MITSUBISHI ELECTRIC RESEARCH LABORATORIES
https://www.merl.com

Understanding Dynamic Compute Allocation in Recurrent Transformers

Moosa, Ibraheem Muhammad; Lohit, Suhas; Wang, Ye; Chatterjee, Moitreya; Yin, Wenpeng

TR2026-090 June 30, 2026

Abstract

Token-level adaptive computation seeks to reduce inference cost by allocating more computation to harder tokens and less to easier ones. How- ever, prior work is primarily evaluated on natural- language benchmarks using task-level metrics, where token-level difficulty is unobservable and confounded with architectural factors, making it unclear whether compute allocation truly aligns with underlying complexity. We address this gap through three contributions. First, we introduce a complexity-controlled evaluation paradigm using existing algorithmic and synthetic language tasks with parameterized difficulty, enabling direct testing of token-level compute allocation. Second, we propose ANIRA, a unified recur- rent Transformer framework that supports per- token variable-depth computation while isolating compute allocation decisions from other model factors. Third, we use this framework to conduct a systematic analysis of token-level adaptive computation across alignment with complexity, generalization, and decision timing. Our results show that compute allocation aligned with task complexity can emerge without explicit difficulty supervision, but such alignment does not imply algorithmic generalization: models fail to extrapolate to unseen input sizes despite al- locating additional computation. We further find that early compute decisions rely on static structural cues, whereas online halting more closely tracks algorithmic execution state

International Conference on Machine Learning (ICML) 2026

© 2026 MERL. This work may not be copied or reproduced in whole or in part for any commercial purpose. Permission to copy in whole or in part without payment of fee is granted for nonprofit educational and research purposes provided that all such whole or partial copies include the following: a notice that such copying is by permission of Mitsubishi Electric Research Laboratories, Inc.; an acknowledgment of the authors and individual contributions to the work; and all applicable portions of the copyright notice. Copying, reproduction, or republishing for any other purpose shall require a license with payment of fee to Mitsubishi Electric Research Laboratories, Inc. All rights reserved.

Mitsubishi Electric Research Laboratories, Inc.
201 Broadway, Cambridge, Massachusetts 02139

Understanding Dynamic Compute Allocation in Recurrent Transformers

Ibraheem Muhammad Moosa [*1] **Suhas Lohit** [2] **Ye Wang** [2] **Moitreya Chatterjee** [2] **Wenpeng Yin** [1]

Abstract

Token-level adaptive computation seeks to reduce inference cost by allocating more computation to harder tokens and less to easier ones. However, prior work is primarily evaluated on natural-language benchmarks using task-level metrics, where token-level difficulty is unobservable and confounded with architectural factors, making it unclear whether compute allocation truly aligns with underlying complexity. We address this gap through three contributions. First, we introduce a complexity-controlled evaluation paradigm using existing algorithmic and synthetic language tasks with parameterized difficulty, enabling direct testing of token-level compute allocation. Second, we propose ANIRA, a unified recurrent Transformer framework that supports per-token variable-depth computation while isolating compute allocation decisions from other model factors. Third, we use this framework to conduct a systematic analysis of token-level adaptive computation across alignment with complexity, generalization, and decision timing. Our results show that compute allocation aligned with task complexity can emerge without explicit difficulty supervision, but such alignment does not imply algorithmic generalization: models fail to extrapolate to unseen input sizes despite allocating additional computation. We further find that early compute decisions rely on static structural cues, whereas online halting more closely tracks algorithmic execution state. Code: `https://github.com/merlresearch/ANIRA`

1. Introduction

The difficulty of next-token prediction varies substantially across tokens within a sequence, depending on context, structure, and the underlying computational demands of the task. Nevertheless, most large language models (LLMs) allocate a fixed amount of computation to every token. As inference cost dominates the deployment of LLMs, this shortcoming has motivated a growing body of work on adaptive computation, including approaches that scale test-time compute by extending reasoning traces in token space (Wei et al., 2022) or by increasing latent computation through recurrence (Graves, 2016; Dehghani et al., 2019; Banino et al., 2021; Hao et al., 2025; Geiping et al., 2025).

Recent advances have begun to explore *token-level* adaptive computation more explicitly, allowing models to allocate different amounts of compute to different tokens (Bae et al., 2025; Zhu et al., 2025). While these methods demonstrate promising efficiency–performance tradeoffs, they are typically evaluated on natural-language benchmarks using task-level metrics. In such settings, token-level difficulty is not directly observable and is entangled with architectural choices, routing mechanisms, and training heuristics. As a result, it remains unclear whether learned compute allocation policies genuinely align with underlying computational complexity, or whether they reflect incidental correlations that are difficult to interpret or validate. This limitation points to a more fundamental challenge: *token-level adaptive computation makes token-level claims, yet is rarely evaluated under conditions where token-level difficulty is well-defined or testable*. Without explicit notions of token-level complexity, adaptive compute policies cannot be meaningfully interpreted, compared, or falsified. Addressing this gap requires evaluation protocols in which difficulty is controllable and observable, and experimental setups that isolate compute allocation decisions from other confounding model factors.

In this work, we study token-level adaptive computation through the lens of *complexity-controlled evaluation*. We analyze token-level adaptive compute models on a suite of algorithmic and synthetic language tasks in which difficulty can be explicitly parameterized at the token or input level, enabling a novel evaluation of whether adaptive computation tracks underlying computational demands. To support controlled experimentation, we introduce **ANIRA** (Adaptive Neural Iterative Reasoning Architectures), a unified recurrent Transformer framework that enables per-token variable-depth computation while minimizing architectural

* IMM performed this work as an intern at MERL.
[1]Pennsylvania State University [2]Mitsubishi Electric Research Labs (MERL). Correspondence to: Suhas Lohit <slohit@merl.com>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

confounds. ANIRA is not proposed as a performance-optimized alternative to existing models, but as a *controlled experimental vehicle* for studying how and when token-level compute allocation emerges. ANIRA supports two decision mechanisms: an *early-allocation* variant (ANIRA-E), which commits to a compute budget based on shallow representations, and an *online-halting* variant (ANIRA-O), which makes stepwise decisions conditioned on intermediate computation. This design allows us to isolate the effect of decision timing on learned compute policies under identical training objectives and compute regularization.

Using this framework, we conduct a systematic empirical study of token-level adaptive computation across multiple dimensions, including alignment with task complexity, generalization to unseen input sizes, training dynamics, and the structure of learned allocation policies. Our findings reveal that while compute allocation aligned with complexity can emerge without explicit difficulty supervision, such alignment does not imply algorithmic generalization. Moreover, we show that early and online decision mechanisms lead to qualitatively different compute strategies, reflecting reliance on structural cues versus algorithmic execution state.

Overall, our contributions are four-fold: i) We introduce a **complexity-controlled evaluation paradigm** for token-level adaptive computation, based on algorithmic and synthetic language tasks with explicit, parameterized difficulty, enabling principled testing of whether compute allocation aligns with true computational complexity; ii) We describe **ANIRA**, a unified and controllable recurrent Transformer framework that supports per-token variable-depth computation while isolating compute allocation decisions from other architectural factors, enabling controlled comparisons between early and online decision mechanisms; iii) Using this framework, we provide a **systematic analysis** of token-level adaptive computation, showing that complexity-aligned compute allocation can emerge without explicit supervision, but does not guarantee algorithmic generalization, and that decision timing fundamentally shapes learned compute policies; and iv) We analyze the **training dynamics** of adaptive computation and identify a consistent two-phase regime—learning followed by compute reduction—providing insight into how adaptive compute policies are acquired and refined during training.

2. Related Work

A pioneering work in adaptive computation is ACT (Graves, 2016), which showed that recurrent neural networks can dynamically choose the number of recurrent computation steps at inference time, adapting computation to task complexity. Universal Transformers (Dehghani et al., 2019) extended this idea to Transformer architectures by repeatedly applying a shared recurrent Transformer block, with optional adaptive halting. PonderNet (Banino et al., 2021) addressed stability issues in ACT by learning a distribution over halting times and regularizing it with a prior, and was applied to both recurrent networks and Transformers.

A related line of work studies adaptive depth at the sequence level. Depth-Adaptive Transformers (Elbayad et al., 2020) train sequence-to-sequence Transformers with intermediate predictions and adaptively determines how much depth to use for a sequence. Encoder-only early-exit methods such as DeeBERT (Xin et al., 2020), FastBERT (Liu et al., 2020), and PABEE (Zhou et al., 2020) attach classifiers to intermediate layers and allow an input example to exit once intermediate predictions are sufficiently confident or stable.

Token-level adaptivity instead varies computation across positions within a sequence, which is especially natural for autoregressive generation. In standard non-recurrent Transformers, Depth-Adaptive Transformers (Elbayad et al., 2020) also consider token-specific decoder depth, while CALM (Schuster et al., 2022) performs confidence-based early exiting during generation. Mixture-of-Depths (Raposo et al., 2024) chooses which layers to use for computation for every token.

More recently, recurrent/looped Transformers have been shown to be useful for various reasoning tasks. In particular, Huginn (Geiping et al., 2025) uses a recurrent Transformer to scale up test-time reasoning, but is not trained to perform sequence- or token-level adaptive computation. Closer to our setting, recent work has begun to combine recurrence with token-level adaptivity for LLMs. Mixture-of-Recursions (Bae et al., 2025) explores routing strategies over recursive computation. In concurrent work, Ouro (Zhu et al., 2025) uses stepwise halting to scale recurrent Transformers, with evaluation primarily on natural-language benchmarks.

A limitation of such evaluations is that token-level difficulty is not directly observable, making it hard to determine whether compute allocation tracks any controlled notion of complexity. In contrast, we introduce a complexity-controlled evaluation protocol in which token-level difficulty is explicitly parameterized, enabling direct tests of whether learned halting decisions align with analytically defined sources of complexity.

