# Neural Algorithmic Reasoning Must Explain When Neuralization Adds Value

**Yu He** [1] **Robert R. Nerem** [2] **Timo Stoll** [3] **Semih Cantürk** [4 5] **Dobrik Georgiev** [6] **Chendi Qian** [3] **Solveig Wittig** [3] **Floris Geerts** [7] **Stefanie Jegelka** [8 9] **Ellen Vitercik** [1] **Yusu Wang** [2] **Nikolaos Karalias** [* 8] **Christopher Morris** [* 3]

## Abstract

Neural algorithmic reasoning (NAR) has emerged as a promising direction for integrating mathematical and algorithmic reasoning into end-to-end trainable neural pipelines, aiming to bring the structure and reliability of algorithms into practical pipelines, such as real-world mathematical problem solving. This position paper argues that, despite growing interest, NAR remains a broad and ambiguously defined collection of ideas. This vagueness obscures its relationship to neighboring areas, such as differentiable programming and machine learning for combinatorial optimization, and risks slowing progress. We argue that NAR has distinctive leverage, but realizing this leverage requires clearer answers to two central questions: what should count as NAR, and when is neuralizing an algorithmic procedure genuinely justified? To this end, we propose a working definition of NAR, clarify where it should and should not be applied, and identify theoretical goals for understanding correctness, generalization, and benefits of neuralized algorithms. We further propose empirical protocols that emphasize cross-paradigm comparisons and track practical value beyond synthetic algorithmic tasks. Together, these objectives provide a roadmap for developing NAR into a coherent research area with broader scientific and practical impact.

## 1. Introduction

Algorithmic procedures are central to mathematical reasoning, such as proof search, tactic selection, dynamic programming, optimization, and verification routines. These algorithmic procedures provide structure and sometimes formal guarantees. However, in realistic AI-for-math pipelines, the inputs to these procedures are often learned, uncertain, or only partially specified: a model may be required to infer a formal statement from natural language, propose a proof state, or generate candidate hypotheses that must be checked by downstream algorithms. Bridging this gap is a timely and promising interest for future breakthroughs for real-world mathematical reasoning systems.

*Neural algorithmic reasoning* (NAR) (Veličković et al., 2020; Xu et al., 2020; Veličković & Blundell, 2021) emerges as a promising direction to address this gap. NAR aims to endow neural networks with the ability to learn, execute, and internalize discrete algorithms, integrating mathematical and algorithmic reasoning into trainable neural systems, enabling end-to-end optimization, and leveraging the data distribution. In principle, it combines the robustness of algorithmic procedures with the adaptability of neural networks, offering a path toward more structured and data-efficient learning. For example, when solving real-world mathematical problems, this perspective is especially relevant when symbolic or algorithmic components must interact with informal language, partial inputs, or distribution-specific data.

However, despite its growing popularity, NAR still lacks a clear scope and definition. The lack of clarity leaves NAR ambiguously positioned among neighboring paradigms, creating confusion about what belongs to the field and what should count as progress. For example, NAR has evolved to encompass a broad set of goals, including imitating classical algorithms (Veličković et al., 2020), aligning neural architectures with algorithmic structure (Xu et al., 2020), and learning new algorithmic behaviors (Schutz et al., 2025). At the same time, NAR's broad framing of combining algorithms and machine learning makes it appear to overlap with several neighboring fields—including differentiable programming (Blondel & Roulet, 2025), neural program synthesis (Parisotto et al., 2016), neuro-symbolic learning (Smet & Raedt, 2025), machine learning for combinatorial optimization (Cappart et al., 2023), learning augmented algorithms (Mitzenmacher & Vassilvitskii, 2020), data-driven algorithm design (Balcan, 2020), and decision-focused learning (Mandi et al., 2024)—making NAR hard to delineate and hard to justify as a distinct area of study. Without a clearly

[*]Main senior contributors. [1]Stanford University [2]University of California San Diego [3]RWTH Aachen University [4]Mila–Quebec AI Institute [5]Université de Montréal [6]Graphcore [7]University of Antwerp [8]Massachusetts Institute of Technology [9]Technical University of Munich. Correspondence to: Yu He <heyu@stanford.edu>.

The $3^{rd}$ *AI for Math Workshop* at the $43^{rd}$ International Conference on Machine Learning (ICML), Seoul, South Korea, 2026. Copyright 2026 by the author(s).

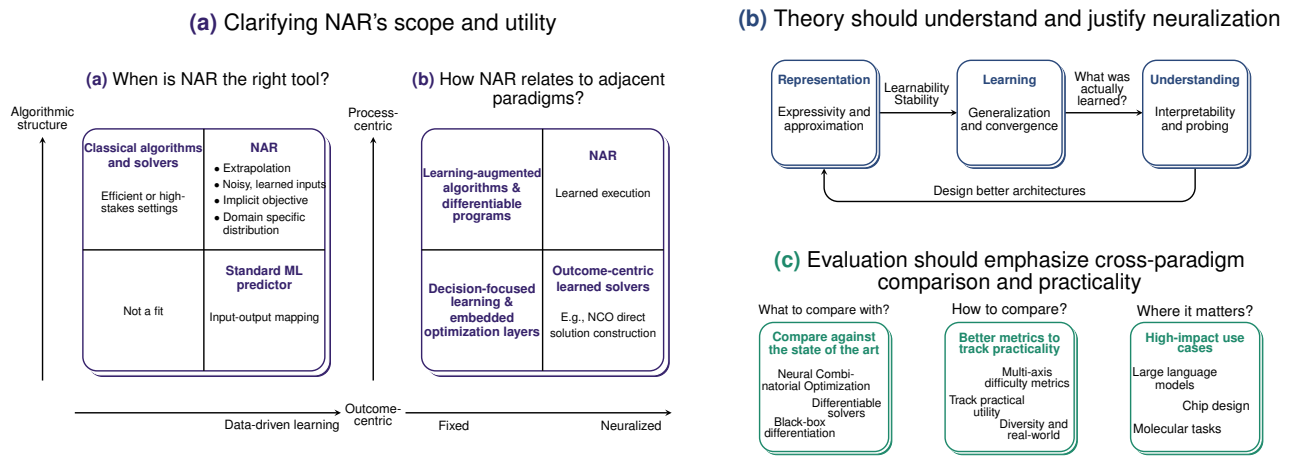


*Figure 1.* We propose a working definition of NAR, clarifying its scope and utility, alongside theoretical and empirical proposals to justify neuralization and demonstrate its leverage.

established scope or shared standards for theoretical and empirical validation, the field is susceptible to fragmentation, overstated claims of novelty, and unfair comparison.

We posit that NAR offers unique leverage for bringing robust, structured algorithmic thinking into real-world learning systems requiring mathematical and algorithmic abilities, but this leverage risks being diluted by an ambiguous scope. As NAR attracts broader attention, the field must clarify what should count as NAR and when neuralizing an algorithmic procedure is genuinely justified. Without such clarity, NAR risks becoming a scattered set of loosely related methods rather than a coherent field with clear goals and meaningful ways to measure progress. Specifically, our position paper contributes to

- a working definition for NAR as learned algorithmic execution, involving learnable algorithmic modules embedded in an end-to-end optimizable pipeline.
- We clarify NAR's scope and utility by identifying where it has unique leverage, where it should not be used, and how it should be situated relative to adjacent fields.
- We recommend theoretical goals for NAR to understand and explain when neuralization is justified, grounded in expressivity, generalization, and interpretability.
- We propose empirical protocols that compare NAR against the strong baselines from neighboring paradigms and that track practical value beyond synthetic algorithmic settings.
- We outline actionable objectives to help the community implement this agenda and further establish NAR as a more coherent research area with broader impact.

## 2. Clarifying NAR's scope and utility

Despite growing attention, NAR remains a broad and loosely connected collection of ideas without a clear definition or scope. The seminal paper that introduced the term NAR (Veličković & Blundell, 2021) proposed an encoder-processor-decoder blueprint. However, beyond motivating the integration of algorithmic reasoning into neural models, it did not attempt to define NAR properly. To ground this paper's central concepts, we first propose a working definition of NAR.

**Definition 1** (Neural Algorithmic Reasoning System)**.** *A neural algorithmic reasoning system* is a neural network pipeline that learns to carry out an algorithmic procedure using its own internal computation at inference (rather than calling an external algorithm). Concretely, an NAR system consists of (i) neural modules $\mathcal{N} = (N_1, N_2, \ldots, N_n)$ composed into an end-to-end pipeline, (ii) a training set $\mathcal{D}$, and (iii) a loss function $\mathcal{L}$. The pipeline is designed with an *algorithmic inductive bias*, so its algorithmic behavior is attributable to some subset of modules in $\mathcal{N}$.

Here, *neural modules* include standard building blocks such as *feed-forward neural networks* and *message-passing graph neural networks* (MPNNs) (Gilmer et al., 2017). The phrase *algorithmic inductive bias* is deliberately broad: it may arise from *algorithmic alignment* (Xu et al., 2020), but also from trajectory supervision (hints (Veličković et al., 2022)) or curricula that teach primitives. We require that one or more learnable modules emulate, approximate, or internalize an algorithmic computation (rather than parameterizing a call to a fixed solver). Observable algorithmic behavior includes matching a reference algorithm $\mathcal{A}$ on a distribution of instances and extrapolating across instance sizes.

Under this definition, examples of NAR include: (i) *Graph-algorithm execution with MPNNs*, e.g., an MPNN trained

to compute quantities depending on shortest paths. (ii) *Algorithm-trajectory emulation*, where models are trained on intermediate states (traces/trajectories) of a target algorithm. In contrast, if the forward pass executes a *fixed* differentiable algorithm, or if a generic predictor is trained *without* an algorithmic inductive bias, it falls outside our intended scope.

Central to our framing is the interplay between *algorithmic priors* and *inductive bias*. Algorithmic priors are task-specific beliefs about which algorithmic procedures are effective for a given problem (e.g., the breadth-first search algorithm for a path-finding task). These priors are incorporated into the system through inductive biases, e.g., by architectural alignment or trajectory supervision.

