A Minimum Description Length Approach to Grammar Inference in Neural Networks

Matan Abudy[1]* MATAN.ABUDY@GMAIL.COM
Orr Well[1]* ORRWELL@MAIL.TAU.AC.IL
Emmanuel Chemla[2] EMMANUEL.CHEMLA@ENS.PSL.EU
Roni Katzir[1] RKATZIR@TAUEX.TAU.AC.IL
Nur Lan[2] NUR.LAN@ENS.PSL.EU

[1] *Tel Aviv University,* [2] *École Normale Supérieure*

Abstract

Accurate generalization to symbolic grammars from realistic amounts of data lies at the heart of many reasoning tasks. Yet it remains a challenge for state-of-the-art neural networks, even when the architecture is known to be expressive enough to represent the target grammar. Using a variety of formal language tasks we show that the choice of regularization method plays a crucial role: when trained on formal languages with standard regularization (L_1, L_2, or none), expressive RNN architectures not only fail to converge to correct solutions but are actively pushed away from perfect initializations. In contrast, applying the Minimum Description Length (MDL) principle to balance model complexity with data fit provides a theoretically grounded regularization method. Using MDL, perfect solutions are selected over approximations, independently of the optimization algorithm. We propose that unlike existing regularization techniques, MDL introduces the appropriate inductive bias for grammar inference, which effectively counteracts overfitting and promotes generalization.

1. Introduction

Modern neural networks can closely approximate solutions to highly complex problems. However, tasks requiring formal reasoning and accurate generalization still show a substantial gap between state-of-the-art models and human intelligence. For example, although models perform well on small-digit arithmetic, they often fail on large-digit numbers in counting, addition, and multiplication (Gambardella et al., 2024; Yehudai et al., 2024; Brown et al., 2020). They also struggle with multi-step, recursive and compositional reasoning (Dziri et al., 2023; Valmeekam et al., 2022; Guzman et al., 2024), fail tests that break co-occurrence patterns (Nezhurina et al., 2024; Wu et al., 2024; Bertolazzi et al., 2024) and are sensitive to slight changes in task presentation (Vafa et al., 2025; Mirzadeh et al., 2024). In some cases, chain-of-thought approaches do not lead to the desired behavior either (Nezhurina et al., 2024; Dziri et al., 2023).

All of these tasks involve underlying rules, syntactic or semantic in nature. Models struggle especially with inductive reasoning, where they are required to infer these rules from observed data (Hua et al., 2025). This inability to generalize beyond the training

* Equal contribution.

© 2026 M. Abudy, O. Well, E. Chemla, R. Katzir & N. Lan.

distribution could be explained if the models rely on heuristics and approximations rather than learning the actual rules. For some models and tasks, it was shown that this is indeed the case (Zhang et al., 2022; Nikankin et al., 2024; Vafa et al., 2025).

Problems may arise from the choice of architecture, objective, or optimization algorithm. Some neural network architectures, including Transformers (Vaswani et al., 2017), are in theory expressive enough to represent correct solutions to a wide range of problems (Siegelmann and Sontag, 1992; Weiss et al., 2021; Pérez et al., 2021; El-Naggar et al., 2023; Stogin et al., 2024). We therefore ask what part of the problem lies in the choice of the objective function itself. As the problem is with generalization, we focus on the part of the objective meant to counteract overfitting: regularization. A regularization term is a component added to the objective function in order to distinguish between models with similar training performance and favor solutions that generalize beyond the training set rather than overfit to it.

We propose Minimum Description Length (MDL) regularization as an inductive bias that can substantially improve performance on tasks that require systematic generalization. Our work focuses on formal languages, but many of the failed tasks mentioned above can be framed in a similar way. Formal languages thus provide a natural benchmark for evaluating generalization in neural networks.

Contributions. We provide empirical evidence for the adequacy of MDL as a regularization method in neural networks. The current work makes three main contributions: (i) Extending previous work on MDL regularization to context-free grammars beyond counting and to more naturalistic grammars. (ii) Providing a systematic comparison of MDL with common regularization methods such as L_1, L_2, and absence of regularization, showing that only MDL consistently preserves or compresses perfect solutions, while other methods push models away from these solutions and degrade their performance. (iii) Establishing the success of MDL across learning settings (architecture search vs. fixed-architecture training) and optimization algorithms (genetic search vs. gradient descent).

All code and experimental data are publicly available at `github.com/taucompling/mdl-reg-approach`.

1.1. Minimum Description Length (MDL)

The MDL (Solomonoff, 1964; Rissanen, 1978) approach to learning avoids overfitting by balancing data fit with model complexity. Maximizing data fit is common to standard training approaches, but MDL imposes an additional requirement of model simplicity, which prevents memorization of accidental patterns. Since identification of meaningful regularities allows compression, data fit is measured by its description length. Similarly, model complexity is measured by the model's encoding length.

Formally, we aim to find a model that minimizes the MDL objective $|G| + |D : G|$, where $|G|$ is the encoding length of a grammar (i.e., a hypothesis) represented by the model, and $|D : G|$ is the description length of the data given this grammar. Both terms are essential: including $|D : G|$ in the objective ensures that the model fits the data well, while including $|G|$ controls for the complexity of the model itself.

For neural networks, $|D : G|$ under Shannon-Fano coding (Shannon, 1948) is equivalent to log surprisal, or the standard cross-entropy (CE) loss. MDL can thus be viewed as a

form of regularization: minimizing the MDL score is equivalent to minimizing CE loss plus a regularization term that reflects the information content (complexity) of the network.

There are *a priori* reasons to favor MDL over standard regularization. Standard methods constrain weight magnitudes, but even numerically small weights can "smuggle" information — for example, by exploiting their high precision to encode a bit string after the decimal point (e.g., 0.001010101...). This also shows that sparsity alone does not guarantee generalization: with unlimited precision, a single non-zero weight is, in principle, enough to encode an entire corpus. MDL offers a principled alternative: unlike standard regularization, it accounts for the full information content of the network and penalizes any form of information smuggling or memorization.

MDL is also cognitively motivated, as human learning behavior aligns with MDL and the related Bayesian framework (assuming a simplicity prior). For example, Orbán et al. (2008) show that subjects identify basic elements in images in line with Bayesian predictions. Other experiments show similar generalization patterns in learning new words from very few examples ("fast mapping") (Xu and Tenenbaum, 2007; Schmidt, 2009; Tenenbaum et al., 2011).

1.2. Related work

MDL and related Bayesian methods have been applied to a wide range of linguistic phenomena, where generalization from limited data is crucial (Horning, 1969; Berwick, 1982; Stolcke, 1994; Grünwald, 1995; De Marcken, 1996). In phonology, the only learners to date that infer adult-like linguistic knowledge from distributional evidence are based on MDL (Rasin and Katzir, 2016; Rasin et al., 2021).

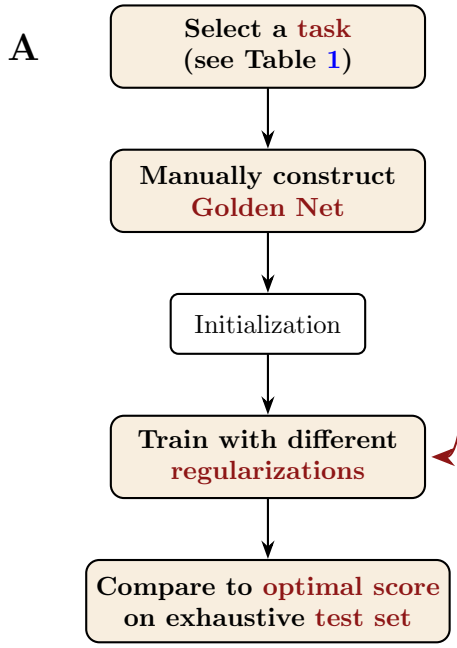
Using a simplicity criterion for artificial neural networks dates back to Hinton and Van Camp (1993); Zhang et al. (1993); Zhang and Mühlenbein (1995) and Schmidhuber (1997). More recently, MDL has been applied to balance memorization and generalization in Hopfield networks (Abudy et al., 2023), and to train a ReLU network to classify primes (Chatterjee and Sudijono, 2024).

Most relevant, Lan et al. (2022) trained Recurrent Neural Networks (RNNs) with MDL to perfectly learn several formal languages, but failed on the more complex Dyck-2 language (see Table 1 for a definition) due to search limitations. Lan et al. (2024) showed that Long Short-Term Memory (LSTM) networks trained on the formal language $a^n b^n$ failed to generalize to the correct grammar with L_1 or L_2 regularization, even when a perfect solution was known to exist in the search space; only MDL identified the correct solution as a minimum of the objective function.

2. Method

2.1. Overview

We define a battery of next-token prediction tasks based on formal languages, each represented by a probabilistic grammar. For each task, we manually construct a "golden" RNN that perfectly solves it by matching the true data distribution. Initializing the learning process with this golden network allows us to examine how each regularization method interacts with correct solutions, without requiring the search to discover them from scratch.



B

	Exp. 1	Exp. 2	Exp. 3
Search			
Genetic algorithm	✔	✔	
Gradient descent			✔
Update			
Weights	✔	✔	✔
Architecture	✔	✗	✗
Regularization Method			
MDL	✔	✔	✗
None	✗	✗	✔
None ($\|G\|$ Limit)	✔	✔	✗
L_1	✔	✔	✔
L_2	✔	✔	✔

Figure 1: **Method overview.** We compare the relative benefit of different regularization methods with the same pipeline (A) across training regimes (B).

