CAN TRANSFORMERS REALLY DO IT ALL? ON THE COMPATIBILITY OF INDUCTIVE BIASES ACROSS TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformers are remarkably versatile and their design is largely consistent across a variety of applications. But are they optimal for any given task or dataset? The answer may be key for pushing AI beyond the mere scaling of current designs.

Method. We present a method to optimize a transformer architecture for a given dataset, which we use as a tool to study optimal task-specific inductive biases. The method replaces the most important non-linearities (GeLUs, softmax) with components optimized on held out data. We then use each resulting new architecture with other datasets as a way to evaluate the compatibility between pairs of tasks.

Findings. On a range of popular algorithmic tasks, our method identifies new architectures with dramatic improvements in learning speed, generalization, and stability across seeds. These designs prove very task-specific, which means that the tasks require inductive biases very different from those of standard transformers. On a range of code and language modeling datasets, we also find architectures with consistent, yet smaller improvements. These designs now transfer much better across datasets, domains (English vs. computer code), and tokenizations.

Implications. These results show that standard transformers are rarely a local optimum in the space of architectures. We show that alternative designs can perform better, but they often sacrifice universality. This calls for future work on architectures that could serve multiple objectives such as fluency and robust reasoning.

1 INTRODUCTION

Inductive biases of transformers. The recent history of machine learning has seen a uniformization of models across tasks and modalities. Most state-of-the-art models for vision, language, and speech for example are based on transformers, barring only relatively minor differences (Vaswani et al., 2017). The success of this general solution over task-specific designs has prompted the hypothesis that transformers implement very generic inductive bias[1] such as a *simplicity bias* akin to Occam's razor (Goldblum et al., 2023). The simplicity bias of neural networks depends on architectural choices such as their activation functions (Teney et al., 2024; 2025). Yet, considering the space of all possible architectures, the following question remains (Q1).

Are transformers a unique and optimal solution endowed with generic inductive biases?

Uneven performance across domains. Transformers perform remarkably well for many applications, e.g. when trained as large language models (LLMs). Paradoxically, they also fail to learn elementary tasks such as arithmetic operations (Nikankin et al., 2024). These failures demonstrate limitations of transformers and have motivated new designs such as positional encodings (Cai et al., 2025; Jelassi et al., 2024) and alternative attention mechanisms (Katharopoulos et al., 2020; Saratchandran et al., 2024b; Schlag et al., 2021). But these new designs are rarely adopted beyond toy tasks. This suggests that the inductive biases of standard transformers are not as well suited to domains as different as e.g. natural language and arithmetic. This raises another question (Q2).

Should we even seek to address such different domains with the same learning method?

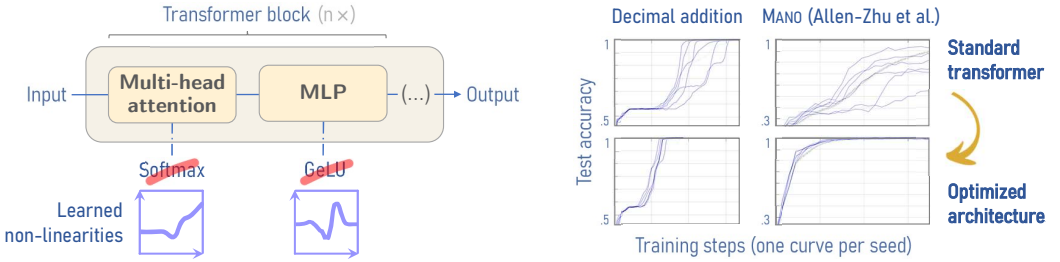


Figure 1: **Our approach to discover better task-specific inductive biases.** (Left) We replace the main non-linearities in a transformer (softmax, GeLUs) with parametrized components optimized for specific tasks. (Right) The optimized architectures allow us to train models with dramatically better convergence, generalization, and stability across seeds, on algorithmic tasks and code/language modeling datasets. We also mix-and-match the new architectures across tasks (not pictured) to evaluate the compatibility of inductive biases across tasks.

The above questions matter for developing future learning systems. Although recent progress in AI stems from scaling up models and data (Mayilvahanan et al., 2025), this growth is not infinitely sustainable, and better learning efficiency seems possible given the capabilities of biological systems. This fundamentally requires improving the inductive biases of our learning methods. Understanding the inductive biases of transformers (Q1) is a step in this direction. And understanding the compatibility of different tasks (Q2) will help select better proxies and incentives for future progress.

Our approach. We address the above questions with a method that optimizes the inductive biases for a specific task by tweaking the transformer architecture. We replace non-linearities (GeLUs, softmax) with parametrized ones, optimized on held-out data. This yields new architectures that match or surpass standard transformers. The improvement in learning speed and/or generalization indicates how far the standard transformer is from a local optimum in the space of architecture for a specific task (Q1). We also mix-and-match these new architectures across tasks to assess how the inductive biases tuned for one task perform for another, thus assessing their compatibility (Q2).

Findings. We study two domains: algorithmic skills and language modeling. For algorithmic skills, we use toy tasks commonly used to evaluate architectures, see e.g. Allen-Zhu (2025). For nearly all considered tasks, our approach finds architectures that dramatically improve learning speed, generalization, and stability across random seeds (Section 3.1). Task-specific variants of transformers can thus be **vastly superior to standard designs, using only minor modifications** like replacing the GeLUs. Our cross-task evaluation also reveals that the new architectures are quite task-specific. This can explain why many hand-crafted components from the literature (e.g. attention mechanisms, positional encodings) are rarely adopted beyond toy tasks. It also challenges the view that a single architecture can be optimal for a vast set of tasks (Goldblum et al., 2023).

For language modeling, we evaluate multiple datasets of natural language and computer code. In most cases, we also find optimized architectures that slightly improve over a baseline transformer. We stress that these improvements are practically not directly useful, because standard components are more computationally efficient. But they matter indirectly, because they are evidence that **standard transformers are neither a unique nor a local optimum** in the space of architectures. In contrast to algorithmic tasks, the cross-task evaluation shows that the improvements can transfer across natural language datasets and tokenization levels (character vs. subword). Overall, the results suggest that standard transformers are intrinsically better suited to modeling natural language than code, and clearly ill-equipped to learn algorithmic skills.

Our contributions are summarized as follows.

- **A method to optimize a transformer architecture** for any given dataset (Section 2). We replace GeLUs and softmaxes with parametrized components optimized on held-out data. The optimized architecture can then be used with standard training to evaluate its suitability to any other dataset.

[1]The *inductive biases* of a learning algorithm correspond to a prior over the space of functions (Mitchell, 1980; Mingard et al., 2021) that favors particular (types of) functions among the many that fit the data. We focus on biases encoded in architectures, rather than choices of optimizer, objective function, initialization, etc.

- **An application to algorithmic tasks** (Section 3). We find that optimized architectures dramatically improve learning speed, generalization, and stability across seeds. They also prove very task-specific, showing the utility of inductive biases very different from standard transformers'.
- **An application to language modeling** (Section 4). We obtain small, albeit consistent improvements, showing that standard transformers are neither unique nor optimal designs, even for common code and natural language modeling tasks.

We discuss implications for the development of future learning systems in Section 6.

2 Proposed Method to Optimize and Evaluate Architectures

Goal. We consider, as a baseline architecture, a standard decoder-only transformer (GPT-2-style, see details in Appendix B). Our goal is to evaluate whether this choice is optimal for specific tasks and datasets. We also seek to identify better variants, as a proxy for identifying the inductive biases best suited to each task. Evaluating the new architectures across tasks can then measure the compatibility of pairs of tasks. All the tasks we consider are formulated as sequence completion of natural language, computer code, or abstract tokens.

Replacing non-linearities with parametrized functions. We replace the main non-linearities in a transformer with parametrized components that can be optimized (see Figure 1). Indeed, the main difference between a transformer and a simple linear model hinge on a few non-linear operations in the attention and MLP layers, which we will alter to obtain different inductive biases.