While Definition 1 provides a working characterization of NAR, it still leaves open important questions about its scope and utility. Historically, NAR has evolved to span heterogeneous goals—from imitating known procedures (Veličković et al., 2020) to discovering new algorithms from data (Schutz et al., 2025)—with blurred boundaries with many adjacent fields such as neuro-symbolic learning (Smet & Raedt, 2025) and neural combinatorial optimization (Cappart et al., 2023). This ambiguity hinders fair comparison and practical justification. This motivates the next step of our framing: when does "neuralizing" an algorithmic procedure offer capabilities that classical algorithms, differentiable solvers, or black-box learning do not? We use this question to delineate where NAR has unique leverage and situate contributions within the broader landscape.

### 2.1. Where NAR has unique leverage

NAR has unique leverage when both algorithmic structure and data-driven learning matter. Its value comes from two sources: (i) embedding algorithmic structure into a model, and (ii) implementing that structure as a neural, trainable component in a larger pipeline.

**Benefits from algorithms.** Embedding algorithmic structure can yield benefits that are hard to obtain from generic function approximation alone. A primary motivation is extrapolation, especially size generalization, i.e., when learned computation reflects stable algorithmic invariants, it is more likely to transfer across instance sizes (Xu et al., 2020). In addition, algorithmic structure can improve learnability and sample efficiency by biasing learning toward effective procedures, thereby reducing the data required to acquire multi-step computation (Xu et al., 2021). It also promotes stability and robustness through structured intermediate computations, improves interpretability with recognizable procedures, and supports composability by enabling reusable modules (Eberle et al., 2025).

**Benefits from neuralization.** NAR is particularly useful when the inputs or objectives are underspecified. It can support a *perception-to-reasoning* pipeline in which the "algorithm" operates on noisy, partial, or learned signals, e.g., latent graphs, uncertain costs, or soft constraints, while remaining trainable end-to-end and preserving useful algorithmic inductive biases. Similarly, when the *objective* is implicit—given via demonstrations, labels, or preferences—the model can learn the goals from data, as in domains like chip design where goals are hard to formalize by human.

A further advantage is the ability to leverage distributional information. Classical algorithm design often treats inputs in a worst-case sense, whereas many real-world instances are drawn from domain-specific distributions. Here, NAR can combine algorithmic structure with domain knowledge; e.g., vehicle-routing problems may repeatedly arise within the same geographic region, with destinations drawn from a limited set of sites. Finally, neuralized procedures may be designed to exploit parallel hardware, making them attractive when latency or throughput is a central constraint.

### 2.2. Where NAR should *not* be used

NAR is often hard to justify when *either* a classical algorithm *or* a standard neural network would be the better tool. For example, neural components need not be introduced if the task is well-structured and classical methods already provide efficient and verifiable solutions. This is especially important in high-stakes settings that require certificates, such as optimality or feasibility guarantees. In such cases, NAR is better treated as an auxiliary component rather than as an uncertified replacement for the solver. Even in lower-stakes settings, neuralizing an already efficient procedure can be difficult to justify if the learned model is slower or less stable under distribution shifts.

In addition, if the goal is simply to learn an input-output mapping without needing to represent or execute a reusable multi-step procedure, then standard neural models are often the better choice. Moreover, when the primary motivation is differentiable end-to-end learning, black-box differentiation or implicit-gradient approaches (Niepert et al., 2021; Petersen, 2022) may be more appropriate, since they do not require learning an entire algorithmic procedure. Overall, NAR work should explain why end-to-end learning of an algorithmic procedure is needed, rather than directly differentiating an existing solver, using a classical algorithm, or training a conventional neural model.

### 2.3. Situating and comparing NAR with neighboring approaches

NAR does not exist in isolation. Adjacent communities, e.g., neuro-symbolic learning (Smet & Raedt, 2025), differentiable programming (Blondel & Roulet, 2025), learning-

augmented algorithms (Mitzenmacher & Vassilvitskii, 2020), machine learning for combinatorial optimization (Cappart et al., 2023; Scavuzzo et al., 2024), and decision-focused learning (Mandi et al., 2024), pursue overlapping goals under different terminology. Without careful comparison, communities risk rediscovering ideas and overstating novelty. We propose the following framework to situate and compare NAR with neighboring fields.

**A framework for positioning methods.** Following Definition 1 and the above discussions, NAR should be positioned around its distinctive target: learning, or neuralizing, an algorithmic procedure—a structured multi-step computation with an algorithmic inductive bias—rather than only learning solutions, costs, objectives, heuristics, or solver parameters. Two questions help distinguish NAR from neighboring approaches, as summarized in Figure 1: *(Q1) Is the algorithmic procedure fixed or neuralized?* and *(Q2) Is the learning target process-centric or outcome-centric?*

*(Q1) Is the algorithmic procedure fixed or neuralized?* In NAR, some part of the algorithmic procedure is represented by learnable modules that emulate, approximate, or adapt a multi-step computation. This distinguishes NAR from differentiable programming, embedded optimization, or decision-focused learning, in which a fixed, hand-specified solver may be placed within a learning pipeline but is not itself learned.

*(Q2) Is the learning target process-centric or outcome-centric?* NAR is *process-centric*: it learns a reusable multi-step procedure, such as search, dynamic programming, or iterative refinement. Outcome-centric methods instead optimize directly for the final solution or decision, without necessarily modeling the computation that produces it. Thus, machine learning for combinatorial optimization overlaps with NAR when it neuralizes a structured procedure; if it simply predicts solutions/costs/heuristics, it is better viewed as neural combinatorial optimization or end-to-end learning.

**How to situate NAR contributions.** First, state the intended benefit of algorithmic structure, such as extrapolation, sample efficiency, robustness, or average-case performance. Secondly, state how the algorithmic prior is encoded, such as through architectural alignment. Together, these clarify which prior is assumed, how it is implemented, and what evidence should count as success.

**Actionable objectives.** The discussion above suggests a single practical requirement for NAR papers: they should pass a *neuralization test*. Rather than treating neuralization as self-justifying, papers should explain why the relevant algorithmic procedure should be learned, adapted, or embedded as a trainable component, instead of being handled by a classical algorithm, a fixed differentiable solver, or a standard neural model. Future NAR papers should answer the following questions.

1. **What is the win case for NAR?** State the setting in which NAR is expected to have unique leverage. Examples include noisy or learned inputs, implicit objectives, differentiable pipelines, amortized inference, exploitable distributional knowledge, or hardware-driven latency constraints.
2. **What exactly is being neuralized?** Specify whether the method learns a procedure, a solution, an objective, a heuristic, a cost model, solver parameters, or some combination of these. This clarifies whether the contribution is genuinely process-centric or more closely related to adjacent fields such as neural combinatorial optimization and differentiable programming.
3. **Why not use a simpler alternative?** Compare against the most relevant non-NAR baseline: a classical algorithm or solver when the task is well specified; a standard neural model when the goal is input-output prediction; or a differentiable solver, black-box differentiation, implicit-gradient method, or hybrid pipeline when the goal is end-to-end learning.
4. **What value is demonstrated?** Name the intended advantage (e.g., extrapolation, sample efficiency, robustness, differentiability, interpretability, composability, latency, or average-case performance). Then report evidence that tests this claim (e.g., solution quality, feasibility, runtime, training cost, robustness under shifts, failure rates, or preservation of certificates).

## 3. Theory should explain and justify neuralization

A central obstacle for NAR is not only the lack of guarantees, but the lack of a theory explaining when neuralization is justified. The key question is not merely whether a neural model can imitate an algorithm, but whether, and when, replacing or relaxing that algorithm with learnable computation is trainable and useful. Theory should therefore formalize the neuralization test introduced above.

This is important for hard tasks such as combinatorial optimization, where algorithms provide worst-case guarantees but may ignore useful distributional structure. NAR is justified when it exploits an instance distribution $\mathcal{D}$ to improve typical-case performance, amortize inference, or handle noisy and partial inputs.

Hence, we propose to advance theory along three axes, e.g., expressivity, to characterize which procedures NAR architectures can represent and what is lost through relaxation; optimization and generalization, to explain when training recovers algorithmic behavior; and interpretability, to determine whether a model has learned the intended procedure or

a useful alternative.

### 3.1. Neural representations of algorithmic procedures

Understanding the expressivity of NAR should be tied to neuralization. A target algorithm $\mathcal{A}$ is not merely an input–output map, but a structured procedure with intermediate states, invariants, branching, merging, and iteration. Thus, a neuralized algorithm $\mathcal{N}_\theta$ should not only be evaluated by its final output, but by which parts of the procedure it can approximate or usefully improve.

Prior work shows that architectures such as MPNNs (Loukas, 2020; Nerem et al., 2025; Xu et al., 2020; Yau et al., 2024; Qharabagh et al., 2026), transformers (de Luca & Fountoulakis, 2024; de Luca et al., 2025b), and fully-connected networks (de Luca et al., 2025a) can represent certain algorithms. However, we still lack a principled account of when an architecture can solve a given algorithmic problem, and with what resources.

This is especially important because many algorithms rely on discrete operations, including exact comparisons, branching, and $\arg\min$. Neural models often replace these with smooth relaxations, which enable differentiability but may change the procedure's semantics. Theory should explain when such relaxations preserve invariants, remain stable, and control error accumulation over many steps, as in work that studies whether transformers can simulate regression algorithms (Akyürek et al., 2023; Bai et al., 2023).

Finally, expressivity should be considered together with computational cost and approximation quality. If a neuralized procedure is slower or less reliable than the algorithm it emulates, it needs a compensating benefit, e.g., amortized inference, improved average-case performance on a distribution $\mathcal{D}$, robustness to noisy inputs, differentiability in a larger pipeline, or speed–accuracy tradeoffs with GPUs. For hard combinatorial problems, this may mean distribution-aware guarantees, such as lower expected optimality gaps on $\mathcal{D}$. Without such an advantage, neuralization is hard to justify.

**Actionable objectives.** We propose the following objectives for deriving a theory of which procedures can be neuralized, and when doing so is valuable.

