

Bridging Relational Learning Paradigms, Relational Learning over Microbiological data

Introducing popper2rdf and rdf2popper Python libraries, presenting two case studies using KG-Microbe and KG-Covid19 knowledge graphs

Prague University of Economics and Business

Presentation outline

- Bridging Inductive Logic Programming and Rule Learning over Knowledge Graphs
 - popper2rdf
 - rdf2popper
 - Transformation success evaluation
 - WN18RR experiment – performance comparison (Popper vs. RDFRules)
- Case studies (RDF Rule Learning)
 - KG-Microbe
 - Building classifier to predict media
 - KG-COVID19
 - 2 tasks on finding interactions between entities
- Conclusion

Different formats, different methods, same results?

- Inductive Logic Programming (systems like Aleph, Popper) → highly expressive but at times struggles with scalability and often requires negative examples (**Closed World Assumption**),
- Rule Learning over Knowledge Graphs (e.g., AMIE+, RDFRules) → scalable and designed to work without negative examples, but lacks ILP's expressivity (**Open World Assumption**)

```
size(piece_a, 4.7).  
fact/ground atom
```

```
<pieceA> <hasSize> "4.7".
```

RDF (Resource Description
Framework)
in N-triples format
(W3C
Recommendation)

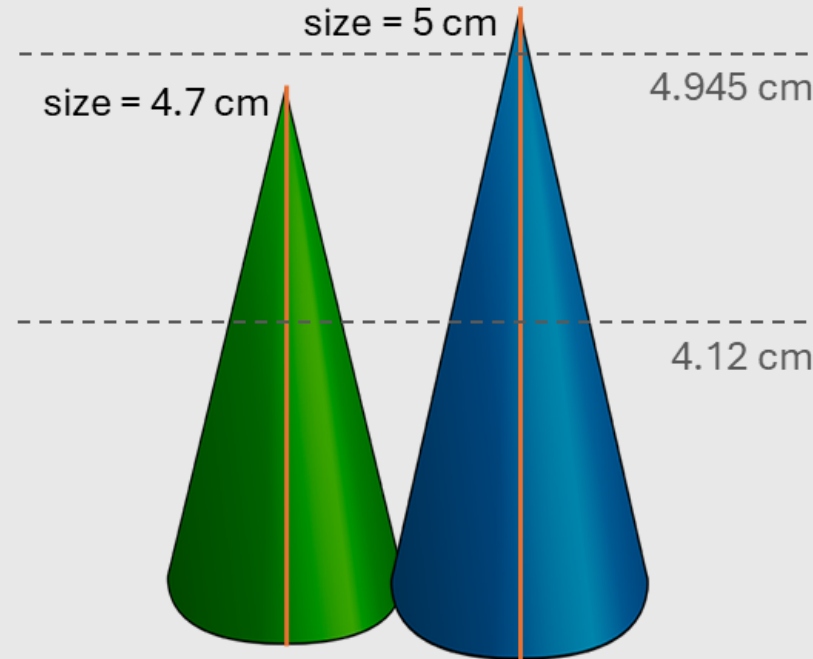
numeric-zendo1 example

RDFRules output

```
( ?c <size#discretized_level_1>
[ 4.945 ; 9.94 ] ) ^
( ?c <contact> ?b ) ^
( ?a <piece> ?b )
⇒ ( ?a <zendo> true )
```

Popper (numsynth-aaai23) output

```
zendo(A): piece(A,B),
contact(B,C),
size(C,D),geq(D,4.12).
```



example of a game configuration

popper2rdf - from Prolog atoms to RDF triples

- Transforms Prolog atoms (currently supports Popper datasets) into RDF triples – these express binary relationships

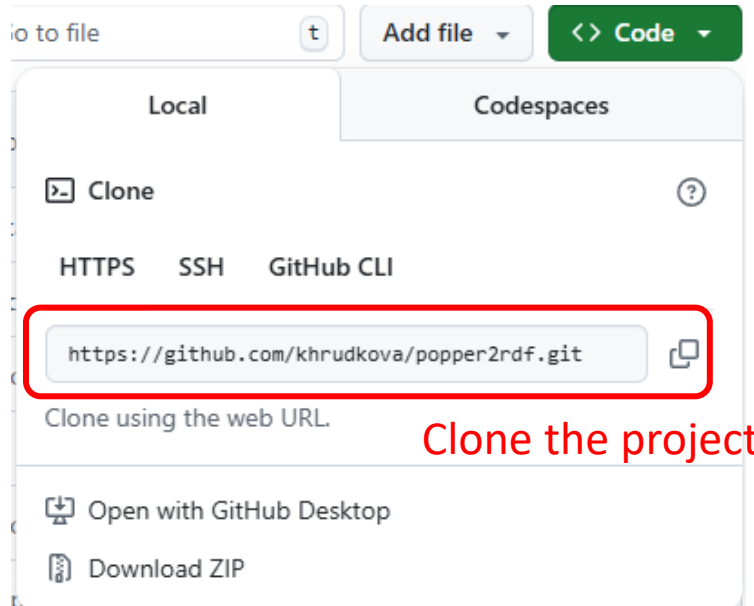
```
<pieceA> <hasSize> "4.7".  
hasSize(PieceA, „4.7“)
```

- Key challenges:
 - Unary atoms - mapped using a generic has_property predicate.
 - N-ary predicates - uses RDF reification to preserve the complete relationship
 - Positive/negative examples – encoded directly into the RDF triple
 - For unary atoms, boolean literals (true/false) are used in the object position; for binary/n-ary atoms, a prefix not_ is applied to the predicate.