We explore three training settings. Experiments 1 and 2 use a genetic algorithm (GA) to evolve networks based on their regularized objective score. Experiment 1 mutates both architecture and weights, while Experiment 2 restricts mutations only to weights. Experiment 3 uses gradient descent (GD) to train the golden network's weights; MDL is excluded due to its non-differentiability. Hyperparameters are detailed in Appendix D.

Performance is evaluated by comparing CE loss on a test set to the theoretical optimum. Figure 1 illustrates the general setup.

2.2. Architecture

We use a general formulation of an RNN, which we refer to as a "free-form RNN". This is a directed graph consisting of units with biases and weighted connections. Connections may be forward or recurrent, and each unit may have a different activation function.

These networks are at least as expressive as ReLU RNNs (as any ReLU RNN can be represented as a free-form RNN) and are therefore capable of performing tasks that require counting (El-Naggar et al., 2023), like some of the tasks described below. The existence of golden networks that perfectly solve all of our tasks further supports their expressivity, including capabilities beyond simple counting.

Table 1: Summary of tasks and example strings.

Task	**Examples**
$\boldsymbol{a^n b^n}$ Equal number of ordered a's and b's	`ab` `aaabbb`
$\boldsymbol{a^n b^n c^n}$ Equal number of ordered a's, b's and c's	`abc` `aaabbbccc`
Dyck-1 Well-matched parentheses	`()()` `((()))(())`
Dyck-2 Well-matched parentheses and brackets	`()[]` `[()](([()]))`
Arithmetic Syntax Nested addition formulas	`((1+1)+(1+1))` `(((1+((1+1)+1))+1)+1)`

Toy-English
Fragment of English with relative clauses
	`dogs think that dogs sleep`
	`dogs think that dogs chase dogs`
	`dogs chase dogs dogs chase`

2.3. Tasks

Each task is defined by a formal language, with datasets generated from its corresponding probabilistic grammar. Table 1 lists representative examples, and full grammar definitions appear in Appendix A.

The tasks are designed to test particular computational abilities: $a^n b^n$ and Dyck-1 require counting; $a^n b^n c^n$ requires two counters, and Dyck-2 requires a stack. The Arithmetic formal language involves nested addition formulas over symbols of '1' and requires a stack to track argument positions. Toy-English is a minimal English fragment with one noun phrase, one transitive verb, one intransitive verb, and one sentential complement verb, supporting object relative clauses. Although the last two tasks are meant to represent more naturalistic grammars, these languages are still defined only by syntax, with no notion of semantic correctness.

2.4. Golden networks

We adopt a strong notion of correctness: a golden network not only assigns the highest probability to the correct next symbol at each timestep; its entire output probability distribution matches the true grammar distribution.

For $a^n b^n$, $a^n b^n c^n$ and Dyck-2, we use golden networks previously discovered by Lan et al. (2022). The golden network for Dyck-1 was found in a GA run under MDL. For Arithmetic Syntax and Toy-English, we constructed golden networks manually and empirically verified their perfect performance. Network diagrams are available in Appendix B.

2.5. Regularization methods

MDL. Following Lan et al. (2022), we calculate the regularization term $|G|$ by encoding the structure of the network: unit count, each unit's type, activation and bias, and the connections (recurrent vs. forward).

Beyond architectural complexity, $|G|$ also reflects the complexity of the weights. Weights and biases are stored as signed fractions, with bit strings formed by concatenating the sign bit, numerator and denominator (each encoded using the prefix-free scheme from Li and Vitányi (2008)). This encoding reflects an intuitive notion of simplicity that goes beyond absolute size: a weight is simple if it has a concise fractional form. For example, 0.1 encodes as 1/10 (simple), while the smaller 0.02234 expands to 1117/50,000 (complex). Thus, 1/10 receives a shorter code. This also illustrates that a naive binary encoding fails to capture our intuitions — 1/10 is simple despite lacking an exact base-2 floating-point representation.

$\boldsymbol{L_1}$**.** A regularization method that encourages sparsity by penalizing absolute weight values:

$$\lambda \sum_{i=1}^{d} |w_i|$$

where λ is the regularization coefficient, d is the number of network weights and w_i is the ith weight.

$\boldsymbol{L_2}$**.** A regularization method aimed at keeping weights small by penalizing their squared values:

$$\lambda \sum_{i=1}^{d} w_i^2$$

No regularization (with a limit on $|G|$). Minimizing CE loss alone leads to overfitting. Specifically, in GA experiments where the architecture itself is allowed to grow, overfitting can take the form of an "exploding" network that grows indefinitely to memorize the training set. To prevent this, we impose a complexity ceiling — three times the golden network's $|G|$ — which charitably introduces an additional level of regularization.

2.6. Evaluation

Accuracy can be misleading: A model can reach 100% next-token accuracy via `argmax` prediction, while misrepresenting the true distribution. And when the underlying grammar is non-deterministic, even a model that matches the true distribution may assign the highest probability to an incorrect symbol (for further discussion see Lan et al. (2023)). Instead of accuracy, we evaluate models using their test set $|D : G|$, equivalent to CE loss. To avoid infinite scores (due to $\log 0$), we smooth the network output distribution by adding 10^{-10} to zero probabilities.

Sampling test strings from the grammar, as done for training, may miss low-probability strings; therefore, we construct exhaustive test sets that contain all strings up to a length threshold, weighted by their true probabilities. These test sets approximate the full grammar distribution; while low training $|D : G|$ may indicate overfitting, low test $|D : G|$ should reflect generalization.

However, as the test set is still a finite approximation, overfitting it remains possible. Thus, we evaluate models not by the absolute value of their test $|D:G|$ score, but by its proximity to the analytically computed optimal score, derived by summing log probabilities under the true distribution. Although a test score below the optimum does not necessarily reflect a better solution, one above the optimum clearly reflects a flawed solution.

3. Experiments and results

3.1. Experiment 1: Genetic architecture search

Training with GD is unsuitable for MDL, as the $|G|$ term is non-differentiable. Following Lan et al. (2022), we optimize networks using a genetic algorithm (GA) (Holland, 1992). We use the Island Model (Gordon and Whitley, 1993; Adamidis, 1994; Cantú-Paz et al., 1998), where the population is divided into equal "islands" that evolve independently with occasional migration. Within each island, networks are mutated and selected based on their regularized objective score. Pseudo-code appears in Appendix C.

The GA mutates networks by adding or removing units or connections, modifying weights and biases, and changing activation functions. Allowed activations include linear, ReLU, tanh, and task-specific functions used by the golden network, enabling the search to leverage the same activations.

Table 2 reports final scores for network complexity ($|G|$) and data fit ($|D:G|$) with regard to the training and test sets, as well as the relative deviation from the optimal score $\Delta(\%) = \frac{|D:G| - \text{Optimal}}{\text{Optimal}} \times 100$. For a given dataset, a positive delta indicates underfitting this dataset, while a negative value indicates overfitting it. Asterisks mark scores that were originally infinite due to a zero probability assigned to the correct next symbol, corrected via smoothing. Figure 2 visualizes these deviations.

MDL-selected networks yield the lowest and closest to optimal test $|D:G|$ scores across all tasks. In addition, MDL tends to preserve the golden network during search, unlike the other regularizers. However, when golden networks are constructed manually they can be larger than necessary; in those cases, MDL compresses them significantly while retaining near-optimal $|D:G|$, producing more compact models.

3.2. Experiment 2: GA in weight-training setting

The genetic algorithm can modify the network architecture during optimization, making it more akin to an architecture search procedure than to standard gradient descent training of the weights. One might argue that since L_1 and L_2 are typically applied during training with a fixed architecture, comparisons to MDL should also be made in that setting.

We believe that the comparison in Experiment 1 is justified: in practice, regularization also guides architecture search, as the process usually also involves weight training. However, we can also compare MDL to standard regularization in a pure weight-training setting. To do that, we initialize the GA population solely with instances of the golden network and restrict mutations to weights and biases, keeping the architecture fixed.

The results are provided in Table 3. Although MDL regularization can effectively control for both architectural and weight complexity, it outperforms L_1 and L_2 even in the

Table 2: **GA architecture search results.** For each task and regularization method, we evaluate the GA-selected network on training samples and an exhaustive test set, reporting the relative gap $\Delta(\%) = \frac{|D:G| - \text{Optimal}}{\text{Optimal}} \times 100$. Asterisks mark originally infinite scores due to zero-probability errors. MDL yields the smallest test gaps and typically preserves or even compresses the golden network.

