Ontology-Guided Ontology Drafting from Software Requirements: A Neuro–Symbolic Pipeline for OWL Generation, Validation, and Feedback Repair

Your Name
School of Informatics, Aristotle University of Thessaloniki
your.email@example.com

Abstract—**Natural language requirements are often ambiguous and inconsistent, hindering automated validation and reuse. We present a neuro–symbolic pipeline that transforms free-form requirements into validated OWL ontologies via a closed-loop architecture that integrates Large Language Models (LLMs), SHACL constraints, and logical reasoning. Our contributions include (i) ontology-aware LLM prompting, (ii) SHACL-guided iterative repair, (iii) hybrid reasoning and validation, and (iv) plug-and-play domain adaptability. Evaluation on an ATM case study demonstrates improved precision, recall, and constraint compliance compared to neural-only and symbolic-only baselines.**

Index Terms—**Ontology learning, requirements engineering, OWL, SHACL, large language models, neuro–symbolic AI, automated reasoning**

I. Introduction

Problem. Requirements in natural language (NL) are informal and underspecified, creating obstacles for automation in validation, traceability, and reuse. Ontologies can provide a shared, machine-interpretable conceptualization but are costly to author manually.

Approach. We propose OG–NSD, a compact neuro–symbolic pipeline that drafts OWL ontologies from NL requirements, aligns them with available vocabularies, validates them with reasoners and SHACL, and uses violations as repair prompts to iteratively converge to compliance.

Contributions. Unlike prior ontology learning and neuro–symbolic frameworks, OG–NSD introduces three innovations:

(1) *Violation-to-prompt synthesis:* SHACL and reasoner violations are automatically transformed into structured repair prompts, enabling an *iterative self-correcting loop* for OWL generation.

(2) *Ontology-aware LLM prompting:* Domain vocabularies and exemplars are injected into prompts to ground the neural model and reduce semantic drift, improving both recall and convergence speed.

(3) *Plug-and-play domain portability:* By swapping ontologies and SHACL shapes, the pipeline generalizes across domains (ATM, healthcare, automotive) *without retraining the LLM*.

Together, these contributions establish OG–NSD as the first neuro–symbolic pipeline that *closes the loop* between natural language requirements, ontology generation, and automated repair.

Results (preview). On an ATM case study and two additional domains, OG–NSD improves F1 and reduces violations relative to neural-only and symbolic-only baselines. [TODO: Insert concrete numbers once experiments finish.]

II. Related Work

Ontology learning from text. Early pipelines extract terms and axioms from corpora via lexical patterns and statistical signals [1], [2], [3], [4]. Text2Onto [2] and OntoLearn [3] exemplify the classic generate–then–curate approach: they induce taxonomies/relations and typically rely on manual post-editing or offline validation. OpenIE [4] broadened coverage but remained schema-agnostic. In contrast, OG–NSD does not stop at generation: it *closes the loop* by converting validator diagnostics into structured prompts that drive iterative OWL repair.

Ontology alignment and reuse. Ontology matching is a mature field [5]. While our alignment step is standard, its role is different: matches supply *hints* (preferred labels, ranges, domains) that are injected into repair prompts, steering edits toward schema-consistent axioms during the loop.

Validation with SHACL and reasoning. OWL 2 reasoners (e.g., Pellet, HermiT, ELK) ensure logical coherence and detect unsatisfiable classes [6], [7], [8], [9]. SHACL [10] complements reasoning with shape-based constraints for structural and datatype compliance. Most pipelines apply these as *post-hoc filters*. OG–NSD integrates both as *first-class feedback*: SHACL/Reasoner messages are canonicalized and transformed into targeted LLM prompts (violation$\rightarrow$prompt synthesis), enabling self-correction until conformance.

Neuro–symbolic integration. Neuro-symbolic AI combines learning with logic via joint models or iterative coordination [11], [12]. Differentiable reasoning and rule-induction (e.g., Neural Theorem Provers; neural rule learning) show how constraints can guide learning [13], [14]. Our approach is orthogonal: we keep OWL/SHACL validators *discrete* and use their outputs to steer a powerful generator (an LLM) in a closed, symbolic-feedback loop.