- An MLP layer is defined as: $\boldsymbol{x} \leftarrow \boldsymbol{W}'\,\phi\big(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}\big)+\boldsymbol{b}'$ where $\boldsymbol{x}$ is a vector of activations, $\boldsymbol{W}$, $\boldsymbol{W}'$, $\boldsymbol{b}$, $\boldsymbol{b}'$ learned weights and biases, and $\phi\colon\mathbb{R}\to\mathbb{R}$ an element-wise non-linearity. In the baseline architecture, ϕ is a GeLU. In our model, $\phi_{\boldsymbol{\theta}_{\mathrm{MLP}}}$ is a 1D linear spline parametrized by learnable keypoints $\boldsymbol{\theta}_{\mathrm{MLP}}$, capable of approximating a variety of functions (details in Appendix B).
- An attention layer in the baseline transformer is defined as: $\boldsymbol{x} \leftarrow \mathrm{softmax}\big(\mathbf{Q}\,\mathbf{K}^{\top}\big)\,\mathbf{V}$, where $\boldsymbol{x}$ is the output vector of activations and $\mathbf{Q},\mathbf{K},\mathbf{V}$ are linear projections of the input. This is a special case of the kernel version of attention: $\boldsymbol{x} \leftarrow \sum_{j=1}^{n} K(\mathbf{Q}_i,\mathbf{K}_j)\,\mathbf{V}_j \,/\, \sum_{j=1}^{n} K(\mathbf{Q}_i,\mathbf{K}_j)$ where the similarity between $\mathbf{Q}$ and $\mathbf{K}$ is measured with a kernel function $K(\mathbf{Q},\mathbf{K})$. In the baseline transformer, $K_{\mathrm{smax}}(\mathbf{Q},\mathbf{K}) = \exp\big(\mathbf{Q}^{\top}\mathbf{K}/\sqrt{d}\big)$. In our model, we introduce a learnable non-linearity $\phi'\colon\mathbb{R}\to\mathbb{R}$ giving $K(\mathbf{Q},\mathbf{K}) = \phi'(\mathbf{Q})^{\top}\phi'(\mathbf{K})$. We implement ϕ' as a linear spline $\phi'_{\boldsymbol{\theta}_{\mathrm{A}}}$ with keypoints $\boldsymbol{\theta}_{\mathrm{A}}$ that can be optimized.

Two-stage setting. Our experiments proceed in two stages. In stage I, we optimize the architecture for a chosen dataset $\mathbb{D}$ by training both the model's weights and its parametrized non-linearities $(\boldsymbol{\theta}_{\mathrm{A}},\boldsymbol{\theta}_{\mathrm{MLP}})$ on $\mathbb{D}$. In stage II, the non-linearities are frozen, and we retrain the model in a standard manner from scratch on any dataset $\mathbb{D}'$. The models obtained from stage II are thus fairly comparable with the baseline architecture.[2] When $\mathbb{D}' \neq \mathbb{D}$, i.e. a "mix-and-match" setting, stage II serves to evaluate whether the inductive biases optimized for $\mathbb{D}$ suit the learning of $\mathbb{D}'$.

Optimizing architectures. Our method may seem similar to prior work about learning activation functions (e.g. (Alexandridis et al., 2025)) but their goals are very different. These works seek to improve performance by continuously updating the activation during training. Whereas we seek to identify inductive biases that can remain hard-encoded in the architecture and further reused to train new models with other seeds and datasets (stage II). We make this possible with a **two-loss training**. During stage I, we hold out a fraction of the training data (e.g. 20%) that we use solely for optimizing the non-linearities, while we optimize the weights in a standard manner on the training set. This prevents a co-adaptation, that could make the non-linearities overfit particular weights or seed. This is particularly important for our experiments on algorithmic toy tasks, and even more so for improving length generalization[3] (Section 3.1). In this latter case, we hold out an out-of-distribution (OOD) split of data (see Section 3.1), such that the weights are optimized for one range of sequence lengths, and the architecture for a different wider range. This forces the architecture to capture an inductive bias for length generalization. In stage II, the non-linearities are frozen, and the model weights are trained in a standard manner on the whole training split of the target dataset.

[2] In stage II, $(\boldsymbol{\theta}_{\mathrm{A}},\boldsymbol{\theta}_{\mathrm{MLP}})$ are frozen and better viewed as pre-tuned hyperparameters than extra model capacity.

[3] The benefit of the two-loss training is smaller for language modeling because the models are heavily over-parametrized and never at risk of overfitting the training data.

A second innovation to prevent the co-adaptation of weights and non-linearities in stage I is **multi-model training**. We train M models in parallel (e.g. $M = 4$) that use different seeds but share the non-linearities being optimized. The resulting optimized architecture is naturally more likely to generalize in stage II to other weights and datasets (see Appendix D). This also proves particularly helpful for algorithmic tasks because the variance across seeds of the baseline architecture is often high. We provide a complete description of our method as Algorithm 1 in the appendix.

Rational for splines. We parametrize our non-linearities as linear splines because they offer the most unbiased tractable parametrization for an $\mathbb{R} \to \mathbb{R}$ function. For example, a spline can represent the identity function as easily as a step function or a sine wave. Prior work on trainable activation functions enforces priors of smoothness or monotonicity e.g. with small MLPs (Apicella et al., 2021; Greydanus & Kobak, 2020)). These would struggle to capture sharp transitions like in Figure 6. We also favor *linear* splines over higher-order (e.g. cubic) ones because they behave nearly identically while being much cheaper, as evaluated by Teney et al. (2025, Appendix D).

3 Experiments on Algorithmic Reasoning Tasks

In this section, we apply the proposed method to a set of tasks commonly used to evaluate the algorithmic skills of transformers, detailed in Table 1. These tasks are elementary but remarkably challenging and often used to highlight limitations of transformers. All the tasks are formulated as sequence completion. Each sequence comprises an "input" part, followed by a separator then an "output" part. The models are trained with a next-token prediction objective on the latter part of training sequences. Unless otherwise noted we use i.i.d. sets of training, validation, and test data.

Experimental setup. For each task $\mathbb{D}$, we first train the baseline architecture and tune its hyper-parameters (width, depth, learning rate, batch size, etc.) for high accuracy and fast convergence on the validation set. We then run the proposed method (stage I, $M = 8$) to optimize the architecture for $\mathbb{D}$. We then re-train a model from scratch with the optimized architecture (stage II), keeping the same hyperparameters (we saw no further improvements by re-tuning them). In Section 3.2, we also re-train models on other tasks $\mathbb{D}'$ as a way to evaluate the generality of the optimized architecture and the compatibility of $\mathbb{D}$ and $\mathbb{D}'$. All results are averages over 6 random seeds.

Table 1: Algorithmic tasks used in our experiments. They are similarly-sized in term of complexity and required model capacity, except for MANO (Allen-Zhu, 2025) which is relatively more complex.

Task	Examples

MEMORIZE. Simple memorization of a mapping between a two-integer key and an integer value, with all integers in [1,32]. Each sequence consists of the key, a separator, and the value. This task has no test set: performance is simply the training accuracy (Zhong & Andreas, 2024).	`23 12 | 10`
	`11 32 | 27`
	`31 19 | 18`

PARENTHESES. Recognition of Dyck language. Each sequence contains parentheses followed by a separator and a marker indicating whether they are balanced or not. Sequences lengths are in [1,20] in the training set, and [21,40] in the validation and test sets (Zhong & Andreas, 2024).	`( ) ( | <unbalanced>`
	`( ( ) ( ) ) | <balanced>`
	`) ( ) ( ) | <unbalanced>`

ADDMOD. Modular addition mod N, with 95% of the N^2 examples used for training (Zhong & Andreas, 2024). We use N=97.	`12 3 | 15` `96 2 | 1`

HAYSTACK. Needle-in-a-haystack recall. The model gets a sequence $[m_1, c_1 \dots m_k, c_k, m_u]$ of markers m_k and values c_k. It must search for the first occurrence of m_u and return its successor c_u (Zhong & Andreas, 2024). We use $k \in [1,10]$ and $m_k, c_k \in [1,64]$.	`2 p 9 k 3 b 9 | k`
	`8 a 2 b 8 | a`
	`2 p 9 k 3 b 5 x 5 | x`

ADD. Decimal addition of 4-digit numbers with digit-wise tokens. (Zhong & Andreas, 2024).	`1 0 0 9 + 1 0 9 2 | 2 1 0 1`
ADDREVERSED. ADD with reversed numbers, known to be easier to learn (Lee et al., 2023) .	`9 0 0 1 + 2 9 0 1 | 1 0 1 2`
COPY. Repeating the input. Elementary but unsolved for length generalization (Cai et al., 2025). Tokens in [1,8]. Seq. lengths in [2,10] for training, [2,15] for validation, [16,20] for testing.	`2 8 | 2 8` `9 4 8 7 8 3 | 9 4 8 7 8 3`

MANO. Synthetic task proposed by Allen-Zhu (2025) to evaluate large pretrained models. Each sequence specifies nested arithmetic operations mod N with number-level tokens. Our scaled-down version uses N=7 and a number of operations per sequences in [1,3].	`(1*3)+4 | 0`
	`(2-(6-1))*3 | 5`
	`(3*(5-6))-1 | 3`

3.1 Improvements on Individual Tasks

Faster convergence. The most striking improvement with optimized architectures is the learning speed (Figure 2). For the ADD and MANO tasks for example, convergence occurs 2–3× faster. The learning rate of the baseline was tuned to its maximum stable value for every task.