Task	Regularizer	$\lvert G\rvert$	Train $\lvert D:G\rvert$	Test $\lvert D:G\rvert$	Δ^{Train}_{Optim} (%)	Δ^{Test}_{Optim} (%)
$a^n b^n$	(Golden)	(139)	(1531.77)	(2.94)	(0.0)	(0.0)
	MDL ($\lvert G\rvert$)	139	1529.39	2.94	**-0.2**	**0.1**
	L_1	460	1498.12	2.94*	-2.2	0.2
	L_2	600	1514.66	3.27*	-1.1	11.4
	None (Lim. $\lvert G\rvert$)	430	1488.17	3.32*	-2.8	13.0
$a^n b^n c^n$	(Golden)	(241)	(1483.91)	(2.94)	(0.0)	(0.0)
	MDL ($\lvert G\rvert$)	241	1483.91	2.94	**0.0**	**0.0**
	L_1	366	1482.88	2.94	-0.1	0.1
	L_2	602	1476.71	3.03*	-0.5	3.2
	None (Lim. $\lvert G\rvert$)	736	1460.67	2.96	-1.6	0.6
Dyck-1	(Golden)	(113)	(1544.48)	(1.77)	(-0.0)	(0.0)
	MDL ($\lvert G\rvert$)	113	1544.22	1.77	**-0.0**	**0.1**
	L_1	753	1488.17	1.90*	-3.6	7.5
	L_2	1140	1479.37	1.96*	-4.2	10.7
	None (Lim. $\lvert G\rvert$)	353	1505.61	2.03*	-2.5	15.1
Dyck-2	(Golden)	(579)	(2121.53)	(2.32)	(-0.0)	(0.0)
	MDL ($\lvert G\rvert$)	327	2130.28	2.37	**0.4**	**2.0**
	L_1	1629	2009.61	2.70*	-5.3	16.1
	L_2	2248	1996.82	2.82*	-5.9	21.3
	None (Lim. $\lvert G\rvert$)	1757	1992.89	2.61*	-6.1	12.5
Arithmetic	(Golden)	(967)	(2581.29)	(3.96)	(0.0)	(0.1)
	MDL ($\lvert G\rvert$)	431	2581.07	3.94	**0.0**	**-0.4**
	L_1	1268	2540.22	3.99*	-1.6	1.0
	L_2	1200	2560.62	4.01*	-0.8	1.4
	None (Lim. $\lvert G\rvert$)	3004	2477.42	4.23*	-4.0	7.0
Toy-English	(Golden)	(870)	(2232.74)	(4.49)	(0.0)	(0.0)
	MDL ($\lvert G\rvert$)	414	2231.96	4.57*	**-0.0**	**2.0**
	L_1	1215	2209.25	5.05*	-1.0	12.6
	L_2	1330	2202.41	5.43*	-1.3	21.0
	None (Lim. $\lvert G\rvert$)	2625	2142.40	6.25*	-4.0	39.4

fixed-architecture setting, confirming that it is the appropriate regularizer for simple weight training as well as for architectural search.

3.3. Experiment 3: Gradient descent

One might object that the optimization algorithm plays an important role in promoting generalization, and propose that unlike GA, GD allows standard regularization to avoid

Table 3: **GA weight-training results.** For each task and regularization method, we evaluate the network selected by GA when only mutations of weights and biases are allowed. We measure performance on training samples and on an exhaustive test set, reporting the relative gap $\Delta(\%) = \frac{|D:G| - \text{Optimal}}{\text{Optimal}} \times 100$; * marks originally infinite scores due to zero-probability errors. Once again, MDL yields the smallest gaps from the optimum and tends to preserve or simplify the golden network.

Task	Regularizer	$\lvert G\rvert$	Train $\lvert D:G\rvert$	Test $\lvert D:G\rvert$	Δ^{Train}_{Optim} (%)	Δ^{Test}_{Optim} (%)
$a^n b^n$	(Golden)	(139)	(1531.77)	(2.94)	(0.0)	(0.0)
	MDL ($\lvert G\rvert$)	139	1529.39	2.94	**-0.2**	**0.1**
	L_1	217	1528.92	2.94	-0.2	0.2
	L_2	268	1528.96	2.94	-0.2	0.1
	None (Lim. $\lvert G\rvert$)	385	1524.71	2.94	-0.5	0.2
$a^n b^n c^n$	(Golden)	(241)	(1483.91)	(2.94)	(0.0)	(0.0)
	MDL ($\lvert G\rvert$)	239	1483.91	2.94	**0.0**	**0.0**
	L_1	362	1482.88	2.94	-0.1	0.1
	L_2	445	1482.94	2.94	-0.1	0.0
	None (Lim. $\lvert G\rvert$)	733	1481.57	2.94	-0.2	0.1
Dyck-1	(Golden)	(113)	(1544.48)	(1.77)	(-0.0)	(0.0)
	MDL ($\lvert G\rvert$)	113	1544.22	1.77	**-0.0**	**0.1**
	L_1	259	1537.35	1.79	-0.5	1.1
	L_2	408	1537.28	1.79	-0.5	1.2
	None (Lim. $\lvert G\rvert$)	355	1534.32	1.79	-0.7	1.3
Dyck-2	(Golden)	(579)	(2121.53)	(2.32)	(-0.0)	(0.0)
	MDL ($\lvert G\rvert$)	490	2121.53	2.32	**-0.0**	**0.0**
	L_1	1202	2064.86	2.39*	-2.7	2.8
	L_2	1426	2055.48	2.58*	-3.1	11.2
	None (Lim. $\lvert G\rvert$)	1731	2031.26	2.78*	-4.3	19.5
Arithmetic	(Golden)	(967)	(2581.29)	(3.96)	(0.0)	(0.1)
	MDL ($\lvert G\rvert$)	635	2581.29	3.96	**0.0**	**0.1**
	L_1	1420	2555.63	3.97*	-1.0	0.4
	L_2	1553	2548.16	4.15*	-1.3	5.1
	None (Lim. $\lvert G\rvert$)	2890	2471.51	4.79*	-4.2	21.3
Toy-English	(Golden)	(870)	(2232.74)	(4.49)	(0.0)	(0.0)
	MDL ($\lvert G\rvert$)	552	2237.59	4.52	**0.2**	**0.8**
	L_1	1491	2186.61	5.05*	-2.1	12.7
	L_2	1704	2182.59	6.25*	-2.2	39.3
	None (Lim. $\lvert G\rvert$)	2610	2158.71	6.21*	-3.3	38.4

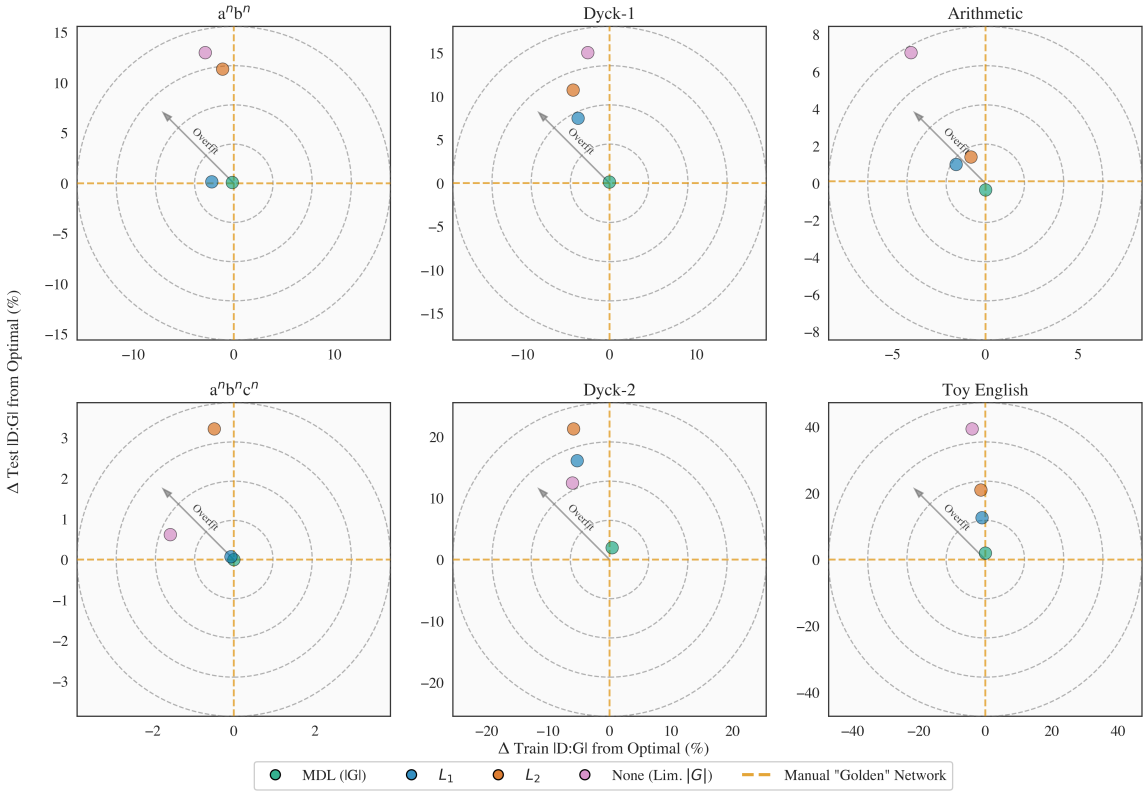


Figure 2: Relative deviation (Δ%) from optimal $|D : G|$ for each final network in Experiment 1, on train (x-axis) and test (y-axis), grouped by task. While MDL remains at or close to the analytical optimum (centered), all other objectives reach a training $|D : G|$ below optimal and a test $|D : G|$ above it, indicating overfitting to the training data.

overfitting. While the dynamics of GD are not fully understood, we argue that it cannot fix a flawed objective. This has already been shown for LSTMs on a^nb^n (Lan et al., 2024); we extend the analysis to new tasks and free-form RNNs.

