PROJECT AUTO-WORLD: TOWARDS AUTOMATED BENCHMARKING OF NEURAL RELATIONAL REASONERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reasoning about relational structures remains a significant challenge for neural models, particularly when they must systematically apply learned knowledge to problem instances that are harder than those seen in training. Progress is hampered by the difficulty of evaluating such generalization, since *a priori*, it is rarely clear what makes an instance hard. We study how this issue can be addressed by using large language models (LLMs) to automate benchmark generation, learning to produce increasingly challenging instances in an end-to-end manner. Concretely, given a world parametrized by Datalog rules, and an Edge Transformer as the reasoning evaluator, we use LLM-driven evolutionary search (based on FunSearch) and autonomous agentic search to discover sampling functions that yield hard problem instances. We also show that the Edge Transformer can be improved using this data such that it generalizes well to further data perturbations. Finally, we show that the same machinery can be applied to novel worlds proposed by LLMs, opening the door to autonomous research on neural relational reasoning.

1 INTRODUCTION

The problem of inferring relationships between entities in a knowledge graph (KG) has received widespread attention (Bordes et al., 2013; Meilicke et al., 2019; Zhu et al., 2021). Despite the vast literature on this topic, however, most approaches are not capable of *systematic* reasoning (Sinha et al., 2019; Das et al., 2025), i.e. they are not capable of applying the learned regularities in a systematic way to solve problems that are harder than those on which they were trained. This presents a stark contrast with classical approaches, such as Inductive Logic Programming (Cropper et al., 2022), where learned rules can be chained to make arbitrarily complex predictions. This observation has sparked a line of work that is focused on systematic neural relational reasoning (Minervini et al., 2020; Bergen et al., 2021; Lu et al., 2022c; Cheng et al., 2023b; Khalid & Schockaert, 2025).

To evaluate the systematic reasoning capabilities of these models, we need to create training/test splits in which the test examples are more difficult than the training instances, *in some sense*. Crucially, however, this requires that we know what makes problem instances hard. Previous work has primarily considered the number of inference steps for this purpose (Sinha et al., 2019). However, the number of inference steps is not the only metric of difficulty that makes sense. For instance, Khalid & Schockaert (2025) focused on problems where the answer can only be derived by combining multiple relational paths, finding that the number of such paths has a strong impact on problem difficulty. As another example, Das et al. (2025) found that existing models struggle when reasoning involves edges that are not on any path between the considered head and tail entities. Accordingly, they suggested using the number of such off-path edges as a difficulty metric. This creates a conundrum: when we are evaluating reasoning models, we do not know *a priori* what makes problem instances hard for these models. We can evaluate them w.r.t. known difficulty metrics, but these metrics may have blind spots, i.e. strong performance on the resulting training/test splits might mask their real limitations. Figure 1 illustrates this with an example that is hard for current models, despite not being identified as such by existing metrics (we will circle back to this example in 8.4 to discuss why we think it is difficult in a way that is not explained by known metrics). Furthermore, generating problem instances which are hard w.r.t. a given difficulty metric can be challenging in itself. The most common strategy is to create problem instances by randomly sampling KGs and then masking some edges, but the

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

conditions that make reasoning hard are often unlikely to arise in these random graphs. While some authors have explored more structured sampling strategies for this reason (Cohen, 2019; Khalid & Schockaert, 2025), such strategies cannot guarantee that problem instances have the "right" structure, given that we do not even know in advance what structures are needed to make reasoning challenging.

Contributions. Inspired by the successes of LLM-based approaches to scientific discovery (Novikov et al., 2025; Romera-Paredes et al., 2024), we ask whether LLMs can automate the evaluation of relational reasoning models. We adapt FunSearch (Romera-Paredes et al., 2024) into an evolutionary loop in which the LLM proposes Python *priority functions* that drive the sampling of candidate problem instances, using the evaluator's failure rate as fitness. This leads to self-improving samplers that generate progressively harder instances. Our analysis shows that the difficulty of the sampled problems cannot be (fully) explained by known difficulty metrics. We also show that surprisingly strong samplers can be obtained by prompting Claude Opus 4.6 in a dialogue setting. Further, we find that augmenting the Edge Transformer's training data with instances generated by the learned samplers yields a stronger evaluator (SUPERET) that largely resists further evolutionary perturbations within the FunSearch loop. However, replacing this loop with an end-to-end `autoresearch` style (Karpathy, 2026) agent leads to samplers that can still challenge the SUPERET.

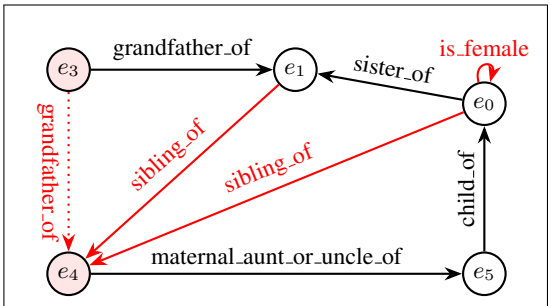


Figure 1: Example of a discovered reasoning instance which challenges existing models. Initial facts are shown as black edges, selected inferred facts as red edges. The task is to infer the relationship between e_3 and e_4.

2 BACKGROUND

Knowledge representation A KG $\mathcal{G}$ is a set of triples $\mathcal{G} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, with $\mathcal{E}$ a set of entities and $\mathcal{R}$ a set of relations. We can think of KGs as labelled graphs; e.g., if $\mathcal{G}$ contains a triple (e, r, f), we say that there is an edge of type r between e and f. At the same time, we can also think of KGs as sets of logical assertions. For example, the triple $(bob, father_of, alice)$ can be viewed as the assertion that *bob* is the father of *alice*. Abusing notation, we will sometimes view KGs as sets of relational facts $r(e, f)$ rather than sets of triples (e, r, f). Following standard practice, we use **Datalog rules** to encode the knowledge that allows us to reason about KGs.[1] An example of such a rule is as follows:

$$father_of(X, Z) \leftarrow father_of(X, Y) \wedge brother_of(Y, Z)$$

This rule encodes that whenever some entity X is known to be the father of Y and Y is known to be the brother of Z, we can infer that X is the father of Z. Given a KG $\mathcal{G}$ and a set of Datalog rules $\mathcal{P}$, we write $\mathcal{G} \cup \mathcal{P} \models r(e, f)$ to denote that $r(e, f)$ can be inferred from $\mathcal{G}$ using the rules in $\mathcal{P}$. For instance, in the example from Figure 1, we can infer $grandfather_of(e_3, e_4)$ from the given KG and the rules from NoRA (Das et al., 2025). In this context, we often refer to KGs $\mathcal{G}$ as **worlds**. The rules in $\mathcal{P}$ capture the regularities that hold in these worlds; they are therefore sometimes referred to as **world rules**. In addition, we often also have access to a set of **constraints** $\mathcal{C}$. Each constraint corresponds to a conjunction of literals that can never be jointly satisfied (e.g. we cannot have $mother_of(X, Y) \wedge father_of(X, Y)$). In the following, when generating worlds, we always guarantee that the constraints in $\mathcal{C}$ are satisfied, to prevent them from being non-nonsensical.

Difficulty metrics Current strategies for evaluating the systematic generalization abilities of neural reasoning models rely on metrics for assessing the difficulty of inferences of the form $\mathcal{G} \cup \mathcal{P} \models r(e, f)$. The most commonly used metric is the **inference depth (D)**, which is defined as the number of inference steps (i.e. rule applications) that are needed to infer $r(e, f)$. While standard Graph Neural Networks (GNNs) and Recurrent Neural Networks (RNNs) struggle to generalize to longer paths, experiments on benchmarks such as CLUTRR suggest that state-of-the-art models are rather robust

[1] Further details on Datalog can be found in Appendix 7.1

in this respect (Cheng et al., 2023a; Khalid & Schockaert, 2025). Das et al. (2025) introduced two additional difficulty metrics, both of which capture the intuition that off-path reasoning is harder than path-based reasoning. The **Off-Path Edge Count (OPEC)** is defined as the number of triples that are used to infer $r(e, f)$ which do not correspond to an edge of a path from e to f. The **Backtrack Load (BL)** instead measures the amount of off-path reasoning based on the close relation between inference depth and path length. It is defined as the ratio of the number of inference steps to the number of entities involved. The higher this ratio, the more the reasoning process intuitively deviates from path-based reasoning. Das et al. (2025) found that all tested models struggled to generalize to reasoning problems with higher OPEC or BL than the training examples.

Learning to reason Let $\mathcal{P}$ be a set of Datalog rules. An example of a $\mathcal{P}$-reasoning problem is a tuple $(\mathcal{G}, s, t, \mathcal{L})$ where $\mathcal{G}$ is a KG, s and t are entities from $\mathcal{G}$ (called the source and target entities), and $\mathcal{L}$ is the set of all relationships between e and f that can be inferred from $\mathcal{G}$, i.e. $\mathcal{L} = \{r \mid \mathcal{P} \cup \mathcal{G} \models r(e, f)\}$. Given a set of training examples $(\mathcal{G}_1, s_1, t_1, \mathcal{L}_1), ..., (\mathcal{G}_n, s_n, t_n, \mathcal{L}_n)$, the task of *learning to reason* is to train a model which can answer queries of the form $(\mathcal{G}_{n+1}, h_{n+1}, t_{n+1}, ?)$, i.e. predict the relationships that hold between two entities in an unseen KG $\mathcal{G}_{n+1}$. All KGs $\mathcal{G}_i$ share the same set of relations $\mathcal{R}$, but their entities are distinct. Intuitively, the model thus has to learn the rules from $\mathcal{P}$ based on the given training examples, and then apply these rules to the unseen test examples. We typically evaluate models on test examples which are harder than the training examples, according to some metric of difficulty. This means that the models are required to reason about the learned rules, rather than relying on statistical shortcuts, and tests their ability to generalize in a systematic way.

Learning to reason is a long-standing challenge in AI (Cropper et al., 2022), which remains largely unsolved. In this paper, we will primarily use Edge Transformers (ETs) for this task (Bergen et al., 2021). While various models have been proposed that can generalize well on path-based reasoning tasks Minervini et al. (2020); Lu et al. (2022c); Cheng et al. (2023b); Khalid & Schockaert (2025), most of these models struggle with off-path reasoning. ETs are theoretically capable of capturing arbitrary formulas from the logic C_3, i.e. the three-variable fragment of first-order logic with counting quantifiers (Müller et al., 2024), and can thus in principle learn the required knowledge. Empirically, ETs have also been found to generalize well in several settings Bergen et al. (2021); Müller et al. (2024), and they achieved by far the strongest results on the NoRA benchmark (Das et al., 2025).

3 METHODOLOGY

Problem setting Let M be a neural reasoning model that was trained on $\mathcal{P}$-reasoning examples (e.g. an ET). We want to find a sampling method which generates $\mathcal{P}$-reasoning examples that are hard for M. Our main focus is on generating graphs $\mathcal{G}$ from which difficult queries can be constructed. To make this more precise, we want to find a sampler S, i.e. a generative model from which we can sample worlds $\mathcal{G}$, satisfying a given set of constraint $\mathcal{C}$, which maximizes the following score:

$$\textit{difficulty}(\mathsf{S}; \mathsf{M}) = 1 - \mathbb{E}_{\mathcal{G} \sim \mathsf{S}}[\min_{h,t \in \mathcal{E}_\mathcal{G}} \min_{r \in \mathcal{L}_{\mathcal{G},h,t}} \mathsf{M}_r(\mathcal{G}, h, t)] \tag{1}$$

where we write $\mathcal{E}_\mathcal{G}$ for the set of entities in world $\mathcal{G}$, $\mathcal{L}_{\mathcal{G},h,t}$ for the ground truth answer to the query $(\mathcal{G}, h, t, ?)$, and $\mathsf{M}_r(\mathcal{G}, h, t)$ for the probability that $r(h, t)$ holds according to M.

3.1 EVOLUTIONARY SEARCH

We aim to learn samplers S that maximize *difficulty*$(\mathsf{S}; \mathsf{M})$ for a given model M. We focus on samplers that generate worlds by adding one edge at a time. To decide which edge to add in each step, we first randomly sample a set of N_{cand} candidate triples (e, r, f). These candidate triples are then scored by a **priority function** ρ, which depends on the partial world constructed thus far and the candidate triple. The highest scoring triple, according to this priority function, is added to the world. The problem of learning a sampler is thus reduced to that of learning a priority function.

To learn this priority function, we use an approach which is based on FunSearch (Romera-Paredes et al., 2024), an LLM-based evolutionary framework aimed at finding high-performing greedy algorithms for combinatorial problems. FunSearch is based on two key insights. First, given a candidate priority function, we can use an LLM to suggest an improved priority function (encoded as a Python method). Second, the quality of any given priority function ρ can be robustly assessed using

a task-specific evaluator. In our case, this evaluator uses the priority function ρ to generate a number of KGs $\mathcal{G}_1, ..., \mathcal{G}_{N_{\text{evalgraphs}}}$, and then uses the given model M to obtain an empirical estimate of (1):

$$\textit{fitness}(\rho) = 1 - \frac{1}{N_{\text{evalgraphs}}} \sum_{i=1}^{N_{\text{evalgraphs}}} \min_{h,t \in \mathcal{E}} \min_{r \in \mathcal{L}_{\mathcal{G}_i,h,t}} \mathsf{M}_r(\mathcal{G}_i, h, t)$$

We refer to this estimate as the fitness of the priority function ρ. To obtain a priority function with a high fitness score, following FunSearch, we maintain a database of previously generated priority functions. In each step, we select two of these priority functions and ask the LLM to suggest an improved one, where the selection is biased towards the functions with the highest fitness. To encourage sufficient exploration, FunSearch maintains multiple databases of priority functions, called islands, where the evolution process is applied to each of these islands independently. After N_{iter} iterations, half of the islands are removed, namely those whose best-scoring priority functions have the lowest fitness. These islands are then replaced by new databases, which are obtained by sampling the best-performing priority functions from the remaining islands. Each set of N_{iter} iterations is referred to as a **cycle**. Further details can be found in Appendix 7.3.

The evolutionary approach is aimed at finding queries that are hard for some model M. If this is successful, then a stronger model might be obtained by adding such queries to the training data. We want to analyze whether this process can be repeated. After a fixed number of N_{cycles} have been completed, we therefore re-train the ET on a training set that includes queries that were sampled from the best overall priority function found so far. We then repeat the process with this new model. We define a **round** as a period of N_{cycles} during which a fixed model M is used, after which it is retrained.

3.2 Auto-research agents

As a natural extension of the evolutionary approach, we consider an agent-in-the-loop procedure similar to `auto-research` (Karpathy, 2026). Different from the approach from Section 3.1, the agent is able to build context over multiple experimental runs. Moreover, it has the ability to run code and perform rudimentary analysis via tool calls and inspections. We initiate the loop by telling the agent to read a `program.md` file, which is provided Appendix 7.4. For this experiment, we refactored our codebase to its minimal set (a simple ET training loop, a problem instance generator utilizing the sampling function, etc.) to make sure the agent's context is minimally obfuscated. A coding agent (Claude Opus 4.6) iteratively edits a single Python program containing the priority function for a single round and the rest of protocol is kept the same as before. The agent's objective is to drive ET evaluation accuracy down while keeping the priority function as simple as possible.

3.3 Model-agnostic LLM-generated strategies

We also wanted to assess the potential of directly relying on the coding abilities of frontier LLMs. To this end, we generated a number of different samplers through a dialogue with Claude Opus 4.6. This led to four general samplers: a sampler which uses hill-climbing to optimize graphs for deep derivations (**Cl-hill**), a sampler which uses backward chaining to ensure there is at least one deep derivation (**Cl-bw**), a sampler which builds graphs from small self-contained patterns which trigger specific derivation rules (**Cl-motif**), and a hybrid strategy obtained by prompting Claude to combine ideas from the three aforementioned methods with the specific aim to challenge GNNs (**Cl-atlas**). These samplers can, in principle, be used for any set of Datalog rules $\mathcal{P}$ and constraints $\mathcal{C}$. We also obtained, as part of this dialogue, a number of samplers that were specific to the NoRA world: a sampler which starts from a few family skeleton templates (**Cl-N-temp**), a sampler which starts from a minimal two-person married seed and greedily expands the graph one edge at a time (**Cl-N-greedy**), and a variant of Cl-bw with heuristics that are specific to the NoRA world (**Cl-N-bw**). Further details on these samplers, and how they were obtained, can be found in Appendix 7.2.