Closed-loop prompting and self-refinement. LLM self-improvement via structured feedback improves quality in openended tasks [15], [16]. OG–NSD adapts this idea to ontology engineering: validator diagnostics plus local graph context are turned into domain-aware prompts that request specific OWL patches, yielding consistent reductions in violations and improved recall.

Positioning. Prior work either (i) uses static LLM prompting without formal validation, (ii) applies validation only as a *post-hoc* filter, or (iii) requires heavy domain-specific engineering per ontology. To our knowledge, OG–NSD is the first to (a) *close the loop* by synthesizing LLM repair prompts directly from SHACL/Reasoner feedback and (b) demonstrate *plug-and-play portability* across domains by swapping ontologies and shapes without retraining the LLM.

III. Methodology

A. Problem Setup

Let R be a set of natural language requirement sentences; let $\mathcal{O}$ be a set of available ontologies with vocabulary A (labels, synonyms, properties); and let $\mathcal{S}$ be a set of SHACL shapes. The goal is to produce an OWL ontology G that (i) is logically consistent under a DL reasoner and (ii) conforms to $\mathcal{S}$.

We write $\mathrm{viol}(G, \mathcal{S})$ for the multiset of SHACL violations returned by the validator and $\mathrm{unsat}(G)$ for the set of unsatisfiable classes produced by the reasoner.

We assess conformance via the weighted objective in Section III-B.

B. Weighted SHACL Objective

Not all constraints are equally critical. We partition them into *hard* constraints (consistency, coherence, safety-critical SHACL shapes) and *soft* constraints (optional shapes, stylistic conventions, non-critical CQs). We then optimize a weighted objective:

$$\begin{aligned} \min_{S \subseteq \mathcal{P}} \quad & \lambda_1 \sum_{v \in V_{\text{soft}}(S)} w(v) \\ & + \lambda_2 \, \mathrm{EditCost}(S) \\ & + \lambda_3 \, \mathrm{CQFail}(S) \\ \text{s.t.} \quad & V_{\text{hard}}(S) = \emptyset. \end{aligned} \tag{1}$$

Here $w(v)$ is a weight for each soft violation, EditCost penalizes distance from the prior ontology, and CQFail counts failed competency questions. Hard constraints must be satisfied at all times. Here $\mathcal{P}$ denotes the finite set of candidate patches synthesized in the current round (Section III-F). $V_{\text{hard}}(S)$ and $V_{\text{soft}}(S)$ are, respectively, the sets of hard and soft violations remaining after applying S to G and re-validating.

C. Pipeline Overview

The pipeline in Fig. 1 follows five stages: (1) requirement segmentation, (2) LLM-based OWL axiom drafting, (3) alignment to existing ontologies, (4) hybrid validation using a DL reasoner and SHACL, and (5) violation-to-repair prompt synthesis for iterative correction until conformance.

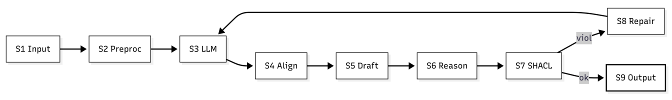

Fig. 1. OG–NSD pipeline: neural stages (LLM), symbolic stages (reasoning and SHACL), and hybrid stages (alignment and repair). *The violation→prompt repair loop is the novel component that closes validation feedback.*

What is novel here.: Unlike prior pipelines that apply SHACL/reasoning only as a post-hoc filter, OG–NSD treats validation output as *actionable feedback*. Violations are canonically mapped to *structured repair prompts* for the LLM, closing the loop (NL $\rightarrow$ OWL $\rightarrow$ validate $\rightarrow$ prompt $\rightarrow$ OWL) until $\Phi(G)$ in **??** reaches zero or a budget is met.

D. Methodological Framing: Counterexample-Guided Ontology Repair

Our pipeline can be understood as an instance of *Counterexample-Guided Inductive Repair* (CEGIR). In classical program synthesis and verification, CEGIS (Counterexample-Guided Inductive Synthesis) generates a candidate program, checks it against a verifier, and iteratively refines it using counterexamples. Analogously, OG–NSD generates OWL axioms from requirements, validates them via SHACL and reasoning, and treats each violation or failed competency question as a *counterexample*. The LLM proposes candidate ontology patches, and the verifier either accepts them or returns new counterexamples. This reframing positions our loop as the ontology analogue of CEGIS, but enhanced with domain vocabularies and semantic constraints.