Background knowledge atoms to RDF triples transformation table

Prolog BK atoms	Transformed RDF triples
<code>red(apple).</code>	<code><apple> <has_property> <red> .</code>
<code>mother(princess_diana,prince_william).</code>	<code><princess_diana> <mother> <prince_william> .</code>
<code>dimension(picture,210,297) .</code>	<code><prefix/dimension(picture,210,297).> <dimension_p1> <picture> . <prefix/dimension(picture,210,297).> <dimension_p2> "210" . <prefix/dimension(picture,210,297).> <dimension_p3> "297" .</code>

popper2rdf – usage (in Jupyter notebook)



```
from popper2rdf import popper2rdf
```

← import

```
# path to the folder with dataset name
```

```
input_files_path = 'input'
```

← Source data folder

```
output_files_path = 'output'
```

← Output data folder

```
popper2rdf('numeric-zendo1', input_files_path, output_files_path)
```

Enter dataset name (folder name)

Source data

bias.pl

bk.pl

exs.pl

Output data

numeric-zendo1.nt

```
<http://vseILPconverter/position(p10_3,8.31,0.03).> <position_p3> "0.03"^^<http://vseILPconverter/position(p48_1,1.16,9.93).> <position_p2> "1.16"^^<http://vseILPconverter/position(p16_2,9.6,3.04).> <position_p1> <p16_2> .
<p26_0> <orientation> <lhs> .
<http://vseILPconverter/position(p25_0,7.48,3.71).> <position_p2> "7.48"^^<http://vseILPconverter/position(p52_1,0.46,4.96).> <position_p3> "4.96"^^<http://vseILPconverter/position(p28_3,8.4,7.58).> <position_p3> "7.58"^^<http://vseILPconverter/position(p8_0,1.12,3.24).> <position_p3> "3.24"^^<http://vseILPconverter/position(p34_1,4.96,8.95).> <position_p3> "8.95"^^<http://vseILPconverter/position(p1_1)> <rotation> "4.38"^^<http://www.w3.org/2001/XMLSchema#double> .
```

RDF triples to background knowledge atoms transformation table

RDF triple	Transformed Prolog atom
<code><apple> <generic predicate> <red> .</code>	<code>red(apple).</code>
<code><apple> <rdf:type> <fruit> .</code>	<code>fruit(apple).</code>
<code><princess_diana> <mother> <prince_william> .</code>	<code>mother(princess_diana,prince_william).</code>

rdf2popper – usage (in Jupyter notebook)

to file t Add file <> Code

Local Codespaces

Clone

HTTPS SSH GitHub CLI

git@github.com:khrudkova/rdf2popper.git

Use a password-protected SSH key.

Open with GitHub Desktop

Download ZIP

```
from rdf2popper import rdf2popper

nt_file_name = "numeric-zendo1" # enter .nt file name
positive_exs_pred = "zendo" # enter predicate symbol denoting positive examples
negative_exs_pred = "not_zendo" # enter predicate symbol denoting negative examples
rdf2popper(nt_file_name, positive_exs_pred, negative_exs_pred, 'popper')
```

Enter dataset name (folder name)

Examples predicate symbol declarations

numeric-zendo1.nt

Source data

bias.pl

bk.pl

exs.pl

Output data

```
head_pred(zendo,1).
body_pred(position_p3,2).
body_pred(position_p2,2).
body_pred(position_p1,2).
body_pred(orientation,2).
body_pred(size,2).
body_pred(rotation,2).
body_pred(color,2).
body_pred(piece,2).
body_pred(contact,2).
```

Minimal bias generated automatically

Clone the project

import

Source data file

Mode

Transformation success

- **Goal: Evaluate if the core information and learnable patterns are preserved across data formats**
- Two-Part Evaluation:
 - **Prolog** → **RDF**: Learn rules with Popper on the original data → transform the data to RDF and learn rules with RDFRules → compare the results
 - **Prolog** → **RDF** → **Prolog**: Transform the original Prolog data to RDF and then immediately back to Prolog → learn rules with Popper on this "round-tripped" data and compare them to the original Popper rules
- Exact/similar solutions = successful transformation

Metrics and datasets

- Similarity:
 - **Quantitative** = standard metrics like Precision, Recall, and F_1 -score on the *entire* dataset
 - **Qualitative** = a set of principles to determine if rules are "similar," allowing for minor differences in variable names or numerical boundaries
- A collection of four well-known and interpretable ILP datasets were chosen
 - **numeric-zendo1** & **numeric-zendo2**: an inductive logic game used to test handling of numerical values and n -ary relations
 - **trains1**: a classic dataset for relational learning about train properties
 - **imdb3**: a dataset containing information about movies, actors, and directors

Prolog → RDF Transformation

- For **3 out of 4 datasets**, the rules learned by RDFRules on the transformed data were considered similar to Popper's original rules
- **trains1** and **imdb3** – solutions were equivalent
- **numeric-zendo1** – a near-exact match was found as minor differences were caused by RDFRules' use of discretization for numerical values
- **numeric-zendo2** – the solutions **were not similar as the original Popper rule required summing numerical values, a function that RDFRules does not support**

trains1

```
f(A):- has_car(A,C),
three_wheels(C),has_car(A,B),
long(B),roof_closed(B).
```

Prolog
(Popper)

```
( ?c <has_property> <roof_closed> ) ^
( ?b <has_property> <three_wheels> ) ^
( ?a <has_car> ?c ) ^ ( ?a <has_car> ?b )
⇒ ( ?a <f> true )
```

RDFRules

The solutions are semantically equivalent. RDFRules successfully replicated the Popper rule. Note the use of the generic `<has_property>` predicate, which is how unary Prolog atoms (like `roof_closed` and `three_wheels`) were transformed into RDF.

imdb3

Prolog (Popper)

```
f(A,B):- movie(C,B),director(B),actor(A),movie(C,A).
f(A,B):- movie(C,B),movie(C,A),gender(A,D),gender(B,D).
```

```
( ?d <movie> ?b ) ∧ ( ?b <gender> ?c ) ∧ ( ?d <movie>
?a ) ∧ ( ?a <gender> ?c )
⇒ ( ?a <f> ?b ) ( ?b <has_property> <director> ) ∧ (
?c <movie> ?b ) ∧ ( ?c <movie> ?a ) ∧ ( ?a
<has_property> <actor> ) ⇒ ( ?a <f> ?b )
```