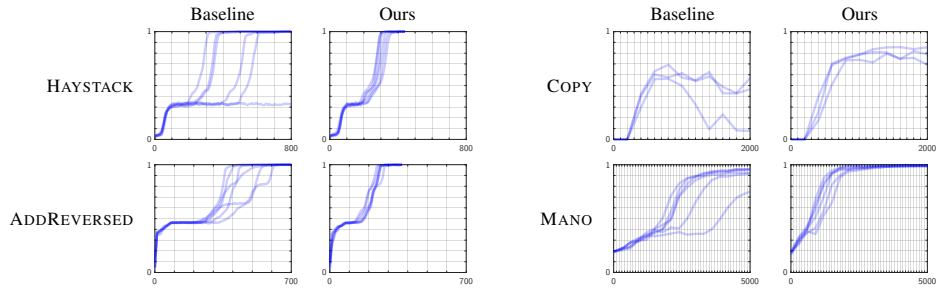


Figure 2: Training curves (test accuracy vs. training step, one curve per random seed) of models trained on algorithmic tasks with a baseline transformer or our optimized architectures. The latter converge much faster and show less variance across seeds. See Appendix C for other tasks.

Reduced variance. On some tasks, baseline transformers show huge variance in accuracy and training speed across random seeds. This suggest tasks that are underspecified (Teney et al., 2021; 2022) and misaligned with the model's inductive biases (Zhou et al., 2024). In these cases, the optimized architectures eliminate the problem and make the training much more reliable (Figure 2).

Better generalization. For some tasks, baseline transformers do not reach perfect test accuracy though they perfectly fit the training data. This shows again a misalignment between the target function and the inductive biases. Optimized architectures solve this problem (see e.g. MANO, Figure 2).

Improved length generalization. An outstanding challenge for transformers is the generalization to sequences longer than seen during training. Even the COPY task is unsolved and a baseline transformer completely fails on unseen lengths (Figure 3). Among the plethora of existing partial solutions, the Alibi positional encodings (Press et al., 2021) bring non-trivial accuracy on slightly longer sequences. We use our method to optimize the Alibi architecture. We use the two-loss mechanism of Algorithm 1 to optimize the transformer weights on lengths 2–10 and the non-linearities on 2–15. This forces the optimized architecture to capture an inductive bias for length generalization. As a result, a model trained with the optimized architecture reaches higher accuracies on longer sequences. While this is not a complete solution to length generalization, it shows that inappropriate inductive biases in the base architecture are one of the obstacles to length generalization.

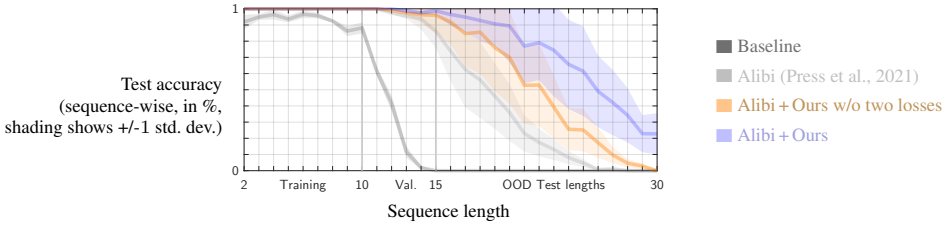


Figure 3: Length generalization on the COPY task. The baseline completely fails on unseen lengths ($\gg 10$). Alibi positional encodings (Press et al., 2021) help. Optimizing the Alibi architecture with our method further improves the accuracy and extends the benefits to longer sequences.

Performance with smaller models. We train models of different widths for each task. Results in Figure 4 show that the accuracy drops more sharply on some tasks with the baseline architecture than optimized ones. Intuitively, when the architecture is already aligned with the task, less capacity is needed in its weights. Equivalently, a fixed number of parameters offers more capacity.

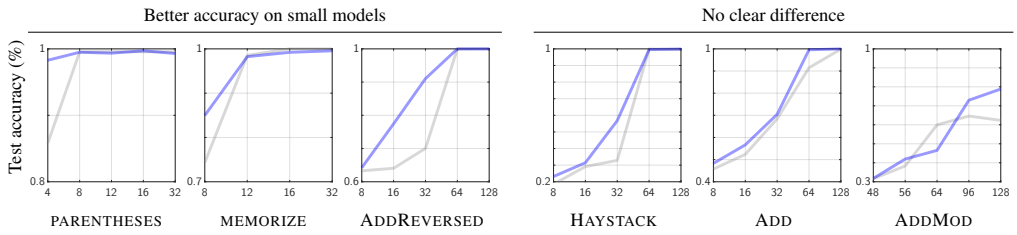


Figure 4: Test accuracy of models of different widths (X axis). On some tasks, optimized architectures (■) maintain higher accuracy than the baseline (■) when reducing the width of the model.

3.2 Compatibility of Optimized Architectures Across Algorithmic Tasks

We now train models on each task $\mathbb{D}$ using architectures optimized for any other task $\mathbb{D}'$ to evaluate the pairwise compatibility of their inductive biases. The results in Figure 5 show that the optimized architectures are very task-specific. Few of the benefits transfer across tasks, mostly across closely related tasks like ADD and ADDREVERSED. Many perform worse than a standard transformer. This shows that the specialization to our algorithmic tasks comes at the cost of universality. These tasks are very narrow however and it remains an open question whether the negative impact is inevitable. A future step to study this question could be a multi-task optimization in Algorithm 1.

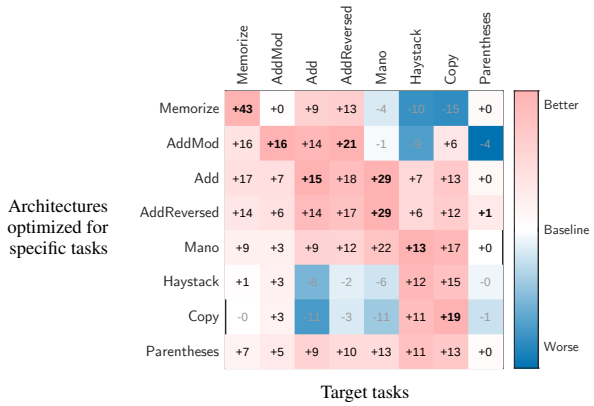


Figure 5: Compatibility of architectures across algorithmic tasks. We plot the absolute difference in test accuracy (%) with the baseline after a fixed number of steps (details in Appendix B). The best option per task (column) is usually on the diagonal, meaning that the optimized architectures are quite task-specific, while still yielding some positive transfer.

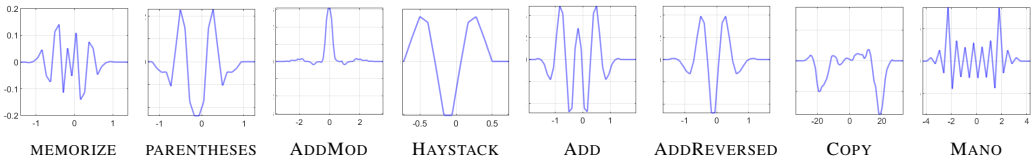


Figure 6: MLP non-linearities optimized for each algorithmic task.

Take-away. On algorithmic tasks, optimized architectures can dramatically outperform standard transformers, but the benefits are quite task-specific. This means that these tasks require inductive biases very different from those of standard transformers.