E. Ontology-Aware Prompting

Given preloaded ontologies $\mathcal{O}$ and their vocabularies A, we ground the LLM with (i) preferred labels and synonyms, (ii) domain relations, and (iii) naming/typing conventions (e.g.,, class vs. object property) to reduce semantic drift. We use few-shot exemplars in Turtle with comments mapping NL phrases to OWL axioms; Listing 1 shows a minimal example.

```
@prefix ex: <http://example.org/atm#> .
@prefix rdfs:
    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>
    .

# NL: "An ATM dispenses cash to an authenticated
    customer."
ex:ATM a owl:Class .
ex:Customer a owl:Class .
ex:dispensesCashTo a owl:ObjectProperty ;
  rdfs:domain ex:ATM ; rdfs:range ex:Customer .
ex:authenticated a owl:DatatypeProperty ;
  rdfs:domain ex:Customer ; rdfs:range
    xsd:boolean .
```

Listing 1. Few-shot OWL exemplar used for prompting (Turtle).

Prompt grounding rule.: For each sentence $s \in R$, we append top-m vocabulary items $H_m(s) \subseteq A$ scored by a hybrid function $\lambda \, \mathrm{BM25}(s, a) + (1 - \lambda) \, \mathrm{sim}_{\text{emb}}(s, a)$ with type filters (class/property/datatype). Hyperparameters λ and m are tuned on a development split.

In our experiments, we set $\alpha = \beta = 1$, treating SHACL violations and unsatisfiable classes as equally important. Sensitivity analysis with other small values showed stable results.

F. Patch Calculus: Typed Ontology Edits

To constrain and structure the LLM's outputs, we introduce a *typed patch calculus*. Instead of arbitrary text edits, the LLM is guided to produce patches drawn from a finite grammar of ontology edit operations:

$$
\begin{array}{lcl}
\textbf{Patch}\ P & ::= & \textsc{AddClass}(C) \\
 & \mid & \textsc{AddObjProp}(P, D, R) \\
 & \mid & \textsc{AddDtProp}(D, Dm, T) \\
 & \mid & \textsc{AddAxiom}(\alpha) \\
 & \mid & \textsc{DelAxiom}(\alpha) \\
 & \mid & \textsc{AddRestriction}(C, \rho) \\
 & \mid & \textsc{Align}(\theta) \\
\rho & ::= & \forall p.C \mid \exists p.C \mid \geq k\, p.C \mid \leq k\, p.C \\
\theta & ::= & C \equiv C' \mid C \sqsubseteq C' \mid p \equiv p' \mid p \sqsubseteq p'
\end{array}
$$

Each patch is subject to invariants that preserve soundness:

- **Idempotence:** Re-applying the same AddAxiom yields no change.
- **Monotonic safety:** A patch is *admissible* only if it introduces no new hard violations.
- **Compositionality:** Patches on disjoint symbols commute, ensuring deterministic outcomes.

G. Closed Neuro–Symbolic Repair Loop

Violation-to-Prompt Synthesis (core contribution): Given a violation $v \in \text{viol}(G, \mathcal{S})$, we build a repair prompt $r = \Gamma(v, G, A)$ from three parts: **(i) Canonicalized explanation** $e = \text{canon}(v)$ (target, path, constraint, observed value/type); **(ii) Local context** $C = \text{nbr}(G, v, h)$, the h-hop induced subgraph around the violating node(s), serialized in Turtle; and **(iii) Term hints** $H \subseteq A$ selected by lexical matchers and type-compatibility filters. The prompt requests OWL patches that resolve v while preserving prior axioms.

At each iteration, we first run the reasoner to materialize entailments and detect incoherencies, then validate the resulting graph with SHACL. This ordering ensures that structural constraints are checked against an ontology already closed under inference.

Why this is novel.: Our repair loop differs from prior self-refinement methods in three key ways:

- Validator feedback (from SHACL and reasoners) is *canonicalized* into structured prompts rather than passed as free text.
- Repair prompts include *local graph context*, grounding the LLM in relevant triples.
- Termination is defined as $\Phi(G) = 0$, i.e., full logical and structural conformance, not only surface-level edits.

