Position: Neural Algorithmic Reasoning Requires a Clear Scope, Theoretical Foundations, and Empirical Rigor

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

In recent years, growing interest has focused on whether neural architectures can execute discrete algorithms and thereby integrate algorithmic reasoning into end-to-end trainable neural pipelines, a direction referred to as *neural algorithmic reasoning (NAR)*. The field has shown promising early results across multiple applied domains. However, NAR still lacks a clear objective and scope, leading to fragmented efforts and unclear connections to adjacent areas, such as differentiable programming and machine learning for combinatorial optimization. Moreover, the theoretical foundations of NAR remain limited with few guarantees regarding correctness, generalization, or failure modes. In parallel, benchmarking protocols are under-standardized for assessing meaningful real-world progress. Hence, in this position paper, we argue that for NAR to realize its potential, its scope should be explicitly defined and systematically contrasted with neighboring communities. We further propose grounding NAR in the study of neuralized algorithms, meaning differentiable algorithmic procedures with principled inductive biases and formal guarantees. Finally, we call for rigorous empirical practice, centered on standardized benchmarks and protocols, that enable consistent, interpretable, and impactful measurements of progress.

1. Introduction

Neural algorithmic reasoning (NAR) is an emerging field connecting classical algorithm design with neural computation (Veličković et al., 2020; Xu et al., 2020; Veličković & Blundell, 2021). NAR aims to let neural networks learn, execute, and generalize discrete algorithms, integrating algorithmic reasoning into trainable neural systems and enabling end-to-end optimization. Its promise is illustrated by applied successes in diverse fields ranging from computational neuroscience (Numeroso et al., 2023) to computer networking (Beurer-Kellner et al., 2022). Yet, despite its growing popularity, NAR lacks a clear scope and definition, reflected by inconsistent usage across the literature. It 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). These contributions arise from diverse traditions—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. Depending on perspective, the same method may be viewed as neural execution of an algorithm (Veličković et al., 2020) or as a learned heuristic (Khalil et al., 2017). Coupled with limited theoretical grounding, the field is susceptible to fragmentation, overstated claims of novelty, and difficulties in comparing methods.

Present work. We argue that for NAR to realize its potential, it needs a working definition that clarifies its aims and distinguishes it from adjacent communities. The field also needs stronger theoretical foundations to clarify capabilities and limitations, as well as solid empirical evidence to track progress, including evaluation standards and benchmark design. Concretely, we

- propose a working scope for NAR and sharpen its distinctions from other subfields;
- recommend concrete research goals spanning theory, empirical evaluation, and pathways to real-world impact;
- map current challenges into a structured research

[*]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>.

Preprint. March 4, 2026.

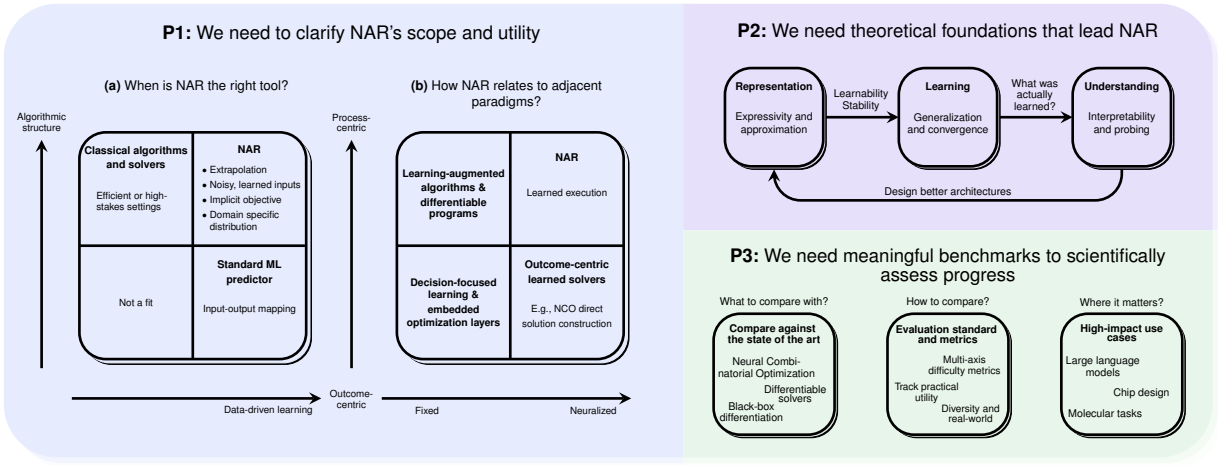

Figure 1. Overview of our three positions, calling for clarifying NAR's scope, theoretical foundations, and meaningful benchmarks.

agenda, linking conceptual ambiguities to actionable objectives.