4 Experiments

Reasoning models can be straightforwardly challenged by presenting them with sufficiently large KGs. Our main focus is therefore on whether it is possible to find *small* KGs which challenge these models. In particular, we restrict our experiments to KGs of at most 8 entities.

Table 1: Cross-evaluation of samplers on NoRA in terms of exact-match accuracy, comparing Claude-generated samplers with the result of four independent runs of the evolutionary process with qwen3-coder-next-80b.

		Test set								
		Cl-atlas	**Cl-motif**	**Cl-N-bw**	**Cl-N-greedy**	**Cl-N-temp**	**Evo1**	**Evo2**	**Evo3**	**Evo4**
Training set	**Cl-atlas**	0.92	0.59	0.26	0.02	0.28	0.16	0.17	0.16	0.16
	Cl-motif	0.58	0.96	0.43	0.19	0.32	0.24	0.25	0.25	0.24
	Cl-N-bw	0.04	0.09	0.99	0.10	0.13	0.08	0.09	0.06	0.09
	Cl-N-greedy	0.05	0.10	0.34	0.96	0.16	0.05	0.06	0.07	0.05
	Cl-N-temp	0.01	0.03	0.06	0.26	0.98	0.01	0.02	0.02	0.02
	Evo1	0.18	0.23	0.23	0.18	0.29	0.68	0.50	0.52	0.50
	Evo2	0.17	0.23	0.28	0.19	0.20	0.53	0.62	0.54	0.60
	Evo3	0.17	0.23	0.31	0.19	0.31	0.52	0.53	0.55	0.54
	Evo4	0.17	0.23	0.15	0.15	0.36	0.52	0.55	0.55	0.52

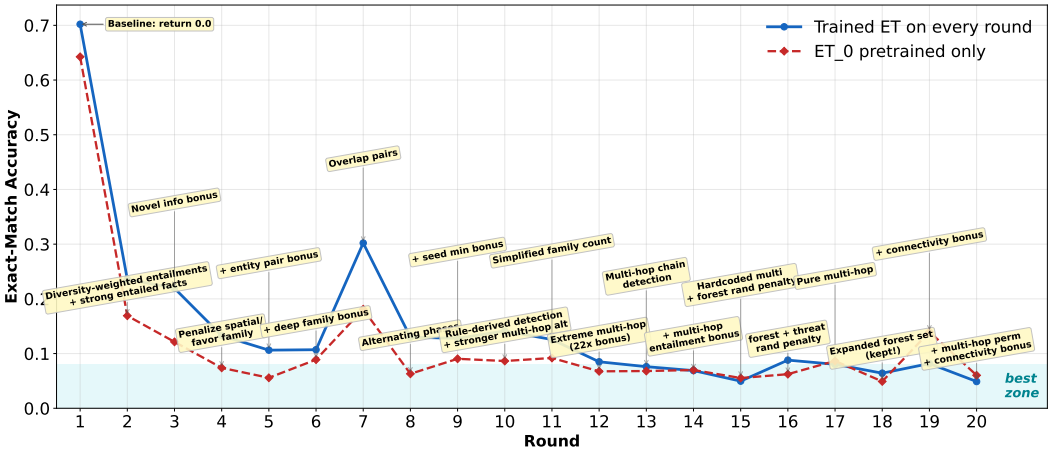


Figure 2: Auto-research results for Claude Opus 4.6 coding agent running in an infinite agent-in-the-loop. Exact-Match accuracies of the ET_0 (trained only once on NoRA 1.1) and the ET that is trained continually on each successive rounds from fresh and historical data generated by each round's priority function can be seen. Summaries of major changes that are introduced per round are also annotated.

Our analysis primarily focuses on the rules and constraints from the NoRA 1.1 benchmark Das et al. (2025). Similar to CLUTRR, this benchmark focuses on the problem of reasoning about family relationships. However, in contrast to previous benchmarks, NoRA requires off-path reasoning, which poses particular challenges for existing methods. NoRA 1.1 consists of a training split and three test splits: test-D, test-OPEC and test-BL (consisting of examples that are systematically harder than the training examples in terms of one of the difficulty metrics). For our analysis, some of the examples from test-D, test-OPEC and test-BL will be used for training the initial Edge Transformer of the Evolutionary search and Auto-Research Flow, since we want to find problems with different forms of difficulty than captured by these test sets.

To allow us to validate our findings on a second domain, we used Claude to generate a completely new set of world rules and constraints (see Appendix 7.5). The resulting domain is called **Iron Coast**. In addition to providing another dataset for evaluating the considered sampling strategies, experiments on this domain will allow us to assess the usefulness of LLM-generated reasoning domains, and thus whether it is feasible to fully automate the process of evaluating neural relational reasoners.

Results are evaluated in terms of exact-match accuracy, i.e. we report the fraction of queries $(\mathcal{G}, h, t, ?)$ for which the predicted set of relations exactly matches the ground truth $\mathcal{L}_{\mathcal{G},h,t}$.

Table 2: Cross-evaluation of priority functions obtained through evolutionary search (for NoRA) using qwen3-coder-next-80b, the priority function obtained at round 20 of the auto-research agent, and the priority function obtained when the process was initialised with the SUPERET model. Results are reported in terms of exact-match accuracy.

		Test set					
		Evo1	**Evo2**	**Evo3**	**Evo4**	**Auto-research**	**SUPERET-evo**
Training set	**Evo1**	0.68	0.50	0.52	0.50	0.03	0.47
	Evo2	0.53	0.62	0.54	0.60	0.03	0.47
	Evo3	0.52	0.53	0.55	0.54	0.04	0.45
	Evo4	0.52	0.55	0.55	0.52	0.03	0.46
	Auto-research	0.12	0.09	0.13	0.09	0.15	0.11
	SUPERET-evo	0.86	0.86	0.87	0.87	0.57	0.70

Evaluation of sampling strategies The difficulty of the queries that are obtained using a given sampling strategy can only be determined relative to a particular model. In Table 1, we therefore present a *cross-evaluation* of different samplers. Each entry in the table shows the performance of an ET that was trained on examples from a given training set and evaluated on queries generated by a particular sampler. We have included 5 of the Claude-generated samplers [2], which were used both to generate training sets and test sets. For instance, the results in the row marked Cl-atlas are those for an ET that was *trained* on examples sampled from Cl-atlas. The column marked Cl-atlas shows the results that were obtained for *test queries* that were sampled from Cl-atlas. We also included samplers obtained from four independent runs of the evolutionary process with Qwen3-next-80b. In particular, the row marked Evo$i$ show results for an ET that was trained on the round 4 training set of a evolutionary process with initial seed i. The columns marked Evo$i$ show results for test queries that were sampled using the priority function developed in round 5. Note that the diagonal elements are thus in-distribution for the Claude samplers, but not for the evolutionary runs. Full details on the experimental set-up can be found in Appendix 7.6.

The results show several clear patterns. First, the samplers obtained from the four runs of the evolutionary process appear to be similar: models trained on Evo1–Evo4 achieve a similar performance on the Evo1–Evo4 test sets. In contrast, models trained on the evolutionary samplers perform poorly on the Claude datasets, and *vice versa*. Among the Claude-based samplers, training on *Cl-motif* examples overall achieves the best results, whereas the NoRA-specific samplers are the weakest in this respect. As expected, the Claude models perform nearly perfectly on in-distribution examples.

We performed a similar cross-evaluation of samplers on the Iron Coast domain, finding broadly the same patterns (see Appendix 8.1). Furthermore, changing the LLM from Qwen3-next-80b to gpt-oss-120b or deepseek-coder-33b only had a minimal impact (see Appendix 8.2).

Figure 2 analyzes the effectiveness of the Auto-research experiment, described in Section 3.2. In each round of this process, a priority function is obtained. The figure shows the performance of two ETs on queries sampled from that priority function: the initial ET (trained on the NoRA dataset, as for the evolutionary approach) and an ET which is trained on a combination of the initial training data and examples sampled from the priority functions from previous rounds. As can be seen, most of the adversarial gain occurs in the first 3 rounds. Towards the later rounds, the performance of both ETs converges, suggesting that the Auto-research agent has found a priority function that challenges the ET, even when recent variations of that priority function are used for training the model.

Training reasoning models on augmented training data In Table 1, we found that both the evolved priority functions and the model-agnostic LLM-generated samplers succeed in challenging the ET. We now investigate whether a more robust ET model can be obtained by combining training examples from all these samplers. Specifically, for this experiment we train a model from 13 sources: the 9 training sets considered in Table 1, together with the training set, test-OPEC, test-BL and test-D splits from NoRA 1.1. We sampled 1300 examples at random from

[2]The remaining two generic Claude samplers generated non-nonsensical worlds, which is related to the fact that the NoRA constraints do not exhaustively avoid all nonsensical worlds. They were thus omitted for this analysis.

Table 3: Analysis of unexplained difficulty, in terms of the score defined in (2).

		Queries					
		Evo	**Claude**	**Auto-research**	**SUPERET-evo**	**test-OPEC**	**test-D**
Model	**Evo**	0.84	0.79	0.88	0.80	0.59	0.60
	Cl-atlas	0.54	0.81	0.87	0.53	0.45	0.44
	Auto-research	0.64	0.79	0.86	0.70	0.53	0.54
	SUPERET-evo	0.90	0.79	0.79	0.89	0.61	0.61
	NoRA-training	0.59	0.77	0.76	0.66	0.51	0.49

Table 4: Priority functions discovered for NoRA, with ET as the evaluator, transfer to other architectures. The table shows results for queries from the original NoRA training set, queries generated with a random priority function, and queries from the discovered priority functions.

	NoRA-Train	**Random**	**Auto-research**	**Evo1**	**Evo2**	**Evo3**	**Evo4**
EpiGNN	0.61	0.05	0.00	0.00	0.00	0.00	0.01
NBFNet	0.72	0.04	0.01	0.02	0.02	0.01	0.02
NCRL	0.55	0.63	0.05	0.22	0.21	0.24	0.22
R-GCN	0.44	0.42	0.00	0.14	0.13	0.13	0.11

each of these sources (which reflects the size of the smallest source). In the following, we refer to the resulting model as SUPERET. We then repeated the evolutionary process for this SUPERET. Figure 3 shows the performance of the (initial) SUPERET after each round of the evolutionary process (in purple), together with the performance of the (initial) ET in four runs of the standard evolutionary process. As can be seen, the performance of the SUPERET remains relatively strong throughout the process, showing that the evolutionary approach struggles to find strategies that challenge this model. Table 2 presents a cross-evaluation of 6 models: ETs trained on the round-4 training data from four standard runs of the evolutionary process (which are distinct from those that were used for training the SUPERET), a model trained on the round-19 training set from the Auto-research experiment, and the round-4 training data from the evolutionary process with the SUPERET. The models are evaluated on the round-5 priority functions of the evolutionary approaches and the round-20 priority function from the Auto-research experiment. This analysis confirms the robustness of SUPERET-evo, i.e. the SUPERET obtained after run 4 of the evolutionary process. We find that, apart from SUPERET-evo, all models perform poorly on the queries obtained from the Auto-research sampler. We see that the model trained on queries from the Auto-research sampler performs poorly overall, suggesting a lack of diversity in the generated worlds.

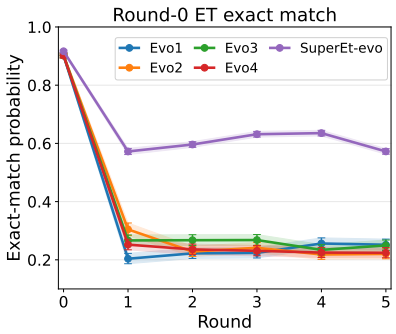


Figure 3: Performance of the original ET after each round of the evolutionary process, comparing a variant initialized with the SUPERET with four runs that were initialized with the standard ET. The results reflect the performance of the initial model on samples generated by the best priority functions at each round.

Transferability of learned priority functions Funsearch and the auto-research agent learn priority functions that generate NoRA problem instances that are challenging for the ET. Table 4 shows that these problem instances are also challenging for other neural reasoning architectures, in particular EpiGNN Khalid & Schockaert (2025), NBFNet Zhu et al. (2021), NCRL Cheng et al. (2023b) and R-GCN Schlichtkrull et al. (2018). In Table 5, we show that the priority functions discovered by the auto-research agent and Funsearch with the SUPERET also transfer across rule bases: priority functions that were discovered for NoRA generate hard problems for Iron Coast, and vice versa. Note that priority functions may refer to specific relations, in which case they cannot be transferred to other rule bases. However, the priority functions considered in Table 5 were generic. Further analysis of the transferability of the discovered priority functions can be found in Appendix 8.3.

Table 5: A sampler that was discovered using the auto-research agent on Iron Coast, with ET as the evaluator, generalizes to NoRA and other architectures (left). Samplers that were discovered on NoRA similarly generalize to Iron Coast (right); we show this for the priority function discovered by the auto-research agent and the one discovered in the SUPERET-evo experiment.

	Iron Coast → NoRA		**NoRA → Iron Coast**		
	Random	**Transferred (AR)**	**Random**	**Transferred (AR)**	**Transferred (SUPERET)**
ET	0.67	0.22	0.68	0.33	0.39
EpiGNN	0.65	0.21	0.50	0.16	0.28
NBFNet	0.30	0.11	0.59	0.20	0.31
NCRL	0.66	0.19	0.64	0.34	0.41
R-GCN	0.46	0.11	0.50	0.16	0.22

Analysis of unexplained difficulty We analyze to what extent the considered samplers have identified novel forms of difficulty, beyond inference depth, OPEC and BL. For a given ET model and test set, we train a classifier to predict whether the model will answer a given query correctly or not, using as features only the inference depth, OPEC and BL of that query (see Appendix 8.4 for details). We then evaluate the extent to which problem difficulty is unexplained, based on the McFadden pseudo-R^2 (R^2_{McF}), as follows:

$$\textit{Unexplained} = 1 - R^2_{\text{McF}} = \frac{l_{\textit{model}}}{l_{\textit{null}}} = \frac{\sum_{q \in Q^+} \log p_{\textit{model}}(q) + \sum_{q \in Q^-} \log(1 - p_{\textit{model}}(q))}{\sum_{q \in Q^+} \log p_{\textit{null}}(q) + \sum_{q \in Q^-} \log(1 - p_{\textit{null}}(q))} \tag{2}$$

where Q^+ and Q^- are the queries that were answered correctly and incorrectly by the ET, for a given set of queries obtained from the considered sampler; $p_{\textit{model}}(q)$ is the probability assigned by our classifier that q will be answered correctly; and $p_{\textit{null}}(q)$ is the probability assigned by an uninformed model, i.e. $p_{\textit{null}}(q) = \frac{|Q^+|}{|Q^+|+|Q^-|}$. In Table 3, we report the *Unexplained* value for different ET models and sampling strategies. This value reflects the extent to which there are aspects of difficulty in the queries that are unexplained by inference depth, OPEC and BL. A value of 1 reflects a situation where knowing the inference depth, OPEC and BL of a query does not provide any information about its difficulty. In contrast, a value of 0 reflects a situation where these difficulty metrics fully determine whether a query is answered correctly. Each row in the table corresponds to an ET that was trained on a particular training set: the round-5 training data of a standard evolutionary run, queries sampled from Cl-Atlas, queries sampled using the round-20 training data from the auto-research experiment, the round-5 training data of an evolutionary run based on the SUPERET, and the standard NoRA 1.1 training split. The columns correspond to different samplers that were used for generating the test queries: the round-5 priority function from a standard evolutionary run, Cl-Atlas, the round-20 sampler of the auto-research experiment, the round-5 priority function from the evolutionary process based on the SUPERET and the test-OPEC and test-D test splits from NoRA 1.1.