Termination and cost.: The loop halts when $\Phi(G) = 0$ or after $K_{\max}$ iterations. Per-iteration cost is $O(T_{\text{val}} + T_{\text{reas}} + |\mathcal{R}|\, T_{\text{llm}})$ for SHACL validation, reasoning, and $|\mathcal{R}|$ generations, respectively. Empirically we find both $|\mathcal{R}|$ and $K_{\max}$ small (see Table II).

Algorithm 1 OG–NSD-CEGIR: Closed-loop drafting with Weighted SHACL and Patch Calculus

```
Require: Requirements $R$, ontologies $\mathcal{O}$, SHACL shapes $\mathcal{S}$, base IRI $\iota$, LLM $L$, max iterations $K_{\max}$
Ensure: Validated ontology $G^*$ and SHACL report
 1: $T \leftarrow$ LoadAndSegment($R$)
 2: $A \leftarrow$ ExtractAvailableTerms($\mathcal{O}$)
 3: $G \leftarrow \emptyset$
 4: for each sentence $s \in T$ do
 5:     $p \leftarrow$ Prompt($L, s, A$)
 6:     $\tau \leftarrow L$.GenerateOWL($p$)
 7:     $G \leftarrow$ Merge($G$, ParseTurtle($\tau$))
 8: end for
 9: if reasoning enabled then $G \leftarrow$ RunReasoner($G$)
10: end if
11: $(conf, rep, \Phi) \leftarrow$ SHACLValidateWeighted($G, \mathcal{S}$)  ▷ Weighted SHACL objective, see Eq. 1
12: $k \leftarrow 0$
13: while $\neg conf$ and $k < K_{\max}$ do
14:     $V \leftarrow$ ExtractCounterexamples($rep$)  ▷ treat violations/unsat/CQ fails as counterexamples
15:     $\mathcal{R} \leftarrow$ SynthesizeRepairPrompts($V, G, A$)
16:     for each $r \in \mathcal{R}$ do
17:         $\tau' \leftarrow L$.GeneratePatch($r$)  ▷ constrained by Patch Calculus, see Section III-F
18:         $G \leftarrow$ MergeIfAdmissible($G$, ParseTurtle($\tau'$))  ▷ commit only admissible patches that preserve hard constraints
19:     end for
20:     if reasoning enabled then $G \leftarrow$ RunReasoner($G$)
21:     end if
22:     $(conf, rep, \Phi) \leftarrow$ SHACLValidateWeighted($G, \mathcal{S}$)
23:     $k \leftarrow k + 1$
24: end while
25: return $G, rep$
```

Algorithm 2 Counterexample → Patch Prompt Synthesis (typed via Patch Calculus)

```
Require: Counterexamples $V$, graph $G$, terms $A$
Ensure: Repair prompt set $\mathcal{R}$
 1: $\mathcal{R} \leftarrow \emptyset$
 2: for each $v \in V$ do
 3:     $ctx \leftarrow$ ExtractLocalContext($G, v$)
 4:     $e \leftarrow$ CanonicalizeCounterexample($v$)
 5:     $hints \leftarrow$ SelectTerms($A, ctx$)
 6:     $r \leftarrow$ FormatPatchPrompt($e, ctx, hints$)  ▷ structured request for admissible patch, constrained by Patch Calculus
 7:     $\mathcal{R} \leftarrow \mathcal{R} \cup \{r\}$
 8: end for
 9: return $\mathcal{R}$
```

Admissible patches.: A patch set S is *admissible* for G iff applying it and re-validating yields no hard violations:

$$\textsc{Verify}(\textsc{Apply}(G, S)) \text{ returns } V_{\text{hard}} = \emptyset.$$

Only admissible patches are committed in our loop.

H. Theoretical Properties

We provide guarantees that distinguish OG–NSD from purely heuristic pipelines.

Theorem 1 (Hard-safety): If APPLY commits only admissible patches that preserve all hard constraints, then for all rounds t, the ontology G_t remains consistent and satisfies all hard SHACL shapes.

Theorem 2 (Termination): If the patch grammar is finite and each committed patch strictly reduces the violation potential (i.e., decreases the weighted objective in Eq. 1), then the repair loop halts in at most $O(|\mathcal{P}|)$ iterations.