3. Adaptive Compute Recurrent Transformers

3.1. ANIRA Architecture

In this section, we describe the *Adaptive Neural Iterative Reasoning Architectures* (ANIRA), which enables token-level adaptivity by varying the amount of computation in a recurrent Transformer core. Motivated by recurrent-depth language models (Geiping et al., 2025), we treat the initial and final layers as input/output interfaces and allow variable

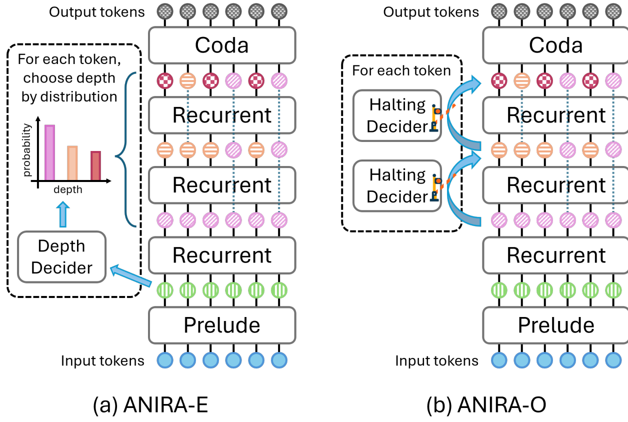


Figure 1. Two variants of ANIRA. Columns correspond to tokens and rows to stages of computation. Circles schematically represent token representations as they pass through the network: changing color or texture indicates continued updating, whereas unchanged color or texture indicates that the token has halted and is subsequently carried forward unchanged. (a) ANIRA-E: a per-token depth decider, applied after the Prelude, predicts a distribution over recurrence depth; the bar chart illustrates this distribution (x-axis: recurrence depth, y-axis: probability). (b) ANIRA-O: a per-token halting decision is made after each recurrence step, determining whether computation continues or halts. After a token halts, later recurrence steps act as identity mappings for that token.

compute only in the recurrent core, letting the model allocate more iterations to harder tokens.

ANIRA follows the Prelude–Recurrent–Coda architecture, akin to Geiping et al. (2025). The *Prelude* is a small stack of causal Transformer layers (Vaswani et al., 2017) producing contextual token representations from the input. The *Recurrent* core is a Transformer block applied for up to D iterations. The *Coda* is a small stack of causal Transformer layers that maps the recurrent output to next-token logits.

Unlike Geiping et al. (2025), ANIRA is token-level adaptive. Adaptivity is learned through a separate decider module that predicts per-token exit depth, i.e., the number of recurrent iterations.[1] We study two decider variants: a *Depth Decider*, which makes an early decision from the Prelude representations, and a *Halting Decider*, which makes online halting decisions from recurrent representations after each iteration.

Architecture notation. Let $x_{1:T} = (x_1, \ldots, x_T)$ be the sequence of input tokens. The Prelude $P(\cdot)$ maps the input tokens to initial representations,

$$h^{(0)}_{1:T} = P(x_{1:T}). \tag{1}$$

A shared recurrent block $R(\cdot)$ is then applied for up to D iterations,

$$h^{(d)}_{1:T} = R\left(h^{(d-1)}_{1:T}\right), \quad d = 1, \ldots, D. \tag{2}$$

Both ANIRA variants define an exit-depth distribution q_i over depths $\{1, \ldots, D\}$ for each token x_i. Given a selected exit depth d_i^*, we define the exit representation

$$z_i = h_i^{(d_i^*)}, \quad i = 1, \ldots, T. \tag{3}$$

The sequence $z_{1:T} = (z_1, \ldots, z_T)$ is then passed to the Coda. The Coda $C(\cdot)$ produces logits

$$o_{1:T} = C(z_{1:T}), \qquad p(x_{t+1} \mid x_{1:t}) = \mathrm{softmax}(o_t). \tag{4}$$

Figure 1 depicts the two variants studied in this paper: early depth decisions (ANIRA-E) and online halting decisions (ANIRA-O).

ANIRA-E. ANIRA-E makes the depth-allocation decision immediately after the Prelude. A depth decider $\psi(\cdot)$ maps the shallow representation $h_i^{(0)}$ to an exit-depth distribution,

$$q_i(d) = \psi(h_i^{(0)})_d, \quad d \in \{1, \ldots, D\}. \tag{5}$$

ANIRA-O. ANIRA-O makes online halting decisions, allowing the exit depth to depend on intermediate recurrent states. For each depth $d = 1, \ldots, D-1$, after computing $h_i^{(d)}$, a halting decider $\phi(\cdot)$ predicts the conditional probability of exiting at that depth:

$$\alpha_i^{(d)} = \phi(h_i^{(d)}) \in (0, 1). \tag{6}$$

These conditional exit probabilities induce an exit-depth distribution q_i. Let $r_i^{(d)}$ denote the probability that token i remains active after depth d. Then

$$r_i^{(0)} = 1, \tag{7}$$
$$q_i(d) = r_i^{(d-1)}\,\alpha_i^{(d)}, \quad d = 1, \ldots, D-1, \tag{8}$$
$$r_i^{(d)} = r_i^{(d-1)}(1 - \alpha_i^{(d)}), \quad d = 1, \ldots, D-1, \tag{9}$$
$$q_i(D) = r_i^{(D-1)}. \tag{10}$$

The final depth D therefore absorbs any remaining probability mass.

Connection to prior work. ANIRA-E commits to a token-specific depth decision from a shallow representation, making it closely related to early depth prediction in Depth-Adaptive Transformers (Elbayad et al., 2020) and conceptually similar to router-based recursion-depth assignment (Bae et al., 2025). Unlike Elbayad et al. (2020), however, ANIRA-E allocates iterations of a *shared recurrent core*. Different

[1]The deciders are multilayer perceptrons (MLPs) matching the MLPs used elsewhere in the model.

from Bae et al. (2025), it predicts an explicit per-token exit depth rather than allocating recursion via budgeted token routing or selection. ANIRA-O follows prior work on online halting-based adaptive computation (Graves, 2016; Banino et al., 2021; Dehghani et al., 2019). In contrast, we apply online halting per token for causal language modeling inside a shared recurrent core, and enforce early-exit semantics by freezing halted token states for subsequent iterations.

3.2. Training and Inference

3.2.1. Training Objective

We train ANIRA with a cross-entropy loss L_{CE} and a compute regularizer L_{C}. The overall objective is:

$$L = L_{\text{CE}} + \gamma\, L_{\text{C}}, \qquad \gamma \geq 0, \tag{11}$$

where γ controls the strength of compute regularization.

The compute regularizer encourages the model's exit-depth distribution $q_i(d)$ to match a fixed prior $p(d)$ using KL divergence: $L_{\text{C}} = \frac{1}{N}\sum_{i=1}^{N} \text{KL}(q_i \| p)$, where N is the total number of tokens in the batch.

We use an exponential prior over depths: $p(d) \propto b^{-d}, b \geq 1$. Under this prior, L_C decomposes to[2]:

$$\text{KL}(q_i \| p) = -H(q_i) + (\log b)\, \mathbb{E}_{q_i}[d] + \text{const} \tag{12}$$

Thus, L_{C} penalizes expected depth guided by the decider distribution while discouraging degenerate depth distributions via the entropy term. L_{CE} is computed only over answer tokens and L_{C} over all tokens, since compute is incurred for both prompt and answer tokens.

3.2.2. Passthrough Mechanism

To enable efficient execution under `torch.compile` during *training*, the computation graph must remain static. We therefore unroll the recurrent core for a fixed number of D iterations, independent of per-token exit decisions. To preserve early-exit semantics, we must ensure: (i) the Coda takes as input the hidden state at each token's selected exit depth, and (ii) the keys and values from that depth are used for subsequent attention by other tokens. We satisfy both requirements via a *passthrough mechanism*.

Once an exit depth $d_i^* \in \{1, \ldots, D\}$ is selected for token x_i (Section 3.2.3), we freeze its representation. Let $\tilde{h}_{1:T}^{(d)} = R(h_{1:T}^{(d-1)})$ denote the output of the recurrent block at iteration d. For each position i and iteration $d = 2, \ldots, D$, we update:

$$h_i^{(d)} = a_i^{(d)}\, \tilde{h}_i^{(d)} + \big(1 - a_i^{(d)}\big)\, h_i^{(d-1)}, \qquad a_i^{(d)} = \mathbf{1}[d \leq d_i^*]. \tag{13}$$

[2]When $b = 1$, the prior becomes uniform. In that case, $\text{KL}(q_i \| p) = -H(q_i) + \textit{const}$, so up to an additive constant the regularizer reduces to the negative entropy term.

It follows that $h_i^{(d)} = h_i^{(d_i^*)}$ for all $d > d_i^*$; i.e., early-exit tokens present a fixed representation to deeper recurrent iterations. With shared parameters across iterations, this also freezes the corresponding keys and values contributed by early-exit tokens.

3.2.3. Depth Selection During Training and Inference

The passthrough mechanism (Section 3.2.2) requires a discrete per-token exit depth $d_i^* \in \{1, \ldots, D\}$ during training. We obtain this depth by sampling from the exit-depth distribution $q_i(d)$, while keeping the recurrent core unrolled for a fixed D iterations to preserve a static computation graph. At inference time, we replace sampling with deterministic depth selection and stop recurrent computation once the current token's selected depth has been reached.

Training-time depth sampling. For ANIRA-E, the depth decider predicts a categorical distribution over depths from the Prelude representation, $q_i(d) = \psi(h_i^{(0)})_d$, with logits $s_i \in \mathbb{R}^D$. We sample d_i^* using the straight-through Gumbel–Softmax estimator (Jang et al., 2017):

$$d_i^* = \arg\max_{d \in \{1, \ldots, D\}} (s_{i,d} + g_{i,d}), \qquad g_{i,d} \sim \text{Gumbel}(0, 1), \tag{14}$$

and backpropagate through $\text{softmax}\big((s_i + g_i)/\tau\big)$.

For ANIRA-O, the halting decider produces conditional halting probabilities $\alpha_i^{(d)} = \phi(h_i^{(d)})$, which induce an exit-depth distribution $q_i(d)$. Let $F_i(d) = \sum_{j=1}^{d} q_i(j)$ denote the cumulative distribution function. During training, we draw $u_i \sim \text{Uniform}(0, 1)$ and apply inverse-CDF sampling:

$$d_i^* = \min_{d \in \{1, \ldots, D\} : F_i(d) \geq u_i} d. \tag{15}$$

We backpropagate through the discrete choice using a straight-through estimator (Bengio et al., 2013). In both variants, the sampled depth d_i^* is used by the passthrough mechanism, while the recurrent computation remains statically unrolled to depth D.

