

Neural Algorithmic Reasoning Must Explain When Neuralization Adds Value

Yu He¹, Robert R. Nerem² Timo Stoll³ Semih Cantürk⁴ Dobrik Georgiev⁵ Chendi Qian³ Solveig Wittig³ Floris Geerts⁶ Stefanie Jegelka^{7,8} Ellen Vitercik¹ Yusu Wang² Nikolaos Karalias^{*,7} Christopher Morris^{*,3}
¹Stanford ²UCSD ³RWTH Aachen ⁴Mila ⁵Graphcore ⁶Antwerp ⁷MIT ⁸TUM *Main senior contributors

What is NAR?

- **Imitating** classical algorithms?
- **Aligning** neural architectures with algorithmic structure?
- **Learning** new algorithmic behaviors?
- Overlap with **adjacent fields** (NCO, neural symbolic reasoning, differential programming...)

Neural algorithmic reasoning (a working definition)

A neural network pipeline that learns to carry out an **algorithmic procedure** using its own **internal** computation at inference.

An **NAR system** consists of

- neural modules $N = (N_1, N_2, \dots, N_n)$ composed into an end-to-end pipeline
- a training set D
- a loss function L

Algorithmic inductive bias

- Obtained through a subset of modules in N
- Interplay between **algorithmic priors** and **inductive bias**

Clarifying NAR's scope and utility

Where NAR has unique leverage

Benefits from algorithms

- Extrapolation: size generalization
- Learnability and sample efficiency
- Stability and robustness
- Interpretability
- Composability

Benefits from neuralization

- End-to-end perception-to-reasoning pipeline
- Noisy, partial, or learned signals
- Learn implicit objectives from data

(a) When is NAR the right tool?

Algorithmic structure ↑	Classical algorithms and solvers Efficient or high-stakes settings	NAR • Extrapolation • Noisy, learned inputs • Implicit objective • Domain specific distribution
	Not a fit	Standard ML predictor Input-output mapping

(b) How NAR relates to adjacent paradigms?

Process-centric ↑	Learning-augmented algorithms & differentiable programs	NAR Learned execution
	Decision-focused learning & embedded optimization layers	Outcome-centric learned solvers E.g., NCO direct solution construction

← Data-driven learning (Fixed) → Outcome-centric (Neuralized)

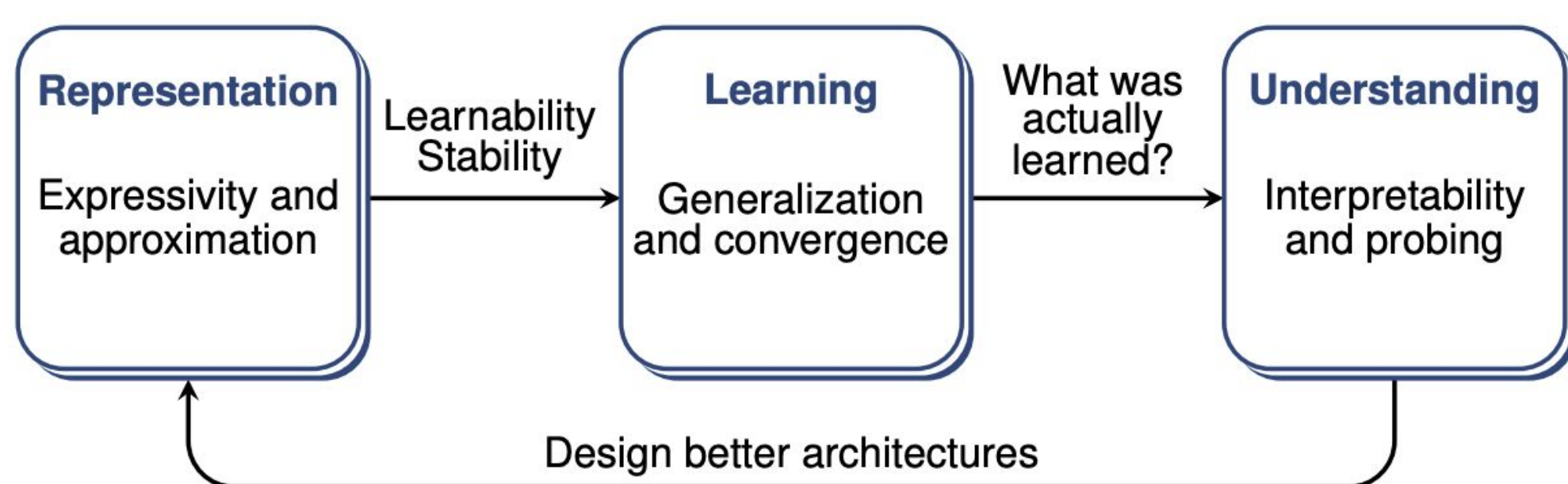
Where NAR should not be used

- A classical **algorithm** or a standard **neural network** would be better
- **High-stakes settings** that require certificates or guarantees
- The goal is learning an **input-output mapping**
- Differentiable learning of an **algorithmic procedure** is not justified

Situating NAR with neighboring approaches

- Is the algorithmic procedure **fixed** or **neuralized**?
- Is the learning target **process-centric** or **outcome-centric**?
- **A neuralization test**: What is the win case for NAR? What exactly is being neuralized? Why not use a simpler alternative? What value is demonstrated?

Theory should explain and justify neuralization



Neural representations of algorithmic procedures

- Characterize **neuralizable primitives** and pipelines
- Analyze relaxations of **discrete operations**
- Bound **error propagation** across algorithmic steps

Learnability and generalizability of neural algorithmic procedures

- Connect **optimization losses** to learned execution
- Test **invariants** under size and distribution shifts
- Separate **true algorithms** from shortcut predictors

Interpreting neuralized algorithmic procedures

- Develop **algorithm-aware** interpretability tools
- Curate benchmarks with **hypothesized** algorithmic structure

Evaluation should emphasize cross-paradigm comparison and practicality

What to compare with? Compare against the state of the art Neural Combinatorial Optimization Differentiable solvers Black-box differentiation	How to compare? Better metrics to track practicality Multi-axis difficulty metrics Track practical utility Diversity and real-world	Where it matters? High-impact use cases Large language models Chip design Molecular tasks
--	--	--

Compare against the state of the art

- An end-to-end **predict-then-optimize** pipeline (e.g., Warcraft shortest-path problem)
- Compare NAR baselines with **neural combinatorial optimization (NCO)** benchmarks

Better metrics to track practicality

- Define **diverse difficulty axes**: sparsity, structure, reasoning depth, encoding mismatch...
- Capture **efficiency, reliability, and robustness**: quality-time AUC, constraint violations, structured shifts...

From synthetic algorithms to high-impact domains

- Design **datasets** from chip design, LLMs, floorplanning, molecular tasks...
- Evaluate in **deployment-like settings**

Full paper