Unlike GA, GD cannot handle non-differentiable activations. For tasks that require such activations, GD is therefore inapplicable; but for a^nb^n, $a^nb^nc^n$, and Dyck-1, we found golden networks that only use differentiable activations. We train them via backpropagation, starting from the golden weights. Since the architecture is fixed, we approximate $|G|$ by the encoding of the weights, ignoring architectural details. Since MDL cannot be optimized with GD, we compare training with L_1, L_2, or no regularization.

Table 4 reports the results, which suggest that our golden networks are not optima under standard regularization methods. In fact, GD pushes networks away from the proposed golden solution and degrades test performance. Incidentally, we also note that $|G|$ increases significantly, suggesting that MDL would not have selected these overfitting solutions.

Table 4: **Gradient descent results.** Starting from the golden weights, we train using each differentiable regularizer and report the relative gap $\Delta(\%) = \frac{|D:G| - \text{Optimal}}{\text{Optimal}} \times 100$. All objectives drift away from the golden solution and degrade its performance. For Dyck-1, we use a different differentiable golden network with softmax outputs than in the GA experiments, hence the slightly different golden scores.

Task	Regularizer	$\approx \lvert G \rvert$	Train $\lvert D:G \rvert$	Test $\lvert D:G \rvert$	Δ^{Train}_{Optim} (%)	Δ^{Test}_{Optim} (%)
$a^n b^n$	(Golden)	(274)	(1532.33)	(2.94)	(0.0)	(0.1)
	None	5524	1529.48	2.94	-0.1	0.2
	L_1	5854	1543.98	2.95	0.8	0.5
	L_2	5610	1531.69	2.95	-0.0	0.3
$a^n b^n c^n$	(Golden)	(599)	(1484.15)	(2.94)	(0.0)	(0.0)
	None	20043	1482.91	2.94	-0.1	0.1
	L_1	20541	1491.15	2.96	0.5	0.9
	L_2	19909	1487.84	3.00	0.3	2.1
Dyck-1	(Golden)	(148)	(1544.28)	(1.78)	(-0.0)	(0.8)
	None	4876	1544.03	1.78	-0.0	1.0
	L_1	5138	1561.23	1.86	1.1	5.0
	L_2	4874	1562.96	1.84	1.2	4.4

4. Discussion

Our golden networks demonstrate that free-form RNNs can express exact solutions for every task we studied. Ideally, a golden network should correspond to a (at least local) minimum: for an appropriate objective, departing from a perfect solution is acceptable only if it leads to an equivalent or better solution, so training initialized with a golden network should converge to a network that is at least as good. Across all of our experiments, only the MDL-based objective behaved in this appropriate manner.

For an infinite language, CE loss only approximates true error asymptotically; with finite data, minimizing CE alone leads to overfitting. We show that standard regularization does not prevent this: for every model we trained with L_1 or L_2, test-set CE increased after training.

Because they focus solely on weight magnitudes, L_1 and L_2 ignore other important aspects of network complexity, such as the number of units, activations and parameter information content. Even when only weights and biases are manipulated during the search, MDL mitigates overfitting better than standard regularization: minimizing the MDL term $|G|$ prioritizes weights that are simple to describe and penalizes any form of memorization through small but high-precision weights.

Gradient descent is sometimes claimed to induce a bias toward generalization (Cohen-Karlik et al., 2022; Mingard et al., 2025). We have shown that even with GD, standard regularization does not reach perfect solutions; in fact, these solutions are not even minima of the regularized objectives. Success with GD may simply result from imperfect optimization: as GD is prone to getting stuck in local minima, it might simply not fully minimize the

objective — otherwise, we would have gotten a fully overfitting model. But taking the wrong objective and only partially minimizing it cannot make it right; and it is not surprising that better results are obtained when the right regularization term is chosen from the start.

Even LLMs face the need to generalize from a small amount of data in some practical scenarios, e.g., in-context learning (Brown et al., 2020); formal languages provide a natural benchmark because they require exact generalization. The fact that models struggle, despite expressive architecture and effective optimization, indicates that the problem lies in the objective. The same problem is likely to show up in more complex tasks as well: if a model cannot acquire a simple formal grammar that we know it can express, then it may approximate natural language extremely well, but it is unlikely that it will fully model its underlying structure. Indeed, we propose that the choice of regularization also underlies the current model failures mentioned in the introduction. Prior work (Lan et al., 2024) demonstrated that MDL outperforms L_1 and L_2 in LSTMs. We extended these findings to free-form RNNs. Although it has not yet been tested empirically, MDL-based regularization could yield the same benefits for Transformers as well.

Some may object that approximations have proven good enough in real-world tasks such as natural language processing. While approximations are inevitable in noisy or ambiguous settings, an approximation that follows the general rule despite some noise is not the same as one that embodies a misunderstanding of the rule itself. Some MDL-selected networks approximate operations (e.g. true zero probabilities are approximated using softmax) but also preserve the underlying rule structure by effectively implementing mechanisms like counters and stacks.

MDL offers a principled framework for designing learning objectives that really aim for perfect generalization. By promoting generalization, regularization based on MDL may allow models to learn from smaller amounts of data. And since it selects models that are smaller, the resulting models are potentially simpler to analyze, which can also improve interpretability.

Currently, the main limitation of our approach is that the MDL objective is not differentiable, making it better suited to evolutionary search methods that have not yet matched the hardware and software optimizations available for GD. The promise of stronger generalization thus motivates two complementary paths: (i) hardware and software advances that enable non-differentiable optimization to run at better speeds and efficiency, and (ii) differentiable approximations or surrogate losses that preserve MDL's bias toward simplicity and generalization.

Additionally, the tasks we evaluated are relatively small in scale. This allowed us to ensure that they were perfectly understood, but future work should explore the benefits of MDL-based regularization when extended to larger-scale settings, additional architectures, and broader benchmarks.

References

Matan Abudy, Nur Lan, Emmanuel Chemla, and Roni Katzir. Minimum description length Hopfield networks. *arXiv preprint arXiv:2311.06518*, 2023. Presented at the Associative Memory & Hopfield Networks Workshop at NeurIPS 2023.

Panagiotis Adamidis. Review of parallel genetic algorithms bibliography. *Technical Report*, 1994.

Leonardo Bertolazzi, Albert Gatt, and Raffaella Bernardi. A systematic analysis of large language models as soft reasoners: The case of syllogistic inferences. *arXiv preprint arXiv:2406.11341*, 2024.

Robert Cregar Berwick. *Locality principles and the acquisition of syntactic knowledge*. PhD thesis, Massachusetts Inst. of Technology Cambridge, 1982.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Erick Cantú-Paz et al. A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis*, 10(2):141–171, 1998.

Sourav Chatterjee and Timothy Sudijono. Neural networks generalize on low complexity data. *arXiv preprint arXiv:2409.12446*, 2024.

Edo Cohen-Karlik, Avichai Ben David, Nadav Cohen, and Amir Globerson. On the implicit bias of gradient descent for temporal extrapolation. In *International Conference on Artificial Intelligence and Statistics*, pages 10966–10981. PMLR, 2022.

Carl De Marcken. Unsupervised language acquisition. *arXiv preprint cmp-lg/9611002*, 1996.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, et al. Faith and fate: Limits of transformers on compositionality. *Advances in Neural Information Processing Systems*, 36:70293–70332, 2023.

Nadine El-Naggar, Andrew Ryzhikov, Laure Daviaud, Pranava Madhyastha, and Tillman Weyde. Formal and empirical studies of counting behaviour in relu rnns. In *International Conference on Grammatical Inference*, pages 199–222. PMLR, 2023.

Andrew Gambardella, Yusuke Iwasawa, and Yutaka Matsuo. Language models do hard arithmetic tasks easily and hardly do easy arithmetic tasks. *arXiv preprint arXiv:2406.02356*, 2024.

V Scott Gordon and Darrell Whitley. Serial and parallel genetic algorithms as function optimizers. In *ICGA*, pages 177–183, 1993.

Peter Grünwald. A minimum description length approach to grammar inference. In *International joint conference on artificial intelligence*, pages 203–216. Springer, 1995.

Manuel Guzman, Jakub Szymanik, and Maciej Malicki. Testing the limits of logical reasoning in neural and hybrid models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 2267–2279, 2024.

Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13, 1993.

John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

James Jay Horning. *A study of grammatical inference*. Stanford University, 1969.

Wenyue Hua, Tyler Wong, Sun Fei, Liangming Pan, Adam Jardine, and William Yang Wang. Inductionbench: Llms fail in the simplest complexity class. *arXiv preprint arXiv:2502.15823*, 2025.

Nur Lan, Michal Geyer, Emmanuel Chemla, and Roni Katzir. Minimum description length recurrent neural networks. *Transactions of the Association for Computational Linguistics*, 10:785–799, 2022.

Nur Lan, Emmanuel Chemla, and Roni Katzir. Benchmarking neural network generalization for grammar induction. In *Proceedings of the 2023 CLASP Conference on Learning with Small Data (LSD)*, pages 131–140, 2023.

Nur Lan, Emmanuel Chemla, and Roni Katzir. Bridging the empirical-theoretical gap in neural network formal language learning using minimum description length. *arXiv preprint arXiv:2402.10013*, 2024.

Ming Li and Paul Vitányi. *An introduction to Kolmogorov complexity and its applications*, volume 3. Springer, 2008.