4 Experiments on Language Modeling Tasks

We now apply the same experimental setup as Section 3 to language modeling. We use datasets for computer code (English, Java) and natural language of various complexity levels (Table 2). Our goal is to understand whether different type of data benefit from different inductive biases. Current practices for building LLMs show that data diversity is beneficial (Longpre et al., 2024) and that code is complementary to natural language (Aryabumi et al., 2024; Petty et al., 2024). But because all kinds of data are mixed during training, it is unknown whether they could each exploit or elicit different mechanisms in a model. We also consider versions of the datasets tokenized at the character or subword level (BPE; details in Appendix B). These choices are motivated by Mayilvahanan et al. (2025) who showed that LLM performance is mostly determined by data diversity and tokenization.

4.1 Improvements on Individual Datasets

TinyStories. We compare in Figure 7 models trained with baseline or optimized architectures. The latter do slightly better. The improvement is small but consistent at different model sizes. Training curves (Figure 15) show that the improvement is larger early during training then diminishes. We

Table 2: Datasets used in our experiments for language modeling (see Appendix B for details).

Dataset	Excerpt
TINYSTORIES. Children stories generated with GPT-3.5. It was designed to capture core aspects of natural language (syntax, coherence, compositionality) with a limited vocabulary. This allows smaller-scale experiments than web-scale open-domain corpora (Eldan & Li, 2023).	`Once upon a time, there was a clever little dog named Max. Max loved to run (...)`
SHAKESPEARE. Plays and sonnets by William Shakespeare, often used in early research on language modeling. It includes recognizable patterns of grammar, rhythm, and vocabulary, as well as a unique structure because of the speaker labels and dialogue formatting (Karpathy, 2015).	`BENVOLIO: Good-morrow, cousin. ROMEO: Is the day so young? BENVOLIO: But (...)`
ENWIK8. First 100 M bytes of the English Wikipedia (Mahoney, 2006). We use the clean version from Yong (2025) with only text visible to human readers, without links and meta data. This data provides dense, real-world text with a mix of vocabulary, syntax, and formatting.	`anarchism originated as a term of abuse first used against early working (...)`
CODESEARCHNET-JAVA & -PYTHON. Dataset of computer code originally created to support research on code search and code–text understanding (Husel et al., 2019). We discard comments and descriptions in natural language following Lu et al. (2021) to focus exclusively on code.	`batch, limit = 100, self._next_limit() it = iter(it) (...)`

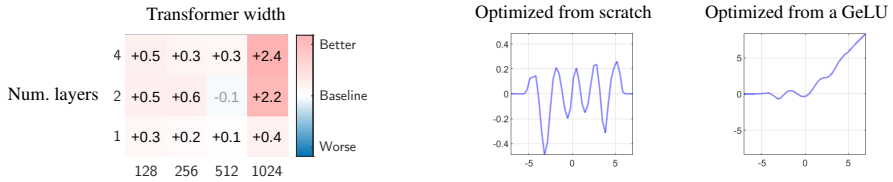


Figure 7: **(Left)** Absolute improvements in token prediction accuracy (%) of the best optimized architectures on TINYSTORIES compared to our baseline transformer. The accuracy is consistently slightly better at different model sizes. **(Right)** Visualization of MLP non-linearities optimized from scratch (results on the left) or from a GeLU initialization (*GeLU + Ours* in Figure 8). Although they resemble generic wavelets, we show in Appendix D that fine details in these functions matter.

find it best to optimize non-linearities only in MLPs (i.e. replacing GeLUs; see Figure 8). Replacing softmaxes with learned components barely matches or underperforms the baseline, indicating a difficult optimization. We experimented with alternative parametrizations that exactly mimic a softmax at initialization. This solution would barely move away from this initialization (not reported in tables), suggesting that a softmax is close to a local optimum.

We visualize in Figure 7 (right) the optimized MLP non-linearities, which are remarkably similar to sine wavelets. We evaluate a non-exhaustive selection of activation functions and attention variants from the literature in Table 3. None of them works better than ours. The gated linear units (GLUs) are a popular design that adds multiplicative interactions to the MLPs. We show that we can also improve them by introducing our learned spline in GLUs in lieu of their internal Swish activations. This provides similar improvements as over standard MLPs, cf. GLU/*Swish* and GLU/*Ours* in Table 3. We also evaluate in Appendix D the importance of fine details in the learned non-linearities. We try to make them more periodic or symmetric, but they then always perform worse.

Table 3: Performance of models trained on TINYSTORIES with existing alternative attention and MLP designs (2 layers, width 256). None works better than ours. See Appendix D for references.

Attention	smax	smax	smax	smax	smax	smax	smax	smax	smax	smax	P1	P3	Adaptive	NormSmax
MLP	Linear	GeLU	Ours	GLU/Swish	GLU/Ours	ReLU	ReLU^2	TanH	Sinc	Gaussian	GeLU	GeLU	GeLU	GeLU
Tr. perplexity	1.78	1.58	**1.57**	1.59	1.58	1.60	1.60	1.71	2.50	1.64	1.62	1.60	1.58	1.58
Val. acc. (%)	59.9	63.7	**64.4**	63.7	64.0	63.5	63.6	61.2	47.7	62.8	63.0	63.7	63.7	63.7

SHAKESPEARE & ENWIK8. These datasets differ from TinyStories in their richer vocabulary and sentence structure. SHAKESPEARE also follows a particular formatting presenting dialogues with speaker labels (see Table 2). The results in Figures 8 & 13 show that *some* optimized architectures slightly improve over the baseline. Optimizing non-linearities in the MLPs is again more useful than in the attention. However, differences with the baseline are small, which suggests that standard transformers are inherently well suited to language modeling.

The improvement is slightly clearer on **character-level datasets** than on tokenized ones (marked -CHAR in Figure 8). We hypothesize that the target function to be learned by the transformer layers for character-level language modeling is more complex, because of the lesser capacity available

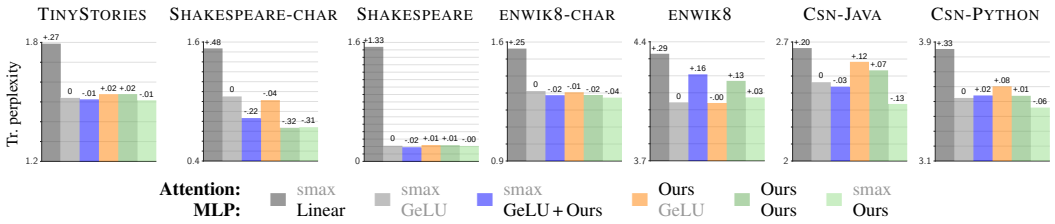


Figure 8: Perplexity on code and natural language (lower is better; numbers on bars correspond to the difference with the baseline architecture). Some optimized architectures perform slightly better than the baseline, often simply with optimized MLP non-linearities ( ). Datasets of code (CSN-JAVA, CSN-PYTHON) also benefit relatively more than datasets of natural language.

in the model's token embeddings (embeddings can otherwise make up a significant fraction of the model parameters for tokenized datasets). This could be the reason why learned non-linearities are particularly helpful, since they can help learn and represent complex functions (Teney et al., 2025).

We also evaluate a version of our optimized **MLP non-linearities initialized as a GeLU** rather than a constant zero (*GeLU + Ours* in Figure 8). With this, the model starts stage I with a non-linearity known to perform well. And because the optimization is non-convex, the optimized solution remains in the local search space near GeLUs (see Figure 7, right). The models trained with these non-linearities perform in-between GeLUs and those optimized from scratch. This means that GeLUs are usually not an optimal solution, not even a local one. But note also that our best solutions are not guaranteed to be *globally* optimal and better ones may exist.

CODESEARCHNET (CSN-JAVA, CSN-PYTHON). The results in Figure 8 show that our optimized non-linearities in MLPs improve again over the baseline. The gains are larger for code than natural language, relative to the gap between the baselines with linear and GeLU MLPs. These larger gains may reflect the larger importance of systematic structure and compositionality in code than natural language. The task of modeling code may thus resemble some of the algorithmic tasks of Section 3, which benefited greatly from optimized architectures. Therefore, the architectures best suited to natural language may not be simultaneously optimal for code.

4.2 COMPATIBILITY OF OPTIMIZED ARCHITECTURES ACROSS LANGUAGE DATASETS

Our final results examine the compatibility of the optimized architectures across language modeling datasets. We consider our seven datasets plus MANO, the most complex of our algorithmic tasks. We train models for every task $\mathbb{D}$ using architectures optimized for any other task $\mathbb{D}'$. The results in Figure 9 show that the variations across architectures are very small. This contrasts with the results on algorithmic tasks (Figure 5). These optimized architectures thus encode much less task-specific specialization. This suggests that the skills required across code and language modeling datasets are much more uniform. We discuss the implications of these results in Section 6.