1. **Identify neuralizable primitives.** For a given architecture, characterize which basic operations—comparison, selection, branching, iteration, or termination—can be implemented by learned neural modules rather than by fixed solvers or hand-coded routines.
2. **Characterize neuralizable pipelines.** Determine which algorithms $\mathcal{A}$ can be represented or approximated by composing such primitives into a pipeline $\mathcal{N}_\theta$, and state the required resources, such as depth, width, recurrence, precision, and parallelism.
3. **Analyze relaxations of discrete operations.** For primitives such as comparisons, branching, and $\arg\min$, construct differentiable relaxations under an instance distribution $\mathcal{D}$ and loss $\mathcal{L}$, and prove when they preserve the intended operation, including under repeated composition.
4. **Bound error propagation.** When $\mathcal{N}_\theta$ replaces components of $\mathcal{A}$ with learned surrogates, establish stability and error-propagation bounds of the form
$$\|\mathcal{N}_\theta(x) - \mathcal{A}(x)\| \leq \Phi\big(\{\varepsilon_c(x)\}_{c\in\mathcal{C}(x)}\big)\,,$$
where local component errors $\varepsilon_c(x)$ accumulate over components $\mathcal{C}(x)$. Such bounds should identify which parts of the algorithm are brittle and which tolerate approximation.
5. **Make computational trade-offs explicit.** Compare the training and inference costs of $\mathcal{N}_\theta$ with the runtime $T_{\mathcal{A}}(n)$ of the target algorithm, accounting for depth, parameters, precision, and hardware. Clarify whether neuralization improves speed, accuracy, robustness, or composability.
6. **Identify distribution-aware advantages.** For hard problems with worst-case classical guarantees, characterize when NAR can exploit structure in $\mathcal{D}$ to improve typical-instance performance, e.g., through better expected approximation ratios, average-case runtime, or robustness.

These objectives shift expressivity theory from the question "Can a neural network simulate this algorithm in principle?" to the more useful question "Can this algorithmic procedure be neuralized in a way that preserves the properties we care about and provides a reason to use NAR?".

### 3.2. Learnability and generalizability of neural algorithmic procedures

Even if an NAR architecture can represent a target procedure, this does not imply that training will find the intended representation or that the learned model will generalize beyond the training dataset. For example, a model may match an algorithm's outputs on small instances while relying on shortcuts or dataset artifacts that fail under distribution shift. Thus, a theory that justifies neuralization must also address whether learned execution is attainable via gradient-based training.

Hence, we propose to leverage *learnable algorithmic invariants*, i.e., latent quantities, relational structures, or update rules that remain stable across instance sizes, perturbations, and distributions. If an NAR model truly learns an algorithmic procedure, then some of these invariants should be preserved across intermediate computations, not merely reflected in final predictions.

To clarify the link between optimization and algorithmic behavior, we propose a hierarchy of learnability. For a model

$\mathcal{N}_\theta$, target procedure $\mathcal{A}$, training objective and regularization term $\mathcal{L}(\theta) + r(\theta)$, training distribution $\mathcal{D}_{\text{train}}$, and deployment distribution $\mathcal{D}_{\text{dep}}$, one can distinguish the following.

1. *Faithful neuralized execution:* there exists parameters $\theta$ that implement the target procedure $\mathcal{A}$, or an equivalent procedure, at the level of intermediate states and final outputs. Study when $\mathcal{N}_\theta$ approximates $\mathcal{A}$ with predictable error, including across multiple computational steps and deployment distribution.
2. *Guaranteed convergence:* gradient-based training converges to parameters that realize an approximate execution, rather than just fitting the training data.
3. *Justified neuralization:* the learned procedure provides a quantifiable advantage over a relevant non-learned algorithm or a generic predictor, e.g., in terms of solution quality, computation time, robustness, differentiability, or integration into a larger pipeline.

Hence, a useful theory should therefore provide bounds of the form

$$\mathcal{L}(\theta) \leq \min_{\psi} \mathcal{L}(\psi) + r(\psi) + \varepsilon \Longrightarrow d_{\text{alg}}(\mathcal{N}_\theta, \mathcal{A}; \mathcal{D}_{\text{dep}}) \leq g(\varepsilon, \Delta),$$

where $d_{\text{alg}}$ measures procedure-level deviation from $\mathcal{A}$, $\Delta$ captures the shift between training and deployment conditions. Therefore, $g$ describes how optimization error and distribution shift affect execution error.

**Actionable objectives.** We propose the following directions for a theory of learnability and generalization that directly supports the justification of neuralization.

1. **Define algorithmic invariants.** Develop metrics that test whether a model learns the intended behavior, including intermediate states, invariants, update rules, and stopping conditions. These metrics should distinguish genuine learned execution from interpolation.
2. **Link losses to procedures.** Determine when minima, or approximate minima, of practical objectives $\mathcal{L}(\theta) + r(\theta)$ correspond to approximate execution of a target procedure.
3. **Analyze optimization dynamics.** Identify when gradient descent converges to algorithmic solutions rather than spurious predictors. Relevant factors include initialization, architecture, hyperparameters, data distribution, and intermediate supervision. Work on combinatorial optimization dynamics, such as policy-gradient analysis (Caramanis et al., 2023), may offer useful starting points.
4. **Explain extrapolation under structured shifts.** Derive generalization guarantees across graph sizes, sequence lengths, noise levels, or shifts common in algorithmic tasks. Such guarantees should connect architectural biases in Section 3.1 to stable learned invariants, rather than relying on i.i.d. generalization.
5. **Characterize advantages over classical baselines.** Formalize when neuralization provably improves over exact algorithms, approximation algorithms, heuristics, differentiable solvers, or distribution-aware non-neural methods. For hard optimization problems, this may require distribution-dependent guarantees. For example, in a maximization setting with instances $x \sim \mathcal{D}_{\text{dep}}$, optimal value $\mathsf{OPT}(x)$, and solution quality $q(\cdot, x)$, one may ask when

$$\mathbb{E}_{x\sim\mathcal{D}_{\text{dep}}}\left[\frac{q(\mathcal{N}_\theta(x), x)}{\mathsf{OPT}(x)}\right] \geq \mathbb{E}_{x\sim\mathcal{D}_{\text{dep}}}\left[\frac{q(B(x), x)}{\mathsf{OPT}(x)}\right] + \gamma,$$

for relevant baselines $B \in \mathcal{B}$ and margin $\gamma > 0$, with the inequality adapted for minimization. Such results would clarify when NAR exploits distributional structure rather than merely replacing a classical algorithm with a learned surrogate.

### 3.3. Interpreting neuralized algorithmic procedures

Interpreting whether an NAR model internalizes the intended algorithmic behavior is essential for understanding and justifying its leverage. First, when embedded in real-world pipelines with noisy inputs (Section 2.1), such analysis tests whether the model learns the intended behavior, adapts it neurally, or exploits shortcuts, thereby justifying NAR. Second, it can support algorithmic discovery (Romera-Paredes et al., 2024): when NAR models outperform the target algorithm on certain distributions (He & Vitercik, 2025), probing may reveal modified procedures useful beyond worst-case regimes.

**Actionable objectives.** We outline concrete directions for developing interpretability tools for NAR and probing the algorithms internalized by trained models.

1. **Develop algorithm-aware interpretability tools.** Design tools that expose learned algorithmic primitives, intermediate states, and compositional structure. These can adapt existing XAI methods: probing hidden activations for algorithmic variables and states (Alain & Bengio, 2018; Naidu et al., 2025); extracting concept- or rule-level structure aligned with target procedures (Georgiev et al., 2022); and using mechanistic interpretability to identify circuits or features that implement algorithmic computation (Nanda et al., 2023; Zhong et al., 2023; He et al., 2025a; Mirjanic et al., 2023; Huben et al., 2024).
2. **Curate benchmarks with hypothesized algorithmic structure.** Construct benchmark tasks with testable hypotheses about the advantages of algorithmic structure, aligning with Eberle et al. (2025). Hypotheses may target which internal variables are represented, which procedures are learned, and which sub-procedures are shared

across tasks.

## 4. Evaluation should emphasize cross-paradigm comparison and practicality

Existing NAR benchmarks (Veličković et al., 2022; Minder et al., 2023; Markeeva et al., 2024; He et al., 2025b; Stoll et al., 2025) primarily focus on synthetic algorithmic tasks, following the similar fashion of the CLRS Benchmark (Veličković et al., 2022) that covers sorting, string, graph, geometry, dynamic programming, and other algorithmic domains. The exclusive focus on algorithmic simulation correctness makes it difficult to compare NAR fairly with neighboring paradigms. This limitation motivates cross-paradigm evaluation to better demonstrate NAR's distinctive value. Moreover, restricting evaluation to synthetic algorithmic datasets limits practical relevance. Since the long-term goal of NAR is to integrate algorithmic reasoning into real-world learning systems, NAR benchmarks should include metrics that track practical utility and connect to high-impact use cases.

### 4.1. Comparing NAR against the state of the art

Careful benchmarking should compare NAR against the strongest relevant alternatives, as discussed in Section 2.3, and isolate its benefits through systematic ablations. Since NAR integrates algorithmic inductive bias into end-to-end trainable pipelines, natural baselines include pipelines that execute algorithms with gradient estimation (Pogancic et al., 2020; Niepert et al., 2021), smooth relaxations of operations such as sorting and sampling (Blondel et al., 2020; Paulus et al., 2020; Petersen, 2022), and embedded differentiable solvers (Amos & Kolter, 2017; Agrawal et al., 2019; Paulus et al., 2024). NAR should also be compared in neural combinatorial optimization settings, where algorithmic alignment has been linked to approximation guarantees (Yau et al., 2024; He & Vitercik, 2025; McCarty et al., 2021) and improved OOD performance (He & Vitercik, 2025; Yau et al., 2024; Veličković et al., 2020).

**Actionable objectives.** Well-designed benchmarks should assess whether integrating a neural component via an explicit NAR module improves downstream performance, generalization, provides guarantees, or yields more stable optimization behavior. We propose the following settings.