Chris Mingard, Henry Rees, Guillermo Valle-Pérez, and Ard A Louis. Deep neural networks have an inbuilt occam's razor. *Nature Communications*, 16(1):220, 2025.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*, 2024.

Marianna Nezhurina, Lucia Cipolina-Kun, Mehdi Cherti, and Jenia Jitsev. Alice in wonderland: Simple tasks showing complete reasoning breakdown in state-of-the-art large language models. *arXiv preprint arXiv:2406.02061*, 2024.

Yaniv Nikankin, Anja Reusch, Aaron Mueller, and Yonatan Belinkov. Arithmetic without algorithms: Language models solve math with a bag of heuristics. *arXiv preprint arXiv:2410.21272*, 2024.

Gergő Orbán, József Fiser, Richard N Aslin, and Máté Lengyel. Bayesian learning of visual chunks by human observers. *Proceedings of the National Academy of Sciences*, 105(7): 2745–2750, 2008.

Jorge Pérez, Pablo Barceló, and Javier Marinkovic. Attention is Turing-complete. *Journal of Machine Learning Research*, 22(75):1–35, 2021. URL https://jmlr.org/papers/v22/20-302.html.

Ezer Rasin and Roni Katzir. On evaluation metrics in optimality theory. *Linguistic Inquiry*, 47(2):235–282, 2016.

Ezer Rasin, Iddo Berger, Nur Lan, Itamar Shefi, and Roni Katzir. Approaching explanatory adequacy in phonology using minimum description length. *Journal of Language Modelling*, 9(1):17–66, 2021.

Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

Jürgen Schmidhuber. Discovering neural nets with low kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–873, 1997.

Lauren A Schmidt. *Meaning and compositionality as statistical induction of categories and constraints*. PhD thesis, Massachusetts Institute of Technology, 2009.

Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 440–449, 1992.

Ray J Solomonoff. A formal theory of inductive inference. part i and ii. *Information and control*, 7(1):1–22, 1964.

John Stogin, Ankur Mali, and C Lee Giles. A provably stable neural network turing machine with finite precision and time. *Information Sciences*, 658:120034, 2024.

Andreas Stolcke. *Bayesian learning of probabilistic language models*. University of California, Berkeley, 1994.

Joshua B Tenenbaum, Charles Kemp, Thomas L Griffiths, and Noah D Goodman. How to grow a mind: Statistics, structure, and abstraction. *science*, 331(6022):1279–1285, 2011.

Keyon Vafa, Justin Chen, Ashesh Rambachan, Jon Kleinberg, and Sendhil Mullainathan. Evaluating the world model implicit in a generative model. *Advances in Neural Information Processing Systems*, 37:26941–26975, 2025.

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Thinking like transformers. In *International Conference on Machine Learning*, pages 11080–11090. PMLR, 2021.

Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. Reasoning or reciting? exploring the capabilities and limitations of language models through counterfactual tasks. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1819–1862, 2024.

Fei Xu and Joshua B Tenenbaum. Word learning as bayesian inference. *Psychological review*, 114(2):245, 2007.

Gilad Yehudai, Haim Kaplan, Asma Ghandeharioun, Mor Geva, and Amir Globerson. When can transformers count to n? *arXiv preprint arXiv:2407.15160*, 2024.

Byoung-Tak Zhang and Heinz Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38, 1995.

Byoung-Tak Zhang, Heinz Muhlenbein, et al. Evolving optimal neural networks using genetic algorithms with occam's razor. *Complex systems*, 7(3):199–220, 1993.

Honghua Zhang, Liunian Harold Li, Tao Meng, Kai-Wei Chang, and Guy Van den Broeck. On the paradox of learning to reason from data. *arXiv preprint arXiv:2205.11502*, 2022.

Appendix A. Task-specific probabilistic grammars

We generate training strings using probabilistic grammars, most of them context-free, and evaluate generalization using exhaustive test sets that enumerate all valid strings up to a specified length or count. Below we describe the grammar and test set construction for each task.

$a^n b^n$, $a^n b^n c^n$

We sample $n \sim \text{Geometric}(p = 0.3)$ and produce either $a^n b^n$ or $a^n b^n c^n$. The test set exhaustively includes strings for all values from $n = 1$ up to the largest n in the training set, plus 1000 additional values. For example, if $n_{\max} = 20$ in training, the test set includes all $n = 1, \ldots, 1020$.

Dyck-1, Dyck-2

Training examples are valid Dyck-1 or Dyck-2 strings sampled from the grammars in Tables 5 and 6 respectively. To avoid excessive runtime and memory load, the maximal possible training string length is 200. The exhaustive test set includes all Dyck-1 or Dyck-2 strings, depending on the task, of length up to 10.

Table 5: Dyck-1 grammar for generating strings

Production Rule	Probability
$S \to [\, S \,]\, S$	0.33333
$S \to \varepsilon$	0.66667

Table 6: Dyck-2 grammar for generating strings

Production Rule	Probability
$S \to [\, S \,]\, S$	0.166665
$S \to (\, S \,)\, S$	0.166665
$S \to \varepsilon$	0.66667

Arithmetic Syntax

Training strings are sampled from the grammar in Table 7. The test set enumerates all syntactically valid expressions of length up to 40.

Table 7: Arithmetic Syntax grammar for generating strings

Production Rule	Probability
$S \to (\mathrm{E} + \mathrm{E})$	1.0
$E \to 1$	0.67
$E \to (\mathrm{E} + \mathrm{E})$	0.33

Toy-English

Training data consists of strings up to length 200. The test set includes all grammatical strings up to length 20.

Table 8: Toy-English grammar for generating strings

Production Rule	Probability
$S \to$ NP VP	1.0
NP $\to$ N	0.75
NP $\to$ N RC	0.25
RC $\to$ NP Vtr	1.0
VP $\to$ Vtr NP	0.34
VP $\to$ Vi	0.33
VP $\to$ Vcp S	0.33
N $\to$ dogs	1.0
Vtr $\to$ chase	1.0
Vi $\to$ sleep	1.0
Vcp $\to$ think that	1.0

Appendix B. Golden networks

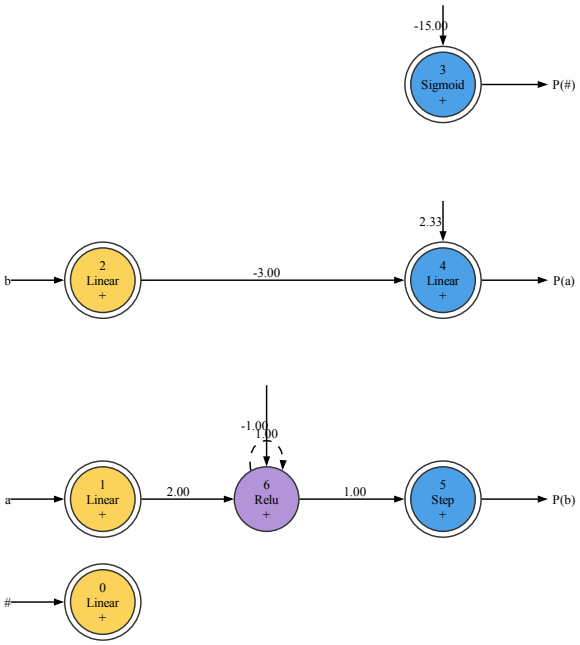


Figure 3: Golden network for the $a^n b^n$ task.

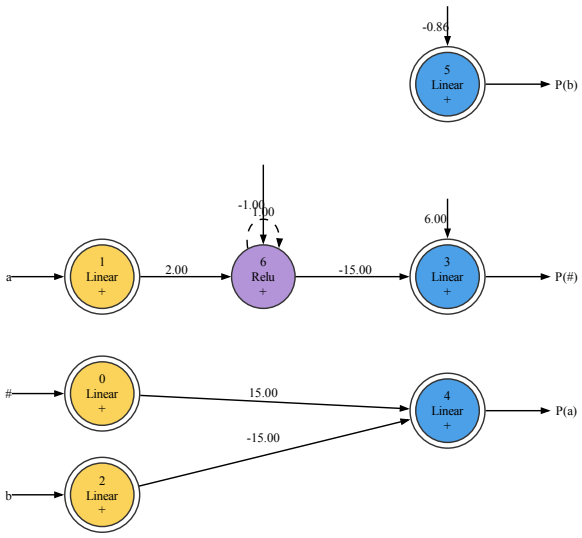


Figure 4: Differentiable golden network for the $a^n b^n$ task, used in Experiment 3.