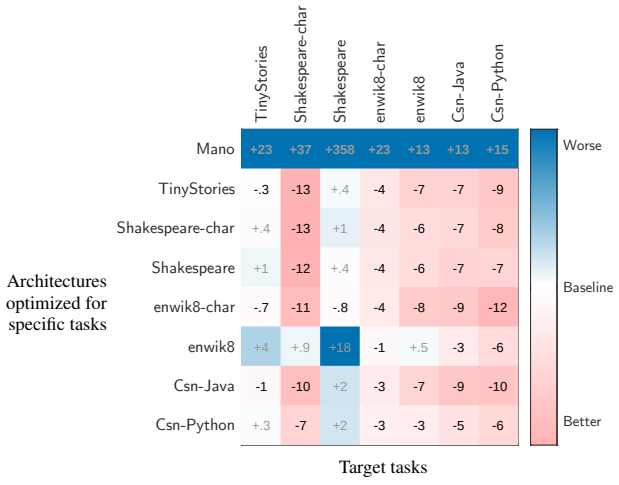


Figure 9: Compatibility of architectures across code and language datasets (relative difference in perplexity with the baseline in %, lower is better). The differences are much less dramatic than with algorithmic tasks (Figure 5), indicating smaller benefit in dataset-specific specialization.

Take-away. For code and natural language modeling, the optimized architectures improve much less than for algorithmic tasks. This means that standard transformers are intrinsically closer to a local optimum in the space of architecture for these tasks than for learning algorithmic skills.

5 Related Work

Understanding inductive biases in NNs. Much of the prior on understanding neural networks (NNs) has focused on their simplicity bias, i.e. their preference for representing functions of low Kolmogorov (Zhou et al., 2023) or spectral complexity (Bhattamishra et al., 2022). The simplicity bias depends primarily on the choice of activation function (Mingard et al., 2019; Teney et al., 2024), and its suitability was questioned (Domingos, 1999) by evaluating alternative activation functions in MLPs (Teney et al., 2024). We extend this inquiry to transformers and larger settings. In particular, we introduce a method to optimize non-linearities in both attention and MLP layers, and apply it to tasks relevant to the state of the art (code, natural language, algorithmic reasoning).

Improving transformers. Current LLMs all use very similar architectures, and Mayilvahanan et al. (2025) show that small design differences play little role in their performance. Prior work has however studied at length the impact of various components of transformers including their nonlinearities (Jha & Reagen, 2025; Newhouse et al., 2025). Proposed improvements include alternative attention mechanisms (Katharopoulos et al., 2020; Saratchandran et al., 2024b; Schlag et al., 2021; Tamayo-Rousseau et al., 2025; Veličković et al., 2024) and activation functions for MLPs (Hu et al., 2025; Teney et al., 2025) and transformers (Mirzadeh et al., 2023; So et al., 2021a). This motivates our work by suggesting that standard transformers are not a uniquely optimal choice of architecture.

Architecture search. Our method to optimize architectures is reminiscent of neural architecture search (NAS) (Goyal et al., 2019; Hong, 2025; Liu et al., 2018; Manessi & Rozza, 2018; Ramachandran et al., 2018; Zoph & Le, 2017). The goals and approach are different though. NAS uses RL or evolutionary algorithms to search through pre-defined design choices. We directly use gradient descent to optimize a relatively unrestricted parametrization of the non-linearities of transformers. Our goal is not to find better models (our designs are often computationally expensive). Instead, our method is a tool to understand the compatibility of the inductive biases required for various tasks.

6 Discussion

We have presented a method to optimize a transformer architecture for specific datasets and used it to study the compatibility of inductive biases across tasks. We found that standard transformers are often suboptimal, but minor tweaks (replacing GeLUs and softmax operations) can substantially improve training speed, generalization, capacity, and stability across random seeds.

Our results show that different tasks benefit from different inductive biases, aligning with the no-free-lunch theorem (Wolpert, 1996). Yet, transformers seem uniquely suitable to a vast range of applications (Goldblum et al., 2023): our results can be seen as probing the limits of this hypothesis.

Architecture vs. scale. Prior work showed that the choice of architecture can become less important with scale (Bachmann et al., 2023; Tay et al., 2022). But this also means that the current need to build ever-larger models may be due to suboptimal inductive biases. In this work, we tweaked transformers to explore the space of inductive biases, but similar effects may be achievable with other means e.g. completely different architectures, initializations (Shinnick et al., 2025), or optimizers.

Do we need domain-specific models? Our results show a higher compatibility across language/code than algorithmic tasks, which are often used to highlight limitations e.g. for length generalization. If these toy tasks really represent desirable capabilities in LLMs, perhaps new architectures are required to combine language and algorithmic capabilities. A future step could be to apply our method to optimize architectures for multiple tasks simultaneously.

Other domains. An extension of this work could examine possible improvements to transformers for other domains such as vision and speech, and whether the improvements transfer across domains.

Limitations. First, our **search space of architectures** is limited. Complex forms of attention (Hashemi et al., 2025) or interactions like gated linear units (GLU, Shazeer (2020)) cannot be represented in our formulation. Further gains are possible with a larger search space, the optimization also becomes more challenging. Second, the **scale of our experiments** is tiny relative to state-of-the-art LLMs. The effects of different architectures may vanish with more data, but improving data efficiency is a key objective of this line of work. So the effects at small scale are particularly relevant. Third, our architectures with optimized non-linearities are **computationally costly**. Our claims are not centered on the performance of these architecture though. They serve instead to better understand the landscape of possible designs for future AI models.

Reproducibility Statement

Appendix B provides a formal description of the proposed method with the values of all hyperparameters. Code is available at `http://github.com/anonymized/anonymized`.

References

Konstantinos Panagiotis Alexandridis, Jiankang Deng, Anh Nguyen, and Shan Luo. Adaptive parametric activation. In *ECCV*, 2025.

Zeyuan Allen-Zhu. Physics of Language Models: Part 4.1, Architecture Design and the Magic of Canon Layers. *SSRN Electronic Journal*, May 2025.

Cem Anil, Yuhuai Wu, Anders Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. Exploring length generalization in large language models. *NeurIPS*, 2022.

Andrea Apicella, Francesco Isgro, and Roberto Prevete. A simple and efficient architecture for trainable activation functions. *Neurocomputing*, 2019.

Andrea Apicella, Francesco Donnarumma, Francesco Isgrò, and Roberto Prevete. A survey on modern trainable activation functions. *Neural Networks*, 2021.

Sanjeev Arora, Nadav Cohen, Wei Hu, and Yuping Luo. Implicit regularization in deep matrix factorization. *NeurIPS*, 2019.

Viraat Aryabumi, Yixuan Su, Raymond Ma, Adrien Morisot, Ivan Zhang, Acyr Locatelli, Marzieh Fadaee, Ahmet Üstün, and Sara Hooker. To code, or not to code? exploring impact of code in pre-training. *arXiv:2408.10914*, 2024.

Gregor Bachmann, Sotiris Anagnostidis, and Thomas Hofmann. Scaling MLPs: A tale of inductive bias. *arXiv:2306.13575*, 2023.

Samuel James Bell and Levent Sagun. Simplicity bias leads to amplified performance disparities. In *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*, pp. 355–369, 2023.

Satwik Bhattamishra, Arkil Patel, Varun Kanade, and Phil Blunsom. Simplicity bias in transformers and their ability to learn sparse boolean functions. *arXiv:2211.12316*, 2022.

Garrett Bingham, William Macke, and Risto Miikkulainen. Evolutionary optimization of deep learning activation functions. In *Genetic and Evolutionary Computation Conference*, 2020.

Ziyang Cai, Nayoung Lee, Avi Schwarzschild, Samet Oymak, and Dimitris Papailiopoulos. Extrapolation by association: Length generalization transfer in transformers. *arXiv:2506.09251*, 2025.

Irit Chelly, Shahaf E Finder, Shira Ifergane, and Oren Freifeld. Trainable highly-expressive activation functions. In *ECCV*. Springer, 2024.

CodeGPT. Microsoft CodeGPT (available on HuggingFace), 2024. `https://huggingface.co/microsoft/CodeGPT-small-py`.