As expected, the *Unexplained* values for the test-OPEC and test-D queries are among the lowest in the table. In contrast, the values for the *Claude* and *Auto-research* test sets are consistently high, for all models, showing that these samplers generate problems whose difficulty is poorly explained by the considered metrics. Finally, for Evo and SUPERET-evo, we see large discrepancies across the different ET models. Focussing on the SUPERET-evo model, for instance, we can see a high unexplained value for the Evo queries. Given the stronger overall performance of this model, this suggests that it performs better than the other models on problems that are only difficult because of their inference depth, OPEC or BL value. The problems on which it struggles, however, have a difficulty which cannot be explained by these metrics.

5 RELATED WORK

Scientific discovery and meta-learning LLMs for automating scientific discovery via methods like FunSearch (Romera-Paredes et al., 2024) and its successor AlphaEvolve (Novikov et al., 2025) (essentially its multi-objective and multi-file editing counterpart) have yielded promising results in various domains including mathematics (Ellenberg et al., 2025), Bayesian and Graph-based Optimization (Aglietti et al., 2024; Verma et al., 2025) and cognitive science (Castro et al., 2025).

They focus on problems with automatically verifiable solutions, e.g., search for mathematical objects with certain properties or designing heuristic algorithms. At its heart, the idea is much older and is related to meta-learning (Schmidhuber, 1987), which attempts to learn algorithms via end-to-end optimization but is encumbered, primarily, by the intrinsic inability to differentiate through complex learning processes such as reinforcement learning (RL). Here, evolution-based optimization is a scalable alternative to RL (Salimans et al., 2017) due its gradient-free nature and has, in fact, also been applied to evolve surrogate objective functions to discover novel loss functions in RL (Lu et al., 2022a; Jackson et al., 2024; Lu et al., 2022b). Within a wider setting, LLM-based evolution has been applied to prompt engineering (Guo et al., 2024; Agrawal et al., 2026), neural architecture search (Nasir et al., 2024) and reward shaping (Hazra et al., 2025; Sapora et al., 2024).

Self-improvement and adversarial learning The idea of self-improvement (Schmidhuber, 1987; Nivel et al., 2013) is closely related to self-play (Samuel, 1959; Sutton et al., 1998), which naturally admits an adversarial interpretation (Goodfellow et al., 2014; Arjovsky et al., 2017) in the form of a two-player minimax game. Our approach is similar in spirit to adversarial learning methods, particularly adversarial imitation learning (Goodfellow et al., 2014; Ghasemipour et al., 2020; Chirra et al., 2026) where reward assignment functions are shaped as a function of the outputs of the underlying policy neural network whose training stability and performance are optimized. Another related thread is the idea of self-challenging (Zhou et al., 2026; Kulikov et al., 2026) or self-play (Sukhbaatar et al., 2018) where a challenger generates training examples for a solver, which can both be optimized together, thereby iteratively producing harder training examples. Nie et al. (2020) have also created a benchmark that specifically challenges a given model, focusing on the task of Natural Language Inference (NLI) and relying on crowdworkers to obtain the problem.

Synthetic benchmark generation The use of LLMs for generating synthetic benchmarks is common practice, e.g., instruction-tuning data for QA and agentic tool-use (Schick & Schütze, 2021; Wang et al., 2023; Shi et al., 2026; Xi et al., 2025) due to inherent scalability of the approach and the scarcity of human data (Singh et al., 2024). For instance, Guinet et al. (2024) generate question and answer pairs for evaluating Retrieval Augmented Generation (RAG) pipelines. The use of synthetic datasets is also common practice for evaluating the reasoning abilities of LLMs in various settings (Dziri et al., 2023; Valmeekam et al., 2022; Mirzadeh et al., 2025; Shojaee et al., 2025; Xie et al., 2025). Seedless, self-challenging, agentic data synthesis using a top-down, taxonomic, self-verification loop have also been proposed (Davidson et al., 2026; Kulikov et al., 2026). However, the evaluation pipelines in such settings rely on predefined sampling strategies, focussing more on generating test examples that cannot have appeared in pre-training than on discovering particular dimensions of difficulty. Another concern is verifying the quality of LLM-based problem instances. In contrast, our approach uses LLM-sampled programs that generate self-challenging instances.

6 Conclusions

We have studied the potential of LLMs for learning to sample problem instances that are challenging for relational reasoning models. Our analysis has focused on three broad strategies: (i) LLM-driven evolutionary search based on FunSearch, (ii) Auto-research agents which autonomously evolve sampling strategies, and (iii) simple dialogue-based prompting of frontier LLMs. We found all strategies to be successful in challenging the Edge Transformer (ET), a state-of-the-art neural relational learning model, despite the fact that the generated problem instances are small (consisting of at most 8 entities). We then considered whether a more robust ET model could be obtained by incorporating the learned samplers into the training data. The resulting model (SuperET) was indeed significantly more robust against the FunSearch-based evolutionary process, although it was still challenged by the sampler that was learned by the Auto-research agent. Our analysis of the learned samplers revealed that they have (implicitly) discovered aspects of difficulty which go beyond what can be explained through known difficulty metrics. Overall, our results show that the considered sampling strategies can play a central role in research on neural relational reasoning, both for stress-testing the robustness of trained models and for generating diverse and informative training examples. In the considered setting, the world rules are assumed to be given, i.e. we use LLM-driven strategies to sample knowledge graphs, but not to sample the rules that determine the regularities that need to be learned. However, we found that this approach can also be successfully applied to LLM-generated world rules. This makes it possible to fully automate the evaluation loop.

AI USE STATEMENT

In this work, generative AI tools were used to generate synthetic data sets. We used Claude Opus 4.6 to create model-agnostic samplers such as Cl-atlas and Cl-motif. These samplers were obtained through conversations with the model, which are summarized in Section 7.2 and shared in the paper's repository. Several commercial generative AI tools were also used to create world rules (Section 7.5), in particular the **Iron Coast** world, as well as additional smaller rule bases used to test the robustness of the samplers. Our implementation of the FunSearch algorithm invokes three open-source LLMs (qwen3-coder-next-80b, gpt-oss-120b and deepseek-coder-33b), whereas our implementation of the `auto-research` algorithm relies on a Claude Opus 4.6 coding agent to generate and edit priority functions.

Generative AI tools were also used as coding assistants, namely for adapting the FunSearch and `auto-research` algorithms to the relational reasoning problem, and more generally throughout the development of the paper's repository. All code produced with such assistance was reviewed and tested by three authors. Moreover, since all generated problems and their solutions are checked with the Clingo ASP reasoner, the correctness of the experimental outputs does not rely on the AI-generated code alone.

The plots presenting the qualitative analysis of the form and content of the successive priority functions obtained with the `auto-research` algorithm (e.g., Figures 2, 7, 8 & 9), and the properties of the graphs sampled with various samplers (e.g., Figures 10, 11, 12 & 13 in the appendix) were generated with the support of generative AI tools; the conclusions drawn from these analyses are the authors' own. Additionally, generative AI tools were used to improve the readability of the paper and to format LaTeX code.

We have not used generative AI tools to develop theoretical models or conceptual frameworks, propose or refine hypotheses, design or provide feedback on research methodology or experiments, or interpret results. Formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, assisting with translation, and cleaning or reformatting data sets are not applicable to this work.

We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

We have not identified immediate ethical concerns or potentially harmful applications arising from this work. All new datasets used in this study are synthetic: the reasoning problems are generated from rule bases and verified automatically with the Clingo ASP reasoner, so no human annotation was required. No personal or sensitive data was collected or used. The use of generative AI tools in this work is described in AI use statement section.

REPRODUCIBILITY STATEMENT

In the HuggingFace repository[3], we release the data used to train the initial model at round 0, the best priority functions obtained in our experiments, and, for every experiment, the datasets of query graphs sampled at each round with the best priority function obtained so far. The FunSearch (Section 3.1) and `auto-research` (Section 3.2) implementations developed for our experiments are available in the `auto-world-rules` GitHub repository[4].

The material obtained through conversations with Claude Opus 4.6 and other proprietary, closed-source LLMs has been made as reproducible as possible. The key prompts are given in the appendix (Section 7.2), and the GitHub repository contains the full text of the conversations together with the generated outputs. For the samplers, this includes the model-agnostic samplers themselves (both NoRA-specific and general samplers), the export of the conversation with Claude Opus 4.6 that led to their final version, the main scripts generated during this conversation to evaluate the correctness and robustness of the samplers, and a summary of the successive corrections and improvements made to

[3] `https://huggingface.co/datasets/autoworldrules/funsearch`

[4] `https://github.com/autoworldrules/auto-world-rules`

the samplers over the course of the conversation. The repository also contains the worlds generated by LLMs: **Iron Coast**, used in our experiments (Section 8.1), as well as the other worlds generated by Claude Opus 4.6 and other closed-source models (GPT-5.1, Claude Sonnet 4.6 and Gemini 3.1 Pro).

REFERENCES

Virginia Aglietti, Ira Ktena, Jessica Schrouff, Eleni Sgouritsa, Francisco JR Ruiz, Alan Malek, Alexis Bellot, and Silvia Chiappa. Funbo: Discovering acquisition functions for bayesian optimization with funsearch. *arXiv preprint arXiv:2406.04824*, 2024.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RQm2KQTM5r.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pp. 214–223. Pmlr, 2017.

Leon Bergen, Timothy J. O'Donnell, and Dzmitry Bahdanau. Systematic generalization with edge transformers. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 1390–1402, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/0a4dc6dae338c9cb08947c07581f77a2-Abstract.html.

Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pp. 2787–2795, 2013. URL https://proceedings.neurips.cc/paper/2013/hash/1cecc7a77928ca8133fa24680a88d2f9-Abstract.html.

Pablo Samuel Castro, Nenad Tomasev, Ankit Anand, Navodita Sharma, Rishika Mohanta, Aparna Dev, Kuba Perlin, Siddhant Jain, Kyle Levin, Noemi Elteto, Will Dabney, Alexander Novikov, Glenn C Turner, Maria K Eckstein, Nathaniel D. Daw, Kevin J Miller, and Kim Stachenfeld. Discovering symbolic cognitive models from human and animal behavior. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 6849–6890. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/castro25a.html.

Kewei Cheng, Nesreen K. Ahmed, and Yizhou Sun. Neural compositional rule learning for knowledge graph reasoning. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023a. URL https://openreview.net/pdf?id=F8VKQyDgRVj.

Kewei Cheng, Nesreen K. Ahmed, and Yizhou Sun. Neural compositional rule learning for knowledge graph reasoning. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023b. URL https://openreview.net/forum?id=F8VKQyDgRVj.

Shashank Reddy Chirra, Jayden Teoh, Praveen Paruchuri, and Pradeep Varakantham. On discovering algorithms for adversarial imitation learning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=hPz3doftL4.

William W. Cohen. Graphlog: A benchmark for logical learning on graphs. In *Proceedings of the 2019 International Symposium on Inductive Logic Programming*, 2019.

Andrew Cropper, Sebastijan Dumancic, Richard Evans, and Stephen H. Muggleton. Inductive logic programming at 30. *Mach. Learn.*, 111(1):147–172, 2022. doi: 10.1007/S10994-021-06089-1. URL https://doi.org/10.1007/s10994-021-06089-1.

Anirban Das, Irtaza Khalid, Rafael Peñaloza, and Steven Schockaert. When no paths lead to rome: Benchmarking systematic neural relational reasoning. In *NeurIPS*, 2025.

Tim R. Davidson, Benoit Seguin, Enrico Bacis, Cesar Ilharco, and Hamza Harkous. Reasoning-driven synthetic data generation and evaluation. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=NALsdGEPhB. J2C Certification.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Xiang Ren, Allyson Ettinger, Zaïd Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/deb3c28192f979302c157cb653c15e90-Abstract-Conference.html.

Jordan S Ellenberg, Cristofero S Fraser-Taliente, Thomas R Harvey, Karan Srivastava, and Andrew V Sutherland. Generative modeling for mathematical discovery. *arXiv preprint arXiv:2503.11061*, 2025.

Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider. Potassco: The potsdam answer set solving collection. *Ai Communications*, 24(2): 107–124, 2011.

Seyed Kamyar Seyed Ghasemipour, Richard Zemel, and Shixiang Gu. A divergence minimization perspective on imitation learning methods. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura (eds.), *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pp. 1259–1277. PMLR, 30 Oct–01 Nov 2020. URL https://proceedings.mlr.press/v100/ghasemipour20a.html.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/file/f033ed80deb0234979a61f95710dbe25-Paper.pdf.

Gauthier Guinet, Behrooz Omidvar-Tehrani, Anoop Deoras, and Laurent Callot. Automated evaluation of retrieval-augmented language models with task-specific exam generation. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, Proceedings of Machine Learning Research, pp. 16773–16801. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/guinet24a.html.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ZG3RaNIsO8.

Rishi Hazra, Alkis Sygkounas, Andreas Persson, Amy Loutfi, and Pedro Zuidberg Dos Martires. Revolve: Reward evolution with large language models using human feedback. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=cJPUpL8mOw.

Matthew Thomas Jackson, Chris Lu, Louis Kirsch, Robert Tjarko Lange, Shimon Whiteson, and Jakob Nicolaus Foerster. Discovering temporally-aware reinforcement learning algorithms. In

The Twelfth International Conference on Learning Representations, 2024. URL https://openreview.net/forum?id=MJJcs3zbmi.

Andrej Karpathy. autoresearch. https://github.com/karpathy/autoresearch, March 2026. GitHub repository.

Muhammad Khalid and Steven Schockaert. Systematic relational reasoning with epistemic graph neural networks. *The Thirteenth International Conference on Learning Representations*, 2025.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2017.

Ilia Kulikov, Chenxi Whitehouse, Tianhao Wu, Swarnadeep Saha, Eryk Helenowski, Weizhe Yuan, Olga Golovneva, Jack Lanchantin, Yoram Bachrach, Jakob Foerster, Xian Li, Han Fang, Sainbayar Sukhbaatar, and Jason Weston. Autodata: an automatic data scientist to create high quality data. April 2026. URL https://facebookresearch.github.io/RAM/blogs/autodata/.

Chris Lu, Jakub Kuba, Alistair Letcher, Luke Metz, Christian Schroeder de Witt, and Jakob Foerster. Discovered policy optimisation. *Advances in Neural Information Processing Systems*, 35:16455–16468, 2022a.

Christopher Lu, Timon Willi, Christian A Schroeder De Witt, and Jakob Foerster. Model-free opponent shaping. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 14398–14411. PMLR, 17–23 Jul 2022b. URL https://proceedings.mlr.press/v162/lu22d.html.

Shengyao Lu, Bang Liu, Keith G. Mills, Shangling Jui, and Di Niu. R5: rule discovery with reinforced and recurrent relational reasoning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022c. URL https://openreview.net/forum?id=2eXhNpHeW6E.

Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. Anytime bottom-up rule learning for knowledge graph completion. In Sarit Kraus (ed.), *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pp. 3137–3143. ijcai.org, 2019. doi: 10.24963/IJCAI.2019/435. URL https://doi.org/10.24963/ijcai.2019/435.

Pasquale Minervini, Sebastian Riedel, Pontus Stenetorp, Edward Grefenstette, and Tim Rocktäschel. Learning reasoning strategies in end-to-end differentiable proving. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 6938–6949. PMLR, 2020. URL http://proceedings.mlr.press/v119/minervini20a.html.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=AjXkRZIvjB.

Luis Müller, Daniel Kusuma, Blai Bonet, and Christopher Morris. Towards principled graph transformers. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/e5419147e53eba322cf12aff266a66f2-Abstract-Conference.html.

Muhammad Umair Nasir, Sam Earle, Julian Togelius, Steven James, and Christopher W. Cleghorn. Llmatic: Neural architecture search via large language models and quality diversity optimization. In Xiaodong Li and Julia Handl (eds.), *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2024, Melbourne, VIC, Australia, July 14-18, 2024*. ACM, 2024. doi: 10.1145/3638529.3654017. URL https://doi.org/10.1145/3638529.3654017.

Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. Adversarial NLI: A new benchmark for natural language understanding. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4885–4901, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.441. URL https://aclanthology.org/2020.acl-main.441/.