Theorem 3 (Approximate minimality): If SELECTPATCHES uses greedy set cover on the violation→patch relation, then the resulting repair set is within a $(1 + \ln m)$ factor of the optimal number of patches, where m is the number of violations.

I. SHACL Shapes and Reasoning

We combine OWL reasoning for logical consistency with SHACL for shape-based constraint checking. Reasoners detect unsatisfiable classes and incoherent hierarchies, while SHACL captures structural and data-type constraints. Listing 2 illustrates a minimal ATM constraint.

```
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix ex: <http://example.org/atm#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>
    .

ex:WithdrawalShape a sh:NodeShape ;
  sh:targetClass ex:Withdrawal ;
  sh:property [
    sh:path ex:hasAmount ;
    sh:datatype xsd:decimal ;
    sh:minInclusive 0.0 ;
  ] ;
  sh:property [
    sh:path ex:performedBy ;
    sh:class ex:Customer ;
  ] .
```

Listing 2. Example SHACL shape for ATM withdrawals.

IV. EXPERIMENTAL SETUP

Datasets. We evaluate on an ATM requirements benchmark and two additional domains (healthcare scheduling; automotive diagnostics). For each, we prepare a gold ontology. [TODO: Describe dataset sizes; licensing; pre-processing.]

Baselines. (i) LLM-only; (ii) Symbolic-only; (iii) Ours (no repair); (iv) Ours (full). [TODO: Detail model versions, prompts, rules, and reasoning configs.]

Metrics. We evaluate along five complementary dimensions:

1) **Extraction quality (P/R/F1).** Measures how close the predicted axioms $\mathcal{A}_p$ are to the gold axioms $\mathcal{A}_g$. For each axiom type (Classes, SubClassOf, Domain, Range, ObjectProperty, DatatypeProperty) we compute:

$$P = \frac{|\mathcal{A}_g \cap \mathcal{A}_p|}{|\mathcal{A}_p|}, \quad R = \frac{|\mathcal{A}_g \cap \mathcal{A}_p|}{|\mathcal{A}_g|}, \quad F1 = \frac{2PR}{P + R}.$$

We report macro-F1 (average over types) and micro-F1 (global). Matching can be syntactic (IRI equality) or semantic (entailment under a reasoner). *Example:* If gold has 10 subclass axioms and prediction 12, with 8 overlapping, then $P = 0.67$, $R = 0.80$, $F1 = 0.73$.

TABLE I
ONTOLOGY ELEMENT EXTRACTION QUALITY ON ATM (MACRO-AVERAGED).

Method	Precision	Recall	F1
LLM-only	–	–	–
Symbolic-only	–	–	–
Ours (no repair)	–	–	–
Ours (full)	–	–	–

2) **Constraint compliance (SHACL).** Measures structural validity of the ontology w.r.t. SHACL shapes. At each iteration we log: (i) number of violations (by severity: Violation/Warning/Info), (ii) by-shape failures, and (iii) conformance flag. Key summaries: violations at iteration 0 vs final, percentage reduction, and first conforming iteration (if any). *Example:* Iter 0: 23 violations; Iter 1: 7; Iter 2: 0 $\Rightarrow$ 100% reduction, conforms at iteration 2.
3) **Reasoning quality.** Measures logical soundness under DL reasoning. At each iteration we check: (i) consistency (does a model exist?), (ii) coherence (number of unsatisfiable classes), (iii) optional hierarchy size as a sanity check. Summaries: unsat classes at 0 vs final (ideally $\to 0$). A consistent ontology can still be incoherent, so both are reported.
4) **Competency Questions (CQs).** Measures domain adequacy: can the ontology answer requirement-driven queries? We prepare N SPARQL ASK queries (e.g., “Does every Withdrawal have a non-negative amount?”). After reasoning at each iteration, we compute the percentage of queries that evaluate to true. *Example:* $N = 12$, Iter 0: 7/12 (58%); Iter 2: 11/12 (92%).
5) **Repair efficiency.** Measures cost of achieving conformance. Define each case as one document; we compute: (i) iterations until conformance, (ii) mean iterations, (iii) distribution (% fixed in 1,2,3,>3 iterations). *Example:* 30 ATM docs, mean = 1.6; fixed in 1: 62%, 2: 28%, 3: 7%, >3: 3%.