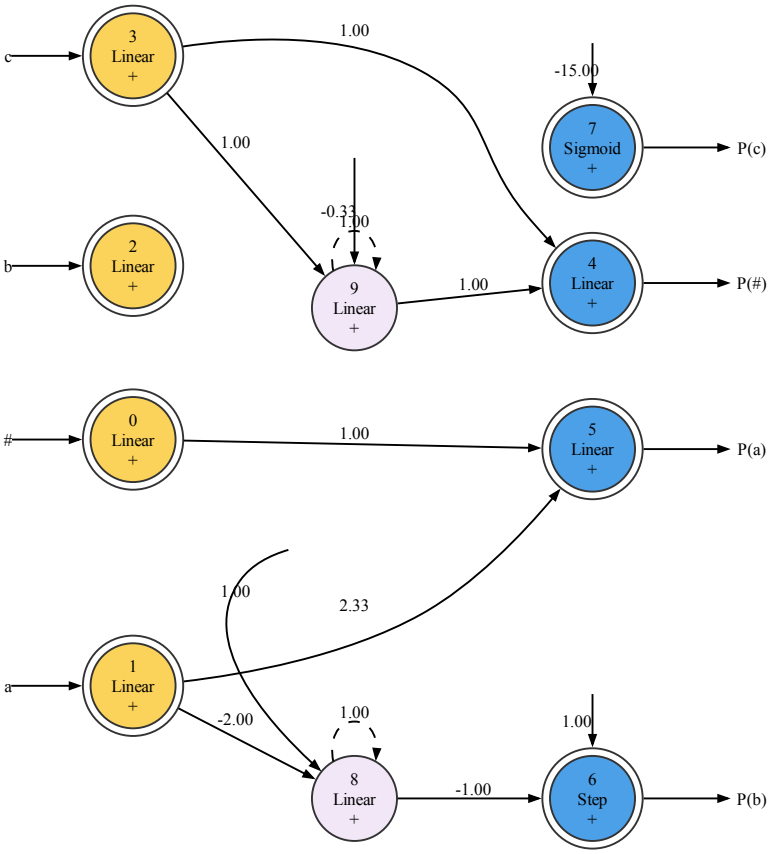


Figure 5: Golden network for the $a^n b^n c^n$ task.

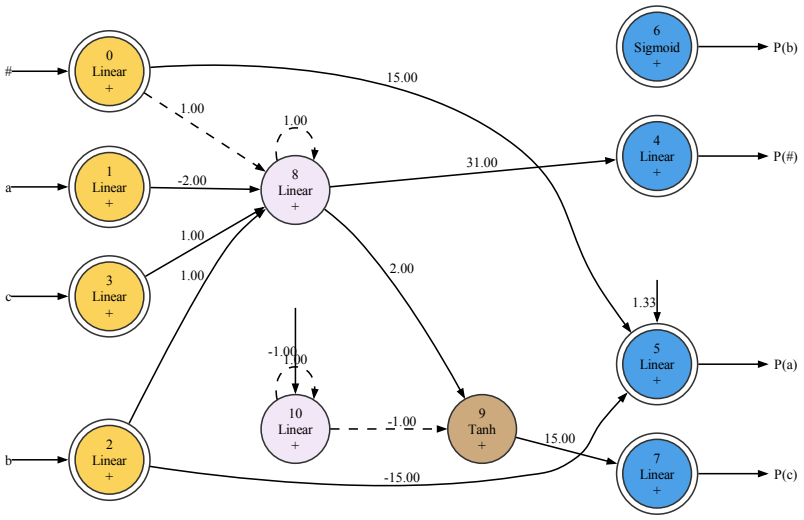


Figure 6: Differentiable golden network for the $a^n b^n c^n$ task, used in Experiment 3.

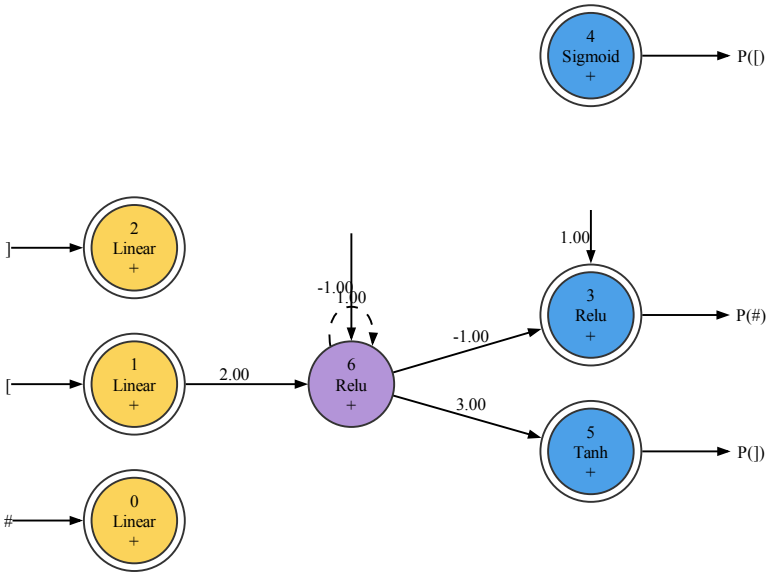


Figure 7: Golden network for the Dyck-1 task.

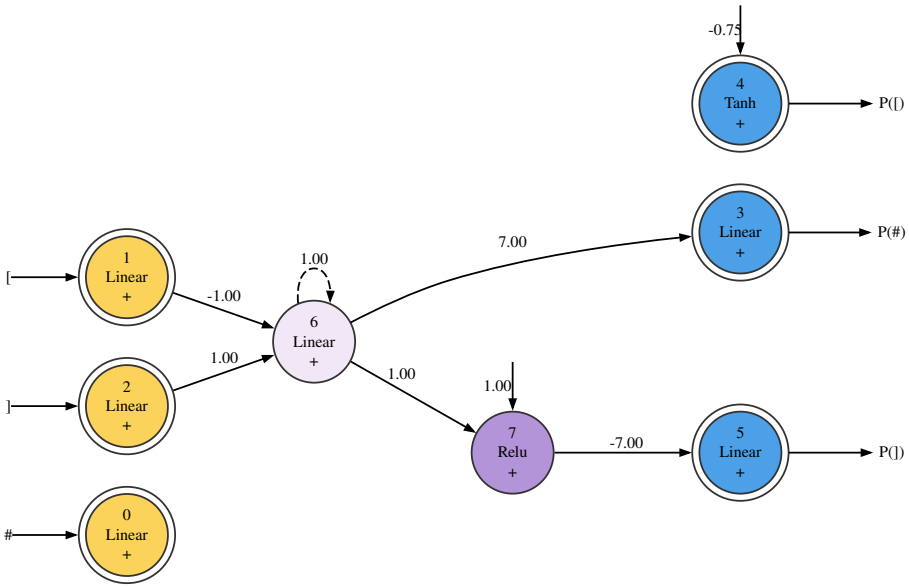


Figure 8: Differentiable golden network for the Dyck-1 task, used in Experiment 3.

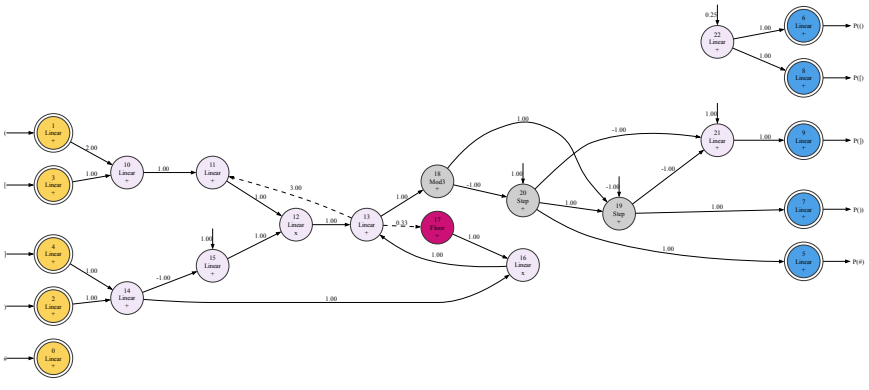


Figure 9: Golden network for the Dyck-2 task.

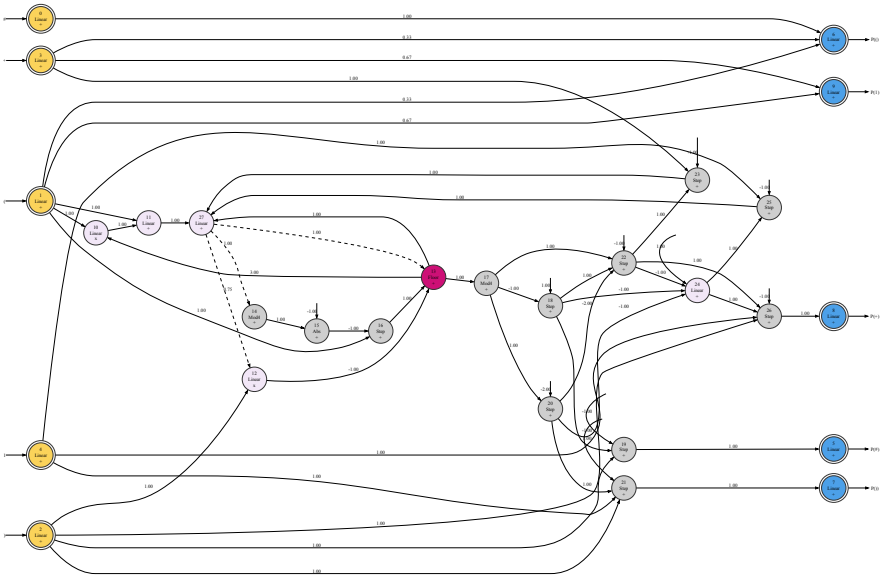


Figure 10: Golden network for the Arithmetic task.