Giacomo De Palma, Bobak Kiani, and Seth Lloyd. Random deep neural networks are biased towards simple functions. *NeurIPS*, 2019.

Kamaludin Dingle, Chico Q Camargo, and Ard A Louis. Input–output maps are strongly biased towards simple outputs. *Nature communications*, 2018.

Pedro Domingos. The role of occam's razor in knowledge discovery. *Data mining and knowledge discovery*, 1999.

Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 2022.

Stanislas Ducotterd, Alexis Goujon, Pakshal Bohra, Dimitris Perdios, Sebastian Neumayer, and Michael Unser. Improving lipschitz-constrained neural networks by learning activation functions. *Journal of Machine Learning Research*, 2024.

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. *NeurIPS*, 36, 2023.

Ronen Eldan and Yuanzhi Li. Tinystories: How small can language models be and still speak coherent english? *arXiv:2305.07759*, 2023.

Richard Futrell and Kyle Mahowald. How linguistics learned to stop worrying and love the language models. *Annual Review of Linguistics*, 2023.

Philip Gage. A new algorithm for data compression. *C Users Journal*, 12(2):23–38, 1994.

Gallant. There exists a neural network that does not make avoidable mistakes. In *IEEE International Conference on Neural Networks*. IEEE, 1988.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2020.

Micah Goldblum, Marc Finzi, Keefer Rowan, and Andrew Gordon Wilson. The no free lunch theorem, kolmogorov complexity, and the role of inductive biases in machine learning. *arXiv:2304.05366*, 2023.

Anirudh Goyal and Yoshua Bengio. Inductive biases for deep learning of higher-level cognition. *Proceedings of the Royal Society A*, 2022.

Mohit Goyal, Rajan Goyal, and Brejesh Lall. Learning activation functions: A new paradigm of understanding neural networks. *arXiv:1906.09529*, 2019.

Sam Greydanus and Dmitry Kobak. Scaling down deep learning with MNIST-1D. *arXiv preprint arXiv:2011.14439*, 2020.

Michael Hahn and Mark Rofin. Why are sensitive functions hard for transformers? *arXiv:2402.09963*, 2024.

Michael Hahn, Dan Jurafsky, and Richard Futrell. Sensitivity as a complexity measure for sequence classification tasks. *Transactions of the ACL*, 2021.

Baran Hashemi, Kurt Pasque, Chris Teska, and Ruriko Yoshida. Tropical attention: Neural algorithmic reasoning for combinatorial algorithms. *arXiv:2505.17190*, 2025.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GeLUs). *arXiv:1606.08415*, 2016.

Katherine L Hermann and Andrew K Lampinen. What shapes feature representations? exploring datasets, architectures, and training. *arXiv:2006.12433*, 2020.

Jinwook Hong. Asnn: Learning to suggest neural architectures from performance distributions. *arXiv:2507.20164*, 2025.

Sara Hooker. The hardware lottery. *CACM*, 64(12):58–65, 2021.

Leyang Hu, Matteo Gamba, and Randall Balestriero. Curvature tuning: Provable training-free model steering from a single parameter. *arXiv:2502.07783*, 2025.

Hamel Husel, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv:1909.09436*, 2019.

Ameya D Jagtap and George Em Karniadakis. How important are activation functions in regression and classification? a survey, performance comparison, and future directions. *Journal of Machine Learning for Modeling and Computing*, 4(1), 2023.

Ameya D Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 2020.

Samy Jelassi, David Brandfonbrener, Sham M Kakade, and Eran Malach. Repeat after me: Transformers are better than state space models at copying. *arXiv:2402.01032*, 2024.

Nandan Kumar Jha and Brandon Reagen. Entropy-guided attention for private llms. *arXiv:2501.03489*, 2025.

Zixuan Jiang, Jiaqi Gu, and David Z Pan. Normsoftmax: Normalizing the input of softmax to accelerate and stabilize training. In *IEEE International Conference on Omni-layer Intelligent Systems (COINS)*. IEEE, 2023.

Keller Jordan, Jeremy Bernstein, Brendan Rappazzo, @fernbear.bsky.social, Boza Vlado, You Jiacheng, Franz Cesista, Braden Koszarsky, and @Grad62304977. modded-nanogpt: Speedrunning the nanogpt baseline, 2024. URL https://github.com/KellerJordan/modded-nanogpt.

Dimitris Kalimeris, Gal Kaplun, Preetum Nakkiran, Benjamin Edelman, Tristan Yang, Boaz Barak, and Haofeng Zhang. SGD on neural networks learns functions of increasing complexity. *NeurIPS*, 2019.

Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness/, 2015.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *ICML*. PMLR, 2020.

Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. The impact of positional encoding on length generalization in transformers. *NeurIPS*, 2023.

Jan Kukačka, Vladimir Golkov, and Daniel Cremers. Regularization for deep learning: A taxonomy. *arXiv:1710.10686*, 2017.

Nayoung Lee, Kartik Sreenivasan, Jason D Lee, Kangwook Lee, and Dimitris Papailiopoulos. Teaching arithmetic to small transformers. *arXiv:2307.03381*, 2023.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *15th European Conference on Computer Vision (ECCV)*, 2018.

Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y Hou, and Max Tegmark. Kan: Kolmogorov-arnold networks. *arXiv:2404.19756*, 2024.

Shayne Longpre, Gregory Yauney, Emily Reif, Katherine Lee, Adam Roberts, Barret Zoph, Denny Zhou, Jason Wei, Kevin Robinson, David Mimno, et al. A pretrainer's guide to training data: Measuring the effects of data age, domain coverage, quality, & toxicity. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, 2024.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, et al. CodeXGlue: A machine learning benchmark dataset for code understanding and generation. *arXiv:2102.04664*, 2021.

Kaifeng Lyu, Zhiyuan Li, Runzhe Wang, and Sanjeev Arora. Gradient descent on two-layer nets: Margin maximization and simplicity bias. *NeurIPS*, 2021.

Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, 2013.

Matt Mahoney. enwik8: First 100m bytes of english wikipedia. Available at `https://mattmahoney.net/dc/textdata.html`, 2006.

Franco Manessi and Alessandro Rozza. Learning combinations of activation functions. In *International Conference on Pattern Recognition (ICPR)*, 2018.

Prasanna Mayilvahanan, Thaddäus Wiedemer, Sayak Mallick, Matthias Bethge, and Wieland Brendel. Llms on the line: Data determines loss-to-loss scaling laws. *arXiv:2502.12120*, 2025.

Raphaël Millière. Language models as models of language. *Mind & Language*, 2024.

Chris Mingard, Joar Skalse, Guillermo Valle-Pérez, David Martínez-Rubio, Vladimir Mikulik, and Ard A Louis. Neural networks are a priori biased towards boolean functions with low entropy. *arXiv:1909.11522*, 2019.

Chris Mingard, Guillermo Valle-Pérez, Joar Skalse, and Ard A Louis. Is SGD a bayesian sampler? well, almost. *Journal of Machine Learning Research*, 2021.

Chris Mingard, Henry Rees, Guillermo Valle-Pérez, and Ard A Louis. Do deep neural networks have an inbuilt occam's razor? *arXiv:2304.06670*, 2023.

Iman Mirzadeh, Keivan Alizadeh, Sachin Mehta, Carlo C Del Mundo, Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, and Mehrdad Farajtabar. Relu strikes back: Exploiting activation sparsity in large language models. *arXiv:2310.04564*, 2023.

Tom M Mitchell. The need for biases in learning generalizations. *Rutgers University CS tech report CBM-TR-117*, 1980.

Aaron Mueller and Tal Linzen. How to plant trees in language models: Data and architectural effects on the emergence of syntactic inductive biases. *Transactions of the ACL*, 2022.

Laker Newhouse, R Preston Hess, Franz Cesista, Andrii Zahorodnii, Jeremy Bernstein, and Phillip Isola. Training transformers with enforced lipschitz constants. *arXiv:2507.13338*, 2025.

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

Isabel Papadimitriou and Dan Jurafsky. Injecting structural hints: Using language models to study inductive biases in language learning. In *Proceedings of ACL*, 2022.

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin A Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *NeurIPS*, 2024.

Jackson Petty, Sjoerd van Steenkiste, and Tal Linzen. How does code pretraining affect language model task performance? *arXiv:2409.04556*, 2024.

Mohammad Pezeshki, Oumar Kaba, Yoshua Bengio, Aaron C Courville, Doina Precup, and Guillaume Lajoie. Gradient starvation: A learning proclivity in neural networks. *NeurIPS*, 2021.