RDFRules

The solutions are equivalent. RDFRules found the same two rules discovered by Popper, demonstrating successful transformation of the relational structure (e.g., identifying co-stars of the same gender, and an actor/director pair who worked on the same movie).

numeric-zendo1

```
zendo(A):- piece(A,B),
contact(B,C),
size(C,D),geq(D,4.12).
```

Prolog (Popper)

```
( ?c <size#discretized_level_1>
[ 4.945 ; 9.94 ] ) ^
( ?c <contact> ?b ) ^
( ?a <piece> ?b ) => ( ?a <zendo> true )
```

RDFRules

The solutions are considered similar. The core logic is identical, but the numerical boundaries differ (4.12 vs 4.945). This is because RDFRules uses discretization (binning) for numerical values, while Popper identifies precise thresholds.

numeric-zendo2

```
zendo(A):- piece(A,D),rotation(D,B),
geq(B,0.94),leq(B,4.309).
zendo(A):- piece(A,D),position(D,B,E),
add(E,B,F),leq(F,6.459).
```

Prolog (Popper)

```
( ?c <position_p2#discretized_level_1> [ 0 ; 5.365 ] ) ^
( ?c <position_p3#discretized_level_1>
[ 0.08588908852349572 ; 5.38 ] ) ^
( ?c <position_p1> ?b ) ^ ( ?a <piece> ?b )
⇒ ( ?a <zendo> true )
```

RDFRules

The solutions are not similar. The Popper solution relies on constraints on size and rotation. The RDFRules solution (showing 1 of 4 rules) instead relies on discretized position attributes (p1, p2, p3). This **divergence likely occurs due to how n-ary relations (like position) are transformed** (reified) and limitations in RDFRules' numerical reasoning capabilities.

Quantitative Evaluation: Prolog → RDF

Baseline: Popper results on Original Prolog Data.

Dataset	Precision	Recall	TP	FN	TN	FP	F_1 ¹
numeric-zendo1	1.00	1.00	30	0	30	0	1.00
numeric-zendo2	1.00	1.00	30	0	30	0	1.00
trains1	1.00	1.00	394	0	606	0	1.00
imdb3	1.00	1.00	4,075	0	117,726	0	1.00

¹ The F_1 metric is not part of the original output of the Popper run and was calculated additionally.

Results: RDRules results on Transformed RDF Data.

Dataset	Total entities	TP+TN	Precision	Recall	F_1
numeric-zendo1	60	58	0.96	0.96	0.96
numeric-zendo2	60	56	0.93	0.93	0.93
trains1	1,000	1,000	1.00	1.00	1.00
imdb3	3,726	3,726	1.00	1.00	1.00

Performance remains perfect for **trains1** and **imdb3**. Minor drops in F1 for **numeric-zendo** datasets (Bottom) are due to the discretization of numerical values by RDRules.

Prolog → RDF → Prolog

- For **3 out of 4 datasets**, Popper learned solutions identical to the original ones after the round-trip
- **numeric-zendo1** and **trains1** – Popper found the exact same solutions with identical coverage after the round-trip
- **imdb3** – Popper found the same solution, but with slightly lower example coverage because the run timed out on the transformed data (most likely due to no continuous ordering)
- **numeric-zendo2** – Popper found a different (though still optimal) solution → the round-trip transformation could not represent the **required summation function**, preventing the original rule from being re-learned

numeric-zendo2 (Prolog $\rightarrow$ RDF $\rightarrow$ Prolog)

Prolog (Popper original)

```
zendo(A):- piece(A,D),rotation(D,B),
geq(B,0.94),leq(B,4.309).
zendo(A):- piece(A,D),position(D,B,E),
add(E,B,F),leq(F,6.459).
```

Prolog (Popper round-trip)

```
zendo(A):- piece(A,B),size(B,D),leq(D,4.069),geq(D,3.0).
zendo(A):- piece(A,D),rotation(D,B),geq(B,5.07),leq(B,5.179).
zendo(A):- piece(A,D),rotation(D,B),geq(B,0.94),leq(B,4.309).
```

The solutions are not similar. Popper found a different solution after the round-trip transformation. The transformation process altered how the underlying data was represented, leading Popper to discover an alternative way to explain the examples

Quantitative Evaluation: Round-trip ($P \rightarrow \text{RDF} \rightarrow P$)

Evaluation of the RDF to Prolog transformation: Popper.

Baseline: Popper results on Original Prolog Data

Dataset	Precision	Recall	TP	FN	TN	FP	F_1 ¹
numeric-zendo1	1.00	1.00	30	0	30	0	1.00
numeric-zendo2	1.00	1.00	30	0	30	0	1.00
trains1	1.00	1.00	394	0	606	0	1.00
imdb3	1.00	1.00	4,075	0	117,726	0	1.00

Results: Popper results on Transformed (Round-trip) Data.

Dataset	Precision	Recall	TP	FN	TN	FP	F_1 ²
numeric-zendo1	1.00	1.00	30	0	30	0	1.00
numeric-zendo2	1.00	1.00	30	0	30	0	1.00
trains1	1.00	1.00	394	0	606	0	1.00
imdb3	1.00	1.00	4,060	0	10,000	0	1.00

^{1, 2} The F_1 metric is not part of the original output of the Popper run and was calculated additionally.