Eric Nivel, Kristinn R Thórisson, Bas R Steunebrink, Haris Dindo, Giovanni Pezzulo, Manuel Rodriguez, Carlos Hernández, Dimitri Ognibene, Jürgen Schmidhuber, Ricardo Sanz, et al. Bounded recursive self-improvement. *arXiv preprint arXiv:1312.6764*, 2013.

Alexander Novikov, Ngân Vu, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery. *CoRR*, abs/2506.13131, 2025. doi: 10.48550/ARXIV.2506.13131. URL https://doi.org/10.48550/arXiv.2506.13131.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nat.*, 625(7995):468–475, 2024. doi: 10.1038/S41586-023-06924-6. URL https://doi.org/10.1038/s41586-023-06924-6.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

Silvia Sapora, Gokul Swamy, Chris Lu, Yee Whye Teh, and Jakob Nicolaus Foerster. EvIL: Evolution strategies for generalisable imitation learning. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 43407–43421. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/sapora24a.html.

Timo Schick and Hinrich Schütze. Generating datasets with pretrained language models. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 6943–6951, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.555. URL https://aclanthology.org/2021.emnlp-main.555/.

Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In Aldo Gangemi, Roberto Navigli, Maria-Esther Vidal, Pascal Hitzler, Raphaël Troncy, Laura Hollink, Anna Tordai, and Mehwish Alam (eds.), *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*, volume 10843 of *Lecture Notes in Computer Science*, pp. 593–607. Springer, 2018. doi: 10.1007/978-3-319-93417-4_38. URL https://doi.org/10.1007/978-3-319-93417-4_38.

Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

Dingfeng Shi, Jingyi Cao, Qianben Chen, Weichen Sun, Weizhen Li, Hongxuan Lu, Fangchen Dong, Tianrui Qin, King Zhu, Minghao Liu, Yuchen Eleanor Jiang, Jian Yang, Ge Zhang, Jiaheng Liu, Changwang Zhang, Jun Wang, and Wangchunshu Zhou. Taskcraft: Automated generation of agentic tasks. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=UJFCyrYM1V.

Parshin Shojaee, Iman Mirzadeh, Keivan Alizadeh, Maxwell Horton, Samy Bengio, and Mehrdad Farajtabar. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity. *CoRR*, abs/2506.06941, 2025. doi: 10.48550/ARXIV.2506.06941. URL https://doi.org/10.48550/arXiv.2506.06941.

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron T Parisi, Abhishek Kumar, Alexander A Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Fathy Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura A Culp, Lechao Xiao, Maxwell Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=lNAyUngGFK. Expert Certification.

Koustuv Sinha, Shagun Sodhani, Jin Dong, Joelle Pineau, and William L. Hamilton. Clutrr: A diagnostic benchmark for inductive reasoning from text. In *Proceedings of EMNLP-IJCNLP*, pp. 4505–4514. Association for Computational Linguistics, 2019. doi: 10.18653/V1/D19-1458. URL https://doi.org/10.18653/v1/D19-1458.

Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SkT5Yg-RZ.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2): 146–160, 1972. doi: 10.1137/0201010. URL https://doi.org/10.1137/0201010.

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Large language models still can't plan (a benchmark for LLMs on planning and reasoning about change). In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022. URL https://openreview.net/forum?id=wUU-7XTL5XO.

Samidha Verma, Arushi Goyal, Ananya Mathur, Ankit Anand, and Sayan Ranu. GRAIL: Graph edit distance and node alignment using LLM-generated code. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=NzoZXju2bL.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13484–13508, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.754. URL https://aclanthology.org/2023.acl-long.754/.

Zhiheng Xi, Yiwen Ding, Wenxiang Chen, Boyang Hong, Honglin Guo, Junzhe Wang, Xin Guo, Dingwen Yang, Chenyang Liao, Wei He, Songyang Gao, Lu Chen, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, Xipeng Qiu, Xuanjing Huang, Zuxuan Wu, and Yu-Gang Jiang. AgentGym: Evaluating and training large language model-based agents across diverse environments. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27914–27961, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1355. URL https://aclanthology.org/2025.acl-long.1355/.

Chulin Xie, Yangsibo Huang, Chiyuan Zhang, Da Yu, Xinyun Chen, Bill Yuchen Lin, Bo Li, Badih Ghazi, and Ravi Kumar. On memorization of large language models in logical reasoning. In Kentaro Inui, Sakriani Sakti, Haofen Wang, Derek F. Wong, Pushpak Bhattacharyya, Biplab

Banerjee, Asif Ekbal, Tanmoy Chakraborty, and Dhirendra Pratap Singh (eds.), *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, IJCNLP-AACL 2025, Mumbai, India, December 20-24, 2025*, pp. 2742–2785. The Asian Federation of Natural Language Processing and The Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.ijcnlp-long.148/.

Yifei Zhou, Sergey Levine, Jason E Weston, Xian Li, and Sainbayar Sukhbaatar. Self-challenging language model agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=9yusqX9DpR.

Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural Bellman-Ford networks: A general graph neural network framework for link prediction. *Advances in neural information processing systems*, 34:29476–29490, 2021.

7 Additional details

7.1 Background on Datalog

We rely on Datalog rules to encode the knowledge for reasoning about KGs. An example of a Datalog rule is as follows:

$$\textit{father_of}(X, Z) \leftarrow \textit{father_of}(X, Y) \wedge \textit{brother_of}(Y, Z)$$

This rule encodes that whenever some entity X is known to be the father of Y and Y is known to be the brother of Z, we can infer that X is the father of Z. Following standard practice, we use uppercase arguments such as X to denote variables and lowercase arguments such as *bob* to denote specific entities (i.e. constants). A literal is an expression of the form $r(\alpha_1, ..., \alpha_n)$ where r is a relation and $\alpha_1, ..., \alpha_n$ are either constants or variables. A Datalog rule is an expression of the form *head* $\leftarrow$ *body*, where *head* is a literal and *body* is a conjunction of literals, such that every variable which appears in *head* also appears in *body*. In this paper, we will not be concerned with literals that have more than two arguments. If all arguments of the literal α are constants, we call α a ground literal. Similarly, a ground rule is a rule in which only ground literals occur. Given a set of Datalog rules $\mathcal{P}$ and a KG $\mathcal{G}$, the grounding $\mathcal{P}_\mathcal{G}$ is defined as the set of all ground rules that can be constructed from the rules in $\mathcal{P}$ by instantiating their variables by entities from $\mathcal{G}$.

Entailment Given a KG $\mathcal{G}$ and a set of Datalog rules $\mathcal{P}$, we write $\mathcal{G} \cup \mathcal{P} \models r(e, f)$ to denote that $r(e, f)$ can be inferred from $\mathcal{G}$ using the rules in $\mathcal{P}$. This entailment relation can be formally defined as follows. We write $\mathcal{G} \cup \mathcal{P} \models_0 r(e, f)$ to denote that $r(e, f) \in \mathcal{G}$. For $i > 1$ we have $\mathcal{G} \cup \mathcal{P} \models_i r(e, f)$ iff $\mathcal{G} \cup \mathcal{P} \models_{i-1} r(e, f)$ or there is a rule $r(e, f) \leftarrow \alpha_1 \wedge ... \wedge \alpha_k$ in $\mathcal{P}_\mathcal{G}$ such that $\mathcal{G} \cup \mathcal{P} \models_{i-1} \alpha_j$ holds for every $j \in \{1, ..., k\}$. We then have $\mathcal{G} \cup \mathcal{P} \models r(e, f)$ iff $\mathcal{G} \cup \mathcal{P} \models_i r(e, f)$ for some $i \in \mathbb{N}$.

Path-based and off-path reasoning When reasoning about knowledge graphs, so-called closed path rules often play an important role (Meilicke et al., 2019). These are rules of the following form:

$$r(X_1, X_n) \leftarrow r_1(X_1, X_2) \wedge ... \wedge r_{n-1}(X_{n-1}, X_n)$$

When all rules in $\mathcal{P}$ are closed path rules, then $\mathcal{G} \cup \mathcal{P} \models r(e, f)$ holds iff there is a path $\mathcal{G}_0 = \{(e, r_1, e_1), (e_1, r_2, e_2), ..., (e_{k-1}, r_k, f)\}$ such that $\mathcal{G}_0 \cup \mathcal{P} \models r(e, f)$. Methods such as NCRL (Cheng et al., 2023a) directly exploit this connection, which simplifies learning significantly. For instance, CLUTRR (Sinha et al., 2019) only involves closed path rules, which is why NCRL performs well on this benchmark. Unfortunately, path-based methods are not sufficient for reasoning about other types of rules, as illustrated in the next example.

Example 1. *The rule base $\mathcal{P}_{nora}$ from NoRA (Das et al., 2025) includes the following rules:*

$$\begin{aligned}
\textit{has_property}(X, \textit{no_daughters}) &\leftarrow \textit{parent_of}(X, Y) \wedge \textit{belongs_to_group}(Y, \textit{male}) \\
&\quad \wedge \textit{has_property}(Y, \textit{no_sisters}) \\
\textit{belongs_to_group}(Y, \textit{male}) &\leftarrow \textit{parent_of}(X, Y) \wedge \textit{has_property}(X, \textit{no_daughters}) \\
\textit{father_of}(X, Y) &\leftarrow \textit{parent_of}(X, Y) \wedge \textit{belongs_to_group}(X, \textit{male})
\end{aligned}$$

Let us furthermore consider the following KG:

$$\mathcal{G} = \{\textit{parent_of}(\textit{bob}, \textit{dave}), \textit{belongs_to_group}(\textit{dave}, \textit{male}), \textit{has_property}(\textit{dave}, \textit{no_sisters}), \\ \textit{parent_of}(\textit{bob}, \textit{john}), \textit{parent_of}(\textit{john}, \textit{alice})\}$$

We have $\mathcal{G} \cup \mathcal{P}_{nora} \models \textit{father_of}(\textit{john}, \textit{alice})$, but this inference relies on knowledge about dave (to infer that bob has no daughters), even though dave is not on any path between john and alice in $\mathcal{G}$.

7.2 Generating model-agnostic samplers using LLMs

The samplers from Section 3.3 were generated and iteratively refined by prompting Claude Opus 4.6 through a dialogue. The LLM was initially requested to create a new set of world rules[5] and then to design an associated sampler in the form of a Python program. Next, we prompted the model to generalize the approach, to make it suitable for any set of rules. Throughout this conversation, we tested the produced samplers by executing the code and collecting sets of story graphs for rule compliance checks. These story graphs were validated locally using the Clingo ASP solver[6] (Gebser et al., 2011), to detect any violation of the rules in the sampled graphs. When errors were found, they were passed back to Claude, with a prompt requesting the model to identify the problem and apply the necessary corrections. It is worth noting that Claude Opus 4.6 does not have access to Clingo in its sandbox. However, it relies on forward derivation chains implemented in the samplers to explore the worlds, and run internal tests related to the correctness of the sampled stories, relying mostly on Python tool calls. The final version of each sampler was obtained after less than 10 update requests. More details about the generation process and the detected errors are given below.

First version of the samplers The model was prompted to write Python programs that sample knowledge graphs from a set of rules (either LLM-generated or user provided). The total number of vertices in the sampled graphs is fixed in advance and passed to the Python program as an argument. We specified in the prompt that the objective was to sample graphs that are *challenging for reasoning models* and would *require systems to make difficult inferences beyond transitivity and hierarchy reasoning steps*. The model is asked to try and design a smart sampling strategy, which could be based on *sampling one edge at a time or expanding a seed graph recursively with subgraph motifs*. We also indicated in the prompt that any creative method can be used. The first world-specific samplers were obtained in this way, including the first version of the Cl-N-temp sampler. The two other NoRa-specific samplers were obtained by prompting the model to design two further algorithms, different from this first sampler and different from each other. We thus obtained the first versions of the Cl-N-greedy and Cl-N-bw samplers. General samplers were obtained by prompting Claude to extend the specific samplers so that they can be used for any set of world rules. Claude Opus 4.6 generated the Cl-hill, Cl-bw, Cl-motif and Cl-atlas samplers in this way.

Errors and refinements Given the initial version of a sampler, we then asked the model to verify, edit and eventually improve the samplers. In general, we found the initial samplers to be quite accurate. We observed several main types of errors when testing the samplers on the NoRa world:

- The arity of the relations was not immediately handled correctly. This was particularly the case for unary predicates expressed with two slots, such are $\textit{person}(a, a)$.
- The required number of entities in the sampled KGs is specified as an argument to the sampler. The initial samplers did not always ensure the sampled graphs had the correct number of entities.
- NoRA's 292 rules create circular dependencies between all 51 predicates. Cycle detection in the rules was not handled in the first versions of the samplers, leading several samplers, such as Cl-bw to be caught in infinite loops.

In addition to NoRA, the general samplers were tested on ten different sets of world rules, which were generated by different LLMs.

[5] described in more details in Appendix 7.5

[6] `https://potassco.org/clingo/`

Generated samplers We now describe the main strategy used by each of the generated samplers.

- **Cl-hill**: this hill-climbing algorithm first analyzes and categorizes the rules, using categories such as base, derived, seedable, symmetric, constraint predicates, join patterns from multi-body rules and transitivity rules. It then seeds an initial set of base facts. Finally, it relies on hill climbing to optimize graphs for deep derivations and fewer facts, by operating several mutations on the initial seed graph and scoring the mutated graphs. The possible mutations include the addition of an entire sub-graph at once, to improve graphs connectivity and add transitive chains. It implements a form of forward chaining to measure the associated proof depths.
- **Cl-bw**: this backward sampling method is goal-directed in the sense that it guarantees that every generated graph contains at least one deep derivation chain by construction. Its graph scoring function is similar to the one used by Cl-hill. It additionally implements Strongly Connected Component (SCC) analysis to detect and handle circular rules[7]. For each target predicate, the sampler traces backwards through the rules to build a *proof skeleton*, i.e. a template proof tree with variables instead of constants. The proof templates are then instantiated with specific constants and the resulting skeletons are combined into graphs, with constants shared across skeletons so the proofs interconnect. A population of graphs is generated with this method, and the best graphs are selected according to a scoring function. Candidates are ranked first by a cheap proxy score, and the top few candidates are then evaluated with the full scoring function.
- **Cl-motif**: similarly to the backward sampler, and unlike the hill climbing sampler, this method has a certain level of rule awareness. Motifs are similar to the proof skeletons from Cl-bw, but at a smaller scale. They are small self-contained patterns that trigger specific derivation rules. They are instantiated with facts and used as building blocks with shared variables to obtain complete graphs. Generated graphs are then ranked with various strategies for a selection of the best ones. This sampler's implementation is general in the sense that is can be used with new sets of rules, but it also contains a few heuristics which are specific to the worlds that were used during the validation and refinement process.
- **Cl-atlas**: this hybrid sampler combines ideas Cl-hill, Cl-bw and Cl-motif. In a first step, a "strategy mix" is chosen at random among four weighted items. Each strategy mix combines two elements such as "backward + joins" or "joins + chains". Once graphs are built following a given strategy mix, hill climbing is applied to improve it by trying random mutations. Cl-Atlas is also the only sampler where we specifically asked, in the Claude prompt, to design a method that would defeat a GNN evaluator. In consequence, (i) the graph scoring function explicitly favors deep reasoning chains, because maximum derivation depth is said to correspond to the number of message-passing steps a graph neural network would need; (ii) predicate diversity is expected to force multi-relational reasoning rather than single-relation pattern matching; and (iii) distractors (irrelevant facts that look useful but are not in any proof) are added because they are expected to make the GNN's discrimination harder.
- **Cl-N-temp**: with this NoRA-specific method, graphs are first generated using a few family skeleton templates (e.g. *nuclear*, *multi-generational* and *all male children* families) and hard-coded fact selection heuristics such as strategic gender omission to trigger gender inference chains. The best graphs are then selected using a score that estimates reasoning complexity from graph topology, based on the presence or number of certain types of predicates. For example, the score favours the presence of multiple underage children in the story because it triggers *living_in_same_place* queries, which typically occur in complex reasoning chains.
- **Cl-N-greedy**: this NoRA-specific algorithms starts from a minimal two-person married seed. The graph is then expanded one edge at a time by scoring candidate mutations (e.g. *add child*, *add marriage*, *flip gender*). Candidate mutations are scored using forward chaining, to measure the inference depth and the number of non-trivial inferences. The method greedily selects the mutation that maximizes this composite score.