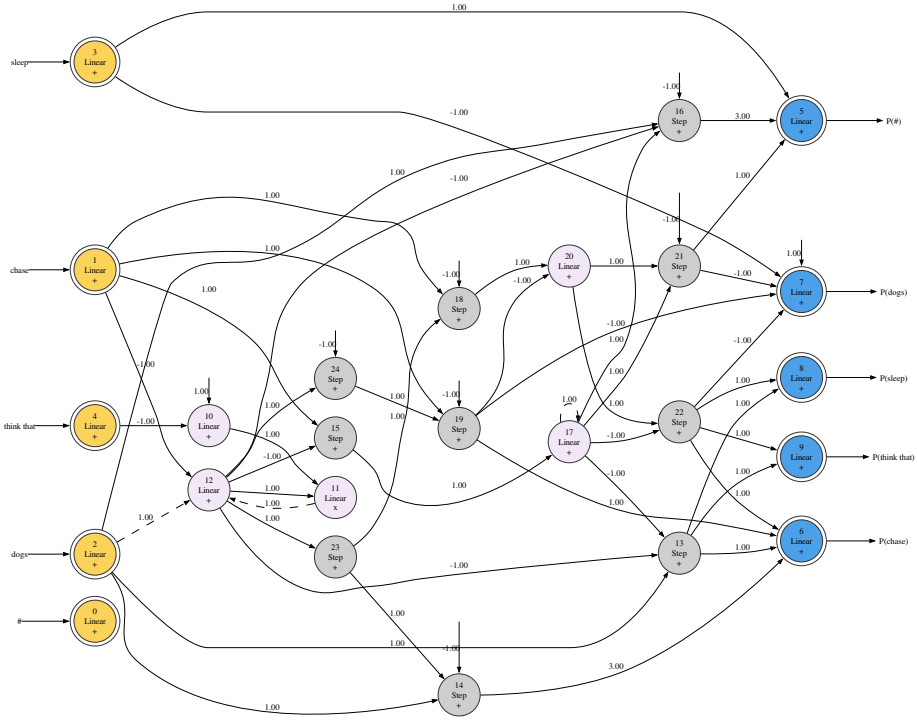


Figure 11: Golden network for the Toy-English task.

Appendix C. Genetic algorithm

Algorithm 1: Genetic Algorithm for Evolving Networks

Function `TournamentSelection(`*pop, objFunc*`):`
    $T \leftarrow t$ random networks from *pop*;
    winner $\leftarrow \arg\min_{objFunc}(T)$;
    loser $\leftarrow \arg\max_{objFunc}(T)$;
    return *winner*, *loser*;
end
population $\leftarrow \emptyset$; `// Population Initialization`
while $|population| < N - 1$ **do**
    generate a random network *net*;
    add *net* to *population*;
end
add *goldenNet* to *population*;
generation $\leftarrow 0$; `// Evolution Loop`
while *generation* $<$ *Gen* **do**
    for N *steps* **do**
        parent, *loser* $\leftarrow$ `TournamentSelection(`*population, objFunc*`)`;
        EVAL($offspring$); `// Evaluate` $offspring$ `using` *objFunc*
        remove *loser* from *population*;
        add $offspring$ to *population*;
    end
    generation $\leftarrow$ *generation* $+ 1$;
end

Appendix D. Hyperparameters

D.1. Genetic algorithm configuration

Table 9: Default genetic algorithm hyperparameters used in Experiments 1 and 2.

Parameter	Value
Number of Islands	500
Training Batch Size	500
Migration ratio	0.01
Migration interval (seconds)	1800
Migration interval (generations)	1000
Number of generations	25,000
Population size	500
Elite ratio	0.001
Allowed activations	Linear, ReLU, Tanh
Allowed unit types	Summation
Tournament size	2
Mutation probability	1.0
Crossover probability	0.0
Maximum network units	1024
Experiment seed	100
Corpus seed	100
Golden network copies in initialization	1

D.2. Task-specific modifications

Unless noted otherwise, all tasks inherit the default configuration from Table 9. The grammars used for corpus generations are detailed in Appendix A.

Table 10: Overrides to allowed activation functions and unit types for each task.

Task	Additional Allowed Activations	Additional Allowed Unit Types
$a^n b^n$	Sigmoid, Unsigned Step	—
$a^n b^n c^n$	Sigmoid, Unsigned Step	—
Dyck-1	Sigmoid	—
Dyck-2	Floor, Modulo-3, Unsigned Step	Multiplication Unit
Arithmetic	Floor, Modulo-4, Absolute Value, Unsigned Step	Multiplication Unit
Toy-English	Unsigned Step	Multiplication Unit

The maximum model complexity for simulations with limited $|G|$ is set to three times the complexity score $|G|$ of the corresponding golden network.

D.3. Backpropagation configuration

Table 11: Backpropagation hyperparameters used in Experiment 3.

Parameter	**Value**
Optimizer	Adam
Learning rate	10^{-4}
Number of epochs	1000
Regularization methods	None, L_1, L_2 (with $\lambda = 1$)
Experiment seed	100
Corpus seed	100

Appendix E. Compute resources

The genetic algorithm experiments were conducted on a SLURM cluster using CPU-only compute nodes. Each GA simulation was run on 252 CPU cores — 250 dedicated to GA islands and 2 reserved for logging and result collection. Runs were capped at 25,000 generations or 48 hours, whichever occurred first.

The gradient descent experiments, being more lightweight, were executed on a personal computer and completed in under an hour.

Appendix F. Additional figures

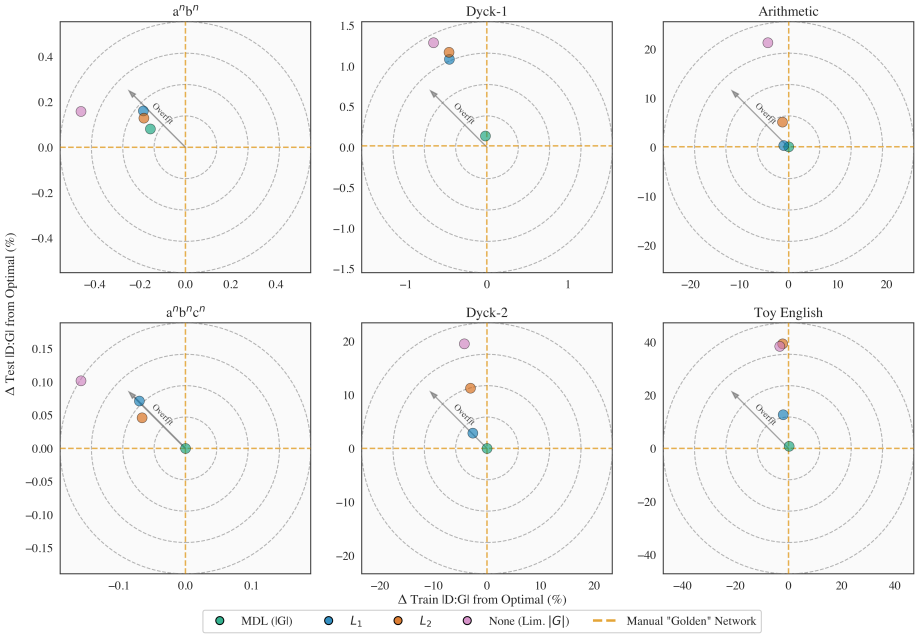


Figure 12: Relative deviation ($\Delta\%$) from optimal $|D:G|$ for each final network in Experiment 2, on train (x-axis) and test (y-axis), grouped by task. Proximity to center indicates a better approximation of the analytical optimum.

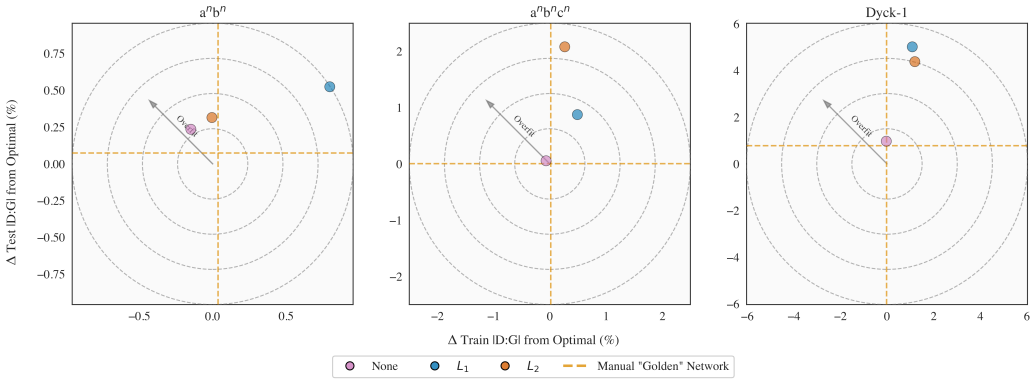


Figure 13: Relative deviation ($\Delta\%$) from optimal $|D:G|$ for each final network in Experiment 3, on train (x-axis) and test (y-axis), grouped by task. Proximity to center indicates a better approximation of the analytical optimum.

Appendix G. Full results

Table 12: **GA architecture search full results.** For each network selected by GA, we report all regularization term values, irrespective of the regularizer used in optimization. Networks are evaluated on training samples, an exhaustive test set, and a non-overlapping test set (strings not seen during training), reporting the relative gap $\Delta(\%) = \frac{|D:G| - \text{Optimal}}{\text{Optimal}} \times 100$. * marks originally infinite scores due to zero-probability errors. Simulations that were stopped due to the 48-hour time limit are marked with †.