** The lower TP (4,060) and TN (10,000) counts occurred because the Popper run timed out on the transformed data before evaluating all examples. Precision and Recall remained 1.00, indicating the learned rules were correct for the subset evaluated.*

Experiment – WN18RR dataset

- Benchmark dataset WN18RR
 - Natively RDF knowledge graph consisting of 89,869 triples
- Testing the scalability of ILP and RDF rule mining tools
 - Popper(v4.4.0) and RDFRules (v1.9.0)
 - Demonstration of case, where user can benefit from the interoperability between methods
- 10 datasets from WN18RR knowledge graph
 - Smallest → 10% random triples from the whole KG
 - Largest → the entire KG
 - Each dataset included 10% more random triples from the KG

Experiment settings

- Learning rules of max length 6
 - Popper bias containing only the head_pred and body_pred definitions (inspired by the WN18RR example from the Popper repository)
 - RDFRules CWA confidence threshold 0.001, Data Coverage Pruning applied
- Running as recursive and non-recursive
 - Recursive as in the result rules can contain the target relation both in the head and in the body of the rule
 - RDFRules natively creates recursive rules, Popper needs declaration in bias
- Popper run both with and without (labeled as *regular*) settings
 - Noisy setting inspired by the ILP example
 - RDFRules does not have an analogous setting
- Target relation = verb_group

Recursive rules

Prolog (Popper)

```
verb_group(A, B) :- verb_group(B, A),
derivationally_related_form(A, C),
derivationally_related_form(C, D), hypernym(C, E),
derivationally_related_form(D, E).
```

RDFRules

```
( ?e <derivationally_related_form> ?d ) ^
( ?c <derivationally_related_form> ?e ) ^
( ?c <hyponym> ?d ) ^
( ?c <derivationally_related_form> ?a ) ^
( ?b <verb_group> ?a ) => ( ?a <verb_group>
?b )
```

Example of a complex recursive rule (where `verb_group` appears in both the head and body). The RDFRules example shown is highly reliable: it has a CWA Confidence of 1.0 and a Support of 84, meaning in all 84 cases where this pattern occurred, the conclusion was correct.

Runtime vs. Number of learned rules

Runtimes (in seconds) at 10%, 50%, and 100% sample sizes.

Experiment	Sample Size	Popper	RDFRules
Regular	10%	7.02s	1.00s
	50%	32.98s	38.40s
	100%	62.06s	513.61s
Noisy	10%	6.33s	1.00s
	50%	31.21s	38.40s
	100%	62.62s	513.61s
Regular + Recursion	10%	Timeout (>1200s)	1.00s
	50%	Timeout (>1200s)	61.28s
	100%	Timeout (>1200s)	853.83s
Noisy + Recursion	10%	Timeout (>1200s)	1.00s
	50%	Timeout (>1200s)	61.28s
	100%	Timeout (>1200s)	853.83s

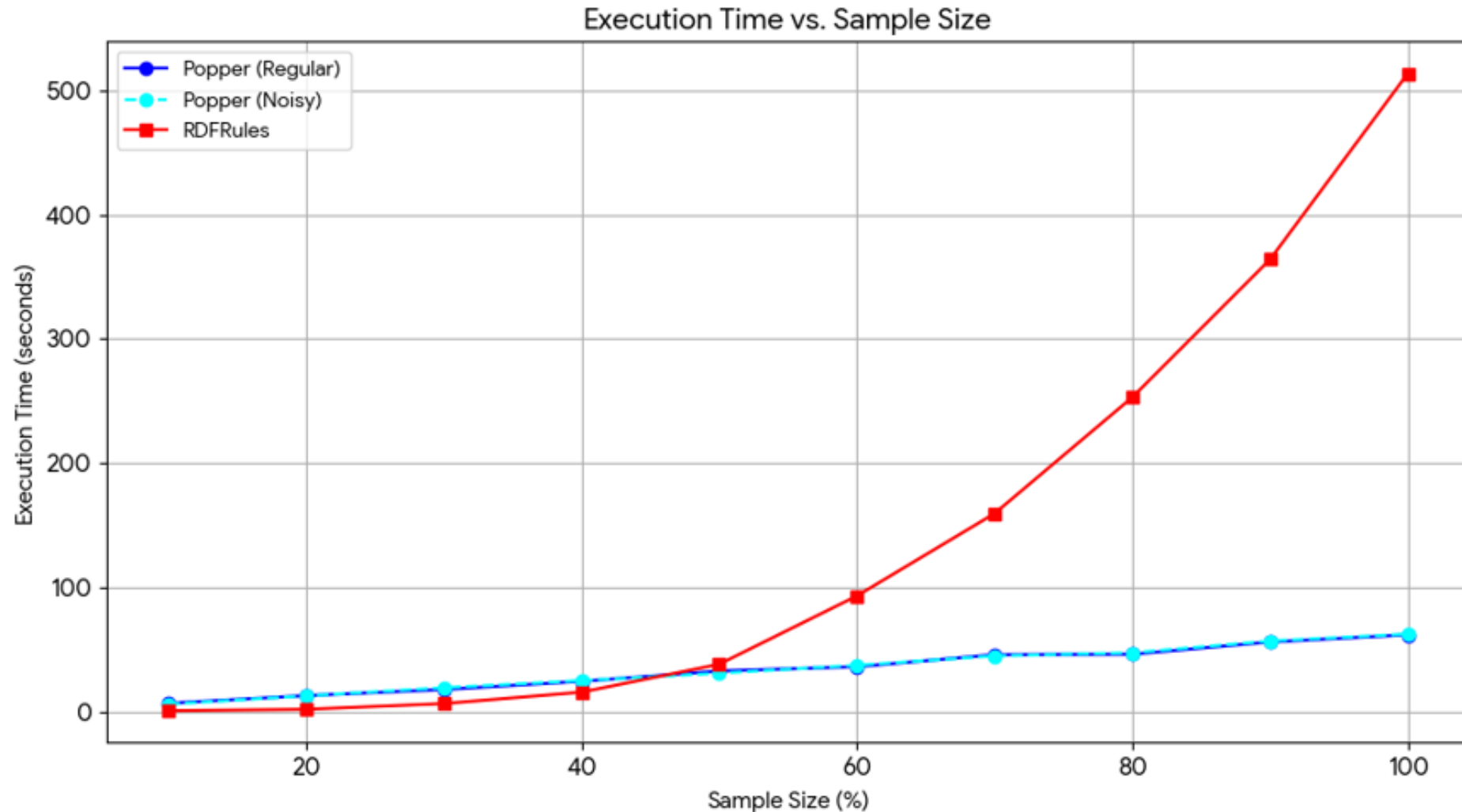
Popper is faster in non-recursive settings on this dataset but consistently times out (>1200s) when recursion is enabled.

