade precision after normalization.
- **Redundant schema expansion:** iterative patches repeatedly add structurally similar triples (e.g., duplicate subclass or domain assertions), causing rapid growth of the asserted graph without corresponding gains in reasoning-level structure.
- **CQ-oriented overgeneration:** repair steps that prioritize competency-question satisfaction introduce additional triples that satisfy ASK queries, but do so by overgenerating loosely constrained schema elements rather than improving alignment with the reference ontology.

These patterns are also visible quantitatively in the iteration-level traces. For example, under the `max_only` policy (Table 6), the number of missing class declarations increases from 37 to 304 across iterations, despite stable CQ performance. This indicates that repair introduces additional schema elements faster than it resolves structural gaps. Similar effects are observed in the aggressive policies, where graph size increases substantially without improving overlap-based precision.

Taken together, these observations explain why repair does not uniformly improve F1: the dominant failure mode is not the inability to fix individual violations, but the uncontrolled accumulation of partially valid schema additions that degrade global ontology quality.

One notable finding is that the best iterative configuration (`max_only`) still does not surpass the strongest raw neural baseline (E1 at low temperature) in F1. This does not invalidate the neuro-symbolic framing; rather, it shows that the current repair policies privilege structural stabilization over pure overlap maximization. For that reason, the current paper should be read as a controlled exploratory benchmark of repair policies rather than as a claim of universal superiority for the full loop.

The strongest raw baseline remains competitive in overlap-based F1, which shows that high-quality initial drafting is already a strong starting point. The value of OG-NSD is therefore not universal score improvement, but a more controlled and auditable workflow that exposes structural defects, logical failures, and competency-question behavior during ontology construction. In this sense, the contribution of the method is methodological rather than purely performance-driven: it enables practitioners to observe, diagnose, and regulate ontology generation rather than relying on a single-shot output.

From a systems perspective, the pipeline incurs additional cost relative to one-shot drafting because each repair round introduces validation, prompt synthesis, and an additional LLM call. In the current pilot study, we report token counts for representative runs and treat runtime and cost primarily as engineering trade-offs rather than as the main optimization target. This reflects the intended use of OG-NSD as a controlled drafting workflow, where interpretability and auditability may justify additional computational overhead in comparison to purely generative baselines.

A brief auxiliary portability check suggests that the OG-NSD architecture is reusable across domains by swapping requirements, shapes, ontologies, and competency questions, but that quality does not transfer automatically. In particular, the weaker healthcare-side behavior indicates that cross-domain robustness remains limited in the current pilot setting. We therefore interpret these supporting runs as evidence of architectural portability rather than of strong domain generalization.

We also used a CQ-oriented instrumentation view as a supporting diagnostic tool to inspect per-iteration functional behavior. In that role, it helped clarify that competency-question gains do not necessarily coincide with better structural ontology quality.

A further implication concerns evaluation design. Structural validity, logical consistency, and competency-question coverage align only partially. This motivates the multi-view evaluation in OG-NSD, as single metrics would obscure key trade-offs between compactness, correctness, and functional adequacy. In other words, the goal of OG-NSD is not to outperform the neural baseline on overlap metrics alone, but to provide a controlled and auditable ontology drafting workflow that exposes structural and reasoning failures during generation.

A natural next step is a human-in-the-loop variant in which an ontology engineer reviews aggregated diagnostics or approves repair patches before integration. Such a setup could help distinguish between locally valid fixes and globally appropriate ontology revisions, especially in cases where automatic repair introduces structurally plausible but semantically weak additions.

7. Threats to Validity

Several threats to validity should be acknowledged. **Internal validity** may be affected by prompt sensitivity, implementation choices in the repair loop, and the specific configuration of the LLM backend. **Construct validity** depends on whether SHACL violations, reasoner outcomes, and CQ success adequately capture ontology quality. **External validity** is limited by the number of domains and by the reliance on ATM and healthcare assets. **Conclusion validity** is further limited by the fact that the current study reports single configuration runs rather than repeated trials with seed variation or uncertainty estimates; the findings should therefore be read as pilot configuration-level evidence rather than as variance-controlled performance claims.