Inference-time depth selection. ANIRA-E deterministically selects the most likely depth,

$$d_i^* = \arg\max_{d \in \{1, \ldots, D\}} q_i(d). \tag{16}$$

Because this decision is made immediately after the Prelude, autoregressive decoding only runs the recurrent core for d_i^* iterations for the current token.

For ANIRA-O, inference selects the first depth at which the cumulative exit probability reaches 0.5:

$$d_i^* = \min\{d \in \{1, \ldots, D\} : F_i(d) \geq 0.5\}. \tag{17}$$

This rule can be evaluated online during autoregressive decoding: after each recurrent iteration, we update $F_i(d)$ for the current token and stop once Eq. (17) is satisfied.

3.2.4. Allocation-Aware KV Caching

During autoregressive decoding, the attention computation at recurrent iteration d requires keys and values for past tokens at the same iteration. However, with a per-token early exit, a past token i may have exited at iteration $d_i^* < d$, so its keys and values at iteration d are not available. We therefore use an allocation-aware KV cache (Elbayad et al., 2020; Geiping et al., 2025). For each past token position i, we cache KV up to its exit iteration d_i^*. When attention at iteration d requests KV for token i, we retrieve the deepest cached entry, i.e., the KV at depth $\min(d, d_i^*)$.

3.2.5. Compute and Memory Savings During Inference

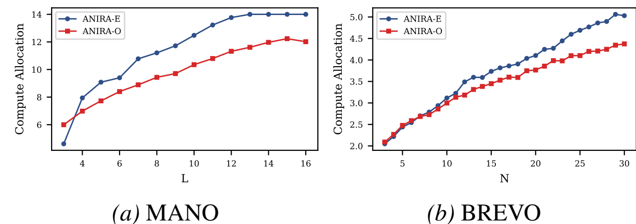
It is easy to see that the compute and KV cache memory used by ANIRA in the recurrent block is proportional to the mean depth allocation $\bar{d} = \frac{1}{T}\sum_{i=1}^{T} d_i^*$ for a length-T sequence. Compared to a non-adaptive model that utilizes the maximum depth D, ANIRA reduces compute and KV cache memory approximately according to the ratio $\bar{d}/D$.

4. Difference in Compute Allocation Policies Between ANIRA-E and ANIRA-O

Activity pattern and execution cost. Let $a_i^{(d)} \in \{0,1\}$ indicate whether token i is *active* at recurrent step d. If $a_i^{(d)} = 0$, token i is frozen (no further updates), but its current representation remains visible to other tokens via attention. Let $A_d = \{i : a_i^{(d)} = 1\}$. The execution-time compute cost can then be defined as $\mathrm{Cost}_{\mathrm{exec}}(\pi) := \sum_{d=1}^{D} |A_d|$, where π denotes the token-level compute-allocation policy.

Decision cost. We separately account for the compute used to *decide* the token-level compute-allocation policy π. Let $\mathrm{Cost}_{\mathrm{dec}}(\pi)$ denote the decision-time compute incurred by policy π. For ANIRA-E, this is the cost of producing all per-token stopping times before running the recurrent steps using a "small" Depth Decider head after the *prelude* layer. For ANIRA-O, this includes the online per-step halting decider computation, a "small" halting head, evaluated on active tokens.

$\mathrm{Cost}_{\mathrm{dec}}(\pi)$ for ANIRA-E is designed to be small enough compared to $\mathrm{Cost}_{\mathrm{exec}}(\pi)$, relative to the task complexity. Without this constraint, the Depth Decider module in ANIRA-E could, in principle, reproduce the recurrent computation used by stepwise halting and thus compute the same stopping times in advance, making the comparison between ANIRA-E and ANIRA-O vacuous. This



(a) MANO *(b)* BREVO

Figure 2. Task complexity vs. mean depth allocation. Both ANIRA variants allocate compute consistent with task complexity while maintaining near-perfect accuracy (94.3% on MANO, 100% on BREVO). Across both tasks, ANIRA-O consistently requires fewer recurrent iterations than ANIRA-E to achieve equivalent performance.

compute-limitedness constraint[3] makes explicit that running the Depth/Halting Decider modules are intended to be *much cheaper* than executing several additional recurrent steps. In our experiments, these decider heads use about 8% of the total parameters.

It is easily shown that it is trivial for the halting decider in ANIRA-O to produce the same compute allocation policy as the depth decider in ANIRA-E: Fix an ANIRA-E policy π^E with exit depth $d_i^* = \pi_i^E$ for the ith input token x_i. Making simple assumptions that the recurrent blocks in ANIRA-O can emulate identity mappings, and that the halting decider block in ANIRA-O is at least as expressive as the Depth Decider block in ANIRA-E, we can trivially define an ANIRA-O policy as π^O by $\pi_i^O(h_{1:T}^{(d)}) = \pi_i^O(h_{1:T}^{(0)}) := \mathbf{1}\{d \geq d_i^*\}$, where the recurrent layers are simply identity mappings. Therefore, for all d, i: $a_i^{(d)} = \mathbf{1}\{d \leq d_i^*\}$ matches the activity pattern induced by π^E and $\mathrm{Cost}_{\mathrm{exec}}(\pi^O) = \mathrm{Cost}_{\mathrm{exec}}(\pi^E)$.

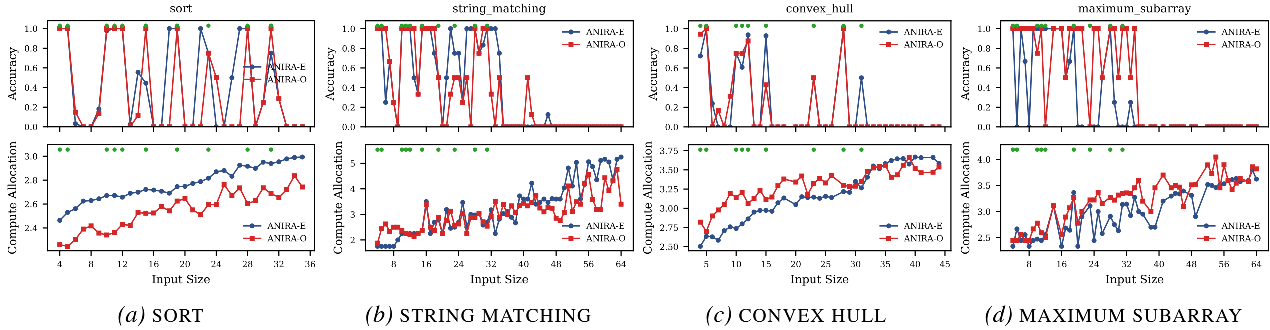
In the reverse direction, we can similarly argue that ANIRA-E cannot emulate all the policies obtained from ANIRA-O, as the halting deciders have access to $h_{1:T}^{(d)}$ obtained from further non-trivial computation using more recurrence steps.

Thus, we generally expect ANIRA-O to be more powerful than ANIRA-E. That is, for the same task performance, we expect ANIRA-E to make *conservative* predictions of the required depth and that ANIRA-O would be able to use fewer recurrences compared to ANIRA-E. However, in practice, the training dynamics and the tasks being solved also play important roles.

5. Complexity Controlled Evaluation Protocol

A central question for token-level adaptive models is whether *the decider learns to allocate compute consistent*

[3] We use $\mathrm{Cost}_{\mathrm{exec}}$ and $\mathrm{Cost}_{\mathrm{dec}}$ as a proxy for the expressivity of the base neural network and decider neural network, as the compute cost is directly measurable.



(a) SORT (b) STRING MATCHING (c) CONVEX HULL (d) MAXIMUM SUBARRAY

Figure 3. CLRS task complexity vs accuracy (top) and compute allocation $\bar{d}$ (bottom). Green markers indicate input sizes seen during training. ANIRA compute allocation tracks task complexity. However, we observe that task accuracy drops sharply at input sizes not covered in training set, indicating interpolation and extrapolation failure.

Table 1. Spearman correlation between per-token expected depth and complexity proxies on LANO generations.

Complexity Proxy	ANIRA-E	ANIRA-O
Parse-space expansion	+0.666	+0.456
Parse convergence	-0.264	-0.111
Addition operations	+0.618	+0.463
Multiplication operations	+0.645	+0.490

with task complexity. We design complexity-controlled evaluation protocol with algorithmic tasks where the task complexity can be rigorously defined and synthetic language where suitable token-level prediction complexity proxies can be designed. This controlled setting lets us test alignment between compute allocation and a known notion of difficulty, which is otherwise unobservable for natural language tokens.

5.1. Algorithmic Tasks

We use the tasks MANO and BREVO from the Physics of Language Models benchmark suite (Allen-Zhu, 2025) and the algorithmic task benchmark CLRS-Text (Markeeva et al., 2024). Next, we briefly introduce the synthetic algorithmic tasks.

MANO: Each instance of MANO is a randomly generated modular-arithmetic expression in prefix notation with operators $\{+, -, \times\}$ and operands in $\{0, \ldots, 22\}$. The target is the expression value modulo 23. A prefix expression with L binary operators can be evaluated in a single left-to-right pass that performs exactly L reductions, so the required computation is $\Theta(L)$. Thus, the *complexity knob* is L.

BREVO: Each instance of BREVO is a directed acyclic graph (DAG) together with a query node. The target is the set of nodes that the query *depends on*, listed in a specific topological order.[4] Computing the transitive predecessor set of the query can be done by a graph traversal over the reachable subgraph, with a time complexity of $\Theta(|V_{\text{reach}}| + |E_{\text{reach}}|)$. Under the benchmark generator, in-/out-degrees are bounded to ≤ 4, so the expected traversal cost scales approximately linearly with N. Thus the number of nodes N in the DAG serves as the *complexity knob*.