Number of learned rules at 10%, 50%, and 100% sample sizes.

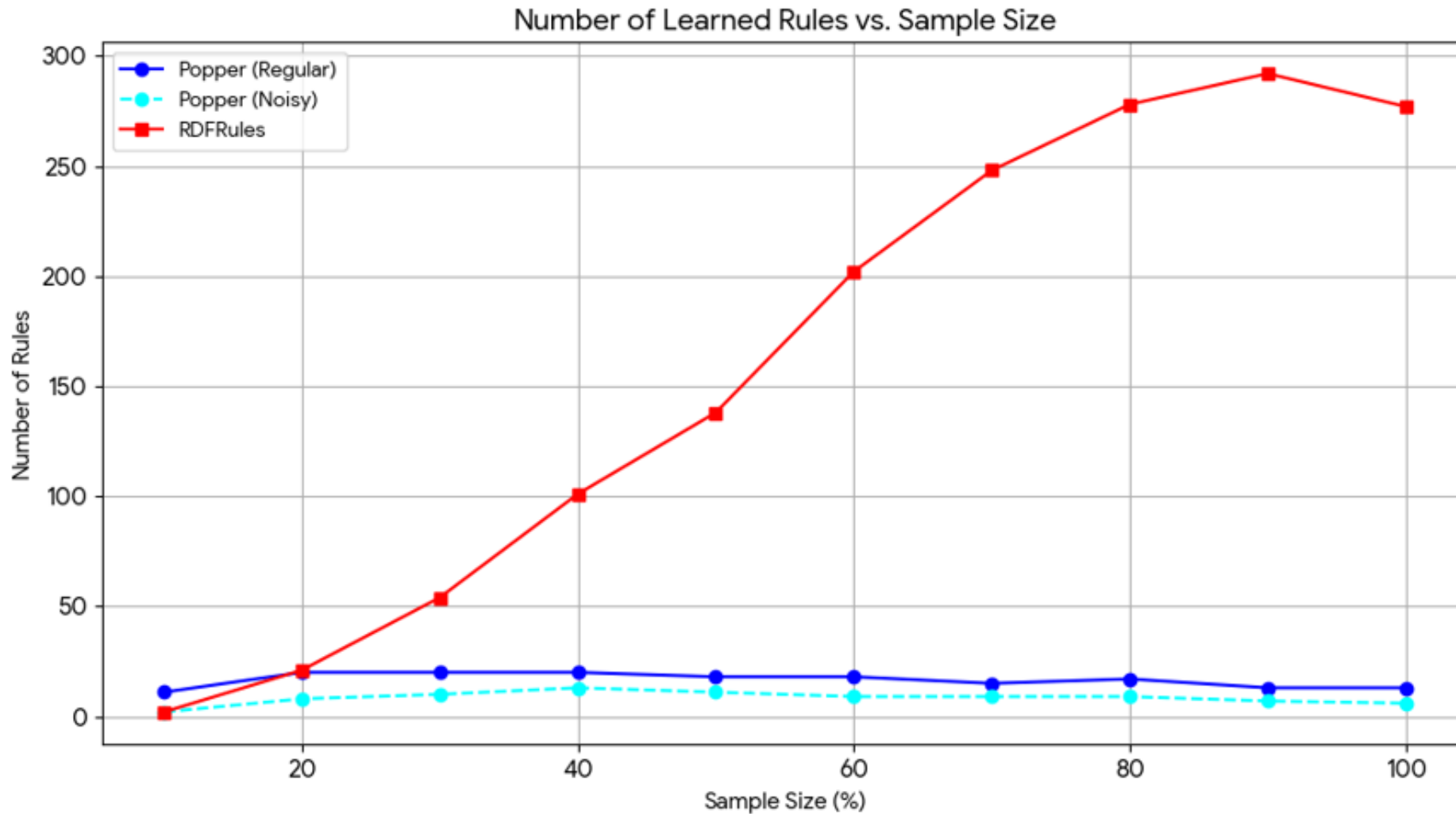
Experiment	Sample Size	Popper	RDFRules
Regular	10%	11	2
	50%	18	138
	100%	13	277
Noisy	10%	2	2
	50%	11	138
	100%	6	277
Regular + Recursion	10%	—	3
	50%	—	303
	100%	—	269
Noisy + Recursion	10%	—	3
	50%	—	303
	100%	—	269

RDFRules generates significantly more rules, reflecting its exhaustive mining approach compared to Popper's minimal program synthesis.