Overall, we posit that NAR needs a clearer scope and relationship to adjacent subfields, supported by theoretically grounded definitions and empirical evaluations that track progress and real-world impact.

See Figure 1 for a schematic illustration of our positions.

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.

P1: We need to clarify NAR's scope and utility

To ground this paper's central concepts, we adopt a loose definition of NAR. The definition is intentionally broad, since strict definitions can impose artificial boundaries in an

emerging area like NAR.

Minimal informal definition. *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 $\mathcal{N}$.

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) 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:

- *Graph-algorithm execution with MPNNs*, e.g., an MPNN trained to compute quantities depending on shortest paths.
- *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 on an algorithmic task *without* an algorithmic inductive bias, it falls outside our intended scope.

Central to our framing is the interplay between *algorithmic priors* and *inductive bias*: priors are task-specific beliefs about effective algorithmic principles, while inductive bias is the architectural tendency that makes algorithm-like computations easier to represent and learn.

NAR is often framed as endowing neural networks with the capacity to learn, execute, or discover algorithmic computation, yet the term spans heterogeneous goals—from imitating known procedures (Veličković et al., 2020) to discovering new algorithms from data (Schutz et al., 2025)—without a shared scope or clear baselines. This ambiguity hinders comparison and practical justification. To mature into a rigorous subfield, NAR needs clearer intended outcomes and principled comparisons: 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, situate contributions within the broader landscape, and motivate a manual and common language for NAR.

P1.1: Where NAR has unique leverage

NAR has unique leverage when both algorithmic structure and data-driven learning matter. We distinguish between the benefits that come from (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. A primary motivation is extrapolation, especially size generalization: when learned computation reflects stable algorithmic invariants, it is more likely to transfer across instance sizes (Xu et al., 2020). Algorithmic structure can also improve learnability and sample efficiency by biasing learning toward effective procedures, reducing the data needed to acquire multi-step computation (Xu et al., 2021). It may also promote stability and robustness via structured intermediate computations, improve interpretability by making behavior attributable to recognizable procedures, and support composability by enabling reusable modules with clearer interfaces and more predictable failure modes (Eberle et al., 2025).

Benefits from neural networks. NAR is particularly useful when inputs or objectives are poorly specified, enabling a *perception-to-reasoning* pipeline, i.e., the "algorithm" may take noisy, partial, or learned inputs (e.g., latent graphs, uncertain costs, soft constraints), while remaining end-to-end trainable and preserving algorithmic biases. Similarly, when the *objective* is implicit (via demonstrations, labels, or preferences) rather than a precise optimization criterion, learning can internalize the goal from data—as in settings where humans optimize against hard-to-quantify objectives, such as chip design.

A further advantage is the exploitation of distributional structure, where instances are drawn from a domain-specific distribution rather than treated as worst-case inputs, as is standard in classical algorithm design. NAR can combine both algorithmic and domain-specific structure (e.g., repeated vehicle routing in a fixed region with destinations from a stable site set). Finally, NAR produces GPU-friendly, parallelizable procedures, such as batched tensor computations that map well to GPU hardware and provide gradients by construction. This motivates the development of general-purpose algorithmic foundation models capable of learning and executing a broad range of procedures.

Actionable objectives. Given this framework, we outline objectives for better understanding when NAR is useful.

1. **Define win-case settings.** Specify canonical settings where NAR is uniquely justified (e.g., noisy or learned inputs, implicit goals, differentiable multi-step pipelines), and document why classical baselines fail.
2. **Name the intended benefit.** Require each NAR claim to state the target advantage (e.g., amortized inference, robustness to noise, differentiable integration, improved efficiency/latency, better average-case performance), and clarify whether it comes from algorithmic structure, neural integration, or their combination.

P1.2: 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 networks need not be introduced if the task is well-structured and classical methods already provide efficient and verifiable solutions. In addition, in high-stakes settings requiring certificates (e.g., optimality or feasibility), replacing an algorithm with a learned surrogate can degrade reliability and interpretability while increasing training and maintenance costs. Even in lower-stakes contexts, neuralizing an already efficient procedure can be hard to justify if the learned model is slower, less stable under distribution shift, or harder to debug.