[7]The generated script suggests that it relies on the algorithm from Tarjan (1972).

- **Cl-N-bw**: this is a NoRA-specific variant of the general backward sampler, with an identical long rule chain sampling goal, but where heuristics specific to the NoRA world are hard-coded. In particular, the algorithm relies on seven hard-coded proof recipes, targeting long derivations in NoRA, instead of building abstract proof skeletons from the available rules. Graphs are built by instantiating and combining proof trees, maximizing the number of deep derivations per entity. Gender information is hidden were possible, to make inference harder.

Note that Cl-N-greedy and Cl-hill are not rule-aware, in the sense that they randomly generate facts and then apply a scoring metric to decide if a given modification should be preserved. In contrast, the two backward samplers and the Cl-motif sampler explicitly rely on the rules and the proof derivations to construct the graphs. Several of the samplers also crucially rely on motifs. In addition to Cl-motif, Cl-hill and Cl-N-temp also took into account the motif hint that was given as part of the prompt. The motifs are implemented with different levels of generalization, from hard-coded templates to more abstract rule-aware skeletons. The Atlas sampler was the last generated sampler. It is the only sampler where the prompt explicitly referred to the task of defeating a GNN-based reasoner. As a result, the sampler relies on distractor edges, and tried to diversify the generated graphs by a random strategy selection initial step. However, all the samplers rely on a similar scoring function, which makes assumptions about what makes a graph challenging for reasoning models, based on known difficulties such as long rule chains. Unlike the evolutionary algorithm, the scoring function is blind to the real bottlenecks encountered by the ET model.

7.3 LLM-BASED IMPROVEMENT OF PRIORITY FUNCTIONS

FunSearch Prompt The LLM is requested to output a priority function that is an improvement over two previously generated priority functions. The prompt contains a full explanation of the evolutionary experimental setting, a detailed description of ASP syntax and the task. A set of callable graph tools are also provided under the form of a list of Python functions. Final format instructions and two previously generated priority functions are included in the body of the prompt.

SUMMARY OF THE PROMPT FOR THE GENERATION OF NEW PRIORITY FUNCTIONS

You are an AI participating in an evolutionary algorithm search to generate candidate priority functions.

TASK OVERVIEW

You are designing a function:

```
priority(cand_fact, definite_rules_program, entailed_facts, facts_program) -> float
```

The purpose of this function is to score how useful a candidate edge would be if added to an existing graph, given what additional edges become entailed in that graph by the `definite_rules_program`.

EVOLUTIONARY PROGRESSION

At iteration 0, `priority_v0` is the baseline and `priority_v1` is incomplete. Your job is to output a complete `priority_v1` that improves on `priority_v0`. After the first iteration, `priority_v0` and `priority_v1` are shown. `priority_v1` is an improvement over `priority_v0` and `priority_v2` is incomplete. Your job is to output a complete `priority_v2` that improves on `priority_v1`.

ARGUMENTS

- `cand_fact (str)`: this is a single directed edge string you are scoring. It is exactly one edge statement ending in a period.
- `definite_rules_program (str)`: this is a string containing many graph-transformation rules, one rule per line. These rules describe when the presence of certain edge patterns in the graph implies that another edge must also exist.
- `entailed_facts (str)`: this is a string containing directed edges that become implied after adding `cand_fact` to `facts_program` and then applying the graph-transformation rules, excluding edges that were already explicitly present in `facts_program` (and excluding the candidate edge itself).
- `facts_program (str)`: this is the existing graph, a string containing directed edges already present, usually one per line, each ending in a period.

GRAPH TOOLKIT

You may call pre-built graph-theoretic functions directly inside your priority function. No import statement is needed; they are automatically available. The available functions are:

```
# --- Path Analysis ---
shortest_path_length(G, src, tgt) -> int | None
    Number of hops on the shortest directed path. Returns None if unreachable.
...
# --- Structure ---
node_degree(G, node, mode='all') -> int
    Degree of a node. mode: 'in' (incoming), 'out' (outgoing), 'all' (both).
...
```

EVOLUTION STRATEGIES

If you only see a complete `priority_v0`, try simple approaches first:

- Count implied edges.
- Find candidate edges whose endpoint nodes are common with nodes already appearing in `facts_program`.
- Return any positive number based on simple metrics.

When you see full demonstrations `priority_v0`, `priority_v1`, and need to complete `priority_v2`:

1. TWEAK THE BEST: take `priority_v1` (the current best) and make small modifications:
   - Adjust constants/weights slightly (e.g., change 0.5 to 0.7, or 2.0 to 1.5).
   - Add a small bonus/penalty term to the existing logic.
   - Modify thresholds or conditions slightly.
   - ...
2. TRY SOMETHING COMPLETELY NEW: ignore previous versions and experiment with a fresh idea.

THE TWO PRIORITY FUNCTIONS

{`priority_v0`} and {`priority_v1`}

7.4 AUTO-RESEARCH AGENT-IN-THE-LOOP INITIATION PROMPT

PROGRAM.MD

Experimental loop (agent-in-the-loop).
An outer loop over rounds where a coding agent iterates on a single `priority(...)` function per round, then retrains and evaluates an Edge Transformer (ET) end-to-end.
Goal and criteria.
Write a priority function that drives the ET's eval accuracy *down*; lower is better. All else being equal, simpler is better: a small improvement that adds ugly complexity is not worth it; removing code while matching or improving accuracy is a simplification win. Weigh complexity cost against improvement magnitude — a 0.001 gain from twenty new lines of hacky code is rejected; a 0.001 gain from *deleting* code is kept; equal accuracy with simpler code is kept. The first run always establishes the baseline.
Key idea.
One round = one priority function (`priority_fns/round_N.py`). The same function is used to generate `eval.csv` (held-out via seed offset), generate `base_train.csv`, train a new ET on the merged `final_train.csv`, and evaluate the new ET (also backfilling the previous ET on the new round's `eval.csv`). *No internal proposal loop* runs over multiple priority functions inside a round; FunSearch / ProgramDB evolution is intentionally removed.
Per-iteration agent loop.
For round N: (1) *read* previous-round artifacts (metrics, sample data); (2) *decide* a change to `priority(...)` (heuristic tweak, new signal, etc.); (3) *write* `priority_fns/round_N.py`; (4) *run* exactly one round of the pipeline.
Round artifacts.
All outputs live under `RUN_DIR/round_N/`: `best_priority_fn.py` (exact code used), `eval.csv`, `base_train.csv`, `final_train.csv`, `model.pth`, and `eval_summary.csv` (covering this round's eval, the previous round's eval, the base train split, and the merged train split). Run-level `records.pkl` stores per-ET metric dicts including `acc_roundN_eval`, `acc_roundN-1_eval`, `acc_roundN_base_train`, `acc_roundN_final_train`, and `acc_roundN+1_eval` (backfill: previous ET on the new `eval.csv`).
Workflow.
(i) Activate the environment; (ii) create round N's priority file by copying round $N-1$ and editing it; (iii) run one round end-to-end via `scripts/run_multiround.py` with `--init_model_path` pointing at the pretrained ET_0; (iv) read the round's `eval_summary.csv`, `records.pkl`, `eval.csv`, and `best_priority_fn.py`; (v) propose the next priority function. Reusing `--run_dir` naively recreates `round_1`; strict incremental rounds in one run dir would need an explicit resume mode (not implemented).
Parameters and defaults.
`run_multiround.py` reads defaults from a single FullFlow JSON config: `multi_round.*` for training and eval knobs (story counts, ET hyperparameters, merge cap), and `evaluation.*` for base story-generation knobs (entity range, base seed).
Main idea.
The agent is a completely autonomous researcher: try things, keep what works, move on from what doesn't, rewind only very sparingly. *Never stop.* Once the loop has begun, the agent does not pause to ask whether to continue — it does not ask "should I keep going?" or "is this a good stopping point?". The human may be away, and the agent is expected to keep working until the configured number of rounds is complete or it is manually interrupted. If it runs out of ideas, it thinks harder: re-reads referenced papers, re-reads in-scope files for new angles, combines previous near-misses, attempts more radical changes. The loop runs until the human interrupts it.

7.5 GENERATING WORLD RULES USING LLMS

LLMs were prompted to generate world rules that can be used to then create challenging reasoning problems. Our prompt describes the ASP syntax, presents the expected rule structure and provide examples. It also contains a few hints regarding the expected characteristics of challenging sets of rules (e.g. the fact that the rule bases should pose reasoning challenges beyond hierarchies and transitivity). The prompt presented below was iteratively refined by querying several models, namely GPT-5.3, Claude Sonnet 4.6 and Gemini 3.1 Pro. Constraints, such as the minimum number of rules to include in the final output and further format restrictions, were progressively added to the prompt during this exploration step, to ensure that the created worlds are complex enough and the syntax of their rules matches the our samplers' parsers.

We additionally noticed that passing the NoRA rules to the LLMs, after giving these instructions, and asking them to create a new world that is both different from NoRA and more challenging, seemed to lead to more complex and interesting sets of rules. The **Iron Coast** world was obtained with this approach, using Claude Opus 4.6 with our refined prompt.

PROMPT FOR THE GENERATION OF WORLD RULES

Your role is to generate new sets of rules used to construct graphs that pose challenging reasoning problems. Your final response must contain a list of domain base facts and the entire set of created rules.

I'm looking for an interesting rule base that can be used for evaluating the reasoning abilities of AI models. The rule base should be encoded using Answer Set Programming (ASP) syntax. In particular, it can contain the following types of rules.

Facts have the following form.

r(a,b).

This fact states that the relationship r holds between the objects a and b. r(a,a) can also be written r(a).

Next, standard rules have the following form:

r(X,Z) :- u(X,Y), v(Y,Z).

r(X,X) can also be written r(X).

Here X,Y,Z are variables. The rule states that when X is in relation u with Y, and Y is in relation v with Z, then X is in relation r with Z. The number of conditions in a rule can vary. For instance, the following are also valid examples of rules:

r(X,Y) :- s(Y,X).
r(X,Y) :- u(X,Y), v(Y,Z).

In addition to standard rules, we also have choice rules, which have the following form:

{r(X,Y),s(X,Y)} :- u(X,Y).

This choice rule means that whenever X is in relation u with Y, then at least one of r(X,Y) or s(X,Y) should hold. Other examples of choice rules are:

r(X,Y) :- s(Y,X).
r(X,Y) :- u(X,Y), v(Y,Z).

Choice rules are typically used in combination with constraints. These take the following form:

:- r(X,Y), s(X,Y).

This constraint means that r(X,Y) and s(X,Y) cannot be jointly satisfied. Another example is:

:- r(X,Y), s(X,Z), Y!=Z.

This means that r(X,Y), and s(X,Z) cannot both be satisfied when Y is different from Z.

Additional constraints for this project :

- Relations must not have more than two arguments.
- A relation must appear in your final answer only if it is used in more than one rule. Note that we are interested in the creation of rule bases that pose reasoning challenges beyond hierarchies and transitivity. The rules must be connected, and connected in various ways so that the reasoning challenges are diverse.
- Do not use any numerical value in the rules. The following example is not acceptable :

  :- reviewer(R), #count { P : assigned(R, P) } > 2.

- Create at least 40 different rules.
- The rules should be preceded by a list of domain base facts.

These are good examples of rules, that are part of a rule base in the domain of family relations:

maternal_grandmother_of(V,X) :- paternal_grandmother_of(U,X), grandmother_of(V,X), U != V.
is_female(X) :- mother_of(X,Y).
{father(X,Y),mother(X,Y)} :- parent(X,Y).

Please create an extensive rule base which describes some domain of interest and which is likely to be challenging for AI models. Check carefully the consistency of the rules you propose. Your final response must be a complete list of all the rules of the world you created and may contain comments on lines starting with "%".

Table 6: Cross-evaluation of samplers on Iron Coast in terms of exact-match accuracy.

		Test set					
		Cl-Atlas	**Cl-bw**	**Cl-hill**	**Evo1**	**Evo2**	**Evo3**
Training set	**Cl-Atlas**	-	0.01	0.15	0.03	0.00	0.02
	Cl-bw	0.03	-	0.03	0.34	0.20	0.18
	Cl-hill	0.13	0.02	-	0.04	0.00	0.04
	Evo1	0.10	0.26	0.07	-	0.48	0.28
	Evo2	0.03	0.23	0.05	0.34	-	0.25
	Evo3	0.06	0.27	0.06	0.42	0.47	-

Table 7: Cross-evaluation of priority functions obtained through evolutionary search (for NoRA), using three different models: qwen3-coder-next-80b (qw), gpt-oss-120b (gpt) and deepseek-coder-33b (ds). Results are reported in terms of exact-match accuracy. Each model was used in four independent runs.

		Test set											
		qw1	**qw2**	**qw3**	**qw4**	**gpt1**	**gpt2**	**gpt3**	**gpt4**	**ds1**	**ds2**	**ds3**	**ds4**
Training set	**qw1**	0.68	0.50	0.52	0.50	0.51	0.51	0.53	0.49	0.51	0.50	0.48	0.48
	qw2	0.53	0.62	0.54	0.60	0.52	0.51	0.53	0.51	0.51	0.52	0.49	0.49
	qw3	0.52	0.53	0.55	0.54	0.51	0.51	0.53	0.49	0.49	0.49	0.48	0.47
	qw4	0.52	0.55	0.55	0.52	0.49	0.50	0.53	0.49	0.48	0.50	0.48	0.47
	gpt1	0.55	0.52	0.54	0.53	0.65	0.52	0.54	0.52	0.51	0.55	0.50	0.50
	gpt2	0.52	0.50	0.52	0.50	0.51	0.63	0.52	0.51	0.50	0.50	0.49	0.50
	gpt3	0.47	0.39	0.50	0.40	0.41	0.40	0.67	0.40	0.39	0.41	0.41	0.40
	gpt4	0.55	0.52	0.52	0.52	0.55	0.53	0.54	0.59	0.51	0.53	0.48	0.51
	ds1	0.51	0.48	0.49	0.48	0.51	0.50	0.53	0.49	0.50	0.50	0.48	0.46
	ds2	0.45	0.43	0.47	0.44	0.43	0.43	0.51	0.43	0.43	0.45	0.42	0.42
	ds3	0.51	0.47	0.52	0.46	0.46	0.49	0.55	0.47	0.45	0.46	0.47	0.45
	ds4	0.53	0.51	0.53	0.51	0.52	0.51	0.55	0.50	0.51	0.51	0.49	0.52

7.6 EXPERIMENTAL METHODOLOGY

Evolutionary approach: standard runs Before sampling the KG itself, we first randomly choose the number of entities from $\{5, ..., 8\}$ and the ratio of edges to entities from $[2.5, 3]$. We then use the considered sampler to find a KG with these constraints. For the evolutionary search process, we start with an ET which has been trained on 6000 examples. For our main experiments with NoRA, we sample these examples equally from the training, test-D, test-OPEC and test-BL splits. For subsequent rounds, an ET is trained on 6000 examples which are sampled equally from: (i) the training data from the previous round, (ii) the best priority function from the latest round, (iii) test-D, (iv) test-OPEC and (v) test-BL. Further details on the training splits are provided in Section 7.7. We use 10 islands, and for each island $N_{\text{iter}} = 20$ attempts are made in each cycle to improve the priority function. Each round consists of four cycles. We set $N_{\text{evalgraphs}} = 8$. In each step of the sampling process, $N_{\text{cand}} = 20$ candidate edges are scored, with the best-scoring edge being added to the graph.

Evolutionary approach: SUPERET When applying the evolutionary approach to the SUPERET, we make some changes to how the ET is trained after each round, to ensure that the diversity of the initial training data is preserved. In particular, after each round of this process, a new ET is trained on the 16900 original training examples, together with 10400 adversarial examples obtained from the best priority function from the latest round, and 3900 examples obtained from previous rounds (respectively forming 33% and 12.5% of the training data source distribution).