Ofir Press, Noah A Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*, 2021.

Aahlad Manas Puli, Lily Zhang, Yoav Wald, and Rajesh Ranganath. Don't blame dataset shift! shortcut learning due to gradients and cross entropy. *NeurIPS*, 2023.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.

Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *ICML*. PMLR, 2019.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Swish: a self-gated activation function. *arXiv:1710.05941*, 2017.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. In *ICLR*, 2018.

Riccardo Rende, Federica Gerace, Alessandro Laio, and Sebastian Goldt. A distributional simplicity bias in the learning dynamics of transformers. *NeurIPS*, 37, 2024.

Vishwanath Saragadam, Daniel LeJeune, Jasper Tan, Guha Balakrishnan, Ashok Veeraraghavan, and Richard G Baraniuk. Wire: Wavelet implicit neural representations. In *CVPR*, 2023.

Hemanth Saratchandran, Sameera Ramasinghe, Violetta Shevchenko, Alexander Long, and Simon Lucey. A sampling theory perspective on activations for implicit neural representations. *arXiv:2402.05427*, 2024a.

Hemanth Saratchandran, Jianqiao Zheng, Yiping Ji, Wenbo Zhang, and Simon Lucey. Rethinking softmax: Self-attention with polynomial activations. *arXiv:2410.18613*, 2024b.

Simone Scardapane, Steven Van Vaerenbergh, Simone Totaro, and Aurelio Uncini. Kafnets: Kernel-based non-parametric activation functions for neural networks. *Neural Networks*, 2019.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *ICML*. PMLR, 2021.

Noam Shazeer. Glu variants improve transformer. *arXiv:2002.05202*, 2020.

Zachary Shinnick, Liangze Jiang, Hemanth Saratchandran, Anton van den Hengel, and Damien Teney. Transformers pretrained on procedural data contain modular structures for algorithmic reasoning. *arXiv preprint arXiv:2505.22308*, 2025.

David So, Wojciech Mańke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V Le. Searching for efficient transformers for language modeling. *NeurIPS*, 34, 2021a.

David R So, Wojciech Manke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V Le. Primer: Searching for efficient transformers for language modeling, 2022. *arXiv:2109.08668*, 2021b.

Leon René Sütfeld, Flemming Brieger, Holger Finger, Sonja Füllhase, and Gordon Pipa. Adaptive blending units: Trainable activation functions for deep neural networks. In *Intelligent Computing: Proceedings of the Computing Conference*. Springer, 2020.

Remi Tachet, Mohammad Pezeshki, Samira Shabanian, Aaron Courville, and Yoshua Bengio. On the learning dynamics of deep neural networks. *arXiv:1809.06848*, 2018.

Camilo Tamayo-Rousseau, Yunjia Zhao, Yiqun Zhang, and Randall Balestriero. Your attention matters: to improve model robustness to noise and spurious correlations. *arXiv:2507.20453*, 2025.

Yi Tay, Mostafa Dehghani, Samira Abnar, Hyung Won Chung, William Fedus, Jinfeng Rao, Sharan Narang, Vinh Q Tran, Dani Yogatama, and Donald Metzler. Scaling laws vs model architectures: How does inductive bias influence scaling? *arXiv:2207.10551*, 2022.

Damien Teney, Ehsan Abbasnejad, Simon Lucey, and Anton van den Hengel. Evading the simplicity bias: Training a diverse set of models discovers solutions with superior ood generalization. *arXiv:2105.05612*, 2021.

Damien Teney, Maxime Peyrard, and Ehsan Abbasnejad. Predicting is not understanding: Recognizing and addressing underspecification in machine learning. In *ECCV*. Springer, 2022.

Damien Teney, Armand Mihai Nicolicioiu, Valentin Hartmann, and Ehsan Abbasnejad. Neural redshift: Random networks are not random functions. In *CVPR*, 2024.

Damien Teney, Liangze Jiang, Florin Gogianu, and Ehsan Abbasnejad. Do we always need the simplicity bias? looking for optimal inductive biases in the wild. In *CVPR*, 2025.

Guillermo Valle-Perez, Chico Q Camargo, and Ard A Louis. Deep learning generalizes because the parameter-function map is biased towards simple functions. *arXiv:1805.08522*, 2018.

Bhavya Vasudeva, Deqing Fu, Tianyi Zhou, Elliott Kau, Youqi Huang, and Vatsal Sharan. Simplicity bias of transformers to learn low sensitivity functions. *arXiv:2403.06925*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 2017.

Petar Veličković, Christos Perivolaropoulos, Federico Barbero, and Razvan Pascanu. softmax is not enough (for sharp out-of-distribution). *arXiv:2410.01104*, 2024.

Alex Warstadt and Samuel R Bowman. What artificial neural networks can tell us about human language acquisition. *Trends in Cognitive Sciences*, 2020.

Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers. *arXiv:2301.08727*, 2023.

David H Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 1996.

David H Wolpert. The supervised learning no-free-lunch theorems. *Soft computing and industry: Recent applications*, 2002.

Zhi-Qin John Xu, Yaoyu Zhang, Tao Luo, Yanyang Xiao, and Zheng Ma. Frequency principle: Fourier analysis sheds light on deep neural networks. *arXiv:1901.06523*, 2019.

Xiulin Yang et al. (anything) goes? a crosslinguistic study of (im)possible language learning in lms. In *Proceedings of ACL*, 2024.

York Yong. Clean version of the wikipedia. Available at `https://www.kaggle.com/api/v1/datasets/download/yorkyong/text8-zip`, 2025.

Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. Are transformers effective for time series forecasting? In *AAAI*, 2023.

Zhongwang Zhang, Pengxiao Lin, Zhiwei Wang, Yaoyu Zhang, and Zhi-Qin John Xu. Initialization is critical to whether transformer fits composite function by inference or memorizing. *arXiv:2405.05409*, 2024.

Ziqian Zhong and Jacob Andreas. Algorithmic capabilities of random transformers. *arXiv:2410.04368*, 2024.

Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Josh Susskind, Samy Bengio, and Preetum Nakkiran. What algorithms can transformers learn? a study in length generalization. *arXiv:2310.16028*, 2023.

Yongchao Zhou, Uri Alon, Xinyun Chen, Xuezhi Wang, Rishabh Agarwal, and Denny Zhou. Transformers can achieve length generalization but not robustly. *arXiv preprint arXiv:2402.09371*, 2024.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

APPENDIX

A ADDITIONAL RELATED WORK

Inductive biases in deep learning are due to choices of architecture (Goyal & Bengio, 2022) and of the learning algorithm (optimizer, objective, regularizers (Kukačka et al., 2017)). We focus on the former. The simplicity bias has been studied from both aspects. Most explanations attribute it to loss functions (Pezeshki et al., 2021) and gradient descent (Arora et al., 2019; Hermann & Lampinen, 2020; Lyu et al., 2021; Tachet et al., 2018). But work on untrained networks shows that it can be explained with architectures alone (De Palma et al., 2019; Goldblum et al., 2023; Mingard et al., 2019; Teney et al., 2024; Valle-Perez et al., 2018). Teney et al. (2024) showed that the choice of activation function can modulate the simplicity bias. The **spectral bias** (Rahaman et al., 2019; Kalimeris et al., 2019) or frequency principle (Xu et al., 2019) is a related but different effect related to training dynamics: NNs approximate low-frequency components of the target function earlier during training with SGD.

Simplicity bias in transformers. The hypothesis of a simplicity bias in NNs has also been studied specifically in transformers. Hahn et al. (2021) shows that common models in NLP are biased to learn low-sensitivity functions. Bhattamishra et al. (2022) shows that transformers are more biased for simplicity than LSTMs. Dziri et al. (2023) examine large pretrained models and determine that that tend rely on shortcut learning on simple reasoning tasks. Zhou et al. (2023) focus on length generalization and show that transformers learn the shortest program in the RASP language that fits the training data –a specific form of the simplicity bias. Rende et al. (2024) study BERT-like models and find that they learn simple functions first during the course of training. Zhang et al. (2024) find that the scale of initialization can influence a transformer's learning of a generalizing or memorizing solution. Vasudeva et al. (2024) further study the bias of transformers for learning low-sensitivity functions using the NTK theory. Hahn & Rofin (2024) show that sensitive functions are hard to learn for transformers because they correspond to sharp solutions in their optimization landscape as a side-effect of the simplicity bias.