Conversely, if the goal is simply to learn an input–output mapping and there is no need to represent or execute a reusable multi-step procedure, then standard neural models (and standard machine learning evaluation) are often the more direct choice. Likewise, when the primary motivation is differentiation for end-to-end learning, black-box differentiation or implicit-gradient approaches (Niepert et al., 2021; Petersen, 2022) may be a more direct tool, as they do not require learning an entire algorithmic procedure. Hence, NAR works should explain why end-to-end learning of a procedure is needed, rather than differentiating an existing solver or using a classical algorithm.

Actionable objectives. To ensure these non-win cases are appropriately considered, we outline the following objectives.

1. **State why NAR is the right tool.** Introduce a "why not just run the algorithm?" requirement, i.e., NAR papers should justify why a classical solver or hybrid pipeline is insufficient and provide empirical evidence.

2. **Use a decision checklist.** Evaluate the (i) need for end-to-end training, (ii) noise/partial observability, (iii) optimality/certification requirements, (iv) distribution shift risk, and (v) compute/latency constraints, and explicitly ask whether algorithmic inductive bias is essential.

P1.3: Situating and comparing NAR approaches

As mentioned previously, NAR does not exist in isolation: adjacent communities—including 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)—often pursue overlapping goals under different terminology. Without careful comparison, communities risk rediscovering ideas, overstating novelty, and fragmenting evaluation practices.

A framework for positioning methods. A useful way to position NAR is to emphasize its unique target: learning (or neuralizing) an algorithmic procedure—a structured, multi-step computation with inductive bias—rather than only learning solutions, costs, or solver parameters. We propose two questions to distinguish approaches: *(Q1) Is the algorithmic procedure fixed or neuralized?* and *(Q2) Is the learning target process-centric or outcome-centric?*

(Q1) If an algorithm is embedded in a learning pipeline and made differentiable, but its procedure is fixed, this does not constitute NAR (e.g., Ong et al. (2024)). This separates implicit models such as OptNet (Amos & Kolter, 2017) from NAR: by our definition in **P1**, algorithmic behavior must be attributable to the neural modules, i.e., the network internalizes the computation rather than parameterizing a call to an external solver.

(Q2) NAR is *process-centric*: it aims to learn a reusable multi-step procedure that embodies an algorithmic inductive bias, rather than merely producing correct final outputs. Methods fall within NAR when they incorporate algorithmic priors that prescribe an effective process. Outcome-centric methods instead treat the problem as a black box and focus on direct solution prediction; while often effective, they bypass reasoning steps and rely on pattern matching rather than algorithmic execution. For example, machine learning for combinatorial optimization aligns with NAR when it incorporates priors about effective algorithmic structure, and is better viewed as end-to-end learning when such priors are absent.

How to relate contributions within NAR. Contributions can be situated along two axes: the benefit sought from embedding algorithmic structure, and the mechanism used to translate an algorithmic prior into inductive bias. Benefits may include improved extrapolation, sample efficiency, or stability/robustness; making the target benefit explicit clarifies what is optimized and what constitutes evidence of

success.

The second axis concerns how priors are encoded. A dominant mechanism is algorithmic alignment, where the architecture is chosen to resemble a known algorithmic framework. Complementary mechanisms include structured supervision (e.g., trajectories or intermediate hints specifying desired states) and regularization that promotes simple, stable computations. We suggest future work state these choices directly: which prior is assumed, how it is encoded, and which benefit is targeted.

Actionable objectives. To operationalize this framing, we outline the following actionable objectives

1. **Map the paradigm boundaries.** Publish a mapping from NAR to neighboring paradigms along concrete axes: what is learned (procedure vs. solution vs. objective), where learning enters, what priors are encoded, and what guarantees are sought (none vs. probabilistic vs. certificates).
2. **Benchmark across paradigms.** Encourage cross-paradigm baselining, i.e., for each NAR setting, compare against the strongest competing paradigm for the same intended outcome (e.g., differentiable solvers for end-to-end optimization, classical algorithms and solvers for data-driven speedups).
3. **Declare success criteria.** Clarify intended outcomes up front (e.g., robustness, differentiability, efficiency) so that "success" is comparable across papers.
4. **Build a shared glossary.** Develop a concise NAR glossary defining key terms (procedure vs. predictor; trace/trajectory supervision; architectural alignment; inductive bias; robustness and distribution shift).
5. **Standardize value reporting.** Adopt a *value-over-solver template* including at minimum solution quality, wall-clock runtime/latency, training cost, robustness under structured shifts, and hard-failure rates (e.g., constraint violations).
6. **Provide practitioner guidance.** Offer a practical decision tree, i.e., given task structure and constraints, recommend, e.g., classical algorithms, differentiable solvers, or full NAR, along with the required baselines and metrics for each choice.