LLM service implementation We have evaluated three LLMs for generating priority functions: deepseek-coder-33b-instruct[8], qwen3-coder-next[9], and gpt-oss-120b[10]. All three models were downloaded from HuggingFace[11]. In our experiments, the LLMs are served on a single GPU or on multiple GPUs using the vLLM library[12]. They receive OpenAI API requests sent from the FunSearch algorithm workers, during the evolutionary cycles steps, at each round. Since we found in our experiments that the results are largely insensitive to the chosen model, we believe that experiments with larger LLMs would not justify the cost. In particular, the size of the current models allows us to keep the computation time reasonable (about 40 hours per 5-round run in the main experimental setting of this work), while setting the model context length to a size that leaves enough space for the generation of long priority functions. In particular, deepseek-coder-33b-instruct has a 16K maximum model length. For qwen3-coder-next and gpt-oss-120b, we set the maximal context length to 32K because they tend to generate longer outputs.

7.7 TRAINING DATA MIX STRATEGY

We provide further details on the training data mix strategy that was used for the evolutionary runs, including the Auto-research experiment. Note that, as explained in Section 7.6, the SUPERET was trained with a different strategy. At every round i, three types of data contribute to the training set.

1. **Priority-generated queries** The priority function discovered in round i is used to generate a fresh set of NoRA worlds with 5–8 entities. For each generated world, C candidate facts are scored by the priority function to serve as queries. In our experiments we generate 60 stories per round, yielding roughly $60 \times C$ queries.
2. **Historical data** The training data from previous rounds (for $i > 0$) is retained, so the model still has access to previously discovered hard problem instances.
3. **External datasets** A fixed collection of external training sets are obtained from the original NoRA dataset, in particular the train split and the `test-BL`, `test-D` and `test-OPEC` splits.

The training data for the ET in each round is capped to 6000 queries. To prevent any single source from dominating the training signal, these queries are sampled uniformly from each of the different

[8] `https://huggingface.co/deepseek-ai/deepseek-coder-33b-instruct`

[9] urlhttps://huggingface.co/Qwen/Qwen3-Coder-Next

[10] `https://huggingface.co/openai/gpt-oss-120b`

[11] `https://huggingface.co/`

[12] `https://vllm.ai/`

sources. All sampling uses a fixed random seed for reproducibility. For the first round, only the external datasets are used for training.

7.8 Auto-Research Agent Experiment

To test whether the difficulty signals discovered by FunSearch are artefacts of its particular evolutionary search procedure, we ran an independent replication using a single LLM agent (Claude) as an autonomous researcher. The agent iterates over priority functions one at a time removing every structural bias introduced by FunSearch.

7.8.1 Setup

The agent operates in a sequential outer loop over *rounds*. At round r it:

1. reads the evaluation artifacts from round $r-1$ (accuracy table, sample evaluation instances, the previous priority function (whilst all the rest are also in its context));
2. proposes a single new priority function;
3. generates training and evaluation data using the priority function;
4. trains a fresh Edge Transformer on the merged training set and evaluates it.

The same data mix, merge procedure, and ET hyperparameters described previously are used. The initial model is ET_0, pretrained on the baseline data with no adversarial instances.

7.8.2 Auto-research recovered Difficulty Signals

The priority functions discovered by the agent converged on four structural signals, each independently rediscovering axes found by the FunSearch runs:

1. **Maternal/paternal predicates.** Facts triggering gender-qualified extended-family relations (e.g. `maternal_grandmother_of`, `paternal_uncle_of`) receive the highest weight ($12\times$ in the final function). These require multi-step reasoning that chains a gender constraint through an intermediate relationship.
2. **Multi-hop relations.** Grandparent, aunt/uncle, nibling, and in-law predicates, all requiring ≥ 2 inference steps, are weighted $5\times$.
3. **Entity-pair relationship density.** The agent discovered that entity pairs connected by ≥ 3 distinct relations produce instances the ET struggles with. This targets multi-label prediction difficulty rather than inference depth.
4. **Self-referential fact suppression** Unary property facts (same source and target entity, e.g. `is_male(3,3)`) are penalised, forcing the generator to select relational facts that create richer graph structure.

These signals align with those found by FunSearch, despite the agent using no evolutionary operators, no population diversity, and a single sequential pass.

7.9 Training the Edge Transformer

The training recipe is identical for the ET across all scenarios and is outlined in the following sections.

Multi-label binary cross entropy We use a multi-label version of the Binary Cross Entropy (BCE) loss for the multi-label classification setting for all problems. The logits for each class are transformed using a sigmoid function and then the problem is treated as a binary classification problem with a multi-hot target binary vector.

$$\mathcal{L}_{\text{BCE}} = \sum_{i \in \mathcal{D}, j \in \mathcal{R}} \text{CE}(\sigma(x_{ij}), y_{ij})$$

where i is the sample index, j is the relation index, x_{ij} is the predicted logit and the y_{ij} is the one-hot target class label.

Table 8: Priority functions learned through evolutionary search and the auto-research agent on Iron Coast, with ET as the evaluator, transfer to other architectures. The table shows results for queries generated by a random priority function and queries from the discovered priority functions.

	Random	Auto-research	Evo1	Evo2	Evo3
EpiGNN	0.50	0.19	0.11	0.16	0.18
NBFNet	0.59	0.21	0.18	0.34	0.31
R-GCN	0.50	0.17	0.09	0.19	0.11
NCRL	0.65	0.24	0.41	0.65	0.53

Initialization and compute All trainable parameters for the models are uniformly initialized. All baseline results that were obtained by us were hyperparameter-tuned using grid search, as detailed below. All experiments were conducted using RTX 4090, GH200, and A6000 GPUs. A single training cycle (multipe times per evolutionary round) of the ET can be conducted within a few hours on a single GPU. This includes training and testing a single model on any evaluation split.

Hyperparameter settings We use the Adam optimizer (Kingma & Ba, 2017). All the models were hyperparameter tuned using an economical grid search over key parameters. For ET, a grid search was performed over the number of attention heads, hidden dimension size, the number of message passing rounds, and dropout rate. All the optimal hyperparameters are available in the companion code with the manuscript.

Training Protocol Each round trains a fresh Edge Transformer from scratch on $\mathcal{D}_{\text{train}}$. The data is split $80/20$ into training and validation partitions via a random split. We use the Adam optimiser with learning rate 10^{-3}, batch size 32, and train for 100 epochs with gradient clipping. Validation accuracy is checked every 10 epochs.

8 Additional Experimental Analysis

8.1 Evaluation on Iron Coast

Table 6 repeats this analysis for the Iron Coast domain. For this experiment, we initialized the evolutionary runs with an ET that was trained on examples generated by the Cl-motif sampler. We compare three other generic Claude-generated samplers with the priority functions that were obtained in three independent runs of the evolutionary method. The results clearly shows that challenging queries can be constructed for the Iron Coast domain, which demonstrates the feasibility of a fully automatic evaluation setup, where the world rules, constraints and samplers are all LLM-generated. The priority functions from the three evolutionary runs appear again similar (with models performing similarity on Evo1–Evo3, regardless of which evolutionary priority function was used for training). The three generic Claude-generated samplers all manage to generate challenging problems.

8.2 Comparison of different LLMs

In Table 7, we present a cross-evaluation of the evolutionary priority functions obtained with three different LLMs, where we completed four independent runs for each LLM. The results show that quality of the priority function remains stable across the three models, with the largest model (gpt-oss-120b) performing on par with a much smaller model (deepseek-coder-33b).

8.3 Analysis of transferability

Tables 8 and 9 provide further analysis on the transferability of the learned priority functions. Table 8 shows that priority functions learned for Ivory Coast, with ET as the evaluator, generate problems that are also difficult for other neural reasoning architectures. This complements a similar analysis that was reported for NoRA in Table 4. Finally, Table 9 shows results for priority functions that were learned with NCRL as the evaluator, instead of the ET. We again find that the discovered priority functions generate problems that are hard for other architectures.

Table 9: Priority functions learned through evolutionary search and the auto-research agent on NoRA, with NCRL as the evaluator, transfer to other architectures. The table shows results for queries generated by a random priority function and queries from the discovered priority functions using Funsearch (Evo) and the Auto-research agent.

	Random	Evo	Auto-research
ET	0.68	0.40	0.30
EpiGNN	0.65	0.13	0.02
NBFNet	0.30	0.08	0.03
R-GCN	0.48	0.08	0.00

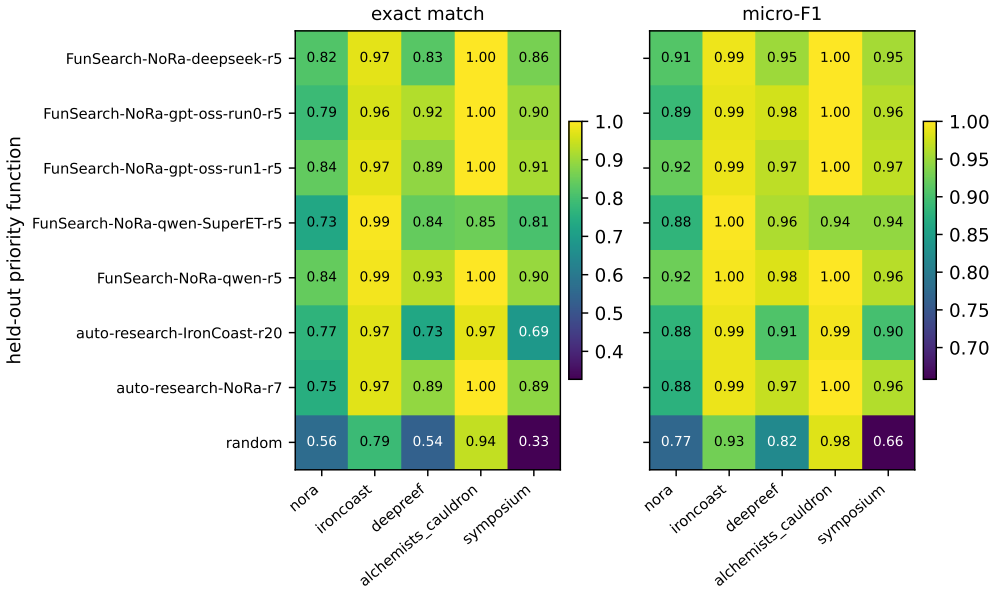


Figure 4: Results of an experiment where we train the edge transformer on 7 out of 8 considered generic priority functions, for 6 different sets of world rules. We then test the model on worlds sampled by the held-out priority function.

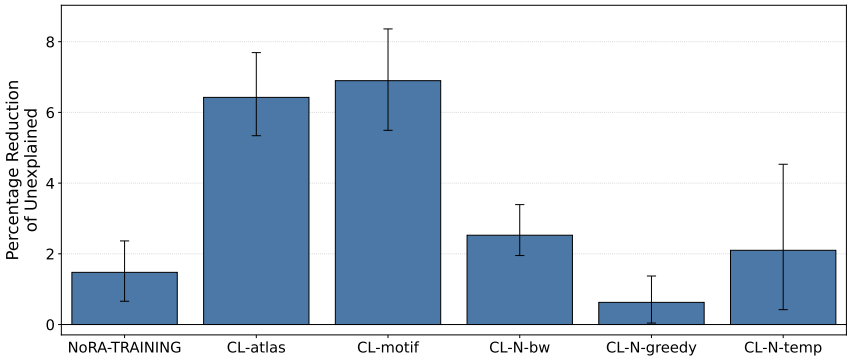


Figure 5: Analysis of percentage reduction in unexplained difficulty when incorporating entailed off-path edges (using queries sampled with priority functions obtained at the end of various standard evolutionary runs).

Figure 4 shows an analysis for eight different generic priority functions: a random priority function, six priority functions discovered with NoRA (5 different configurations of FunSearch and 1 configuration of autoresearch) and one discovered with Iron Coast (autoresearch). For each experiment reported in the figure, we trained an ET on 7 out of 8 priority function and tested the resulting model on worlds sampled with the held-out priority function. In addition to NoRA and IronCoast, we evaluated on three further worlds obtained with Claude (in the same way as IronCoast). The results are overall relatively high, which suggests that the different generic priority functions have mostly discovered similar types of difficulty. We recall that many of the priority functions discovered through FunSearch use strategies that are tailored towards the considered rule base, allowing them to identify other sources of difficulty. Interestingly, in Figure 4, the worst results are obtained when the random priority function is omitted from training. This shows that, while the problems generated by the random priority function are intuitively the easiest, the ET still struggles on these problems when they are not included in the training distribution. Another interesting observation is that omitting the autoresearch priority function obtained from Iron Coast mostly leads to worse results than omitting the autoresearch priority function obtained from NoRA. Since the IronCoast rule base is intuitively easier, finding challenging priority functions is harder, which seems to result in stronger transferability. We can also see that omitting the priority function discovered with the SuperET has a larger impact than omitting the other FunSearch priority functions, which is in line with our findings from the main paper that this priority function generates more challenging problem instances.

8.4 Analysis of problem difficulty

Given a trained ET and a set of queries, we train a classifier to predict whether a given query will be answered correctly by the ET. The features used for classification are the inference depth, OPEC and BL of the query. As the classifier, we use a Gradient Boosting Classifier. We used the scikit-learn implementation (sklearn.ensemble.HistGradientBoostingClassifier) with the following hyperparameters: max_depth = 3, n_estimators = 100 and learning_rate = 0.1. Each fitted tree model is trained for 100 boosting iterations using cross-entropy.

For the analysis of problem difficulty, all ET models are first calibrated for 5 epochs on 6000 examples sampled from test-D, test-OPEC and test-BL. This calibration step is intensionally limited, so the unique characteristics of the models are maintained. The aim of this calibration step is to ensure that all models have had some exposure to problems with higher reasoning depth, OPEC and BL values, as our focus is on exploring what forms of difficulty remain beyond these three difficulty metrics.

Impact of entailed off-path edges When inspecting the generated problem instances from the evolutionary samplers, we identified a class of queries that are difficult because the derivation relies on inferred off-path edges. Figure 1 shows an example of such a query, where the *is_female* loop is an off-path edge. However, since this edge is inferred, the OPEC value of this query is 0. Despite only involving 5 entities, all models included in the analysis from Figure 5 misclassified this example. More generally, we found that incorporating the number of such inferred off-path edges can reduce

the value of the *Unexplained* score. Specifically, let us defined the **Off-Path Entailed Edge Count (OPEEC)** as the number of entailed off-path edges that are used to derive the answer. Figure 5 shows the percentage reduction in the *Unexplained* score when adding OPEEC as a feature, for a number of different models evaluated on test queries obtained by combining, for each evolutionary run, the queries sampled using the priority function produced at the end of round 5. Specifically, the percentage reduction metric, reported in Figure 5, is defined as

$$\textit{Reduction} = 100 \cdot \frac{\textit{Unexplained}_{\textit{base}} - \textit{Unexplained}_{\textit{aug}}}{\textit{Unexplained}_{\textit{base}}}$$

where $\textit{Unexplained}_{\textit{base}}$ is the *Unexplained* score, as defined by (2), for the classifier that only incorporates inference depth, OPEC and BL; $\textit{Unexplained}_{\textit{aug}}$ is the corresponding value for the augmented classifier, which additionally incorporates the number of entailed off-path edges. We report the median value of *Reduction* that was found across 100 paired bootstrap samples (where each bootstrap sample has the same size as the ET training set for that model). To test for statistical significance, we used a one-sided empirical paired bootstrap test, where the null hypothesis is that *Reduction* ≤ 0. The p-value is computed as

$$p = \frac{1 + \text{number of bootstrap samples with } \textit{Reduction} \leq 0}{1 + 100}$$

Figure 5 shows the value of *Reduction* for some of the models. As can be seen, the values are positive for all tested models. The fact that the reduction is positive is statistically significant for all models ($p < 0.05$).

8.5 Analysis of the evolved priority functions

We analyze the best evolved priority functions retained at the end of each evolutionary round using abstract syntax tree (AST) tokens. For each priority function, we parse the source with the standard Python `ast` module and extract four kinds of pseudo-tokens: *calls*, *attribute accesses*, *operators* and *imports*. The same parser also records structural metrics: the number of AST nodes, the maximum nesting depth, the number of `Call` nodes, the number of `Compare` nodes, the number of branching nodes (`If`, `For`, `While`, `Try`, etc.), and the number of distinct tokens. Across Evo1–Evo4, this yields 20 priority functions and 158 distinct raw tokens, giving a 20×158 token-count matrix.