CLRS: The CLRS-Text dataset provides a suite of 30 algorithmic tasks with systematic variation in input size. The training set sparsely covers the size range, enabling analysis of both interpolation and extrapolation. The dataset also provides algorithm traces that provide step-by-step supervision (akin to chain-of-thought). As we focus on latent reasoning, we remove algorithm traces and train from questions alone. This merges trace-distinct variants (e.g., quicksort and bubble sort into a single SORT task), yielding 23 unique tasks. The *complexity knob* of each task in CLRS is the size of the input. For example, the complexity knob of the SORT task is defined as the number of items to sort. A full list of problems and the definition of their input size is provided in the Appendix A.1.

5.2. Synthetic Language

We use the synthetic language task LANO from Allen-Zhu (2025), where sequences are generated from a probabilistic context-free grammar (PCFG). Unlike the algorithmic tasks, LANO has no single difficulty knob; difficulty varies across token positions as the prefix becomes more or less syntactically ambiguous.

We therefore construct token-level difficulty proxies from an incremental parser. Specifically, we run an incremental prefix parser for PCFGs (Nowak & Cotterell, 2023) on model-generated strings and record per-token signals. We group these into (i) *ambiguity proxies* capturing parse-space branching and consolidation (*parse-space expansion* and *parse convergence*) and (ii) *compute proxies* measuring parser work (the number of *additions* and *multiplications* per token). Ideally, compute allocation should correlate

[4] Predecessors are traversed in the lexicographical order by node name, and the query node itself is excluded from the output.

positively with *parse-space expansion* and compute proxies, and negatively with *parse convergence*. Formal definitions are provided in Appendix A.7.

5.3. Defining Compute Allocation

For MANO, BREVO and CLRS, we define compute allocation as the mean allocated depth $\bar{d}$ on the answer tokens using the inference-time depth selection criteria described in Section 3.2.3. For LANO, we define it as the expected depth $\mathbb{E}_{q_i}[d]$ for each token position.[5]

6. Results

6.1. Compute Allocation Tracks Task Complexity

We train the ANIRA models on MANO instances spanning difficulty levels $L \in \{3, \ldots, 16\}$ and separately on BREVO instances with sizes $N \in \{3, \ldots, 30\}$. On CLRS, we train *multitask* ANIRA models across all problems in the CLRS training dataset. On LANO, we train ANIRA models on the PCFG 3b[6] described in Allen-Zhu (2025). We set the maximum number of recurrent iterations to $D = 6$ for all tasks, except MANO where we use $D = 14$. [7]

Figure 2 shows that the ANIRA models learn to allocate compute consistent with task complexity on the MANO and the BREVO tasks.[8] Figure 3 shows the same for a representative selection of tasks from the CLRS dataset.[9] Table 1 shows that the ANIRA models' compute allocation is consistent with the token level prediction complexity on synthetic language. Thus we can conclude that the *ANIRA models allocate compute consistent with complexity even in the absence of explicit complexity signals during training.*

6.2. Adaptivity Does Not Lead to Algorithmic Generalization

As compute allocation tracks task complexity, we might expect ANIRA to learn input-size-invariant algorithmic representations and thus generalize better. However, the per-size accuracy curves in Figure 3 suggest otherwise: both variants exhibit sharp accuracy drops at unseen input sizes, even within the interpolation regime. This indicates that the *models learn size-specific solutions rather than fully exploiting shared algorithmic structure*.

[5]We use expected depth here to reduce the variance from depth sampling as we only have proxies for the complexity.

[6]PCFG 3b has 16 nonterminals, 3 terminals symbols, and 35 production rules with a maximum derivation depth of 7. All branching rules have uniform probability 0.5.

[7]For more details see Table 5 in the Appendix.

[8]Appendix A.3 provides corroborating results on DEPO task.

[9]Appendix A.1 provides results on all CLRS tasks.

6.3. Training Dynamics

To understand how ANIRA learns its compute policy, we study the training dynamics of task accuracy and depth allocation on the MANO and BREVO tasks. It is interesting to observe from Figure 4 and Figure 5 that *the models learn the tasks in an easy-to-hard order*, without explicit supervision.

Further, we see two consistent phases of training: *learning* and *compute reduction*. In the *learning* phase, both variants rapidly increase their compute allocation, often approaching the maximum, suggesting that *the model initially relies on near-full recurrent computation to reduce the task loss*. In the *compute reduction* phase, *the models learn to reduce compute allocation while preserving task performance*.

6.4. ANIRA-E and ANIRA-O Learn Different Compute Allocation Policies

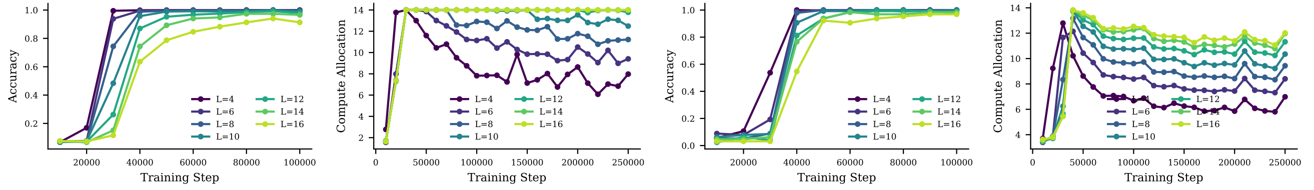
For many tasks, especially when trained per-task (e.g., MANO, BREVO, DEPO) ANIRA-O tends to converge to lower average depth than ANIRA-E, which is consistent with online halting leveraging intermediate recurrent states to stop more aggressively. To further understand the difference in compute allocation policies of ANIRA-E and ANIRA-O, we perform regression analysis on the compute allocation of answer tokens on the BREVO task, against structural and algorithmic state features, extracting features at each token position including direct dependents, DFS traversal depth, search frontier size and the number of newly enabled nodes.

Table 2 shows the linear regression coefficients of each feature controlling for graph size with categorical fixed effects. *Despite similar overall performance, the two models employ fundamentally different strategies.* ANIRA-E's compute allocation is primarily explained by the structural feature hub structure (out-degree), with algorithmic state features contributing minimally. In contrast, ANIRA-O compute allocation is explained primarily by algorithmic execution features DFS depth, frontier size, and newly enabled nodes suggesting it learns to *track algorithmic progress to decide when to halt*.

The architectural choice thus induces qualitatively different learned behaviors. ANIRA-E exploits statistical correlations between structure and difficulty, while ANIRA-O's strategy aligns more closely with the underlying algorithm. Appendix A.8 describes the complete methodology and feature definitions.

6.5. ANIRA Starts Solving the Problem While Processing the Question Tokens

For the MANO prefix-notation arithmetic evaluation task, the task could be solved while processing the question to-



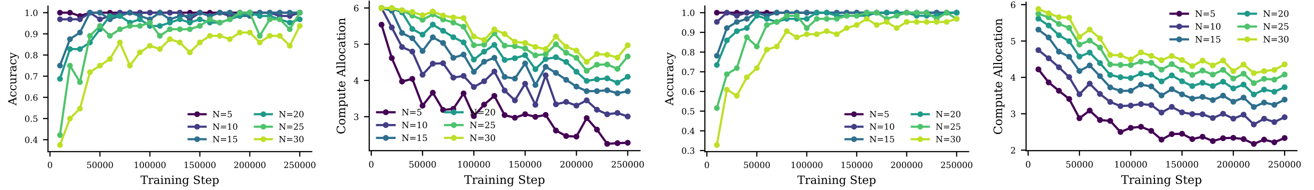
(a) ANIRA-E: Accuracy vs Training steps

(b) ANIRA-E: Compute Allocation vs Training steps

(c) ANIRA-O: Accuracy vs Training steps

(d) ANIRA-O: Compute Allocation vs Training steps

Figure 4. MANO Training Dynamics: ANIRA learn tasks in easy-to-hard order and learning happens in two phases: *learning* and *compute reduction*.



(a) ANIRA-E: Accuracy vs Training steps

(b) ANIRA-E: Compute Allocation vs Training steps

(c) ANIRA-O: Accuracy vs Training steps

(d) ANIRA-O: Compute Allocation vs Training steps

Figure 5. BREVO Training Dynamics: ANIRA learns tasks in easy-to-hard order and learning happens in two phases: *learning* and *compute reduction*.

Table 2. **BREVO: answer-token feature coefficients for compute allocation.** Linear regression coefficients predicting per-answer-token allocated compute from BREVO features, controlling for graph size N with categorical fixed effects. Larger absolute coefficients indicate stronger association. ANIRA-E aligns most with hub structure, while ANIRA-O aligns more with algorithmic-state features. Fit: ANIRA-E R^2=0.7337, ANIRA-O R^2=0.7088.

Feature	ANIRA-E	ANIRA-O
Hub structure (out-degree)	+0.134	+0.168
DFS traversal depth	-0.005	+0.302
Search frontier size	+0.047	+0.171
Newly enabled nodes	−0.079	−0.286

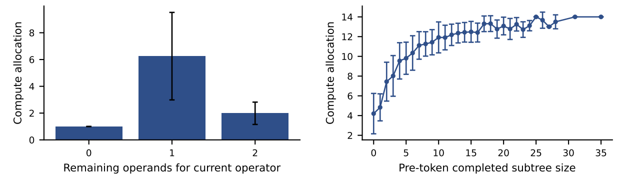


Figure 6. **MANO question tokens: ANIRA-E compute allocation tracks online parse state. Left:** Operator-token compute allocation vs. *remaining operands for current operator* r_t before the token ($r_t = 0$ root operator token; $r_t = 2$ left-child context; $r_t = 1$ right-child context). **Right:** For operators with $r_t = 1$, compute allocation increases with *pre-token completed subtree size* s_t. Error bars denote standard deviation; results use only correctly answered instances.

kens. The adaptive nature of the ANIRA models naturally allows us to study whether the models actually learn this.

We derive *prefix-observable* parse-state features from the question prefix using a deterministic stack parser. For each question token t, we record: (i) *pre-token operator stack depth* d_t, (ii) *remaining operands for current operator* $r_t \in \{0, 1, 2\}$, the number of operands still required by the current top-of-stack operator immediately before consuming token t (setting $r_t = 0$ for the root operator token), and (iii) *pre-token completed subtree size* s_t, defined as the total number of operator nodes in all operator-subtrees completed at the preceding token.