P2: We need theoretical foundations that lead NAR

A central obstacle for NAR is the lack of a rigorous theoretical foundation. While empirical studies show promise, they are not yet supported by a principled understanding of what NAR architectures can represent, how they learn algorithmic behavior, and why they succeed or fail beyond carefully engineered benchmarks. To make NAR scientifically grounded, the field must develop a theory that links architectural expressivity, generalization under distribution shifts, and optimization dynamics.

P2.1: Understand expressivity or approximation abilities of NAR architectures

A rigorous understanding of the expressive power of NAR architectures is necessary before we can assess which algorithms they can learn to approximate. In particular, we need clarity on which algorithmic tasks a given architecture can represent. Although prior work shows that specific architectures, such as MPNNs (Nerem et al., 2025; Xu et al., 2020; Yau et al., 2024; Loukas, 2020) or transformers (de Luca & Fountoulakis, 2024; de Luca et al., 2025), can capture certain algorithmic tasks, we still lack a unified understanding of when an architecture can solve a well-defined computational problem.

A further challenge is that many algorithms rely on discrete branching or exact comparisons. Understanding when smooth transformations can efficiently approximate these operations, and how errors accumulate across multiple computational steps (e.g., Bai et al. (2023) on regression algorithms), is essential for evaluating when NAR architectures apply.

Finally, computational complexity matters. If a neural architecture runs slower than the discrete algorithm it aims to emulate, and distributional information does not yield compensating benefits such as improved approximation, then the motivation for NAR becomes unclear; see also **P1.2**. Developing principled learning-based variants or approximations of classical algorithms, together with clear trade-offs, would be an important step toward making NAR broadly useful to the computer science community.

Actionable objectives. To translate the above positions into concrete research directions, we outline the following actionable objectives, inspired by the Böhm–Jacopini theorem (Böhm & Jacopini, 1966) from programming language theory or a weaker notion thereof, and similar to the approaches taken by Akyürek et al. (2023) and Bai et al. (2023) to analyze whether transformers can simulate regression algorithms.

1. **Expressing discrete algorithmic primitives (at the module level).** For given architectures, identify learnable modules N_i and their composition in a pipeline $\mathcal{N}$ that emulate standard discrete primitives (sequence, selection, iteration) in a way attributable to learned neural computation.
2. **Characterize algorithms that can be simulated (at the pipeline level).** Using the primitive realizations

from (1), characterize which algorithms $\mathcal{A}$ can be represented (or approximated) by composing modules in $\mathcal{N}$, ideally with a basis akin to Böhm-Jacopini, while making limitations from architectural constraints explicit (e.g., depth, scaling, precision, locality).

3. **Continuous relaxation of primitives.** For each discrete primitive from (1), construct differentiable counterparts suitable for learning under $\mathcal{D}$ and $\mathcal{L}$ (e.g., smooth comparisons, soft selection, relaxed arg min), and prove approximation guarantees for the primitive *and* its compositional use in a pipeline.

4. **Error propagation under composed neural computation.** When $\mathcal{N}$ replaces the primitives of $\mathcal{A}$ by continuous surrogates, establish stability conditions and error-propagation bounds under (structurally) recursive composition, e.g.,

$$\|\mathcal{N}_\theta(x) - \mathcal{A}(x)\| \leq \bigoplus_{c \in C(x)} \varepsilon_c(x),$$

where $C(x)$ indexes the (possibly composite) subcomputations invoked on input x, and the bound identifies which learned components control deviation from $\mathcal{A}$.

5. **Analyze the computational complexity of neural algorithmic implementations.** For a target algorithm $\mathcal{A}$ with runtime $T_{\mathcal{A}}(n)$, determine whether a neural realization $\mathcal{N}_\theta$, including training and inference costs under realistic assumptions (depth, parameters, precision, parallelism), achieves

$$T_{\mathcal{N}}(n) \in O(T_{\mathcal{A}}(n)),$$

or whether overhead is justified by advantages induced via $\mathcal{D}$ and $\mathcal{L}$ (e.g., improved approximation ratios, lower average-case runtime, or better GPU utilization).