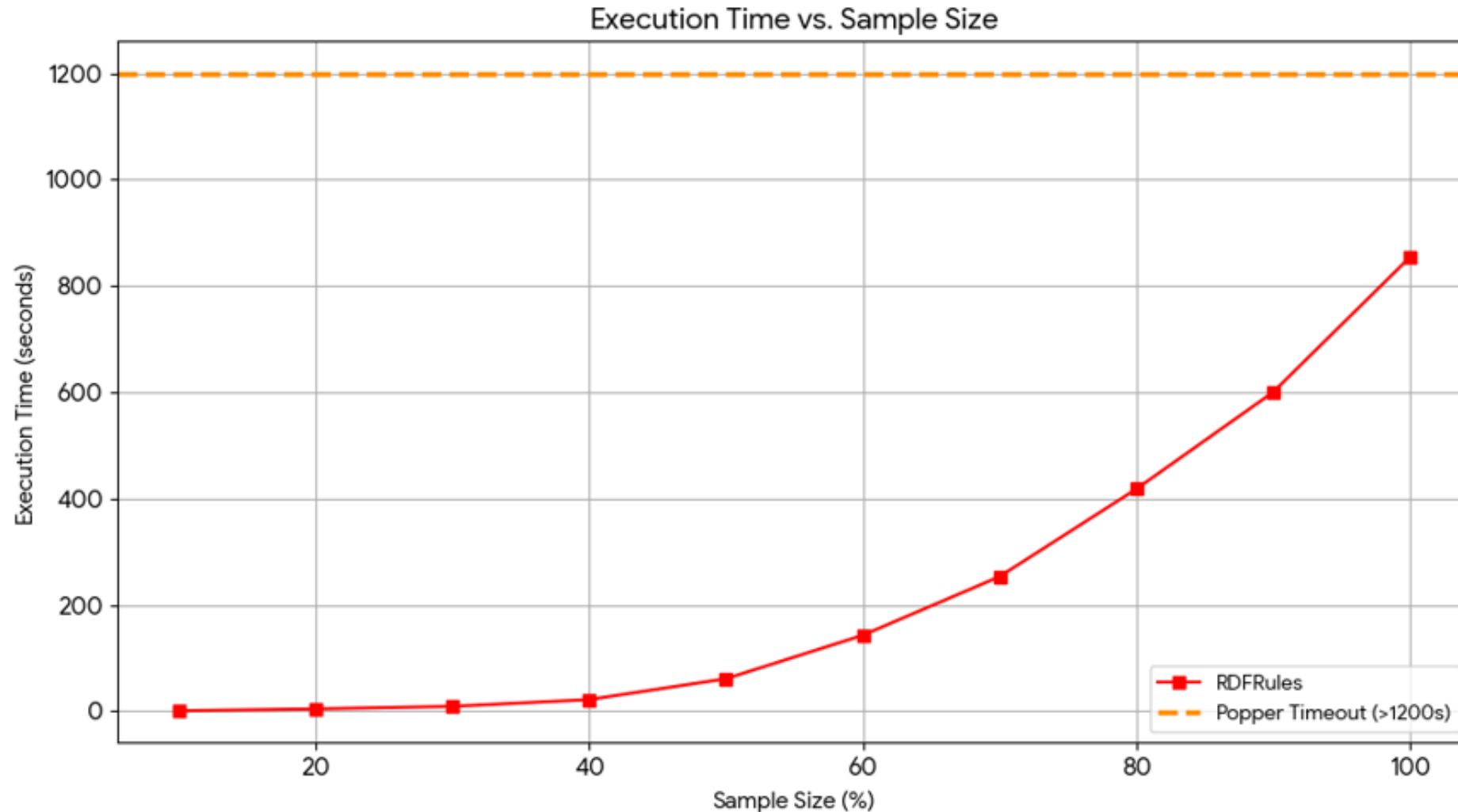
Performance on non-recursive tasks



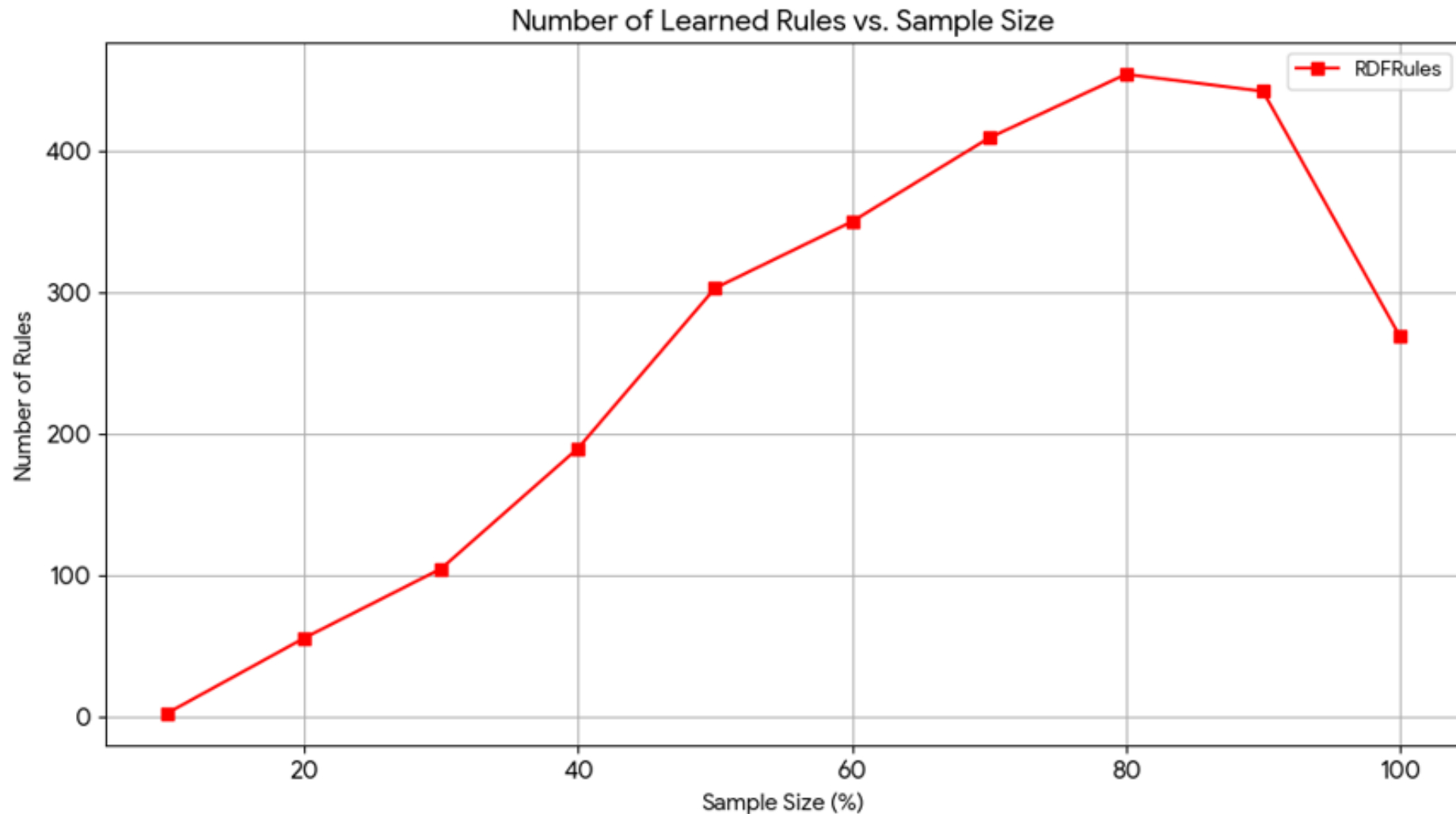
Performance on non-recursive tasks



Performance on recursive tasks



Performance on recursive tasks



Case study: KG-Microbe (Santangelo et al. 2025)

- **Motivation:** Understanding of complex interactions between microbes, their environment, and human health
- A microbe-centric knowledge graph designed for linked (relational) microbial data
- 1,368,732 nodes and 2,763,211 edges

Task Definition

- **Goal:** Find new information about possible cultivation media for microbes
- Task focused on a specific list of 40 media recommended by domain experts
- Knowledge graph rule mining task and building classifier → RDFRules

Top 5 media based on frequency

Medium ID	Medium Name	Occurrence
65	GYM STREPTOMYCES MEDIUM	1920
214	BACTO MARINE BROTH (DIFCO 2216)	1634
693	COLUMBIA BLOOD MEDIUM	1284
830	R2A MEDIUM	1278
92	TRYPTICASE SOY YEAST EXTRACT MEDIUM	1217

Rule Discovery: Approach

- **Objective** → classify media connected to a microbe by the **biolink:occurs_in** predicate
 - This relationship signifies the environment where the microbe was found, lives, or was grown (e.g., natural substance or lab medium)
- 40 separate mining procedures were run, one for each of the target media
 - merged 40 rulesets into 2.64 million rules, further reductions resulted in ruleset consisting of 3,779 rules
- Rules found a prediction for all 407 test cases

Prediction Example: *Gemella sanguinis*

- The mined rule predicts the medium Columbia blood medium for a microbe that consumes D-mannitol, has a coccus cell shape and is capable of pyroglutamyl-peptidase I

```
( ?a biolink:consumes obo:CHEBI_16899 ) ∧  
( ?a biolink:has_phenotype  
cell_shape:coccus ) ∧  
( ?a biolink:capable_of EC:3.4.19.3 )  