Figure 6 shows that ANIRA-E compute allocation is strongly structured by these online state variables: operator tokens receive the most compute when $r_t = 1$, and within this subset, compute increases with the size of the immediately preceding completion event s_t. This suggests that the model allocates extra compute *while processing the question tokens* as the parse state evolves, rather than deferring all computation to the final answer token.

6.6. Natural Language Mathematical Questions

We study adaptive depth allocation in a natural-language setting by augmenting an eight recurrent layer depth-recurrent Llama model[10] from (McLeish et al., 2025) with both ANIRA-E and ANIRA-O. We finetune each model for about 5B tokens on Nemotron-CC-Math-v1-4plus (Mahabadi et al., 2025), a large-scale high-quality math corpus. We evaluate on GSM-Symbolic (Mirzadeh et al., 2025), a

[10] https://huggingface.co/smcleish/Recurrent-Llama-3.2-train-recurrence-8

Table 3. GSM-Symbolic evaluation after finetuning on Nemotron-CC-Math-v1-4plus. We report exact accuracy and the mean allocated depth $\bar{d}$ over answer tokens. The baseline uses fixed depth $D=8$, while ANIRA-E and ANIRA-O allocate variable depth. P1 and P2 add one and two additional clauses relative to MAIN.

	Baseline	ANIRA-E		ANIRA-O	
Split	Acc. (%)	Acc. (%)	$\bar{d}$	Acc. (%)	$\bar{d}$
Main	46.26	45.70	6.060	43.68	4.108
P1	23.54	24.06	6.004	23.02	4.105
P2	5.80	5.80	5.982	5.08	4.001

dataset of grade-school-level mathematical questions created from symbolic templates. Relevant to our study, GSM-Symbolic provides three evaluation splits, MAIN, P1, and P2, where P1 and P2 add one and two additional clauses to a question relative to MAIN. We treat these splits as increasing in difficulty, MAIN $<$ P1 $<$ P2.

Table 3 reports exact accuracy and mean allocated depth over answer tokens. Neither ANIRA variant allocates more depth on the harder splits. At the same time, both variants incur only a modest accuracy decrease relative to the depth-8 baseline on each split, while achieving substantial reductions in compute allocation.

We note that measuring task difficulty is inherently complex for natural language tasks. This experiment shows that compute allocation by ANIRA is not easily interpretable on natural language, and does not appear to correlate with a simple notion of task complexity.

7. Limitations and Future Work

Limitations. First, our experiments focus on standard autoregressive Transformer architectures; we do not study how token-level adaptive computation interacts with other architectural components used in many modern large-scale models, such as Mixture-of-Experts. Second, in our natural language experiments, which focus on mathematical reasoning, the analysis relies on coarse task-level proxies for difficulty. While useful for studying broad trends, such proxies do not directly capture token-level complexity, which is the level most relevant for adaptive computation. More broadly, our natural language evaluation is limited in scope, since mathematical reasoning represents only one subset of natural language tasks. Third, although our analysis shows that ANIRA can reduce inference computation in principle, realizing these gains in practice is non-trivial. Existing scalable inference frameworks, such as vLLM, are heavily optimized for conventional fixed-depth Transformer execution and do not directly support token-level adaptive computation.

Future directions. These limitations point to several promising directions for future work. An important next step is to develop complexity proxies for natural language that are meaningful at the token level. Our results on synthetic CFG already suggest that compute allocation correlates with syntactic complexity; in natural language, one could further investigate semantic complexity, including reasoning difficulty. Another direction is to extend this analysis beyond autoregressive Transformers. In particular, with the recent rise of diffusion-based language models (DLMs), which generate tokens in parallel, it would be valuable to study token-level adaptive computation in DLMs using the systematic framework introduced in this paper. Finally, an important practical direction is to develop efficient inference systems that can fully exploit token-level adaptive decisions.

8. Conclusion

Token-level adaptive computation has largely been treated as an efficiency mechanism evaluated through task-level performance, despite making claims about token-level behavior. This work reframes adaptive computation as a measurement and interpretability problem, arguing that its validity depends on the availability of well-defined, testable notions of token-level difficulty. Our findings suggest that while adaptive compute policies can be learned, their meaning and generalizability are tightly coupled to the structure of the evaluation setting: where token-level complexity is explicit, allocation reflects meaningful signals; where it is not, such as in natural language, interpretation becomes inherently ambiguous. More broadly, this highlights the need for complexity-aware evaluation protocols, principled difficulty proxies, and inductive biases that connect computation to algorithmic structure, if adaptive computation is to serve as a foundation for reliable and interpretable reasoning systems rather than a heuristic optimization.

Acknowledgments

Suhas Lohit, Ye Wang and Moitreya Chatterjee were exclusively supported by MERL. Wenpeng Yin (and partially his student Ibraheem Muhammad Moosa) were supported by NSF DMS-2533995.

Impact Statement

This paper presents work whose goal is to advance the field of machine learning, specifically by studying token-level adaptive computation for language models and proposing a complexity-controlled evaluation protocol. Our methods may help reduce inference cost and energy use by allocating less computation to easier tokens. The broader societal implications are similar to those of prior work on more efficient model architectures and inference systems, and

we do not anticipate impacts beyond these well-established considerations.

References

Allen-Zhu, Z. Physics of language models: Part 4.1, architecture design and the magic of canon layers. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=kxv0M6I7Ud.

Bae, S., Kim, Y., Bayat, R., Kim, S., Ha, J., Schuster, T., Fisch, A., Harutyunyan, H., Ji, Z., Courville, A., and Yun, S.-Y. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=QuqsEIVWIG.

Banino, A., Balaguer, J., and Blundell, C. Pondernet: Learning to ponder. In *8th ICML Workshop on Automated Machine Learning (AutoML)*, 2021. URL https://openreview.net/forum?id=1EuxRTe0WN.

Bengio, Y., Léonard, N., and Courville, A. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. URL https://arxiv.org/abs/1308.3432.

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, Ł. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/pdf?id=HyxXZwJcwH.

Elbayad, M., Gu, J., Grave, E., and Auli, M. Depth-adaptive transformer. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJg7KhVKPH.

Geiping, J., McLeish, S. M., Jain, N., Kirchenbauer, J., Singh, S., Bartoldson, B. R., Kailkhura, B., Bhatele, A., and Goldstein, T. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=S3GhJooWIC.

Graves, A. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Hao, S., Sukhbaatar, S., Su, D., Li, X., Hu, Z., Weston, J., and Tian, Y. Training large language models to reason in a continuous latent space, 2025. URL https://arxiv.org/abs/2412.06769.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkE3y85ee.

Liu, W., Zhou, P., Wang, Z., Zhao, Z., Deng, H., and Ju, Q. Fastbert: a self-distilling bert with adaptive inference time. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 6035–6044, 2020.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Mahabadi, R. K., Satheesh, S., Prabhumoye, S., Patwary, M., Shoeybi, M., and Catanzaro, B. Nemotron-cc-math: A 133 billion-token-scale high quality math pretraining dataset, 2025. URL https://arxiv.org/abs/2508.15096.

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.

McLeish, S., Li, A., Kirchenbauer, J., Kalra, D. S., Bartoldson, B. R., Kailkhura, B., Schwarzschild, A., Geiping, J., Goldstein, T., and Goldblum, M. Teaching pretrained language models to think deeper with retrofitted recurrence, 2025. URL https://arxiv.org/abs/2511.07384.

Mirzadeh, S. I., Alizadeh, K., Shahrokhi, H., Tuzel, O., Bengio, S., and Farajtabar, M. GSM-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=AjXkRZIvjB.

Nowak, F. and Cotterell, R. A fast algorithm for computing prefix probabilities. In Rogers, A., Boyd-Graber, J., and Okazaki, N. (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 57–69, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-short.6. URL https://aclanthology.org/2023.acl-short.6/.

Raposo, D., Ritter, S., Richards, B., Lillicrap, T., Humphreys, P. C., and Santoro, A. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

Schuster, T., Fisch, A., Gupta, J., Dehghani, M., Bahri, D., Tran, V., Tay, Y., and Metzler, D. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472, 2022.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., brian ichter, Xia, F., Chi, E. H., Le, Q. V., and Zhou, D. Chain of thought prompting elicits reasoning in large language models. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J.

Xin, J., Tang, R., Lee, J., Yu, Y., and Lin, J. Deebert: Dynamic early exiting for accelerating bert inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 2246–2251, 2020.

Zhou, W., Xu, C., Ge, T., McAuley, J., Xu, K., and Wei, F. Bert loses patience: Fast and robust inference with early exit. In *Advances in Neural Information Processing Systems*, 2020.