P2.2: Understand generalization and convergence of NAR architectures

Even if an NAR architecture is expressive enough to approximate a target algorithm, this does not imply that training will find such a representation or that the resulting model will generalize. A complete theory of NAR must therefore go beyond expressivity and characterize learnability and stability under gradient-based training by analyzing: (i) the structure of algorithmic solutions in parameter space, (ii) when minima of a regularized loss $\mathcal{L}(\psi) + r(\psi)$ correspond to correct (or approximately correct) executions beyond the training set, and (iii) the optimization dynamics induced by (stochastic) gradient descent. That is, where **P2.1** concerns *what* an architecture can represent, **P2.2** concerns *whether and how* training reaches such representations. A useful viewpoint is that of *learnable algorithmic invariants*, i.e., latent quantities or structures that remain stable under distributional variation and across optimization trajectories, including under perturbations or distribution shift (e.g., larger instances than in the training set). This lens also connects NAR to *geometric deep learning* (Bronstein et al., 2021) and related extensions that generalize group symmetries to monoids, providing mathematical conditions for extrapolation, e.g., via asynchronous message passing (Dudzik et al., 2023) and filter-equivariant functions (Lewis et al., 2025).

To clarify the relationship between optimization and algorithmic behavior, we propose a hierarchy of learnability. For a supervised or unsupervised loss $\mathcal{L}$ and regularization term r, with model $\mathcal{N}_\theta$, one can distinguish:

1. *Exact algorithmic realization:* if $\theta \in \arg\min_\psi \mathcal{L}(\psi) + r(\psi)$ on a given training dataset, then $\mathcal{N}_\theta$ implements the target algorithm.
2. *Approximate algorithmic realization:* For sufficiently small $\varepsilon > 0$, if $\mathcal{L}(\theta) \leq \min_\psi \mathcal{L}(\psi) + r(\psi) + \varepsilon$ on a given training dataset, then the error of the neural model $\mathcal{N}_\theta$ approximating the target algorithm is controlled by a predictable function of ε.
3. *Convergence of gradient descent:* (stochastic) gradient descent converges to such parameters, ideally also quantifying the speed of convergence.

Similar hierarchies could be defined up to inherent representational limitations of the given architectures, building on **P2.1**, including probabilistic versions (e.g., Goel et al., 2022).

Understanding algorithmic generalization further requires characterizing intermediate algorithmic states during execution. If an algorithm decomposes into T computational steps (see **P2.1**), one seeks multi-step error bounds, as in Item 4, to explain how minor errors can compound out-of-distribution.

Actionable objectives. We propose the following directions toward a unified theory of generalization and convergence in NAR, complementing the expressivity analysis of **P2.1**.

1. **Characterize the optimization landscape of algorithmic solutions.** Determine when algorithmic representations from **P2.1** correspond to minima (or approximate minima) of practical training objectives, and how these minima scale with depth, width, and precision.
2. **Link optimization error to algorithmic performance.** Derive bounds of the form

$$\mathcal{L}(\theta) \leq \min_\psi \mathcal{L}(\psi) + r(\psi) + \varepsilon \text{ implies } \|\mathcal{N}_\theta(x) - \mathcal{A}(x)\| \leq g(\varepsilon),$$

where g depends on the algorithmic structure in **P2.1** and architectural inductive biases.
3. **Analyze convergence of gradient descent toward algorithmic solutions.** Identify conditions under which

gradient descent avoids shortcut solutions and converges to parameters aligned with the target algorithm, incorporating initialization, learning rates, and distributional assumptions (e.g., drawing inspiration from work by Caramanis et al. (2023) on policy-gradient methods for combinatorial optimization).

4. **Derived a unified theory for whole algorithm classes instead of isolated algorithms.** Develop learnability criteria that combine input distributions, architectural biases, and optimization dynamics, moving beyond worst-case notions toward guarantees predictive of practical NAR performance (e.g., whether leveraging distributional information can improve performance guarantees such as approximation ratios).
5. **Principled design of NAR foundation models.** Develop methods that generalize to algorithms unseen at train time, either zero-shot (conditioning on a specification) or few-shot (conditioning on examples), ideally allowing dynamic composition and reuse of previously trained components.

Together, these objectives connect representational capacity (**P2.1**), optimization dynamics, and algorithmic generalization, addressing when NAR architectures not only *represent* algorithms but also *learn* and *generalize* them robustly.

P2.3: Interpretability and probing of learned algorithmic procedures

While NAR has shown effectiveness across a range of algorithms, there is comparatively little interpretability-focused work on what procedures a model internalizes, and how. A systematic approach to probing learned NAR models would be valuable for several reasons: it can assess whether the intended algorithm was learned and guide architecture design; it can reveal what algorithm the model actually learns in practice, helping to explain effects such as multi-task learning (Xhonneux et al., 2021; Ibarz et al., 2022) and informing better training methods; and it can enable analysis of the learned algorithm itself, opening the door to algorithm discovery. This includes understanding alignment with the target algorithm and, when models outperform it (He & Vitercik, 2025), identifying new or modified procedures the model has implicitly discovered.