Task	Regularizer	$\lvert G\rvert$	L_1	L_2	Train $\lvert D:G\rvert$	Test $\lvert D:G\rvert$	Δ^{Train}_{Optim} (%)	Δ^{Test}_{Optim} (%)
$a^n b^n$	(Golden)	(139)	(7.00)	(15.00)	(1531.77)	(2.94)	(0.0)	(0.0)
	MDL ($\lvert G\rvert$)	139	7.00	15.00	1529.39	2.94	**-0.2**	**0.1**
	L_1	460	5.46	10.54	1498.12	2.94*	-2.2	0.2
	L_2	600	5.52	2.92	1514.66	3.27*	-1.1	11.4
	None (Lim. $\lvert G\rvert$)	430	33.95	110.80	1488.17	3.32*	-2.8	13.0
$a^n b^n c^n$	(Golden)	(241)	(11.33)	(16.44)	(1483.91)	(2.94)	(0.0)	(0.0)
	MDL ($\lvert G\rvert$)	241	12.33	19.44	1483.91	2.94	**0.0**	**0.0**
	L_1	366	5.40	8.59	1482.88	2.94	-0.1	0.1
	L_2	602	5.49	3.25	1476.71	3.03*	-0.5	3.2
	None (Lim. $\lvert G\rvert$)	736	44.12	131.07	1460.67	2.96	-1.6	0.6
Dyck-1	(Golden)	(113)	(7.00)	(15.00)	(1544.48)	(1.77)	(-0.0)	(0.0)
	MDL ($\lvert G\rvert$)	113	6.00	10.00	1544.22	1.77	**-0.0**	**0.1**
	L_1	753	25.63	31.51	1488.17	1.90*	-3.6	7.5
	L_2	1140	31.02	24.72	1479.37	1.96*	-4.2	10.7
	None (Lim. $\lvert G\rvert$)	353	38.43	189.06	1505.61	2.03*	-2.5	15.1
Dyck-2	(Golden)	(579)	(27.33)	(35.11)	(2121.53)	(2.32)	(-0.0)	(0.0)
	MDL ($\lvert G\rvert$)	327	15.00	23.50	2130.28	2.37	**0.4**	**2.0**
	L_1	1629	35.95	36.82	2009.61	2.70*	-5.3	16.1
	L_2	2248	42.04	36.04	1996.82	2.82*	-5.9	21.3
	None (Lim. $\lvert G\rvert$)†	1757	94.99	204.82	1992.89	2.61*	-6.1	12.5
Arithmetic	(Golden)	(967)	(47.75)	(54.67)	(2581.29)	(3.96)	(0.0)	(0.1)
	MDL ($\lvert G\rvert$)	431	20.25	25.31	2581.07	3.94	**0.0**	**-0.4**
	L_1	1268	21.04	19.73	2540.22	3.99*	-1.6	1.0
	L_2	1200	17.93	15.94	2560.62	4.01*	-0.8	1.4
	None (Lim. $\lvert G\rvert$)†	3004	173.91	408.82	2477.42	4.23*	-4.0	7.0
Toy-English	(Golden)	(870)	(49.00)	(61.00)	(2232.74)	(4.49)	(0.0)	(0.0)
	MDL ($\lvert G\rvert$)	414	24.00	33.50	2231.96	4.57*	**-0.0**	**2.0**
	L_1	1215	15.06	14.18	2209.25	5.05*	-1.0	12.6
	L_2	1330	13.15	8.23	2202.41	5.43*	-1.3	21.0
	None (Lim. $\lvert G\rvert$)	2625	163.22	390.45	2142.40	6.25*	-4.0	39.4

Table 13: **GA weights-only full results.** For each network selected by GA, we report all regularization term values, irrespective of the regularizer used in optimization. Networks are evaluated on training samples, an exhaustive test set, and a non-overlapping test set (strings not seen during training), reporting the relative gap $\Delta(\%) = \frac{|D:G| - \text{Optimal}}{\text{Optimal}} \times 100$. * marks originally infinite scores due to zero-probability errors. Simulations that were stopped due to the 48-hour time limit are marked with †.

Task	Regularizer	$\lvert G\rvert$	L_1	L_2	Train $\lvert D:G\rvert$	Test $\lvert D:G\rvert$	Δ^{Train}_{Optim} (%)	Δ^{Test}_{Optim} (%)
$a^n b^n$	(Golden)	(139)	(7.00)	(15.00)	(1531.77)	(2.94)	(0.0)	(0.0)
	MDL ($\lvert G\rvert$)	139	7.00	15.00	1529.39	2.94	**-0.2**	**0.1**
	L_1	217	3.96	5.43	1528.92	2.94	-0.2	0.2
	L_2	268	4.57	4.38	1528.96	2.94	-0.2	0.1
	None (Lim. $\lvert G\rvert$)	385	95.18	2706.98	1524.71	2.94	-0.5	0.2
$a^n b^n c^n$	(Golden)	(241)	(11.33)	(16.44)	(1483.91)	(2.94)	(0.0)	(0.0)
	MDL ($\lvert G\rvert$)	239	11.33	16.44	1483.91	2.94	**0.0**	**0.0**
	L_1	362	5.07	8.36	1482.88	2.94	-0.1	0.1
	L_2	445	6.16	5.80	1482.94	2.94	-0.1	0.0
	None (Lim. $\lvert G\rvert$)	733	758.90	246247.87	1481.57	2.94	-0.2	0.1
Dyck-1	(Golden)	(113)	(7.00)	(15.00)	(1544.48)	(1.77)	(-0.0)	(0.0)
	MDL ($\lvert G\rvert$)	113	6.00	10.00	1544.22	1.77	**-0.0**	**0.1**
	L_1	259	6.62	8.33	1537.35	1.79	-0.5	1.1
	L_2	408	7.66	5.27	1537.28	1.79	-0.5	1.2
	None (Lim. $\lvert G\rvert$)	355	24.01	125.04	1534.32	1.79	-0.7	1.3
Dyck-2	(Golden)	(579)	(27.33)	(35.11)	(2121.53)	(2.32)	(-0.0)	(0.0)
	MDL ($\lvert G\rvert$)	490	22.33	32.11	2121.53	2.32	**-0.0**	**0.0**
	L_1	1202	31.29	33.04	2064.86	2.39*	-2.7	2.8
	L_2	1426	32.54	29.38	2055.48	2.58*	-3.1	11.2
	None (Lim. $\lvert G\rvert$)	1731	124.94	1003.61	2031.26	2.78*	-4.3	19.5
Arithmetic	(Golden)	(967)	(47.75)	(54.67)	(2581.29)	(3.96)	(0.0)	(0.1)
	MDL ($\lvert G\rvert$)	635	28.75	34.06	2581.29	3.96	**0.0**	**0.1**
	$L_1^\dagger$	1420	16.88	15.22	2555.63	3.97*	-1.0	0.4
	$L_2^\dagger$	1553	20.75	17.68	2548.16	4.15*	-1.3	5.1
	None (Lim. $\lvert G\rvert$)†	2890	185.24	421.52	2471.51	4.79*	-4.2	21.3
Toy-English	(Golden)	(870)	(49.00)	(61.00)	(2232.74)	(4.49)	(0.0)	(0.0)
	MDL ($\lvert G\rvert$)	552	31.00	51.00	2237.59	4.52	**0.2**	**0.8**
	L_1	1491	27.69	22.46	2186.61	5.05*	-2.1	12.7
	L_2	1704	29.66	19.56	2182.59	6.25*	-2.2	39.3
	None (Lim. $\lvert G\rvert$)	2610	139.89	304.89	2158.71	6.21*	-3.3	38.4

Table 14: **Gradient descent full results.** For each final network, we report all regularization term values, irrespective of the regularizer used in training. $|G|$ is approximated by the encoding length of the weights. Networks are evaluated on training samples, an exhaustive test set, and a non-overlapping test set (strings not seen during training), reporting the relative gap $\Delta(\%) = \frac{|D:G| - \text{Optimal}}{\text{Optimal}} \times 100$.

Task	Regularizer	$\approx \lvert G \rvert$	L_1	L_2	Train $\lvert D:G \rvert$	Test $\lvert D:G \rvert$	Δ^{Train}_{Optim} (%)	Δ^{Test}_{Optim} (%)
$a^n b^n$	(Golden)	(274)	(56.86)	(718.73)	(1532.33)	(2.94)	(0.0)	(0.1)
	None	5524	57.87	728.90	1529.48	2.94	-0.1	0.2
	L_1	5854	56.17	707.85	1543.98	2.95	0.8	0.5
	L_2	5610	56.30	707.86	1531.69	2.95	-0.0	0.3
$a^n b^n c^n$	(Golden)	(599)	(91.33)	(1655.78)	(1484.15)	(2.94)	(0.0)	(0.0)
	None	20043	92.54	1661.71	1482.91	2.94	-0.1	0.1
	L_1	20541	90.06	1638.44	1491.15	2.96	0.5	0.9
	L_2	19909	90.37	1638.25	1487.84	3.00	0.3	2.1
Dyck-1	(Golden)	(148)	(20.75)	(104.56)	(1544.28)	(1.78)	(-0.0)	(0.8)
	None	4876	21.14	107.30	1544.03	1.78	-0.0	1.0
	L_1	5138	20.06	100.91	1561.23	1.86	1.1	5.0
	L_2	4874	20.20	100.91	1562.96	1.84	1.2	4.4