⇒ ( ?a biolink:occurs_in medium:693 )
```

RDFRules

Case Study: KG-COVID-19 (Reese et al. 2021)

- **Motivation:** Consolidating biomedical data to find drug repositioning candidates for COVID-19
- Graph contains information on COVID-19, human proteins, genes, chemical compounds, drugs, and related scientific literature
- 377,482 nodes and 21,433,063 edges
- Graph size reduced to process with available resources
 - Metadata removal, missing triples imputation using high confidence rules

Task (2A) Definition

- **Goal:** Perform a mechanistic deep-dive on a specific set of drugs
- **Objective:** Find drugs that m-interact with **ACE1** and, via an intermediary, interact with **ACE2** they link by identifying shared properties or common third proteins
- Knowledge graph rule mining task → RDFRules
- Result contained three rules

Uncovering the Core RAS Pathway

- The identified drugs are **Angiotensin II Receptor Blockers (ARBs)**

```
( ?b <interacts_with> <Q9BYF1> ) ^
( ?a <molecularly_interacts_with> ?b ) ^
( ?a <category> <Drug> ) ^
( ?a <molecularly_interacts_with> <P30556>
)
⇒ ( ?a <molecularly_interacts_with>
<P12821> )
```

RDFRules

Task (2B) Definition

- **Goal:** Analyze common characteristics of all entities (not just drugs) that directly interact with both ACE1 and ACE2
- **Objective:** Move beyond simple pairwise interactions to find significant, higher-order relationships
- Knowledge graph rule mining task → RDFRules

Surfacing a Core Biological Axis

- The rule with the highest support captures a critical triangular relationship at the nexus of the Renin-Angiotensin and Nitric Oxide systems

```
⇒ ( ?a <molecularly_interacts_with>
  <P12821> )
  ( ?a <interacts_with> <Q9BYF1> ) ∧
  ( ?a <interacts_with> <P29475> )
⇒ ( ?a <interacts_with> <P12821> )
```