More broadly, algorithmic interpretability informs our understanding of expressivity (Sanford et al., 2024) and reasoning (Akyürek et al., 2023; Zhou et al., 2024; Yang et al., 2024) of neural models. Understanding what algorithms models learn and employ can also inspire architectural design as an alternative to the scaling paradigm (Eberle et al., 2025).

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

1. **Draw inspiration from explainable AI.** Adapt established interpretability tools to NAR's algorithmic setting, including:
   - **Auxiliary probing models.** Train simple probes on hidden activations of a frozen model (Alain & Bengio, 2018; Naidu et al., 2025) to test whether relevant *algorithmic variables* are encoded and how they evolve across layers, inspecting whether internal state updates align with the target procedure.
   - **Concept-based interpretability.** Use concept-based analysis to extract logical/discrete concepts aligned with *algorithmic rules*; e.g., Georgiev et al. (2022) extract propositional formulas and derive explicit rules for algorithms such as BFS and Kruskal's algorithm.
   - **Mechanistic interpretability.** Reverse-engineer circuits implementing *algorithmic computations*, building on work in small transformers (Nanda et al., 2023; Zhong et al., 2023) and recent NAR studies (e.g., He et al. (2025a)), geometry-level mechanistic analysis (e.g., Mirjanic et al. (2023)), and sparse autoencoders to identify interpretable features (Huben et al., 2024).
2. **Develop algorithm-aware interpretability tools.** Design tools tailored to algorithmic computation that identify learned primitives (e.g., sequence, selection, branching), track intermediate states, and analyze computations across iterations or parallel updates. Beyond evaluation and debugging, such tools could extract procedures from trained models, pushing NAR toward algorithm discovery (Romera-Paredes et al., 2024).
3. **Curate benchmarks with hypothesized algorithmic structure.** Construct benchmark tasks with testable hypotheses about 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. We expand on benchmarks in **P3**.

P3: We need meaningful benchmarks to scientifically assess progress

Existing benchmarks primarily 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.

P3.1: Comparing NAR against the state of the art

Careful benchmarking requires evaluating NAR methods against competing alternative paradigms (see also **P1.3**) and establishing their benefits through systematic ablations. Since one of the central benefits of NAR is its capacity to seamlessly integrate algorithmic inductive bias into an end-to-end differentiable pipeline, it can be directly compared to pipelines that execute algorithms or use smooth approximations. These include backpropagation through algorithms (Pogancic et al., 2020; Niepert et al., 2021), smooth approximations of algorithms and operations like 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). Empirical evaluations of NAR can also naturally be conducted in the setting of neural combinatorial optimization, since previous work has shown that aligning models with algorithms can endow them with approximation guarantees (Yau et al., 2024; He & Vitercik, 2025; McCarty et al., 2021) and better OOD performance in algorithmic tasks (He & Vitercik, 2025; Yau et al., 2024; Veličković et al., 2020).

Actionable objectives. Well-designed benchmarks in this area should enable assessment of whether integrating a neural component via an explicit NAR module leads to improved downstream performance, performance guarantees, better generalization, or more stable optimization behavior. To probe this question, we propose evaluating NAR pipelines in the following settings.

1. **End-to-end predict-then-optimize.** In this setting, a model predicts the parameters of a problem instance, and a bespoke algorithm is called to solve it. Gradient estimation techniques are used to backpropagate through the pipeline, enabling end-to-end training. The parameters may be the graph's edge weights, or, more broadly, the constraints and costs of an optimization problem. A representative example is the Warcraft shortest-path problem (Pogancic et al., 2020), where the goal is to find shortest paths in videogame maps. There, instead of directly predicting a shortest path, the model predicts terrain-dependent traversal costs from map-pixel descriptions, and a shortest-path algorithm is used to solve the problem. Similar examples in this vein include vehicle routing (Baty et al., 2024), integer programming (Paulus et al., 2021), and even Sudoku puzzles (Wang et al., 2019; Paulus et al., 2024). NAR methods can be compared in this setting with black-box backpropagation and related techniques. Consider the following empirical question. In the Warcraft benchmark, can a NAR shortest path pipeline (e.g., a cost predictor combined with a max-aggregation GNN as a shortest path solver) be used to outperform the differentiable predict-then-optimize shortest path approach featured in the original blackbox backpropagation paper (Pogancic et al., 2020) and subsequent work (Berthet et al., 2020; Niepert et al., 2021; Petersen et al., 2024)?
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). An empirical aspect of NAR that requires special attention is OOD generalization. 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). This provides an excellent opportunity to test whether NAR pipelines truly benefit from their algorithmic inductive bias for OOD generalization.