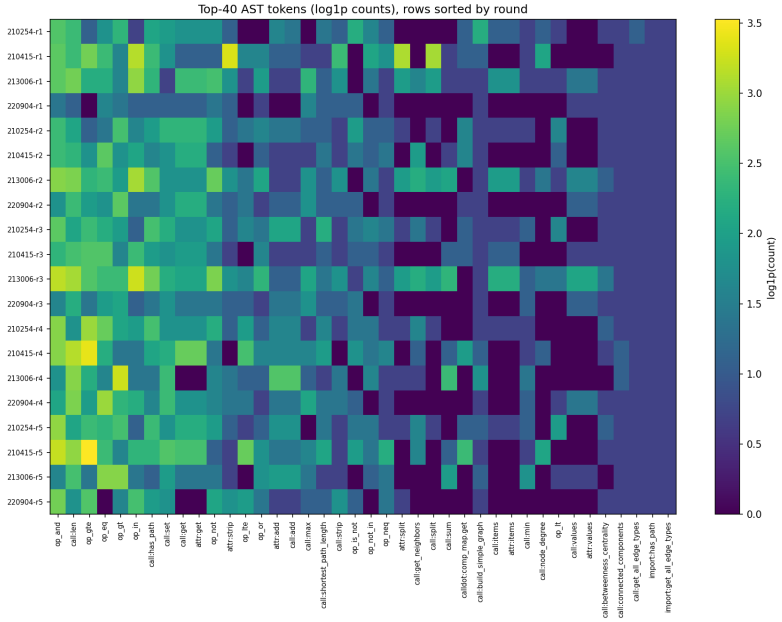
Figure 6a shows the $\log(1 + \text{count})$ abundance of the top-40 most frequent tokens, with rows ordered by round and grouped by run. The dominant columns are logical and relational operators; the graph related tokens (in the right hand side of the heat-map) are sparse, indicating occasional spikes in graph-related queries by individual runs. Centering the token matrix column-wise and taking its SVD gives us a low-dimensional structure in token usage (Figure 6b): the first two singular components together explain roughly 55% of the variance, and the first seven explain at least 90%. Figure 6c then projects the evolution of priority functions onto the first five principal components and traces the trajectory of each run across rounds.

Figure 7 shows an analysis of how much the priority function changes throughout the evolutionary process. For this analysis, we compute the syntactic similarity between the best priority function obtained at the end of each cycle and (i) the priority function obtained in the first cycle and (ii) the priority function obtained in the previous cycle. Syntactic similarity is measured in terms of ROUGE-L. We can see that large changes are infrequent and may occur in different phases of the evolutionary process.

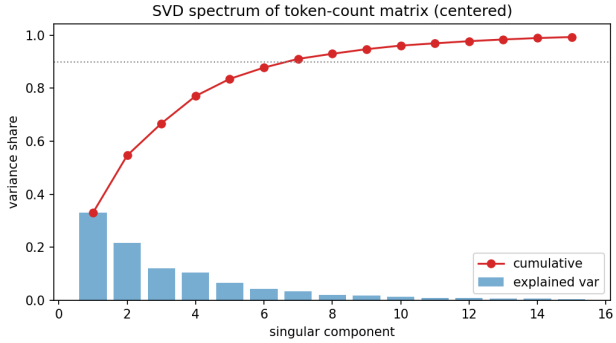
Figure 8 shows how the complexity of the priority function evolves, estimated by its normalized compression distance $\mathrm{NCD}(x, y) = \frac{C(xy) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}}$ where $C(\cdot)$ is the compressed length and xy denotes concatenation.

8.6 Analysis of graphs generated by the evolution process

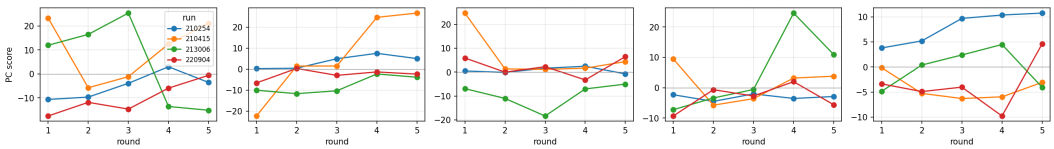
In this section, we perform an exploratory analysis of the knowledge graphs produced by the priority functions from the evolutionary runs (Evo1–Evo4) and the Claude auto-research protocol. Specifically, given that the graph size remains fixed, we investigate how density, path lengths, and the number of parallel edges evolve across rounds, making reasoning a difficult task for the edge transformer.



(a)



(b)



(c)

Figure 6: (a) AST token heatmap: $\log(1+\text{count})$ for the top-40 tokens across the 20 priority functions. (b) SVD spectrum of the column-centered token-count matrix: per-component variance share (bars) and cumulative share (line). The horizontal line marks 90% explained variance. (c) Trajectory of each Evolutionary Search run (Evo1–Evo4) in the PC_1–PC_5 space across rounds.

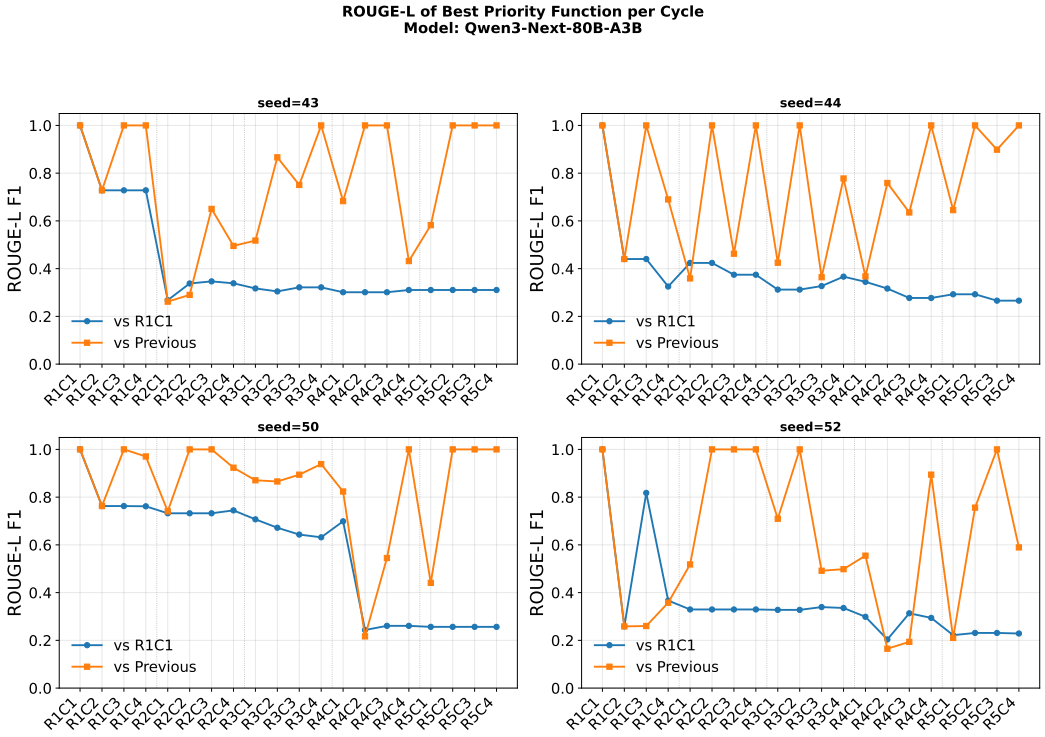


Figure 7: Analysis of the syntactic similarity between the evolved priority functions and the initial one (blue) and the priority function from the previous cycle (orange), measured in terms of ROUGE-L.

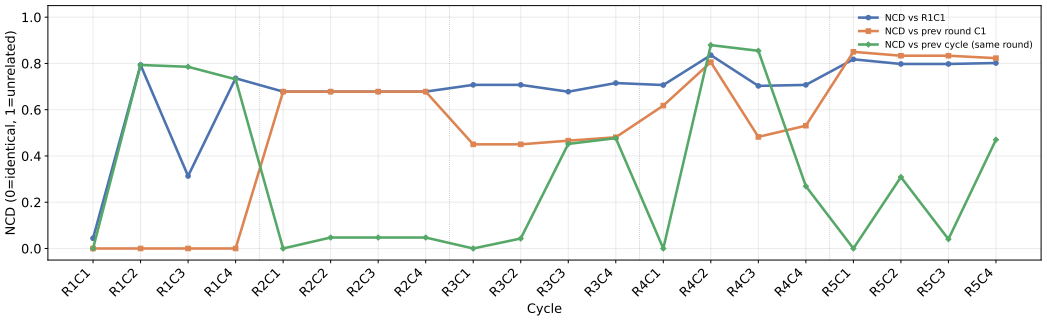


Figure 8: Analysis of the Normalized Compression Distance of the learned priority functions at the current cycle with respect various reference point priority functions.

We test 20 graph connectivity features (derived using `NetworkX`), which can be broadly classified into global structure, source-target reasoning paths, spectral/topological connectivity, and number of possible query labels. We note that evolutionary process has the highest jump (increase/decrease) in most graph features from round zero to round one (see Figures 10 and 11) for both sampling methods. The feature edge label entropy (diversity of relational labels between two nodes), parallel edge ratio (fraction of fact-edges that are redundant in the sense that some other relation already connects the same node pair), and the number of potential query labels shows consistent increase from the baseline (Figures 12 and 13).

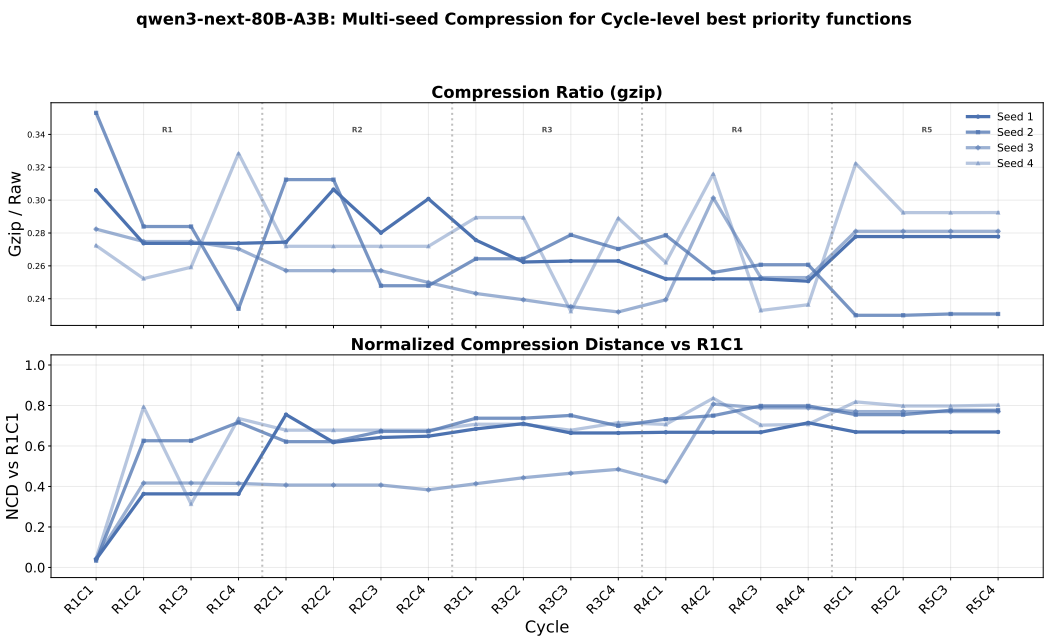


Figure 9: Analysis of the Normalized Compression Distance (NCD) and compression ratio of the learned priority functions at the current cycle with respect to the best priority function at the end of round 1 for multiple seeds. This is a heuristic proxy for Kolmogorov complexity and again confirms what ROUGE-L evolution suggests that syntactically, the priority functions are changing less with respect to their states at the end of round 1.

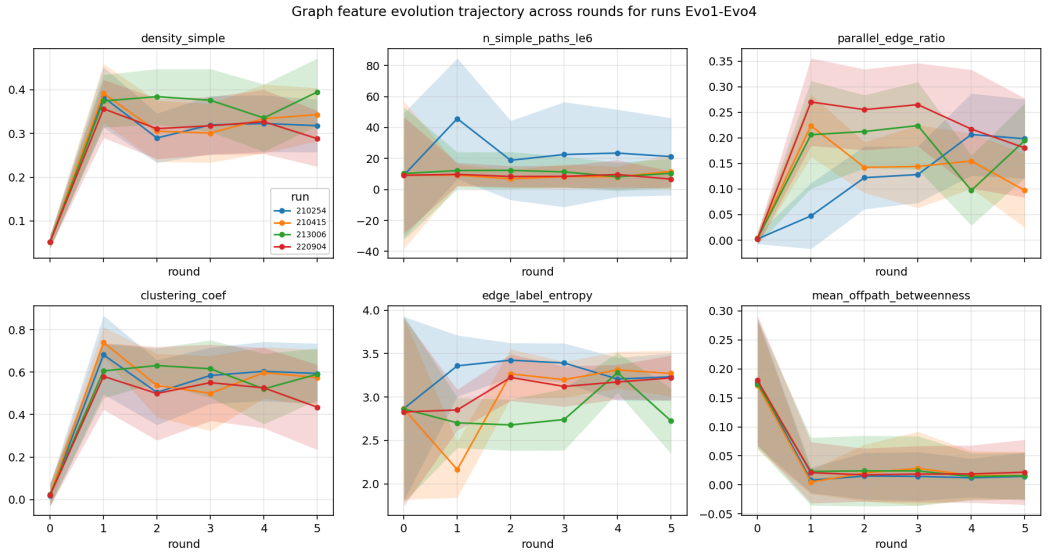


Figure 10: EVOLUTIONARY SEARCH: Evolution of mean graph features, across 5 rounds, for Funsearch runs Evo1-Evo4.

Graph feature trajectories and histogram summaries for the two sampler families are shown in Figures 10, 11, 12, 13. Both sampling methods show increases in the number of parallel edges and edge-label entropy, increasing ambiguity for the edge transformer.

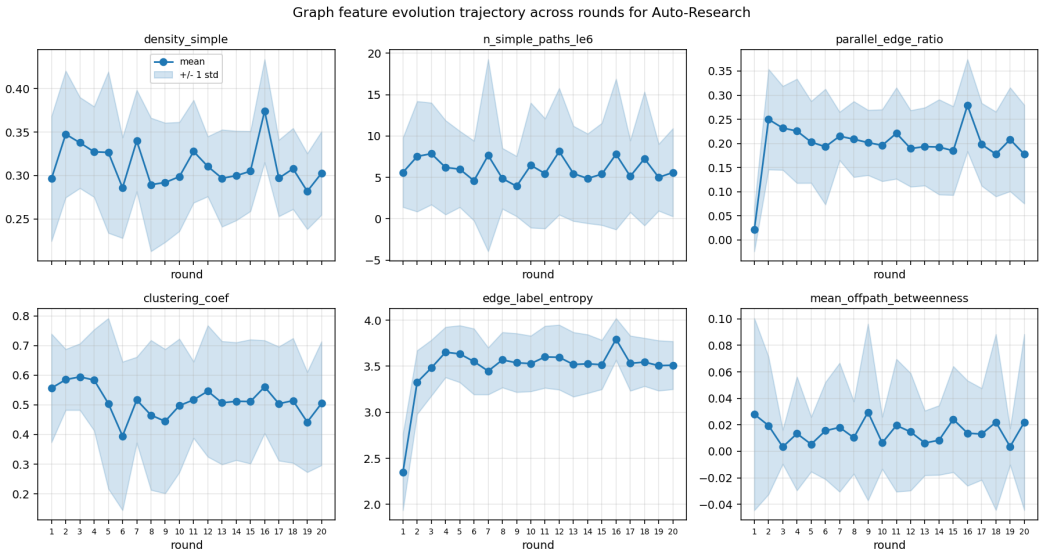


Figure 11: AUTO-RESEARCH: Evolution of mean graph features, across 20 rounds.