1. **An end-to-end predict-then-optimize pipeline.** In this setting, a model predicts the parameters of a problem instance, such as edge weights, constraints, or costs, and a bespoke algorithm solves the resulting problem. Gradient estimation then enables backpropagation through the full pipeline. A canonical case study is the Warcraft shortest-path problem (Pogancic et al., 2020), where a model predicts terrain-dependent traversal costs from map pixels, and a shortest-path algorithm computes the solution. Similar settings arise in vehicle routing (Baty et al., 2024), integer programming (Paulus et al., 2021), and Sudoku (Wang et al., 2019; Paulus et al., 2024), comparing NAR against black-box backpropagation and related approaches.
2. **Neural combinatorial optimization (NCO) benchmarks.** NAR pipelines can be directly compared to state-of-the-art NCO baselines on classic combinatorial optimization problems such as Max-Cut (Nath & Kuhnle, 2024; Tönshoff et al., 2022), Graph Coloring (Tönshoff et al., 2022; Schuetz et al., 2022), Traveling Salesperson (Li et al., 2024; Ma et al., 2025), Boolean Satisfiability (Li et al., 2023; Tönshoff et al., 2022), and Maximum Independent Set (Li et al., 2024; Ma et al., 2025). OOD generalization is a key empirical concern in NAR. State-of-the-art benchmarking for NCO often considers generalization both to larger instances (Tönshoff et al., 2022; Li et al., 2024) and to different instance types (Li et al., 2023; Yolcu & Póczos, 2019; Yau et al., 2024), offering a strong test of whether NAR pipelines truly leverage their inductive bias for OOD generalization.

*Figure 2.* Warcraft optimal-cost accuracy.

| Grid | LPGD | NAR | NAR+Dijkstra |
|---|---|---|---|
| 12×12 | 0.940 | 0.967 | 0.867 |
| 18×18 | 0.919 | 0.963 | 0.842 |
| 24×24 | 0.901 | 0.946 | 0.860 |
| 30×30 | 0.903 | 0.939 | 0.874 |

**Illustrative proof-of-concept.** We illustrate how the *predict-then-optimize* framework enables cross-paradigm comparison on a Warcraft shortest-path benchmark (Pogancic et al., 2020). We reformulate the problem to first *predict* a terrain-dependent cost map from an input image, after which a shortest-path module *converts* the predicted costs into the final path. This formulation allows direct comparison between LPGD (Paulus et al., 2024), which trains through an exact shortest-path solver via black-box differentiation, and NAR, which replaces the solver with a learned Bellman–Ford-embedded graph executor. Using optimal-cost accuracy in Figure 2, we find that NAR outperforms LPGD across all grid sizes. An additional ablation that decodes NAR's predicted costs using Dijkstra's algorithm (NAR+Dijkstra) performs substantially worse, suggesting that the gain also comes from the learned executor's exploitation of latent algorithmic structure. Details of the experiment are in Appendix B.

### 4.2. Better metrics to track practical utility

Current NAR evaluations rely on narrow metrics (primarily final-output accuracy), test size generalization (CLRS

(Veličković et al., 2022), SALSA-CLRS (Minder et al., 2023)) or different distributions (GraphBench (Stoll et al., 2025)), lacking metrics that reflect practical values, such as runtime efficiency, robustness, and anytime behavior.

**Actionable objectives.** To bridge the gap between synthetic algorithmic simulation and real-world readiness, we propose that NAR benchmarks move beyond the current convention on final accuracy and incorporate more diverse difficulty axes and evaluation metrics.

1. **Define diverse difficulty axes that reflect real distributions**. Introduce controlled out-of-distribution axes with progress measures to better reflect the diversity of real-world data. Some examples include (i) *Sparsity and structure:* Evaluate across graph families with distinct sparsity and structural properties, reflecting heterogeneous real-world structure; (ii) *Reasoning depth:* Test whether models trained on shallow steps can compose many steps and recover from intermediate errors; (iii) *Encoding mismatch:* Assess generalization across input representations (e.g., integer to floating-point weights, synthetic to textual/real-world inputs).
2. **Use metrics that track practical utility.** Move beyond final algorithmic simulation accuracy. For example, Aggarwal et al. (2025) proposes Generalization Out-of-Distribution (GOOD), evaluating performance as graph size grows via the area under a score–distribution curve. More broadly, metrics should capture anytime behavior (e.g., quality-time AUC), hard failures (e.g., constraint-violation rates), robustness to structured shifts, and efficiency metrics tailored to the use case.

### 4.3. From synthetic algorithms to high-impact domains

A key motivation of NAR is embedding algorithmic knowledge into real-world learning systems, with success in planning (Deac et al., 2021; He et al., 2022), network configuration (Beurer-Kellner et al., 2022), and computational neuroscience (Numeroso et al., 2023). Yet benchmarks focus mainly on synthetic algorithmic tasks, leaving a gap in real-world domains.

**Actionable objectives.** We outline two objectives for evaluating NAR in high-impact domains.

1. **Design datasets from high-impact domains.** Evaluate on tasks where algorithmic procedures already play a central role, such as Electronic Design Automation (EDA), where chip-design tools rely on efficient algorithms and fine-tuned heuristics. More broadly, datasets should test whether NAR modules can be integrated into practical end-to-end pipelines for joint optimization. We further elaborate on large language models, floorplanning, and molecular tasks, in Appendix C.
2. **Evaluate NAR in deployment-like settings.** Beyond standalone benchmark performance, NAR should be assessed under realistic constraints such as runtime, latency, efficiency, robustness, and integration cost. Some advantages of NAR may only appear when embedded in domain-specific pipelines, so meaningful evaluation will often require collaboration with domain experts.

## 5. Alternative views

The success of scaling has driven major progress in large models (Kaplan et al., 2020). One possible perspective may view that NAR is unnecessary if data and compute alone enable generalization across algorithmic and reasoning tasks (Wei et al., 2022). We argue that, as data and compute limits emerge (Villalobos et al., 2024), NAR offers a complementary path through algorithm-centric architectures (Eberle et al., 2025) with iterative algorithmic priors that may bring better generalization, sample efficiency, and interpretability (Section 2.1).

NAR is connected to several neighboring paradigms as discussed. A skeptical view may see NAR not as a standalone field, but as an extension or relabeling of these paradigms. We believe this concern exactly motivates clearer scoping rather than undermining NAR. Our goal is not to isolate NAR from neighboring paradigms, but to identify cases that benefit in distinct ways from learned algorithmic execution. This is the role of our neuralization test in Section 2.3: to clarify what is being neuralized, why adjacent alternatives are insufficient, and what value learned execution provides.

One may also argue that NAR could benefit from remaining loosely defined or, more broadly, from including any algorithmic components (learnable or not) within machine learning pipelines. This view encompasses works such as fixed algorithms for optimizable systems with efficiency and correctness guarantees (Ong et al., 2024), as well as studies of algorithmic understanding (Yang et al., 2024) and mathematical reasoning (Zhou et al., 2024). We argue that precise scoping is necessary to unify NAR and establish it as a rigorous research area, while overlapping work can be systematically compared through cross-benchmarking.

## 6. Conclusion and future directions

Neural algorithmic reasoning (NAR) offers a promising route for connecting classical algorithms with end-to-end trainable neural systems. However, its impact remains limited by unclear scope and fragmented practice. We offer a sharper definition of NAR grounded in neuralized algorithms. This framing clarifies NAR's boundaries and distinguishes it from adjacent fields. We further introduce new theoretical objectives to justify neuralization, empirical proposals for cross-field comparison and practical evaluation, and concrete,

actionable objectives for community-wide collaboration to expand NAR's scientific and practical impact.

*Taken together,* these points reposition NAR as a coherent research program centered on neuralized procedures and rigorous evaluation, with clearer contrast to neighboring communities and clearer standards for what constitutes meaningful progress.

*Looking ahead,* we envision shaping NAR into a field of *differentiable algorithms with guarantees* that rigorously bridges classical algorithm design and differentiable neural architectures, exploits data distributions and modern GPU hardware, and delivers stronger performance guarantees.

## Acknowledgement

We thank Petar Veličković for feedback on this paper, and Erik Müller for creating the figure. YH is supported by a Cubist PhD Fellowship. CM, CQ, TS, and SW are partially funded by a DFG Emmy Noether grant (468502433) and RWTH Junior Principal Investigator Fellowship under Germany's Excellence Strategy.

## References

Aggarwal, M., Jayalath, D., and Jürß, J. Rethinking test generalisation in neural algorithmic reasoning. In *LoG*, 2025.

Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, J. Z. Differentiable convex optimization layers. *NeurIPS*, 32, 2019.

Akyürek, E., Schuurmans, D., Andreas, J., Ma, T., and Zhou, D. What learning algorithm is in-context learning? investigations with linear models. In *ICLR*, 2023.

Alain, G. and Bengio, Y. Understanding intermediate layers using linear classifier probes. *arXiv preprint arxiv:1610.01644*, 2018.

Alvarez, A. M., Louveaux, Q., and Wehenkel, L. A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29(1):185–195, 2017.

Amos, B. and Kolter, J. Z. Optnet: Differentiable optimization as a layer in neural networks. In *ICML*, 2017.

Ansótegui, C., Sellmann, M., and Tierney, K. A gender-based genetic algorithm for the automatic configuration of algorithms. In *International Conference on Principles and Practice of Constraint Programming*, 2009.

Bai, Y., Chen, F., Wang, H., Xiong, C., and Mei, S. Transformers as statisticians: Provable in-context learning with in-context algorithm selection. In *NeurIPS*, 2023.

Balcan, M. Data-driven algorithm design. In Roughgarden, T. (ed.), *Beyond the Worst-Case Analysis of Algorithms*, pp. 626–645. Cambridge University Press, 2020.

Balcan, M.-F., Deblasio, D., Dick, T., Kingsford, C., Sandholm, T., and Vitercik, E. How much data is sufficient to learn high-performing algorithms? *Journal of the ACM*, 71(5):1–58, 2024.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V. F., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gülçehre, Ç., Song, H. F., Ballard, A. J., Gilmer, J., Dahl, G. E., Vaswani, A., Allen, K. R., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M. M., Vinyals, O., Li, Y., and Pascanu, R. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Baty, L., Jungel, K., Klein, P. S., Parmentier, A., and Schiffer, M. Combinatorial optimization-enriched machine learning to solve the dynamic vehicle routing problem with time windows. *Transportation Science*, 58(4):708–725, 2024.