Zhu, R.-J., Wang, Z., Hua, K., Zhang, T., Li, Z., Que, H., Wei, B., Wen, Z., Yin, F., Xing, H., et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

Problem	Input size n (definition)
Sort	$n = \|\mathbf{x}\|$ for an input array $\mathbf{x} = (x_1, \dots, x_n)$.
Binary search	$n = \|\mathbf{x}\|$ for a sorted array $\mathbf{x} = (x_1, \dots, x_n)$.
Find minimum	$n = \|\mathbf{x}\|$ for an input array $\mathbf{x} = (x_1, \dots, x_n)$.
Select (order statistic)	$n = \|\mathbf{x}\|$ for an input array $\mathbf{x} = (x_1, \dots, x_n)$.
Maximum subarray	$n = \|\mathbf{x}\|$ for an input array $\mathbf{x} = (x_1, \dots, x_n)$.
String matching	$n = \|T\| = \|P\|$ for text $T \in \Sigma^n$ and pattern $P \in \Sigma^n$.
Longest common subsequence	$n = \|X\| = \|Y\|$ for sequences $X, Y \in \Sigma^n$.
Matrix chain multiplication	$n = \|p\|$ for dimension vector $p = (p_0, \dots, p_{n-1})$ (so #matrices $= n - 1$).
Optimal BST	n = number of keys, with probabilities $\{p_i\}_{i=1}^n$ and $\{q_i\}_{i=0}^n$.
Activity selection	n = number of activities/intervals $\{(s_i, f_i)\}_{i=1}^n$.
Task scheduling	n = number of jobs $\{(d_i, w_i)\}_{i=1}^n$.
Convex hull	n = number of planar points $\{(x_i, y_i)\}_{i=1}^n \subset \mathbb{R}^2$.
Shortest path	$n = \|V\|$ for a graph $G = (V, E)$ (vertices).
MST Kruskal	$n = \|V\|$ for a graph $G = (V, E)$ (vertices); outputs edge matrix.
MST Prim	$n = \|V\|$ for a graph $G = (V, E)$ (vertices); outputs predecessor array.
Breadth-first search	$n = \|V\|$ for a graph $G = (V, E)$ (vertices).
Depth-first search	$n = \|V\|$ for a graph $G = (V, E)$ (vertices).
Topological sort	$n = \|V\|$ for a directed graph $G = (V, E)$ (vertices).
Strongly connected components	$n = \|V\|$ for a directed graph $G = (V, E)$ (vertices).
All-pairs shortest paths	$n = \|V\|$ for a graph $G = (V, E)$ (vertices).
Articulation points	$n = \|V\|$ for a graph $G = (V, E)$ (vertices).
Bridges	$n = \|V\|$ for a graph $G = (V, E)$ (vertices).
Bipartite matching	$n = \|U\| + \|W\|$ for bipartite $G = (U, W, E)$.
Segment intersection	constant-size instance (two line segments in $\mathbb{R}^2$; no varying n).

Table 4. Definition of input size n per CLRS problem used for our CLRS-Text analysis.

Model	Task	D	#heads	Prelude/Coda	d_{model}	d_{ff}	maxlen	batch size	LR	γ/b
ANIRA	MANO	14	8	1/1	256	1664	64	512	1e-4	1e-1/1.25
	BREVO	6	8	1/1	512	2048	256	64	1e-3	1e-1/2.0
	DEPO	6	8	1/1	512	2048	256	64	3e-4	1e-1/2.0
	LANO	6	8	1/1	512	2048	512	64	1e-4	1e-2/1.0
	CLRS	6	8	1/1	512	2048	4096	32	1e-4	1e-2/2.0

Table 5. ANIRA hyperparameters. Both ANIRA-E and ANIRA-O employ the same hyperparameters. The hyperparameters LR, γ and b were chosen based on the performance on validation sets with a small grid search for ANIRA-E and reused for ANIRA-O.

A. Appendix

A.1. CLRS-text Task Complexity Knobs

Section 5.1 presented four representative tasks from the CLRS-Text benchmark. Here we present the results for all algorithmic tasks. Table 4 explains the complexity knob definition for each task of the CLRS-text benchmark. Figure 7 shows the accuracy and compute allocation plots for all the algorithmic tasks in the benchmark. Our conclusion from Section 6.1 that ANIRA models learn reasonable compute allocation policies holds generally for all CLRS tasks. Also, we see the same generalization failure discussed in Section 6.2 on all tasks.

A.2. Model and Training Configuration

Table 5 presents the detailed model configuration and hyperparameter used to train the ANIRA models on each task. The ANIRA models were implemented by modifying the Llama 2 model implementation available from Huggingface Transformers library.

For ANIRA with hidden size 512, each of the Prelude, recurrent core, and Coda has about 4.2M parameters. For MANO we use hidden size 256, where each component has about 1.5M parameters. The corresponding decider module has about 1.0M parameters (hidden size 512) and about 0.4M parameters (hidden size 256), respectively. In total the ANIRA models with hidden size of 512 have about 13.7M parameters and the models with hidden size of 256 have about 5.1M parameters.

The deciders are Multilayer Perceptrons (MLPs) matching the MLPs used elsewhere in the model. The architecture consists of layer normalization with learnable affine parameters, followed by a MLP. The first linear layer projects the H-dimensional

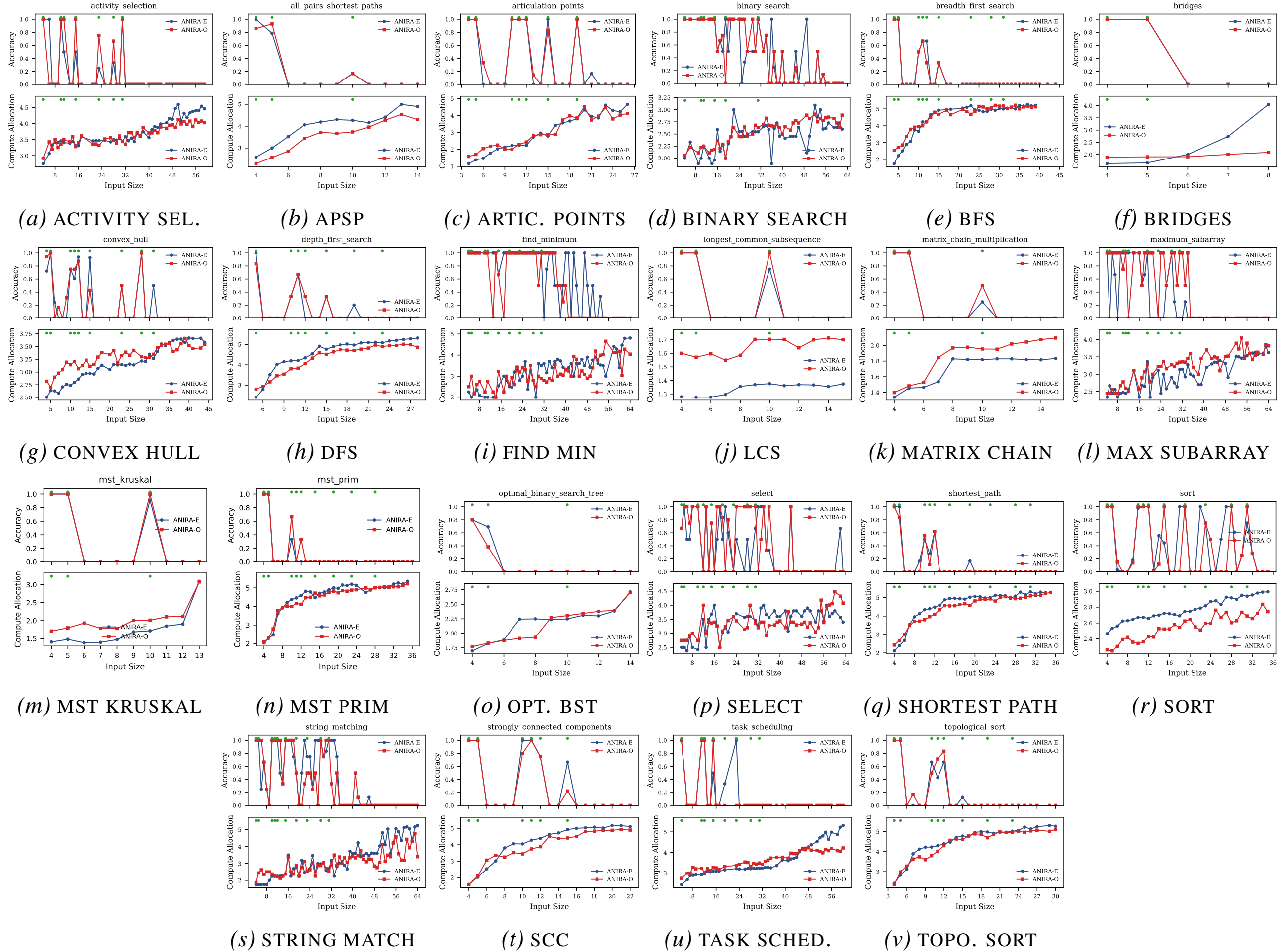


Figure 7. CLRS task complexity vs accuracy (top) and compute allocation $\bar{d}$ (bottom) for all 21 tasks. Green markers indicate input sizes seen during training. ANIRA compute allocation tracks task complexity. However, we observe that task accuracy drops sharply at input sizes not covered in training set, indicating interpolation and extrapolation failure. Note, we ignore SEGMENTS INTERSECTION problem as it's complexity does not vary in the dataset.

hidden state to an intermediate dimension I (set to 4H, except for MANO, for which it is 6.5H) without bias, followed by a SiLU nonlinearity. The second linear layer, which includes a bias term, maps from the intermediate dimension to the output space. The two ANIRA variants differ in their output space. ANIRA-E produces D logits corresponding to a categorical distribution over depth levels. These logits are centered by subtracting their mean before applying the softmax function, which stabilizes training. ANIRA-O instead produces a single logit that is passed through a sigmoid function to obtain the halting probability.

We use the AdamW (Loshchilov & Hutter, 2019) optimizer with a constant learning rate with warmup schedule. Warmup steps were set to 1000. We trained all ANIRA models for 250k steps. Each model was trained on a single NVIDIA A100 GPU. Each training run took about seven hours.

For the natural language experiments in section 6.6, we train the models for 5B tokens, with a batch size of 32 and sequence length of 4096, learning rate of $2e-5$, γ of $1e-1$, on four NVIDIA A100 GPUs, each with 80GB VRAM, which took about a day.