In addition, some comparisons involve exact matching and relaxed reasoning-aware matching under different assumptions. These measures are complementary but not interchangeable, and their interpretation must remain tied to the intended evaluation perspective.

8. Conclusion

We presented OG-NSD, a neuro-symbolic pipeline for drafting OWL ontologies from natural-language requirements. The pipeline combines LLM drafting with ontology-aware prompting, SHACL validation, DL reasoning, competency-question checking, and iterative repair. The experimental study shows that symbolic feedback is useful when it acts as a controlled guidance mechanism, but that overly aggressive repair policies can damage precision, destabilize reasoning, and inflate graph size.

The most balanced iterative policy in our study is `max_only`, while low-temperature neural drafting remains a strong baseline in its own right. More broadly, the results suggest that the central challenge in LLM-driven ontology engineering is not only how to generate ontologies, but how to regulate the interaction between generation, validation, and stopping. OG-NSD therefore contributes a concrete and auditable step toward trustworthy ontology drafting workflows.

Artifact Availability

The implementation, configuration files, and experimental artifacts are publicly available at https://github.com/KristiCami/Ontology-Guided. The repository contains the OG-NSD codebase, runnable scripts, experiment configurations, domain assets, and recorded outputs used to support the claims of the paper.

Declaration on Generative AI

During the preparation of this work, the authors used Generative AI tools to support language editing, proofreading, and minor text refinement. The authors reviewed and edited all AI-assisted output and take full responsibility for the content of the publication.

References

[1] T. R. Gruber. A translation approach to portable ontology specifications. Knowledge Acquisition, 5(2):199–220, 1993.

[2] M. Uschold and M. Grüninger. Ontologies: Principles, methods and applications. The Knowledge Engineering Review, 11(2):93–136, 1996.

[3] M. Grüninger and M. S. Fox. The role of competency questions in enterprise engineering. In Benchmarking — Theory and Practice, pages 22–31. Springer, 1995.

[4] G. K. Q. Monfardini, J. P. A. Almeida, and G. Guizzardi. Use of competency questions in ontology engineering: A survey. Applied Ontology, 18(1):1–38, 2023.

[5] H. Knublauch and D. Kontokostas, editors. Shapes Constraint Language (SHACL). W3C Recommendation, 2017.

[6] W3C OWL Working Group. OWL 2 Web Ontology Language: Document Overview (Second Edition). W3C Recommendation, 2012.

[7] B. Motik, P. F. Patel-Schneider, and B. Parsia, editors. OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax (Second Edition). W3C Recommendation, 2012.

[8] J. García Fernández. Ontology Engineering with Large Language Models. Master's thesis, Delft University of Technology / TNO, 2024.

[9] M. J. Saeedizade, A. S. Lippolis, M. Poveda-Villalón, and O. Corcho. Navigating ontology development with large language models. In Proceedings of ESWC 2024 Workshops, 2024.

[10] A. S. Lippolis, M. J. Saeedizade, M. Poveda-Villalón, and O. Corcho. Ontology generation with metacognitive prompting in large language models. In Proceedings of ESWC 2024 Workshops, 2024.

[11] D. Garijo, M. Poveda-Villalón, E. Amador-Domínguez, Z. Wang, R. García-Castro, and O. Corcho. LLMs for ontology engineering: A landscape of tasks and benchmarking challenges. In Proceedings of the 1st International Workshop on LLMs for Ontology Engineering, 2025.

[12] L. N. DeLong, R. Fernández Mir, and J. D. Fleuriot. Neurosymbolic AI for reasoning over knowledge graphs: A survey. arXiv preprint arXiv:2302.07200, 2024.