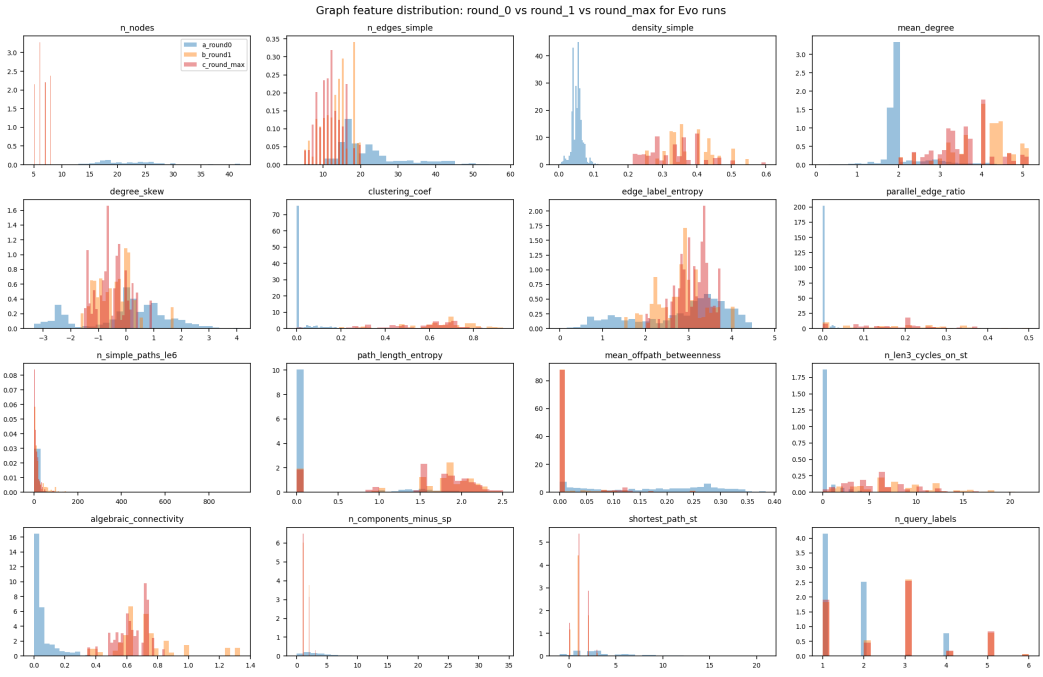


Figure 12: EVOLUTIONARY SEARCH: histograms of selected graph features for round 0, round 1 and round 5.

8.7 EXAMPLE PRIORITY FUNCTIONS

In this section we present the best priority functions generated by all the approaches used in this paper. The FunSearch function is from a Qwen3-80B-A3B model run for 5 rounds with 4 cycles. Note that there are general and world specific features in all the functions. The Auto-Research functions are much more compact and compositional in nature. For instance, round 7 has mostly general features whereas round 20 is more world specific but also contain some features from round 7.

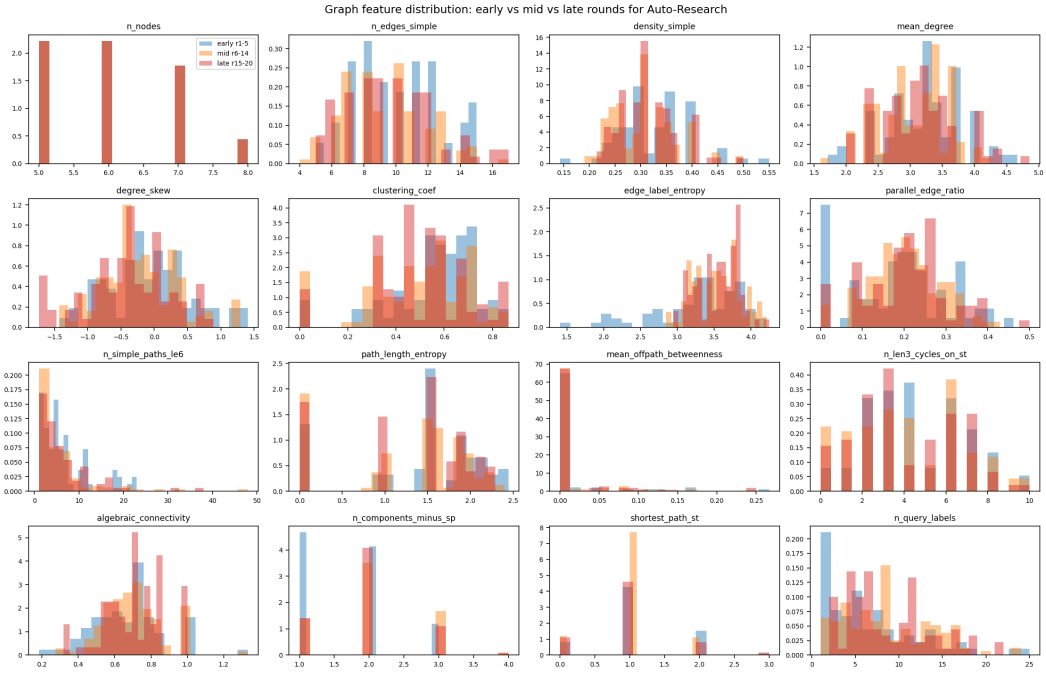


Figure 13: AUTO-RESEARCH: histograms of selected features for early (r=1–5), mid (r=6–14) and late (r=15–20) phases.

ROUND 5 CYCLE 4 FUNSEARCH (BEST PRIORITY FUNCTION)

```
def priority(cand_fact: str, definite_rules_program: str, entailed_facts: str,
     facts_program: str) -> float:
    """Improved version of `priority` that adds scoring for entailed edges that create
     multi-hop cycles and penalizes redundant edge types."""
    from Funsearch.GraphWranglingMethods import betweenness_centrality,
     build_simple_graph, connected_components, count_paths, get_all_edge_types, has_path,
      num_nodes, shortest_path_length
    import re

    # Parse the candidate edge
    try:
        cand_fact_stripped = cand_fact.strip()
        match = re.match(r'^(\w+)\s*\(\s*(\d+)\s*,\s*(\d+)\s*\)\s*\.\s*$',
     cand_fact_stripped)
        if not match:
            return 0.0
        relation, source, target = match.groups()
    except Exception:
        return 0.0

    # Parse the entailed facts into edges
    try:
        entailed_lines = [l.strip() for l in entailed_facts.strip().split('\n') if l.
     strip() and not l.startswith('#')]
    except Exception:
        entailed_lines = []

    if not entailed_lines:
        return 0.0

    # Basic score: number of entailed edges with base multiplier
    score = len(entailed_lines) * 0.35

    # Relation type weights (tuned for fine-grained differentiation)
    if relation in ('parent_of', 'sibling_of', 'spouse_of'):
        score += 1.6
    if relation in ('grandparent_of', 'uncle_of', 'aunt_of', 'nibling_of'):
        score += 0.85
    if relation in ('living_in_same_place', 'school_mates_with', 'colleague_of'):
        score += 0.55

    # Structural analysis
    try:
        G = build_simple_graph(facts_program)
        betweenness = betweenness_centrality(G)
        original_edge_types = get_all_edge_types(G)
        edge_types_seen = set()
        long_range_bonus = 0.0
        bridge_bonus = 0.0
        new_reach_bonus = 0.0
        chain_potential_bonus = 0.0
        indirect_bonus = 0.0
        constraint_bonus = 0.0
        multi_hop_cycle_bonus = 0.0
        redundancy_penalty = 0.0

        n = num_nodes(G)

        # Analyze each entailed edge
        for line in entailed_lines:
            try:
                em = re.match(r'^(\w+)\s*\(\s*(\d+)\s*,\s*(\d+)\s*\)\s*\.\s*$', line.
     strip())
                if not em:
                    continue
                rel_name, src, tgt = em.groups()
                edge_types_seen.add(rel_name)

                # Enhanced novel reachability scoring
                if not has_path(G, src, tgt):
                    score += 2.3
                    new_reach_bonus += 0.5 * (n / max(n, 4))
                else:
                    pl = shortest_path_length(G, src, tgt)
                    if pl is not None:
                        if pl >= 6:
                            long_range_bonus += 3.0
```

ROUND 5 CYCLE 4 FUNSEARCH (BEST PRIORITY FUNCTION) (CONTINUED)

```
                    elif pl >= 5:
                        long_range_bonus += 2.0
                    elif pl >= 4:
                        long_range_bonus += 1.3
                    elif pl >= 3:
                        long_range_bonus += 0.7
            # Bridge detection with stricter thresholds
            if src in betweenness and tgt in betweenness:
                avg_bc = (betweenness[src] + betweenness[tgt]) / 2
                if not has_path(G, src, tgt) and avg_bc > 0.14:
                    bridge_bonus += 2.6
                elif avg_bc > 0.10:
                    bridge_bonus += 0.6

            # Novelty scoring with tighter penalty/bonus balance
            if rel_name in original_edge_types:
                score -= 0.25
                redundancy_penalty += 0.1  # Track redundant types for bonus/penalty
 calculation
            else:
                score += 0.75

            # Shortcut detection (counting paths up to 5 hops for better sensitivity
)
            paths_count = count_paths(G, src, tgt, cutoff=5)
            if paths_count >= 2 and has_path(G, src, tgt):
                score += 0.6

            # Constraint sensitivity: stronger bonus for distinct nodes
            if src != tgt:
                constraint_bonus += 0.2

            # Chain potential bonus: slightly increased weight
            chain_relations = {'child_of', 'parent_of', 'grandparent_of', '
sibling_of', 'spouse_of'}
            if rel_name in chain_relations:
                chain_potential_bonus += 0.28

            # Indirect edge bonus: detect if entailed edge has high betweenness
endpoints
            try:
                if src in betweenness and tgt in betweenness:
                    bc_product = betweenness[src] * betweenness[tgt]
                    if bc_product > 0.006 and not has_path(G, src, tgt):
                        indirect_bonus += 0.35
            except:
                pass

            # Multi-hop cycle detection: detect cycles with path length >= 2
            if src != tgt and has_path(G, src, tgt):
                path_len_to_tgt = shortest_path_length(G, src, tgt)
                if path_len_to_tgt is not None and path_len_to_tgt >= 2:
                    if has_path(G, tgt, src):
                        multi_hop_cycle_bonus += 1.1 * (path_len_to_tgt / max(
path_len_to_tgt, 2))

        except Exception:
            continue

    # Add bonuses with optimized weights
    score += bridge_bonus * 1.5
    score += long_range_bonus * 1.1
    score += new_reach_bonus * 1.2
    score += chain_potential_bonus * 1.8
    score += indirect_bonus * 1.4
    score += constraint_bonus * 1.5
    score += multi_hop_cycle_bonus * 1.3  # NEW: emphasize multi-hop cycles more
strongly

    # Diversity bonus: slightly stronger nonlinearity and cap the bonus
    score += 1.0 * (len(edge_types_seen) ** 0.82)

    # Redundancy penalty scaling
    score -= redundancy_penalty * 0.4

    # Bridge candidate analysis with refined degree criteria
    try:
```

ROUND 5 CYCLE 4 FUNSEARCH (BEST PRIORITY FUNCTION) (CONTINUED)

```
            src_deg = len(set(G.successors(source)) | set(G.predecessors(source)))
            tgt_deg = len(set(G.successors(target)) | set(G.predecessors(target)))

            # Prefer asymmetric connections (low-degree to high-degree)
            if (src_deg <= 1 and tgt_deg >= 5) or (tgt_deg <= 1 and src_deg >= 5):
                score += 1.5
            elif (src_deg <= 2 and tgt_deg >= 4) or (tgt_deg <= 2 and src_deg >= 4):
                score += 0.85
            elif (src_deg <= 1 and tgt_deg >= 4) or (tgt_deg <= 1 and src_deg >= 4):
                score += 0.75

            # Bonus for connecting components
            components = list(connected_components(G))
            if len(components) > 1:
                src_comp = next((c for c in components if source in c), None)
                tgt_comp = next((c for c in components if target in c), None)
                if src_comp and tgt_comp and src_comp != tgt_comp:
                    score += 2.6
        except Exception:
            pass

    except Exception:
        pass  # Fall back to basic score if graph building fails

    return float(score)
```

ROUND 7 AUTO-RESEARCH (BEST PRIORITY FUNCTION)

```
def priority(cand_fact: str, definite_rules_program: str, entailed_facts: str,
     facts_program: str) -> float:
    if not entailed_facts.strip():
        return 0.0

    cand_rel = cand_fact.strip().rstrip('.').split('(')[0] if '(' in cand_fact else ''
    if cand_rel.startswith('not_'):
        return -100.0

    lines = entailed_facts.strip().split('\n')

    # Parse existing entity pairs from the story
    existing_pairs = {}
    for line in facts_program.strip().split('\n'):
        line = line.strip().rstrip('.')
        if '(' in line:
            rel = line.split('(')[0]
            args = line.split('(')[1].rstrip(')').split(',')
            if len(args) == 2:
                pair = (args[0].strip(), args[1].strip())
                existing_pairs.setdefault(pair, set()).add(rel)

    # Score entailed facts: prefer those that ADD relations to existing pairs
    score = 0.0
    entailed_rels = set()
    overlap_count = 0
    for line in lines:
        line = line.strip().rstrip('.')
        if '(' in line:
            rel = line.split('(')[0]
            if rel.startswith('not_'):
                continue
            entailed_rels.add(rel)
            args = line.split('(')[1].rstrip(')').split(',')
            if len(args) == 2:
                pair = (args[0].strip(), args[1].strip())
                if pair in existing_pairs:
                    overlap_count += 1  # adds complexity to existing pairs

    # Prefer: many entailed, many overlapping pairs, diverse relations
    score = float(len(lines)) + overlap_count * 5.0 + len(entailed_rels) * 3.0
    return score
```

ROUND 20 AUTO-RESEARCH (BEST PRIORITY FUNCTION)

```python
def priority(cand_fact: str, definite_rules_program: str, entailed_facts: str,
     facts_program: str) -> float:
    if not entailed_facts.strip():
        return -10.0

    cand_rel = cand_fact.strip().rstrip('.').split('(')[0] if '(' in cand_fact else ''
    if cand_rel.startswith('not_'):
        return -100.0

    # Expanded trivial set: spatial + attributes + non-family social
    trivial = {'living_in', 'living_in_same_place', 'is_person', 'is_place',
               'is_male', 'is_female', 'is_underage', 'no_brothers', 'no_sisters',
               'no_sons', 'no_daughters', 'colleague_of', 'school_mates_with'}

    multi_hop = {'grandmother_of', 'grandfather_of', 'granddaughter_of', 'grandson_of',
                 'maternal_grandmother_of', 'maternal_grandfather_of',
                 'paternal_grandmother_of', 'paternal_grandfather_of',
                 'aunt_of', 'uncle_of', 'niece_of', 'nephew_of',
                 'maternal_aunt_or_uncle_of', 'paternal_aunt_or_uncle_of',
                 'maternal_uncle_of', 'paternal_uncle_of',
                 'great_grandparent_of', 'great_grandchild_of',
                 'sister_in_law_of', 'brother_in_law_of',
                 'father_in_law_of', 'mother_in_law_of',
                 'son_in_law_of', 'daughter_in_law_of'}

    lines = entailed_facts.strip().split('\n')

    existing_rels = set()
    for line in facts_program.strip().split('\n'):
        line = line.strip().rstrip('.')
        if '(' in line:
            existing_rels.add(line.split('(')[0])

    score = 0.0
    entailed_rels = set()
    multi_hop_count = 0
    for line in lines:
        line = line.strip().rstrip('.')
        if '(' in line:
            rel = line.split('(')[0]
            if rel.startswith('not_'):
                continue
            entailed_rels.add(rel)
            if rel in trivial:
                score -= 2.0
            elif rel in multi_hop:
                score += 20.0
                multi_hop_count += 1
            else:
                score += 1.0

    novel_rels = entailed_rels - existing_rels - trivial
    score += len(novel_rels) * 10.0
    non_trivial = entailed_rels - trivial
    score += len(non_trivial) * 3.0
    score += multi_hop_count * 8.0

    return score
```