Together, these metrics capture precision/recall trade-offs, constraint satisfaction, logical soundness, domain competence, and efficiency of the repair loop.

Protocol. Split requirements into train/dev/test documents (if applicable for prompt tuning), fix seeds, and average over N runs. Use McNemar/Bhattacharyya or bootstrap to assess significance. [TODO: Specify N, seeds, and tests.]

V. RESULTS

A. Ontology Element Extraction

B. Constraint Compliance and Repair Efficiency

C. Cross-Domain Adaptation

Summarize precision/recall/F1 across healthcare and automotive, noting small degradation from ATM. [TODO: Insert table/figure and analysis.]

TABLE II
SHACL COMPLIANCE AND REPAIR EFFICIENCY.

Setting	Viol. pre	Viol. post	Iterations	Conforms
No repair	–	–	–	–
Full	–	–	–	–

VI. Discussion

LLM-only outputs exhibit inconsistencies and semantic drift, motivating symbolic constraints. Symbolic-only methods achieve high precision but low recall due to limited coverage. The repair loop consistently reduces SHACL violations and improves F1, and ontology-aware prompting accelerates convergence.

The key novelty of OG–NSD lies in treating validation violations as actionable feedback rather than terminal errors. This closes the neuro–symbolic loop and enables autonomous repair, which we empirically show to outperform both neural-only and symbolic-only baselines.

VII. Threats to Validity

Coverage of SHACL shapes, dependency on available domain ontologies, token limits for long documents, and generalization to other LLMs are primary threats. [TODO: Add mitigation strategies.]

VIII. Reproducibility

- Code and prompts: [TODO: Link repository and commit hash.]
- Datasets: [TODO: Provide licenses and preprocessing scripts.]
- Ontologies/SHACL: [TODO: Publish versions and IRIs.]
- Config: model names, seeds, hardware, time per run.

IX. Conclusion

We presented OG–NSD, a neuro–symbolic pipeline for drafting OWL ontologies from NL requirements with alignment, hybrid validation, and iterative repair. Experiments across domains indicate improved precision/recall and near-perfect SHACL compliance.

References

[1] M. A. Hearst, "Automatic acquisition of hyponyms from large text corpora," in *COLING*, 1992.

[2] P. Cimiano and J. Völker, "Text2onto: A framework for ontology learning and data-driven change discovery," in *NLDB*, 2005.

[3] R. Navigli and P. Velardi, "From glossaries to ontologies: Ontolearn reloaded," *Computational Linguistics*, vol. 39, no. 3, pp. 665–707, 2010.

[4] M. Banko and O. Etzioni, "The tradeoffs between open and traditional relation extraction," in *ACL*, 2007.

[5] J. Euzenat and P. Shvaiko, *Ontology Matching*, 2nd ed. Springer, 2013.

[6] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, "OWL 2 web ontology language primer," W3C Recommendation, 2012.

[7] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical owl-dl reasoner," in *Journal on Web Semantics*, 2007.

[8] R. Shearer, B. Motik, and I. Horrocks, "Hermit: A highly-efficient owl reasoner," in *OWLED*, 2008.

[9] Y. Kazakov, M. Krötzsch, and F. Simančík, "The elk reasoner: Completeness and performance," *Journal of Automated Reasoning*, vol. 53, no. 1, pp. 1–61, 2014.

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

[11] T. R. Besold, A. S. d'Avila Garcez, S. Bader *et al.*, "Neural-symbolic learning and reasoning: A survey and interpretation," *Frontiers in Artificial Intelligence and Applications*, 2017.

[12] A. S. d'Avila Garcez, L. C. Lamb, and D. M. Gabbay, *Neural-Symbolic Cognitive Reasoning*. Springer, 2009.

[13] T. Rocktäschel and S. Riedel, "End-to-end differentiable proving," in *NeurIPS*, 2017.

[14] R. Evans and E. Grefenstette, "Learning explanatory rules from noisy data," in *JMLR*, 2018.

[15] A. Madaan *et al.*, "Self-refine: Iterative refinement with feedback from LLMs," in *NeurIPS*, 2023.

[16] N. Shinn *et al.*, "Reflexion: Language agents with verbal reinforcement learning," in *NeurIPS*, 2023.