A.3. Results on DEPO: K-Step Successor in a Directed Cyclic Graph

DEPO evaluates multi-hop retrieval ability. Each instance defines a directed cycle over N nodes. The input presents this cycle as a shuffled list of directed edges, followed by up to $\min(N, 10)$ queries. Each query specifies a starting node and a step count K, and the target is the Kth successor of the start node along the cycle. We serialize queries as `<query-k>`

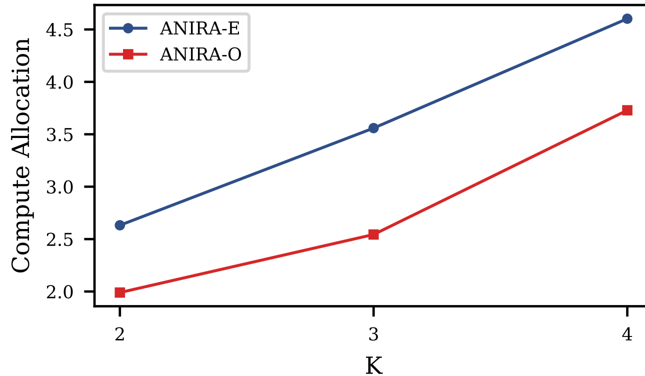


Figure 8. Compute allocation vs. complexity for DEPO

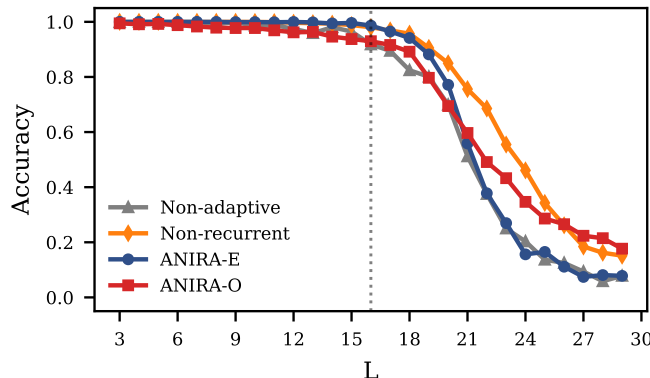


Figure 9. MANO extrapolation: expression length L vs. accuracy. The dashed line indicates the maximum L seen during training.

`<q> <ans> <y> <eoa>`, where the query token `<query-k>` encodes K in the token identity (e.g., `<query-03>`). Node names are single tokens, so each query's answer is a single token; supervision is applied only to that answer token.

Complexity knob. Recovering the Kth successor requires composing K successor steps, yielding an inherent sequential cost $\Theta(K)$.

Compute allocation. We train a single model on a mixture of instances with varying N and K up to maximum values 50 and 4, and then evaluate compute allocation using fixed-K evaluation sets, stratified by K (averaging over the evaluated N values). Using the inference-time depth selections described in Section 3.2.3, we compute the mean allocated depth $\bar{d}$ over the answer tokens. Figure 8 show that $\bar{d}$ increases with K for both variants, indicating that ANIRA learns to allocate more recurrent computation to instances requiring longer iterative retrieval.

A.4. Extrapolation Beyond the Training Range

We probe out-of-distribution generalization by evaluating on MANO instances with expression length L beyond the training range $L \in \{3, \ldots, 16\}$. Figure 9 shows accuracy vs L for ANIRA-E, ANIRA-O, a non-adaptive model and standard Transformer model without weight sharing.

All four models achieve near-perfect accuracy within the training range, but performance degrades sharply once L exceeds the training maximum, indicating limited extrapolation to longer expressions. ANIRA-O degrades more gracefully and maintains higher accuracy than both ANIRA-E and the non-adaptive baseline at the largest L, but the overall accuracy remains low in the extrapolation regime. Neither recurrence/weight sharing nor adaptive depth allocation substantially improves extrapolation, although online halting provides a modest robustness benefit.

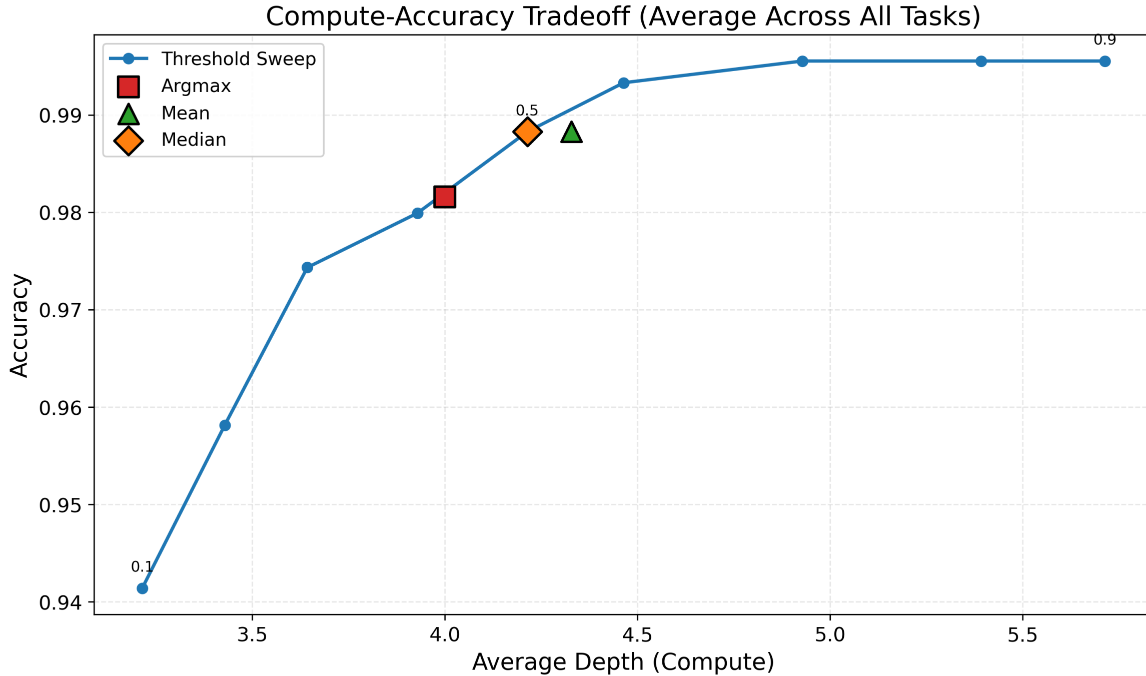


Figure 10. ANIRA-E: Accuracy for different depth choice during inference and cumulative threshold sweep.

A.5. Pareto Frontier for ANIRA-O

The ANIRA-O model allows different choices for the depth choice during inference. In the main paper, we exclusively used the mode of the distribution during inference. On Figure 10, we show the model's accuracy for mean and median choice for the BREVO task. The different choices lead to minor variance in accuracy and compute allocation.

Further, we study the compute-accuracy tradeoff by changing the cumulative threshold. Figure 10 shows that by choosing different thresholds we can trade off between task accuracy and compute cost.

A.6. MANO: Question Token Compute Allocation Piecewise model

The following piecewise parametric model was found to fit ANIRA-E question-token compute allocation $\hat{c}(t)$ on MANO:

$$\hat{c}(t) = \begin{cases} 2.678 & \text{operand}, \ r_t = 1 \\ 1.511 & \text{operand}, \ r_t = 2 \\ 1.000 & \text{operator}, \ r_t = 0 \\ 4.088 + 0.625\, s_t + 0.261\, d_t & \text{operator}, \ r_t = 1 \\ 1.104 + 0.337\, d_t & \text{operator}, \ r_t = 2 \end{cases} \tag{18}$$

which attains $R^2 \approx 0.63$, indicating that most systematic variation in question-time compute is explained by a small set of prefix-observable parse-state features.

A.7. Incremental Parser Features

We extract syntactic complexity features using an incremental prefix PCFG parser (Nowak & Cotterell, 2023). The parser maintains two chart data structures. The first is $\beta[i, X, k]$, which stores *inside probabilities*, i.e., the probability that nonterminal X derives the substring spanning positions $[i, k)$ according to the grammar. The second is an auxiliary structure $\gamma[i, j, Y, Z]$, which accumulates probabilities over partial derivations where nonterminal Y has been recognized over positions $[i, j)$ and nonterminal Z is expected to continue from position j. Intuitively, γ tracks intermediate parsing states where the parser has identified a left child constituent and is searching for a matching right child.

After appending a token at position N, we compute the following features:

- **active_gamma_slice** (parse-space expansion): The number of non-zero entries in γ at the most recent split point, defined as $|\{(i, Y, Z) : \gamma[i, N-1, Y, Z] \neq 0\}|$. This quantity measures the expansion of the parse space, where higher values indicate a greater number of candidate continuations under consideration.
- **active_beta_end** (parse convergence): The number of non-zero entries in β that terminate at the current position, defined as $|\{(i, X) : \beta[i, X, N] \neq 0\}|$. This quantity counts the number of completed constituent spans at the current position.
- **ops_add** and **ops_mul**: The total number of addition and multiplication operations, respectively, performed during the processing of the current token. These quantities measure the arithmetic intensity of each parsing step.

A.8. BREVO Answer Token Compute Allocation Analysis Details

This appendix provides the methodological details and full regression results for the BREVO answer-token compute allocation analysis summarized in Section 6.4.

A.8.1. Task Description and Data Collection

The BREVO task requires the model to output the dependencies of a query node in a directed acyclic graph (DAG), ordered by the deterministic depth-first search (DFS) postorder traversal used to construct the target answer. Graph instances range from $N = 3$ to $N = 30$ nodes. We analyze compute allocation on 210,640 answer tokens from 26,641 samples for ANIRA-E and 14,046 answer tokens from 1,718 samples for ANIRA-O.

For each answer token, the model produces a distribution over possible compute depths. We summarize this distribution by its expected depth,

$$c_{it} = \sum_{d=1}^{D} d\, p_{itd},$$

where p_{itd} is the model's predicted probability of using depth d for answer token t in example i. We use this continuous quantity as the regression target as compared to the actual depth (argmax/median), it explains 11% additional variance while avoiding discretization artifacts.

A.8.2. Feature Extraction

At each answer token position, we extract features that characterize graph structure and the algorithmic execution state. Structural features include: graph size N, hub structure (out-degree), in-degree, and subtree size. Algorithmic state features include: DFS traversal depth, active frontier size (nodes in search queue), newly enabled nodes (dependencies satisfied by current output), and graph distance to query node.

A.8.3. Regression Methodology

Our analysis proceeds in four stages. First, we address multicollinearity by removing features with pairwise correlations $|r| > 0.8$ (retaining the feature more correlated with the target) and computing Variance Inflation Factors (VIF), retaining only features with VIF < 5. This eliminates `out_degree_with_query` ($r = 0.93$ with `out_degree_dep`) and `max_pred_exp_depth` ($r = 0.998$ with `avg_pred_exp_depth`).