RDFRules

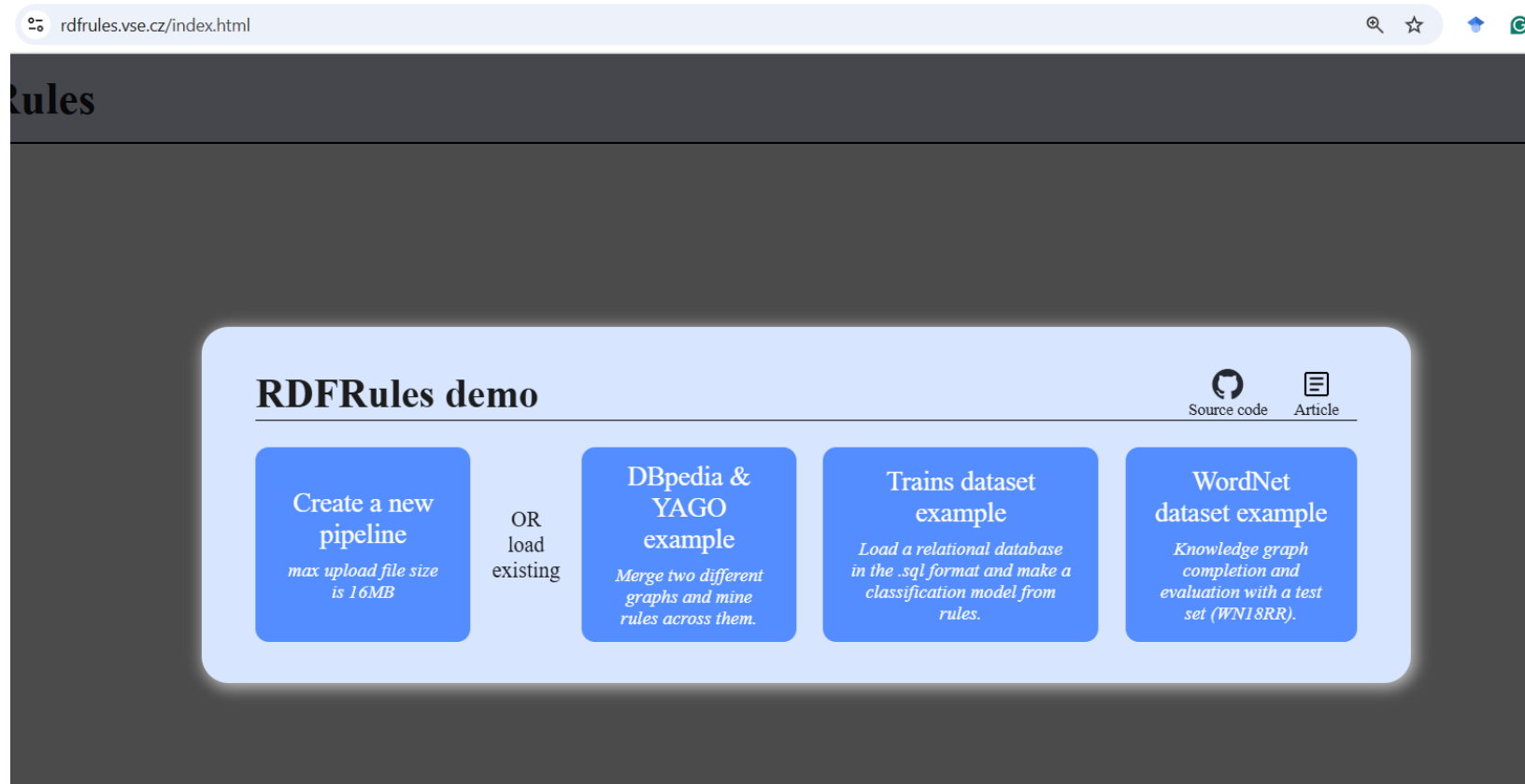
Conclusion

- **popper2rdf** and **rdf2popper** provides first insight at bridging the gap between the worlds of ILP and RDF rule mining
- The main contribution is providing the community with the first practical, open-source tools to make these powerful paradigms interoperable
- The tools were tested on four benchmark datasets and results show that transformation is viable and effective for enabling interoperability for declarative relational data
- The utility of these tools was demonstrated in a comparative experiment on the WN18RR benchmark dataset, which highlighted the distinct strengths of each paradigm
 - ILP (Popper): On non-recursive tasks, Popper was significantly faster and more scalable than RDFRules as the data size increased,
 - However, RDFRules generated more rules, which may lead to better predictive performance
 - RDF Rule Learning (RDFRules): RDFRules successfully handled large-scale recursive rule learning, a task where Popper timed out at all sample sizes, on non-recursive tasks, RDFRules was slower but learned a substantially larger number of rules

Conclusion

- Practical case studies demonstrating the utility of rule mining on real-world knowledge graphs:
 - KG-Microbe: An analysis of the KG-Microbe graph used RDFRules to find new information about microbial cultivation media by building classifier to predict media for microbes
 - "Columbia blood medium" for microbes based on specific phenotypic and metabolic traits (e.g., *Gemella sanguinis* consuming D-mannitol, having a coccus shape, and pyroglutamyl-peptidase I capability)
 - KG-Covid-19: A study on the KG-Covid-19 graph successfully identified drug repositioning candidates
 - The rules uncovered by RDFRules corresponded to known drug classes, such as Angiotensin II Receptor Blockers (ARBs) , and surfaced critical biological relationships, like the triangular nexus between the Renin-Angiotensin and Nitric Oxide systems

Demo - rdfrules.vse.cz



- Links:

- khrudkova/popper2rdf
- khrudkova/rdf2popper

References

Andrew Cropper and Rolf Morel. Learning programs by learning from failures. *Machine Learning*, 110(4):801–856, 2021.

Galárraga, L., Teflioudi, C., Hose, K., & Suchanek, F. M. (2015). Fast rule mining in ontological knowledge bases with AMIE++. *The VLDB Journal*, 24(6), 707-730.

Santangelo, B. E., Hegde, H., Caufield, J. H., Reese, J., Kliegr, T., Hunter, L. E., ... & Joachimiak, M. P. (2025). KG-Microbe-Building Modular and Scalable Knowledge Graphs for Microbiome and Microbial Sciences. *bioRxiv*, 2025-02.

Reese, J. T., Unni, D., Callahan, T. J., Cappelletti, L., Ravanmehr, V., Carbon, S., ... & Mungall, C. J. (2021). KG-COVID-19: a framework to produce customized knowledge graphs for COVID-19 response. *Patterns*, 2(1).

Václav Zeman, Tomáš Kliegr, and Vojtěch Svátek. RDFRules: Making RDF rule mining easier and even more efficient. *Semantic web*, 12(4):569–602, 2021.

Thanks for your attention