P3.2: Design principles for better NAR benchmarks

Current NAR evaluations are limited, i.e., they rely on narrow metrics (primarily final-output accuracy), test generalization mainly via training on small graphs and testing on larger ones (CLRS (Veličković et al., 2022), SALSA-CLRS (Minder et al., 2023)) or different distributions (GraphBench (Stoll et al., 2025)), and often miss runtime efficiency, robustness, and anytime behavior. More principled difficulty axes, broader metrics, and more diverse datasets are needed to provide empirical evidence for expressivity (**P1**) and generalization (**P2**), while tracking practical utility.

Actionable objectives. To address these limitations, we propose evaluation standards to systematically benchmark NAR.

1. **Define controlled and explicit difficulty axes.** Introduce clearly defined out-of-distribution axes beyond size generalization. Some examples include (i) *Sparsity and topology:* 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 real-world utility.** Go beyond final-output 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, alongside efficiency metrics tailored to the use case.
3. **Develop diverse, real-world NAR datasets.** Broaden benchmarks beyond classical (especially polynomial-time) algorithmic tasks. Geometric algorithms are one promising extension (Chen et al., 2025), with Euclidean inputs that naturally encode positional structure and tasks such as triangulation, range search, and convex hulls. More broadly, move beyond synthetic settings to noisy, practical use cases, and better track progress toward deployment; see **P3.3** for example domains.

Future benchmarks should combine these properties and align with the objectives in **P2.1** and **P2.2**. They should also maintain fair protocols for cross-benchmark comparison. We encourage community-facing efforts (e.g., leaderboards and open-source support) toward a general NAR benchmark spanning diverse algorithms and difficulty dimensions.

P3.3: Connect NAR to high-impact use cases/domains

One key strength of NAR is its ability to embed algorithmic knowledge into neural networks for real-world problem solving, and it has shown practical value in domains such as reinforcement learning (Deac et al., 2021; He et al., 2022), network configuration (Beurer-Kellner et al., 2022), and computational neuroscience (Numeroso et al., 2023). However, many applications impose domain-specific constraints that are not captured by existing benchmarks and remain largely unexplored by NAR. More attention is therefore needed to develop robust, deployable NAR models and expand their impact in high-stakes domains, such as large (language) models, chip design, floorplanning, and molecular tasks. For example, many Electronic Design Automation (EDA) tools for chip design rely on efficient algorithms with fine-tuned heuristics, making them natural candidates for NAR's deployment. We defer details of NAR's potential in other use cases to Appendix A. Furthermore, existing limitations and potential advantages of NAR should be systematically collected to improve realistic benchmark design and integrate NAR into systems in a more principled manner.

Actionable objectives. Complementing issues raised in **P3.2** and **P3.3**, we outline actionable objectives for applying NAR in high-impact domains and addressing deployment challenges.

1. **Integrate NAR into existing algorithmic pipelines.** Many classical algorithms are used as fixed components in larger pipelines; as a result, many deployed pipelines are not fully differentiable. We propose integrating NAR modules into such pipelines to enable end-to-end learning and joint optimization, thereby adapting algorithmic procedures to real-world problems and aligning with the advantages outlined in **P1**.
2. **Evaluate potential impact of NAR in real-world settings.** While benchmarks provide controlled evaluations, real-world deployments often involve more complex settings and constraints, requiring metrics that are often underemphasized in benchmarks (e.g., runtime, latency, efficiency). Moreover, NAR may behave differently when embedded in an application pipeline versus when evaluated on standalone tasks. Therefore, NAR should be assessed under realistic conditions, both on benchmarks derived from applications and within end-to-end pipelines.
3. **Explore advantages of NAR through deployment.** Aligning with **P1.1** and **P3.2**, we need to investigate emerging properties of NAR beyond raw performance. Identifying and validating such properties in real-world applications is challenging and often requires domain expertise; we therefore encourage domain communities to evaluate NAR in situ. Some advantages may only emerge through deployment in domain-specific settings, alongside theoretical and empirical analysis.