Second, we perform ablation analysis, measuring each feature's marginal contribution as $\Delta R^2 = R^2_{\text{full}} - R^2_{\text{without feature}}$. Third, we select features with $\Delta R^2 > 0.5\%$ for the final model. Fourth, we validate feature importance by parameterizing N as a categorical variable (one-hot encoded with the first category as baseline), isolating feature effects within graph size to test whether the features explain compute allocation or act as merely proxy for problem scale.

A.8.4. Results: ANIRA-E Model

Table 6 presents results for both linear and categorical N parameterization. Graph size dominates the linear model, contributing $\Delta R^2 = 28.29\%$, with coefficient $+0.059$ indicating approximately 0.06 additional layers per node. Hub

structure contributes $\Delta R^2 = 4.41\%$ with coefficient $+0.152$. Remaining features (subtree size, frontier size, newly enabled nodes) contribute less than 1% each.

For the categorical N parameterization, the substantial improvement ($\Delta R^2 = +4.71\%$) indicates strong non-linear size dependence. Hub structure maintains high feature importance ($\Delta R^2 = 3.51\%$), indicating it is used by ANIRA-E to determine compute allocation beyond just graph size N. Other features remain below 1% ΔR^2 under categorical N, suggesting that their contributions are comparatively small after controlling for graph size.

Table 6. ANIRA-E feature importance under linear and categorical N parameterization. Ratio indicates $\Delta R^2_{\text{categorical}}/\Delta R^2_{\text{linear}}$, measuring feature retention when controlling for non-linear size effects.

Feature	Linear ΔR^2 (%)	Linear coef.	Categorical ΔR^2 (%)	Categorical coef.	Ratio
N (graph size)	28.29	+0.059	—	—	—
Hub structure	4.41	+0.152	3.51	+0.134	0.80
Subtree size	0.70	−0.022	0.75	−0.026	1.08
Frontier size	0.64	+0.049	0.55	+0.047	0.86
Newly enabled	0.55	−0.081	0.46	−0.079	0.84
DFS depth	—	—	< 0.5	−0.005	—
Intercept	—	+3.21	—	—	—
R^2	68.66%		73.37%		

A.8.5. Results: ANIRA-O Model

Table 7 presents results for both linear and categorical N parameterization. Unlike ANIRA-E, importance is distributed across features. In the linear model, DFS depth leads with $\Delta R^2 = 5.16\%$ (coefficient $+0.303$), followed by graph size ($\Delta R^2 = 4.83\%$, coefficient $+0.032$), newly enabled nodes ($\Delta R^2 = 3.75\%$), frontier size ($\Delta R^2 = 3.63\%$), and hub structure ($\Delta R^2 = 3.02\%$).

For categorical N parameterization, the model reaches $R^2 = 70.88\%$. The minimal improvement ($\Delta R^2 = +0.89\%$) indicates nearly linear size dependence. Critically, all features maintain their importance, confirming they track algorithmic execution state independent of problem scale. DFS depth, newly enabled nodes, and frontier size exhibit ratios of 0.99, demonstrating robustness to N parameterization.

Table 7. ANIRA-O feature importance under linear and categorical N parameterization. Ratio indicates $\Delta R^2_{\text{categorical}}/\Delta R^2_{\text{linear}}$, measuring feature retention when controlling for non-linear size effects.

Feature	Linear ΔR^2 (%)	Linear coef.	Categorical ΔR^2 (%)	Categorical coef.	Ratio
N (graph size)	4.83	+0.032	—	—	—
Hub structure	3.02	+0.171	2.85	+0.168	0.94
Frontier size	3.63	+0.170	3.60	+0.171	0.99
Newly enabled	3.75	−0.287	3.72	−0.286	0.99
DFS depth	5.16	+0.303	5.12	+0.302	0.99
Intercept	—	+2.09	—	—	—
R^2	70.00%		70.88%		

A.8.6. Model Comparison

Table 8. Comparison of linear vs categorical N parameterization. The magnitude of R^2 improvement when using categorical N reveals the degree of non-linear size dependence in each model's compute allocation strategy.

Model	R^2 (Linear N)	R^2 (Categorical N)	Improvement
ANIRA-E	68.66%	73.37%	+4.71%
ANIRA-O	70.00%	70.88%	+0.89%

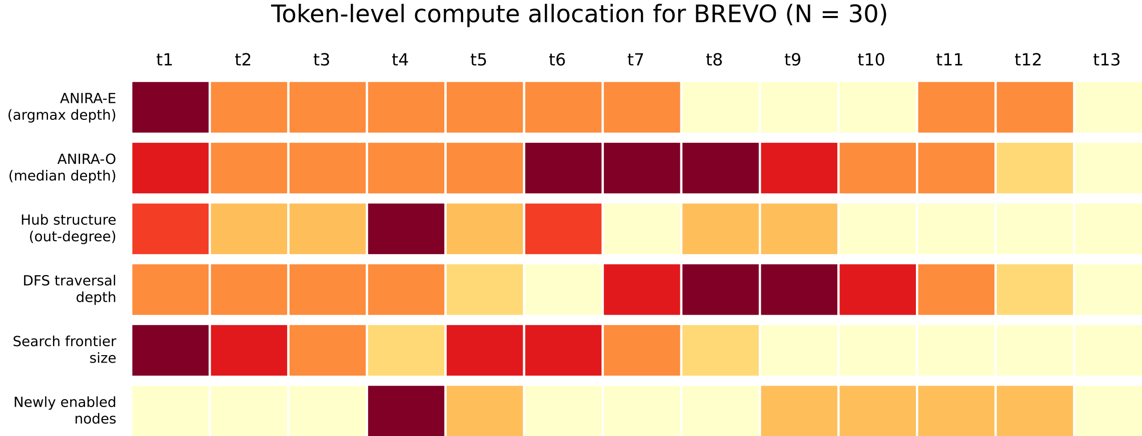


Figure 11. Token-level compute allocation on a representative BREVO example with graph size $N = 30$. Columns are answer tokens, predicted perfectly by both ANIRA-E and ANIRA-O. The top two rows show compute allocated by ANIRA-E and ANIRA-O, while the remaining rows show graph- and traversal-derived token features. Darker cells indicate larger values.

Table 8 compares regressions that use a linear graph-size term with regressions that use categorical graph-size parameterization. For ANIRA-E, tbe substantial improvement of the categorical N parametrization model ($+4.71\%$), indicates non-linear graph-size effects. For ANIRA-O, the slight improvement ($+0.89\%$), indicates that its size dependence is closer to linear. This pattern is consistent with the feature-importance results above: ANIRA-E allocation is determined by graph size and hub structure, whereas ANIRA-O allocation is explained more by DFS depth, frontier size, and newly enabled nodes.

A.8.7. Discussion

The categorical N analysis reveals qualitatively different learned behaviors despite similar task performance. ANIRA-E's compute allocation strategy relies primarily on structural heuristics, with graph size contributing 28% of explained variance (linear model) and showing strong non-linear effects. Hub structure maintains moderate importance (3.51%) when controlling for size, but algorithmic state features contribute minimally.

In contrast, ANIRA-O exhibits balanced feature importance with no dominant predictor. The algorithmic state features (DFS depth, frontier size, newly enabled nodes) maintain their explanatory power when controlling for graph size (ratios ≈ 0.99). The minimal categorical N improvement ($+0.89\%$) indicates ANIRA-O's compute allocation scales approximately linearly with problem size.

These findings suggest that architectural choices (predicting depth vs deciding when to halt) induce fundamentally different learned strategies. ANIRA-E learns to estimate problem difficulty from structure and allocate compute accordingly, while ANIRA-O learns to track algorithmic state and decide when computation is sufficient.

A.8.8. Formal Definition of the Token-Level Features

Let $G_i = (V_i, E_i)$ be the DAG for example i, where an edge $u \to v$ means that u is a direct dependency, or predecessor, of v. Let q_i be the query node. The gold answer sequence

$$A_i = (a_{i1}, \ldots, a_{iT_i})$$

is obtained by running the deterministic BREVO DFS from q_i over predecessor edges in sorted order, emitting nodes in postorder, and then removing q_i. Let $\mathcal{A}_i = \{a_{i1}, \ldots, a_{iT_i}\}$ be the corresponding answer set. For answer position t, write $v_{it} = a_{it}$ for the emitted node and

$$M_{i,t} = \{a_{i1}, \ldots, a_{it}\}, \qquad M_{i,0} = \emptyset$$

for the set of answer nodes emitted through position t. All features below are computed at the answer-token position corresponding to v_{it} and are restricted to $\mathcal{A}_i$.

Hub structure (out-degree). Let

$$\mathrm{Succ}_i(v) = \{u : (v, u) \in E_i\}.$$

The hub structure feature is the answer-restricted out-degree of the emitted node:

$$h_{it} = |\mathrm{Succ}_i(v_{it}) \cap \mathcal{A}_i|.$$

It counts direct answer nodes that depend on v_{it}; the query node itself is not included. Equivalently, it measures how many future answer nodes directly depend on the current node.

DFS depth. Let $r_i(v)$ be the recursion depth at which v is emitted by the deterministic DFS used to construct A_i. The query node starts at depth 0, and each recursive call to a predecessor increases depth by one. For nodes reachable through multiple paths, $r_i(v)$ is the depth from the first DFS visit. The DFS-depth feature is

$$r_{it} = r_i(v_{it}).$$

Frontier size. Before emitting token t, define the topological frontier

$$F_{i,t-1} = \{u \in \mathcal{A}_i \setminus M_{i,t-1} : \mathrm{Pred}_i(u) \cap \mathcal{A}_i \subseteq M_{i,t-1}\}.$$

These are remaining answer nodes whose dependencies within the answer set have already been emitted. The frontier-size feature is

$$f_{it} = |F_{i,t-1}|.$$

Newly enabled nodes. After emitting v_{it}, define $F_{i,t}$ analogously using $M_{i,t}$. The newly-enabled feature is

$$g_{it} = |F_{i,t} \setminus (F_{i,t-1} \setminus \{v_{it}\})|.$$

It counts answer nodes that become topologically available because v_{it} has just been emitted.