Bengio, Y., Lodi, A., and Prouvost, A. Machine learning for combinatorial optimization: A methodological tour d'horizon. *Eur. J. Oper. Res.*, 290(2):405–421, 2021.

Beurer-Kellner, L., Vechev, M. T., Vanbever, L., and Veličković, P. Learning to configure computer networks with neural algorithmic reasoning. In *NeurIPS*, 2022.

Blondel, M. and Roulet, V. The elements of differentiable programming. *arXiv preprint arXiv:2403.14606*, 2025.

Blondel, M., Teboul, O., Berthet, Q., and Djolonga, J. Fast differentiable sorting and ranking. In *ICML*, 2020.

Cappart, Q., Chételat, D., Khalil, E., Lodi, A., Morris, C., and Veličković, P. Combinatorial optimization and reasoning with graph neural networks. *Journal of Machine Learning Research*, 24:130:1–130:61, 2023.

Caramanis, C., Fotakis, D., Kalavasis, A., Kontonis, V., and Tzamos, C. Optimizing solution-samplers for combinatorial problems: The landscape of policy-gradient method. In *NeurIPS*, 2023.

Chen, B., Li, C., Dai, H., and Song, L. Retro*: Learning retrosynthetic planning with neural guided a* search. In *ICML*, 2020.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms*. MIT Press, 2 edition, 2001.

de Luca, A. B. and Fountoulakis, K. Simulation of graph algorithms with looped transformers. In *ICML*, 2024.

de Luca, A. B., Giapitzakis, G., and Fountoulakis, K. Learning to add, multiply, and execute algorithmic instructions exactly with neural networks. In *NeurIPS*, 2025a.

de Luca, A. B., Giapitzakis, G., Yang, S., Veličković, P., and Fountoulakis, K. Positional attention: Expressivity and learnability of algorithmic computation. In *ICML*, 2025b.

Deac, A., Veličković, P., Milinkovic, O., Bacon, P., Tang, J., and Nikolic, M. Neural algorithmic reasoners are implicit planners. In *NeurIPS*, 2021.

DeepSeek-AI, Liu, A., Mei, A., Lin, B., Xue, B., Wang, B., Xu, B., Wu, B., Zhang, B., Lin, C., Dong, C., Lu, C., Zhao, C., Deng, C., Xu, C., Ruan, C., Dai, D., Guo, D., Yang, D., Chen, D., Li, E., Zhou, F., Lin, F., Dai, F., Hao, G., Chen, G., Li, G., Zhang, H., Xu, H., Li, H., Liang, H., Wei, H., Zhang, H., Luo, H., Ji, H., Ding, H., Tang, H., Cao, H., Gao, H., Qu, H., Zeng, H., Huang, J., Li, J., Xu, J., Hu, J., Chen, J., Xiang, J., Yuan, J., Cheng, J., Zhu, J., Ran, J., Jiang, J., Qiu, J., Li, J., Song, J., Dong, K., Gao, K., Guan, K., Huang, K., Zhou, K., Huang, K., Yu, K., Wang, L., Zhang, L., Wang, L., Zhao, L., Yin, L., Guo, L., Luo, L., Ma, L., Wang, L., Zhang, L., Di, M. S., Xu, M. Y., Zhang, M., Zhang, M., Tang, M., Zhou, M., Huang, P., Cong, P., Wang, P., Wang, Q., Zhu, Q., Li, Q., Chen, Q., Du, Q., Xu, R., Ge, R., Zhang, R., Pan, R., Wang, R., Yin, R., Xu, R., Shen, R., Zhang, R., Liu, S. H., Lu, S., Zhou, S., Chen, S., Cai, S., Chen, S., Hu, S., Liu, S., Hu, S., Ma, S., Wang, S., Yu, S., Zhou, S., Pan, S., Zhou, S., Ni, T., Yun, T., Pei, T., Ye, T., Yue, T., Zeng, W., Liu, W., Liang, W., Pang, W., Luo, W., Gao, W., Zhang, W., Gao, X., Wang, X., Bi, X., Liu, X., Wang, X., Chen, X., Zhang, X., Nie, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yu, X., Li, X., Yang, X., Li, X., Chen, X., Su, X., Pan, X., Lin, X., Fu, X., Wang, Y. Q., Zhang, Y., Xu, Y., Ma, Y., Li, Y., Li, Y., Zhao, Y., Sun, Y., Wang, Y., Qian, Y., Yu, Y., Zhang, Y., Ding, Y., Shi, Y., Xiong, Y., He, Y., Zhou, Y., Zhong, Y., Piao, Y., Wang, Y., Chen, Y., Tan, Y., Wei, Y., Ma, Y., Liu, Y., Yang, Y., Guo, Y., Wu, Y., Wu, Y., Cheng, Y., Ou, Y., Xu, Y., Wang, Y., Gong, Y., Wu, Y., Zou, Y., Li, Y., Xiong, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Wu, Z. F., Ren, Z. Z., Zhao, Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Xie, Z., Zhang, Z., Hao, Z., Gou, Z., Ma, Z., Yan, Z., Shao, Z., Huang, Z., Wu, Z., Li, Z., Zhang, Z., Xu, Z., Wang, Z., Gu, Z., Zhu, Z., Li, Z., Zhang, Z., Xie, Z., Gao, Z., Pan, Z., Yao, Z., Feng, B., Li, H., Cai, J. L., Ni, J., Xu, L., Li, M., Tian, N., Chen, R. J., Jin, R. L., Li, S. S., Zhou, S., Sun, T., Li, X. Q., Jin, X., Shen, X., Chen, X., Song, X., Zhou, X., Zhu, Y. X., Huang, Y., Li, Y., Zheng, Y., Zhu, Y., Ma, Y., Huang, Z., Xu, Z., Zhang, Z., Ji, D., Liang, J., Guo, J., Chen, J., Xia, L., Wang, M., Li, M., Zhang, P., Chen, R., Sun, S., Wu, S., Ye, S., Wang, T., Xiao, W. L., An, W., Wang, X., Sun, X., Wang, X., Tang, Y., Zha, Y., Zhang, Z., Ju, Z., Zhang, Z., and Qu, Z. Deepseek-v3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

Dong, B., Fan, Y., Sun, Y., Li, Z., Pan, T., Xun, Z., and Wang, J. Maximum score routing for mixture-of-experts. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025.

Eberle, O., McGee, T. A., Giaffar, H., Webb, T. W., and Momennejad, I. Position: We need an algorithmic understanding of generative AI. In *ICML*, 2025.

Eisenmann, H. and Johannes, F. Generic global placement and floorplanning. In *Proceedings 1998 Design and Automation Conference. 35th DAC. (Cat. No.98CH36175)*, pp. 269–274, 1998.

Eysenbach, B., Salakhutdinov, R. R., and Levine, S. Search on the replay buffer: Bridging planning and reinforcement learning. In *NeurIPS*, 2019.

Gasse, M., Chételat, D., Ferroni, N., Charlin, L., and Lodi, A. Exact combinatorial optimization with graph convolutional neural networks. In *NeurIPS*, 2019.

Georgiev, D., Barbiero, P., Kazhdan, D., Veličković, P., and Liò, P. Algorithmic concept-based explainable reasoning. In *AAAI*, 2022.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *ICML*, 2017.

Gomes, C. and Selman, B. Algorithm portfolios. *Artificial Intelligence*, 126:43–62, 2001.

Gupta, R. and Roughgarden, T. A PAC approach to application-specific algorithm selection. In *ITCS*, 2016.

He, J., Jenne, H., Vargas, M., Brown, D., Mishne, G., Wang, Y., and Kvinge, H. MINAR: Mechanistic interpretability for neural algorithmic reasoning. In *New Perspectives in Graph Machine Learning*, 2025a.

He, Y. and Vitercik, E. Primal-dual neural algorithmic reasoning. In *ICML*, 2025.

He, Y., Veličković, P., Liò, P., and Deac, A. Continuous neural algorithmic planners. In *LoG*, 2022.

He, Y., Li, Y., White, C., and Vitercik, E. Can LLMs reason structurally? an evaluation via the lens of data structures. *arXiv preprint arXiv:2505.24069*, 2025b.

Huang, G., Hu, J., He, Y., Liu, J., Ma, M., Shen, Z., Wu, J., Xu, Y., Zhang, H., Zhong, K., Ning, X., Ma, Y., Yang, H., Yu, B., Yang, H., and Wang, Y. Machine learning for electronic design automation: A survey. *ACM Trans. Des. Autom. Electron. Syst.*, 26(5), June 2021.

Huben, R., Cunningham, H., Smith, L. R., Ewart, A., and Sharkey, L. Sparse autoencoders find highly interpretable features in language models. In *ICLR*, 2024.

Hutter, F., Hoos, H. H., Leyton-Brown, K., and Stützle, T. ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1): 267–306, 2009. ISSN 1076-9757.

Kadioglu, S., Malitsky, Y., Sellmann, M., and Tierney, K. ISAC-instance-specific algorithm configuration. In *ECAI*, 2010.

Kahng, A. B., Lienig, J., Markov, I. L., and Hu, J. *VLSI physical design: from graph partitioning to timing closure*, volume 312. Springer, 2011.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models, 2020.

Kedem, G. and Watanabe, H. Graph-optimization techniques for ic layout and compaction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 3(1):12–20, 1984. doi: 10.1109/TCAD.1984.1270052.

Khalil, E., Le Bodic, P., Song, L., Nemhauser, G., and Dilkina, B. Learning to branch in mixed integer programming. In *AAAI*, 2016.

Khalil, E. B., Dai, H., Zhang, Y., Dilkina, B., and Song, L. Learning combinatorial optimization algorithms over graphs. In *NeurIPS*, 2017.