2. Alternative views

NAR is connected to several neighboring paradigms and communities, such as neural combinatorial optimization (Cappart et al., 2023), black-box optimization (Pogancic et al., 2020), or differentiable programming (Blondel & Roulet, 2025). From one perspective, many contributions labeled as NAR may be viewed as extensions of these related frameworks. This view highlights the opportunity to clarify distinctions between these fields, as outlined in **P1**, and to establish cross-community baselines and evaluation protocols, as outlined in **P3**.

Relatedly, in many settings, running a classical algorithm or embedding a differentiable solver may be the most appropriate approach. Moreover, augmenting existing algorithms with learned heuristics or surrogate models can provide clear empirical benefits. From this viewpoint, the distinctive value of NAR lies in the subset of problems outlined in **P3.3**, where internalizing an algorithmic procedure as a trainable component enables capabilities that are otherwise difficult to obtain.

Finally, regarding our call for stronger theoretical foundations, one could argue that theory inevitably abstracts away key real-world details. However, as argued in **P2**, theoretical guarantees clarify a field's capabilities and limitations and, in this position paper, are complemented with a call for rigorous benchmarks and application-driven evaluations.

3. Conclusion and future directions

Neural algorithmic reasoning has the potential to connect classical algorithmic structure with end-to-end trainable neural systems, but an unclear scope and fragmented practice currently limit its impact. In this position paper, we argued for a sharper definition of what NAR is and is not, and for grounding the area in the study of neuralized algorithms, meaning differentiable procedures with principled inductive biases and a solid theoretical foundation, providing formal guarantees. We also emphasized that progress should be demonstrated through rigorous and comparable empirical evidence rather than isolated empirical demonstrations.

Taken together, these points reposition NAR as a coherent research program centered on neuralized procedures and rigorous evaluation, with clearer connections 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

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.

We thank Petar Veličković for feedback on this paper. We also thank Erik Müller for creating the figure.

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.

Berthet, Q., Blondel, M., Teboul, O., Cuturi, M., Vert, J.-P., and Bach, F. Learning with differentiable perturbed optimizers. *NeurIPS*, 2020.

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.

Böhm, C. and Jacopini, G. Flow diagrams, Turing machines and languages with only two formation rules. *Commun. ACM*, 9(5):366–371, 1966.

Bronstein, M. M., Bruna, J., Cohen, T., and Veličković, P. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges, 2021.

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.

Chen, S., Ciolli, O., Sidiropoulos, A., and Wang, Y. Effective neural approximations for geometric optimization problems. In *NeurIPS*, 2025.

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., Yang, S., Veličković, P., and Fountoulakis, K. Positional attention: Expressivity and learnability of algorithmic computation. In *ICML*, 2025.

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.

Dudzik, A. J., von Glehn, T., Pascanu, R., and Veličković, P. Asynchronous algorithmic alignment with cocycles. In *LoG*, 2023.

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.

Goel, S., Kakade, S. M., Kalai, A., and Zhang, C. Recurrent convolutional neural networks learn succinct learning algorithms. In *NeurIPS*, 2022.

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.

Ibarz, B., Kurin, V., Papamakarios, G., Nikiforou, K., Bennani, M., Csordás, R., Dudzik, A. J., Bošnjak, M., Vitvitskyi, A., Rubanova, Y., Deac, A., Bevilacqua, B., Ganin, Y., Blundell, C., and Veličković, P. A generalist neural algorithmic learner. In *LoG*, 2022.

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.

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, 2004.

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.

Lewis, O., Ghani, N., Dudzik, A. J., Perivolaropoulos, C., Pascanu, R., and Veličković, P. Filter equivariant functions: A symmetric account of length-general extrapolation on lists. In *NeurIPS 2025 Workshop on Symmetry and Geometry in Neural Representations*, 2025.

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.

Petersen, F., Borgelt, C., Sutter, T., Kuehne, H., Deussen, O., and Ermon, S. Newton losses: Using curvature information for learning with differentiable algorithms. *NeurIPS*, 2024.

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

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.

Sanford, C., Fatemi, B., Hall, E., Tsitsulin, A., Kazemi, M., Halcrow, J., Perozzi, B., and Mirrokni, V. Understanding transformer reasoning capabilities via graph algorithms. In *NeurIPS*, 2024.

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.

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.

Xhonneux, L. A. C., Deac, A., Veličković, P., and Tang, J. How to transfer algorithmic reasoning knowledge to learn new algorithms? In *NeurIPS*, 2021.

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. High-impact domains for NAR

In the following, we discuss high-impact domains 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, 2004) 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.