Kim, J., Seo, Y., and Shin, J. Landmark-guided subgoal generation in hierarchical reinforcement learning. In *NeurIPS*, 2021.

Kleinhans, J., Sigl, G., Johannes, F., and Antreich, K. Gordian: Vlsi placement by quadratic programming and slicing optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(3):356–365, 1991.

Kool, W., van Hoof, H., and Welling, M. Attention, learn to solve routing problems! In *ICLR*, 2019.

Lavagno, L., Scheffer, L., and Martin, G. *EDA for IC Implementation, Circuit Design, and ProcessTechnology (Electronic Design Automation for Integrated Circuits Handbook)*. CRC Press, Inc., USA, 2006. ISBN 0849379245.

Li, Y., Guo, J., Wang, R., Zha, H., and Yan, J. Fast t2t: Optimization consistency speeds up diffusion-based training-to-testing solving for combinatorial optimization. *NeurIPS*, 2024.

Li, Z., Guo, J., and Si, X. G4satbench: Benchmarking and advancing sat solving with graph neural networks. *arXiv preprint arXiv:2309.16941*, 2023.

Lin, S., Liu, J., and Wong, M. D. Gamer: Gpu accelerated maze routing. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–8, 2021.

Lobjois, L. and Lemaître, M. Branch and bound algorithm selection by performance prediction. In *AAAI*, 1998.

Loukas, A. What graph neural networks cannot learn: depth vs width. In *ICLR*, 2020.

Lu, Y.-C., Hsiao, H.-H., and Ren, H. Invited paper: LLM-enhanced gpu-optimized physical design at scale. In *ICCAD*, 2025.

Ma, C.-Y. and Liao, C.-S. A review of protein-protein interaction network alignment: From pathway comparison to global alignment. *Computational and Structural Biotechnology Journal*, 18:2647–2656, 2020.

Ma, J., Pan, W., Li, Y., and Yan, J. Coexpander: Adaptive solution expansion for combinatorial optimization. In *ICML*, 2025.

Mandi, J., Kotary, J., Berden, S., Mulamba, M., Bucarey, V., Guns, T., and Fioretto, F. Decision-focused learning: Foundations, state of the art, benchmark and future opportunities. *Journal of Artificial Intelligence Research*, 2024.

Markeeva, L., McLeish, S., Ibarz, B., Bounsi, W., Kozlova, O., Vitvitskyi, A., Blundell, C., Goldstein, T., Schwarzschild, A., and Veličković, P. The CLRS-text algorithmic reasoning language benchmark. *arXiv preprint arXiv:2406.04229*, 2024.

Marple, D., Smulders, M., and Hegen, H. Tailor: A layout system based on trapezoidal corner stitching. *IEEE transactions on computer-aided design of integrated circuits and systems*, 9(1):66–90, 1990.

Martins, R. M. F. and Lourenço, N. C. C. Analog integrated circuit routing techniques: An extensive review. *IEEE Access*, 11:35965–35983, 2023.

McCarty, E., Zhao, Q., Sidiropoulos, A., and Wang, Y. Nnbaker: A neural-network infused algorithmic framework for optimization problems on geometric intersection graphs. *NeurIPS*, 2021.

Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., and Alon, U. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

Minder, J., Grötschla, F., Mathys, J., and Wattenhofer, R. SALSA-CLRS: A sparse and scalable benchmark for algorithmic reasoning. In *LoG*, 2023.

Mirjanic, V. V., Pascanu, R., and Veličković, P. Latent space representations of neural algorithmic reasoners. In *LoG*, 2023.

Mitzenmacher, M. and Vassilvitskii, S. Algorithms with predictions. In Roughgarden, T. (ed.), *Beyond the Worst-Case Analysis of Algorithms*, pp. 646–662. Cambridge University Press, 2020.

Naidu, P., Wang, Z., Bergen, L., and Paturi, R. Quiet feature learning in algorithmic tasks. In *Mechanistic Interpretability Workshop at NeurIPS 2025*, 2025.

Nanda, N., Chan, L., Lieberum, T., Smith, J., and Steinhardt, J. Progress measures for grokking via mechanistic interpretability. In *ICLR*, 2023.

Nath, A. and Kuhnle, A. A benchmark for maximum cut: Towards standardization of the evaluation of learned heuristics for combinatorial optimization. *arXiv preprint arXiv:2406.11897*, 2024.

Nerem, R. R., Chen, S., Dasgupta, S., and Wang, Y. Graph neural networks extrapolate out-of-distribution for shortest paths. *arXiv preprint arXiv:abs/2503.19173*, 2025.

Niepert, M., Minervini, P., and Franceschi, L. Implicit MLE: backpropagating through discrete exponential family distributions. *NeurIPS*, 2021.

Numeroso, D., Bacciu, D., and Veličković, P. Learning heuristics for A*, 2022.

Numeroso, D., Bacciu, D., and Veličković, P. Dual algorithmic reasoning. In *ICLR*, 2023.

Ong, E., Huszár, F., Lio, P., and Veličković, P. Parallelising differentiable algorithms removes the scalar bottleneck: A case study. In *ICML 2024 Workshop on Differentiable Almost Everything: Differentiable Relaxations, Algorithms, Operators, and Simulators*, 2024.

Parisotto, E., Mohamed, A., Singh, R., Li, L., Zhou, D., and Kohli, P. Neuro-symbolic program synthesis. *arXiv preprint arXiv:abs/1611.01855*, 2016.

Paulus, A., Rolínek, M., Musil, V., Amos, B., and Martius, G. Comboptnet: Fit the right np-hard problem by learning integer programming constraints. In *ICML*, 2021.

Paulus, A., Martius, G., and Musil, V. Lpgd: A general framework for backpropagation through embedded optimization layers. *arXiv preprint arXiv:2407.05920*, 2024.

Paulus, M., Choi, D., Tarlow, D., Krause, A., and Maddison, C. J. Gradient estimation with stochastic softmax tricks. *NeurIPS*, 2020.

Petersen, F. Learning with differentiable algorithms. *arXiv preprint arXiv:2209.00616*, 2022.

Pogancic, M. V., Paulus, A., Musil, V., Martius, G., and Rolínek, M. Differentiation of blackbox combinatorial solvers. In *ICLR*, 2020.

Qharabagh, M. F., de Luca, A. B., Giapitzakis, G., and Fountoulakis, K. Learning to execute graph algorithms exactly with graph neural networks, 2026.

Reed, J., Sangiovanni-Vincentelli, A., and Santomauro, M. A new symbolic channel router: Yacr2. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 4(3):208–219, 1985.

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J. R., Ellenberg, J. S., Wang, P., Fawzi, O., Kohli, P., and Fawzi, A. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Sandholm, T. Very-large-scale generalized combinatorial multi-attribute auctions: Lessons from conducting $60 billion of sourcing. In Neeman, Z., Roth, A., and Vulkan, N. (eds.), *Handbook of Market Design*. Oxford University Press, 2013.

Savinov, N., Dosovitskiy, A., and Koltun, V. Semi-parametric topological memory for navigation. In *ICLR*, 2018.

Scavuzzo, L., Aardal, K. I., Lodi, A., and Yorke-Smith, N. Machine learning augmented branch and bound for mixed integer linear programming. *arXiv preprint arXiv:2402.05501*, 2024.

Schuetz, M. J., Brubaker, J. K., Zhu, Z., and Katzgraber, H. G. Graph coloring with physics-inspired graph neural networks. *Physical Review Research*, 4(4):043131, 2022.

Schutz, A., Darvariu, V.-A., Panagiotaki, E., Lacerda, B., and Hawes, N. Tackling GNARLy problems: Graph neural algorithmic reasoning reimagined through reinforcement learning. *arXiv preprint arXiv:2509.18930*, 2025.

Segler, M. H. S., Preuss, M., and Waller, M. P. Planning chemical syntheses with deep neural networks and symbolic ai. *Nature*, 555(7698):604–610, Mar 2018.

Sherwani, N. A. *Algorithms for VLSI physical design automation*. Springer, 1999.

Smet, L. D. and Raedt, L. D. Defining neurosymbolic ai. *arXiv preprint arXiv:2507.11127*, 2025.

Stoll, T., Qian, C., Finkelshtein, B., Parviz, A., Weber, D., Frasca, F., Shavit, H., Siraudin, A., Mielke, A., Anastacio, M., Müller, E., Bechler-Speicher, M., Bronstein, M., Galkin, M., Hoos, H., Niepert, M., Perozzi, B., Tönshoff, J., and Morris, C. Graphbench: Next-generation graph learning benchmarking. *arXiv preprint arXiv:2512.04475*, 2025.

Tönshoff, J., Kisin, B., Lindner, J., and Grohe, M. One model, any csp: graph neural networks as fast global search heuristics for constraint satisfaction. *arXiv preprint arXiv:2208.10227*, 2022.

Veličković, P. and Blundell, C. Neural algorithmic reasoning. *Patterns*, 2(7):100273, 2021.

Veličković, P., Ying, R., Padovano, M., Hadsell, R., and Blundell, C. Neural execution of graph algorithms. In *ICLR*, 2020.

Veličković, P., Badia, A. P., Budden, D., Pascanu, R., Banino, A., Dashevskiy, M., Hadsell, R., and Blundell, C. The CLRS algorithmic reasoning benchmark. In *ICML*, 2022.

Villalobos, P., Ho, A., Sevilla, J., Besiroglu, T., Heim, L., and Hobbhahn, M. Position: Will we run out of data? limits of LLM scaling based on human-generated data. In *ICML*, 2024.

Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks. In *NeurIPS*, 2015.

Wang, P.-W., Donti, P., Wilder, B., and Kolter, Z. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *ICML*, 2019.

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., and Fedus, W. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856.

Xu, K., Li, J., Zhang, M., Du, S. S., Kawarabayashi, K., and Jegelka, S. What can neural networks reason about? In *ICLR*, 2020.

Xu, K., Zhang, M., Li, J., Du, S. S., Kawarabayashi, K.-I., and Jegelka, S. How neural networks extrapolate: From feedforward to graph neural networks. In *ICLR*, 2021.

Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32(1):565–606, 2008.

Yang, L., Lee, K., Nowak, R. D., and Papailiopoulos, D. Looped transformers are better at learning learning algorithms. In *ICLR*, 2024.

Yau, M., Karalias, N., Lu, E., Xu, J., and Jegelka, S. Are graph neural networks optimal approximation algorithms? In *NeurIPS*, 2024.

Yolcu, E. and Póczos, B. Learning local search heuristics for boolean satisfiability. *NeurIPS*, 32, 2019.

Zaslavskiy, M., Bach, F., and Vert, J.-P. Global alignment of protein-protein interaction networks by graph matching methods. *Bioinformatics*, 25(12):i259–67, June 2009.

Zhang, W. and Dietterich, T. G. A reinforcement learning approach to job-shop scheduling. In *IJCAI*, 1995.

Zhong, Z., Liu, Z., Tegmark, M., and Andreas, J. The clock and the pizza: Two stories in mechanistic explanation of neural networks. In *NeurIPS*, 2023.

Zhou, H., Bradley, A., Littwin, E., Razin, N., Saremi, O., Susskind, J. M., Bengio, S., and Nakkiran, P. What algorithms can transformers learn? a study in length generalization. In *ICLR*, 2024.

Zhou, Y., Lei, T., Liu, H., Du, N., Huang, Y., Zhao, V. Y., Dai, A. M., Chen, Z., Le, Q. V., and Laudon, J. Mixture-of-experts with expert choice routing. In *NeurIPS*, 2022.

Zitnik, M., Li, M. M., Wells, A., Glass, K., Morselli Gysi, D., Krishnan, A., Murali, T. M., Radivojac, P., Roy, S., Baudot, A., Bozdag, S., Chen, D. Z., Cowen, L., Devkota, K., Gitter, A., Gosline, S. J. C., Gu, P., Guzzi, P. H., Huang, H., Jiang, M., Kesimoglu, Z. N., Koyuturk, M., Ma, J., Pico, A. R., Pržulj, N., Przytycka, T. M., Raphael, B. J., Ritz, A., Sharan, R., Shen, Y., Singh, M., Slonim, D. K., Tong, H., Yang, X. H., Yoon, B.-J., Yu, H., and Milenković, T. Current and future directions in network biology. *Bioinformatics Advances*, 4(1):vbae099, 08 2024. ISSN 2635-0041.

## A. Additional related work

To the best of our knowledge, Cappart et al. (2023); Veličković & Blundell (2021) gave the first definition of NAR, building on earlier work on neural executions of graph algorithms (Veličković et al., 2020) and on the reasoning capabilities of (graph) neural networks; see (Battaglia et al., 2018; Xu et al., 2020; 2021) and the references therein.

Several adjacent—though complementary—research directions also aim to combine algorithms with machine-learning predictions, stretching back to the 1990s (Zhang & Dietterich, 1995). A long line of work uses machine learning to reduce runtimes for finding high-quality solutions to NP-hard computational problems. Early approaches focused on algorithm selection (e.g., Lobjois & Lemaître, 1998; Gomes & Selman, 2001; Xu et al., 2008; Kadioglu et al., 2010; Sandholm, 2013) and the configuration of highly parameterized solvers (e.g., Hutter et al., 2009; Ansótegui et al., 2009). With the rise of deep learning, attention shifted toward embedding learned components into exact solvers for NP-hard problems (e.g., Khalil et al., 2016; Alvarez et al., 2017; Gasse et al., 2019; Bengio et al., 2021; Cappart et al., 2023; Scavuzzo et al., 2024), as well as toward *neural combinatorial optimization* (e.g., Vinyals et al., 2015; Khalil et al., 2017; Kool et al., 2019), which aims to learn polynomial-time policies and heuristics that return high-quality feasible solutions to NP-hard problems. In contrast, NAR has a different objective: rather than using learning primarily to accelerate search, it aims to incorporate algorithmic reasoning into end-to-end trainable systems, with an emphasis on systematic generalization.

A related set of paradigms focuses on enabling end-to-end optimization through discrete computations by making algorithmic components differentiable. In particular, *black-box differentiation* (Pogancic et al., 2020) and *differentiable programming* (Blondel & Roulet, 2025) develop techniques for discrete differentiation and differentiable computer programs, respectively. In contrast to NAR, which trains neural networks to perform algorithmic computations, these paradigms focus on differentiating given (often hand-specified) discrete computations.

While the aforementioned research is primarily empirical, a growing body of work studies the interplay between machine learning and algorithm design from a theoretical perspective. For example, *algorithms with predictions* (Mitzenmacher & Vassilvitskii, 2020) incorporate learned predictions into classical algorithms as "advice" to obtain improved performance guarantees when the predictions are accurate, while retaining worst-case robustness. Relatedly, *data-driven algorithm design* develops theory for using machine learning to select or configure parameterized algorithms (Gupta & Roughgarden, 2016; Balcan, 2020; Balcan et al., 2024). These methods retain an explicit algorithmic template and use learning to provide advice or select parameters with provable guarantees, whereas NAR aims to build neural systems that internalize or execute the computation itself, emphasizing learned execution rather than worst-case performance guarantees.

Benchmarks for NAR mostly focus on synthetic algorithmic tasks. The CLRS Benchmark (Veličković et al., 2022) contains 30 classical algorithms from the CLRS textbook (Cormen et al., 2001) spanning sorting, string, graph, geometry, dynamic programming, and other algorithmic domains. SALSA-CLRS (Minder et al., 2023) extends CLRS with distributed and randomized algorithms. GraphBench (Stoll et al., 2025) provides large-scale graph algorithmic tasks with explicit difficulty tiers and size generalization splits. CLRS-Text (Markeeva et al., 2024) translates CLRS into text descriptions for LLMs. DSR-Bench (He et al., 2025b) evaluates LLMs through data-structure operations for fine-grained diagnosis. Despite broader task coverage, a unified standard is missing.

## B. Warcraft experiment details

To demonstrate the utility of our proposed *predict-then-optimize* framework in Section 4.1, we provide a proof-of-concept example on the Warcraft shortest-path benchmark (Pogancic et al., 2020). The input is a terrain image, and the task is to predict a valid source-to-target shortest path on a pixel grid. Instead of directly predicting the path mask, we reformulate the problem as predicting a cost map from the input image, after which a shortest-path module converts the predicted costs into the final path.

We compare two approaches within this framework. The first, LPGD (Paulus et al., 2024), uses an exact shortest-path solver as the inference module and trains through the optimization layer using the LPGD update. In this setting, LPGD can be viewed as a black-box differentiation method applied to the original benchmark. The second, NAR, replaces the exact solver with a learned Bellman–Ford-biased graph executor using min-style cost propagation and predecessor prediction. We report *optimal-cost accuracy*: a prediction is counted as correct if it is a valid source-to-target path and has the same cost as the ground-truth shortest path under the true vertex weights.

Table 2 shows that the NAR model outperforms the direct solver-based approach across all grid sizes. This suggests

that replacing the shortest-path solver with a GNN algorithmically biased toward shortest-path computation can improve downstream path prediction, while also providing a useful cross-paradigm comparison within the predict-then-optimize framework.

To test whether the gain comes only from improved cost prediction, we perform an ablation in which the costs predicted by NAR are decoded using Dijkstra's algorithm rather than the model's learned predecessor predictions. We see that the Dijkstra-decoding ablation performs substantially worse than the full NAR model. This indicates that NAR does not merely learn costs that are useful for a classical shortest-path solver; rather, it also appears to exploit latent algorithmic structure in its learned executor to produce better downstream predictions. These findings support our scoping of NAR use cases in Section 2 and suggest that predict-then-optimize is a promising avenue for NAR applications and future benchmarking.

### B.1. Detailed Breakdown.

We evaluate shortest-path prediction on the Warcraft shortest-path benchmark, where each instance is a node-weighted grid graph derived from a map image. The source is fixed at the top-left node and the target at the bottom-right node. We consider grid sizes $k \in \{12,18,24,30\}$ under 8-neighbor connectivity.

All methods share the same visual front end. Given an input image $x$, the encoder predicts a raw cost map $\hat{c} = f_\theta(x)$, which is converted to nonnegative node costs via the pointwise transform $\tilde{c} = |\hat{c}|$. In all reported runs, we use the `combresnet18` encoder.

**Exact-solver baselines.** For `bb` and `lpgd`, the predicted cost map $\tilde{c}$ is passed directly to Dijkstra's algorithm, yielding a hard shortest-path mask $\hat{y} = \mathrm{SP}(\tilde{c})$. Since this mapping is discrete, gradients are supplied by a custom surrogate backward pass. If $g = \partial\mathcal{L}/\partial\hat{y}$ denotes the upstream gradient and $\tau > 0$ is a temperature parameter, the implemented blackbox-style vector–Jacobian product is

$$\frac{\partial\mathcal{L}}{\partial\tilde{c}} \approx -\frac{\mathrm{SP}(\tilde{c}) - \mathrm{SP}(\tilde{c} + \tau g)}{\tau}.$$

We use $\tau = 20\,0$ throughout.

A crucial implementation detail is that, in this codebase, `lpgd` is mapped to `lpgd_lower`, and both `bb` and `lpgd` follow the same backward branch. Thus, for this shortest-path implementation, `bb` and `lpgd` are effectively the same differentiable solver method; any small numerical differences should therefore be interpreted as run-to-run variability rather than as a substantive algorithmic distinction.

**Bellman–Ford-aligned neural executor.** Our learned executor, `nar_min_sparse`, replaces the exact solver with a recurrent neural procedure that is explicitly structured around Bellman–Ford-style relaxation. For each node $v$, the model constructs local input features from the predicted cost $\tilde{c}_v$, a source indicator, a target indicator, and normalized spatial coordinates. These features are projected to an initial hidden state $h_v^{(0)} \in \mathbb{R}^d$, with hidden dimension $d = 32$ in all reported experiments.

In addition to hidden states, the executor maintains explicit distance estimates. If $s$ denotes the source, then initialization is $d_s^{(0)} = \tilde{c}_s$ and $d_v^{(0)} = M$ for all $v \neq s$, where $M$ is a large constant. At iteration $t$, each incoming neighbor $u \in \mathcal{N}^-(v)$ proposes an updated distance to node $v$ according to

$$p_{u\to v}^{(t)} = d_u^{(t)} + \tilde{c}_v + \delta_{u\to v}^{(t)}, \qquad \delta_{u\to v}^{(t)} = \alpha \tanh\Big(\psi_{\mathrm{corr}}(h_u^{(t)}, h_v^{(t)}, d_u^{(t)}, \tilde{c}_u, \tilde{c}_v, r_{uv})\Big),$$

where $r_{uv}$ is the relative coordinate offset, $\psi_{\mathrm{corr}}$ is an MLP, and the correction scale is fixed to $\alpha = 0\,1$. The distance update then follows a Bellman–Ford-style relaxation,

$$d_v^{(t+1)} = \min\left(d_v^{(t)}, \min_{u\in\mathcal{N}^-(v)} p_{u\to v}^{(t)}\right),$$

with the source distance clamped after every iteration so that $d_s^{(t+1)} = \tilde{c}_s$.

Hidden-state updates are also proposal-aware. For each incoming edge $u \to v$, the executor computes a message feature $m_{u\to v}^{(t)} = \psi_{\mathrm{msg}}(h_u^{(t)}, h_v^{(t)}, p_{u\to v}^{(t)}, r_{uv})$, and aggregates these messages with a softmin weighting over proposals. Concretely, the edge weights are proportional to $\exp(-p_{u\to v}^{(t)}/\tau_{\mathrm{agg}})$, with aggregation temperature $\tau_{\mathrm{agg}} = 0\,25$, so lower-cost proposals

contribute more strongly. The aggregated message is combined with the current hidden state, the local node features, and the updated distance estimate through an MLP residual update followed by layer normalization.

The executor also predicts predecessor logits for each node. These logits combine a learned compatibility score with an explicit bias toward low-cost proposals, so that predecessor choices remain tied to the relaxation dynamics. The final predecessor distribution is converted into a differentiable soft path by initializing unit mass at the target node and propagating that mass backward through the predecessor probabilities for a fixed number of steps. A hard path is obtained by taking argmax predecessor choices and tracing them back from target to source. In the implementation, both the number of execution iterations and the number of backtracking steps default to $2k$, so larger grids receive proportionally longer recurrent computation.

**Training objective.** For the exact-solver methods, training uses only a path-level loss. For `nar_min_sparse`, we optimize the composite objective

$$\mathcal{L} = \lambda_{\text{path}}\mathcal{L}_{\text{path}} + \lambda_{\text{dist}}\mathcal{L}_{\text{dist}} + \lambda_{\text{pred}}\mathcal{L}_{\text{pred}} + \lambda_{\text{sparse}}\mathcal{L}_{\text{sparse}}.$$

Here, $\mathcal{L}_{\text{path}}$ is a Hamming-style loss between the predicted soft path and the ground-truth shortest-path mask; $\mathcal{L}_{\text{dist}}$ is a Smooth-$L_1$ loss between predicted distance traces and exact Bellman–Ford teacher traces; $\mathcal{L}_{\text{pred}}$ is a cross-entropy loss on local predecessor labels extracted from the same teacher; and $\mathcal{L}_{\text{sparse}}$ is an $L_1$ penalty on the executor MLP parameters, encouraging a sparse learned update rule.

The Bellman–Ford teacher is computed exactly from the ground-truth node costs and provides, at every iteration, both the current shortest-known distance at each node and the predecessor responsible for that update. In the reported runs, we set $\lambda_{\text{path}} = 1\,0$, $\lambda_{\text{dist}} = 0\,1$, $\lambda_{\text{pred}} = 0\,2$, and $\lambda_{\text{sparse}} = 10^{-7}$.

**Optimization and evaluation.** All reported runs use Adam with learning rate $5 \times 10^{-4}$, zero weight decay, gradient clipping at norm $1\,0$, batch size 32, and 50 training epochs. All experiments are run with a single random seed. Our primary evaluation metric is *optimal-cost accuracy*, i.e., the fraction of instances for which the decoded path attains the true optimal path cost. We also report *valid path rate*, the fraction of decoded outputs that form a valid source-to-target path, and *decoded optimal-cost accuracy*, obtained by running exact Dijkstra on the model's predicted cost map rather than on the executor's own predecessor chain. The latter is particularly useful for separating poor cost prediction from imperfect learned execution.

**Model selection and reporting.** During training, the model is evaluated after each epoch on the validation split, and the checkpoint with the highest validation `optimal_cost_accuracy` is retained. After training, this selected checkpoint is reloaded and evaluated once on the test split. Thus, validation performance is used for model selection, while the final reported numbers are test-set results.

**Implementation note.** In the saved outputs, `eval_split=val` denotes the split used for model selection, and `report_split=test` denotes the split used for final reporting. Accordingly, the reported `final_eval` metrics correspond to test-set performance of the checkpoint selected on validation.

## C. High-impact domains for NAR

We discuss several high-impact domains that are suitable for designing datasets for NAR.

**Large (Language) Models.** Recent work on LLMs shows that agentic setups and a mixture of expert (MoE) models are of particular interest. As suggested by Dong et al. (2025) and Zhou et al. (2022), the optimal execution path of such models can be found by solving a linear optimization problem or applying the Sinkhorn algorithm to an optimal transport problem. By connecting NAR to these problems, a fully differentiable pipeline could be generated and optimized directly. Furthermore, recent results from agentic task synthesis processes (DeepSeek-AI et al., 2025) allow the benefits of NAR pipelines over classical algorithms to be evaluated. In addition, reinforcement learning algorithms and paradigms such as hierarchical reinforcement learning and agent buffers (Eysenbach et al., 2019) can be improved by incorporating shortest-path algorithms (Kim et al., 2021; Savinov et al., 2018) into the reinforcement learning algorithm. By using classical algorithms within the reinforcement learning framework, NAR could provide a fully differentiable pipeline or, post-training, directly support reinforcement learning. However, the impacts of NAR-based algorithms for LLM training and reinforcement learning are currently not well explored and implemented.

**Chip design.** Chip design involves many hard optimization problems, such as placement (Eisenmann & Johannes, 1998; Kleinhans et al., 1991), constrained graph partitioning (Kahng et al., 2011), routing (Lin et al., 2021), and other combinatorial or geometric optimization tasks (Lavagno et al., 2006). Many Electronic Design Automation (EDA) tools for chip design rely on efficient algorithms with fine-tuned heuristics to solve hard optimization problems, making them natural candidates for NAR's deployment, especially in VLSI design (Sherwani, 1999). For example, during the derivation of an IC chip layout, graph optimization (Kedem & Watanabe, 1984) or corner-stitching algorithms can be used to improve layout design and performance (Marple et al., 1990). Moreover, in design space estimation for EDA, previous work has used genetic algorithms or simulated annealing to enable search for a suitable design space (Huang et al., 2021). Finally, routing requires the application of specialized algorithms suitable for exploring the given search space (Reed et al., 1985) or routing procedure (Martins & Lourenço, 2023), enabling the usage of NAR-based search and path algorithms. With various heuristics and algorithms used in layout design flows and electronic circuit simulation, single steps or complete EDA processes have been evaluated using machine learning methods (Huang et al., 2021; Lu et al., 2025). However, the addition of NAR-based algorithms to existing EDA pipelines or to enhance machine learning-based algorithms has not been evaluated so far and would represent a promising step towards fully machine learning-based EDA systems.

**Materials & molecular design.** The development and optimization of molecules cover a wide range of applications, including biophysics, drug discovery, and nanotechnology. Design at the molecular level naturally supports graph representations and related classical algorithms, such as graph traversal, cycle detection, motif extraction, and constraint satisfaction, making NAR a promising approach for molecular property inference and optimization. For example, a crucial metric in such tasks is whether the output is a *valid* molecule, alongside additional constraints such as stability, non-toxicity, magnetic properties, etc. Consequently, tasks such as material structure prediction and molecular generation can be formulated as constraint satisfaction problems (CSPs). Numeroso et al. (2023) learn the primal and dual formulations of CSPs simultaneously via NAR, leveraging complementary information to improve performance across both tasks. This may provide a blueprint on how NAR can help solve complex combinatorial problems over structural prediction or generation tasks with hard constraints.

**Network biology.** Many tasks in drug discovery complement molecular design with tasks that require searching or planning over network representations of biological systems, such as protein-protein interaction (PPI) or gene regulatory networks (GRN). NAR is a powerful tool for searching and planning over such biological networks more efficiently. One such task is retrosynthesis, which involves identifying reactants and viable biochemical reaction pathways to synthesize a target compound. Retrosynthesis can essentially be modeled as a constrained path-finding problem on (hyper)-graphs, where state-of-the-art methods leverage algorithms like MCTS and A$^*$ to search the solution space effectively: Segler et al. (2018) replace hand-crafted search heuristics with MCTS driven by a policy network, while Chen et al. (2020) effectively demonstrates A$^*$ as a more efficient alternative to MCTS for retrosynthesis. Numeroso et al. (2022) in turn use NAR to learn better heuristics for A$^*$ to outperform Dijkstra's algorithm for path finding. NAR can help go beyond static heuristics, and *learn-to-search* in a not only efficient but also generalizable manner across larger or different chemical spaces for retrosynthesis tasks.

NAR can also help with substructural recognition and motif extraction, which are essential components for applying machine learning to network biology tasks such as PPI alignment (Milo et al., 2002). Network alignment tasks aim to leverage topological similarities between networks from different sources and are essential for transferring biological knowledge across species or for understanding relationships between different species (Ma & Liao, 2020). PPI alignment in particular is a task very much analogous to graph matching (Zaslavskiy et al., 2009), one of the key tasks in the CLRS NAR benchmark. NAR could also emulate graph-traversal methods essential to motif extraction. Zitnik et al. (2024) also argue, in their discussion of future directions in network biology, that NAR, in particular, is well-suited to tackle different paradigms in network biology in a unified manner.