Activation functions are key for introducing non-linearities in NNs. Many options were considered early on, e.g. sine activations in the Fourier Neural Networks from 1988 (Gallant, 1988). ReLUs are often credited for enabling the rise of deep learning by avoiding vanishing gradients (Maas et al., 2013). However they are also essential in inducing the simplicity bias (Teney et al., 2024) which may be just as important. The research community has slowly converged towards smooth handcrafted variants of ReLUs such as GeLUs (Dubey et al., 2022; Hendrycks & Gimpel, 2016; Ramachandran et al., 2017). Some works proposed **learning activation functions** using extra parameters optimized alongside the weights of the network (Alexandridis et al., 2025; Apicella et al., 2019; 2021; Bingham et al., 2020; Chelly et al., 2024; Ducotterd et al., 2024; Jagtap et al., 2020; Scardapane et al., 2019; Sütfeld et al., 2020). See Jagtap & Karniadakis (2023) for a comprehensive review. The goal is to better fit the training data with an activation function that can evolve during training. In contrast, we use meta learning to find an activation function that induces better inductive biases, such that training with this *fixed* activation provides better generalization. This requires bi-level optimization, episodic training, and unbiased parametrization that allows us to learn activations very different from existing ones. **Kolmogorov-Arnold Networks** (Liu et al., 2024) parametrize the connections in a NN, which is equivalent to learning different activation functions across channels and layers. They use a parametrization as splines similar to ours. Their benefits in physics-related problems likely result from the alterations to the inductive biases studied in this paper. Our method differs from **neural architecture search** (White et al., 2023) in its ability to discover novel activation functions from scratch, rather than selecting from predefined candidates (Sütfeld et al., 2020) or from a narrow set of parametric functions (Alexandridis et al., 2025).

Length generalization refers to the ability of a model to generalize to sequences longer than seen during training, especially for algorithmic tasks (e.g. arithmetic operations on numbers with more digits). This remains a challenge despite extensive work on positional encodings, which only partially address the problem (Anil et al., 2022; Kazemnejad et al., 2023; Zhou et al., 2024). This paper shows that other aspects of the architecture can be important. We use the COPY task as proof of concept and show that different MLP activation functions can bring a significant improvement to the existing Alibi encodings (Press et al., 2021).

Learnability and inductive biases. The learnability of any given task is a fundamental question in machine learning. It is well know that inductive biases are indispensable for generalization to unseen data (Mitchell, 1980) and that no learning algorithm is universally useful, as per one of the no-free lunch theorems (Wolpert, 2002)). Meanwhile, neural networks have nevertheless proved widely successful. The broad applicability of transformers, in particular, suggests that their inductive bias has a broad relevance to real-world data (Goldblum et al., 2023). The **simplicity bias** is a broad and vague characterization of these properties. Various studies have established however that the simplicity bias is not universally beneficial (Domingos, 1999; Teney et al., 2025; Zeng et al., 2023) and even responsible for failure cases such as **shortcut learning** (Geirhos et al., 2020; Puli et al., 2023; Teney et al., 2021) or the amplification of biases and performance disparities (Bell & Sagun, 2023). Even the underlying principle supporting the simplicity bias, known as **Occam's razor**, has long been debated in the philosophical literature because it lacks a justification from first principles (Mingard et al., 2023, Appendix A). A prominent argument for simplicity is rooted in algorithmic information theory (Dingle et al., 2018) with results stating essentially that "*a bias in the distribution of target functions must be towards low complexity*". However, this only means that simplicity is a good prior on average, but not necessarily the best choice for any task or dataset.

Studies in linguistics and cognitive science have also examined the question of learnability. This includes studies on the influence of architectures and data on generalization during **language acquisition**, both for humans and machines (Futrell & Mahowald, 2023; Millière, 2024; Warstadt & Bowman, 2020). This explains how syntactic and structural biases arise and how they can be controlled (Mueller & Linzen, 2022; Papadimitriou & Jurafsky, 2022; Yang et al., 2024). Our paper complements this line of work since it helps clarify the impact of architectures on generalization. Our approach is quite different though. Our method allows searching through the space of architectures via the optimization of non-linearities. This matters because current popular designs (e.g. transformers) are contingent on external factors, cf. the **Hardware Lottery** (Hooker, 2021)).

Connection with prior work. This paper is a follow-up the study by Teney et al. (2025) that uses trainable non-linearities to study whether the *simplicity bias* of standard neural architectures is always desirable. It was however limited to MLPs and toy data, and relied on an expensive optimization method unsuitable to modern architectures. In comparison, our main innovations are:

- a formulation of trainable non-linearities that applies to transformers' MLPs and attention layers;
- a tractable optimization method replacing the expensive bi-level approach from prior work;
- the study of mainstream domains (language modeling, algorithmic reasoning);
- the study of cross-task compatibility, whereas prior work focuses on individual datasets;
- a demonstration of massive improvements on algorithmic tasks;
- a PyTorch implementation that allows swapping standard activation functions for optimized ones with a few lines of code, available at `https://github.com/anonymized/anonymized`.

B Implementation Details

Proposed method. We provide a formal description of our method in Algorithm 1.

Algorithm 1 Proposed method (stage I) to optimize a transformer architecture for a specific task.

Input:
Training data $\mathbb{D} = \{\boldsymbol{s}_i\}_{i=1}^{n}$ as token sequences $\boldsymbol{s} \in \mathbf{S}$.
Baseline architecture $\mathcal{T}$ instantiable as next-token prediction model $T_{\boldsymbol{\theta}} : \mathbf{S} \to \mathbf{S}$ of weights $\boldsymbol{\theta}$.
$\mathcal{L}(\cdot,\cdot)$: Loss function. α: Fraction of held-out data. M: Number of parallel models.

Method:
Define a new architecture $\hat{\mathcal{T}}_{\boldsymbol{\theta}_\mathrm{A}\,\boldsymbol{\theta}_\mathrm{MLP}}$ by replacing
– softmaxes with $\Sigma_j\, K(\mathbf{Q}_i, \mathbf{K}_j)\, \mathbf{V}_j \,/\, \Sigma_j\, K(\mathbf{Q}_i, \mathbf{K}_j)$, where $K(\mathbf{Q}, \mathbf{K}) = \phi_{\boldsymbol{\theta}_\mathrm{A}}(\mathbf{Q})^\top \phi_{\boldsymbol{\theta}_\mathrm{A}}(\mathbf{K})$.
– GeLUs with a linear spline $\phi_{\boldsymbol{\theta}_\mathrm{MLP}}$,
where architecture hyperparameters $\boldsymbol{\theta}_\mathrm{A}$ and $\boldsymbol{\theta}_\mathrm{MLP}$ specify the value of splines ϕ at their keypoints.
Instantiate M untrained models of architecture $\hat{\mathcal{T}}$ as $\hat{T}^1_{\boldsymbol{\theta}_1}$ ... $\hat{T}^M_{\boldsymbol{\theta}_M}$.
Split $\mathbb{D}$ into $\mathbb{D}_\mathrm{arch}$ and $\mathbb{D}_\mathrm{wts}$ of sizes $\alpha\, n$ and $(1-\alpha)n$.

while not converged *SGD training loop*
Sample mini-batch $\mathbb{D}^0$ from $\mathbb{D}_\mathrm{arch}$ and $\mathbb{D}^1$ ... $\mathbb{D}^M$ from $\mathbb{D}_\mathrm{wts}$
Eval. loss of each individual model on its own data $\mathbb{D}^m$: $L^m_\mathrm{wts} \leftarrow \Sigma_{\boldsymbol{s} \in \mathbb{D}^m}\, \mathcal{L}\big(\hat{T}^m_{\boldsymbol{\theta}_m}(\boldsymbol{s}), \boldsymbol{s}\big)$
Eval. combined loss of all models together on $\mathbb{D}^0$: $L_\mathrm{arch} \leftarrow \Sigma_m\, \Sigma_{\boldsymbol{s} \in \mathbb{D}^0}\, \mathcal{L}\big(\hat{T}^m_{\boldsymbol{\theta}_m}(\boldsymbol{s}), \boldsymbol{s}\big)$
Update weights of each model: $\forall m,\ \boldsymbol{\theta}_m \leftarrow \mathrm{SGD}(\boldsymbol{\theta}_m, \nabla_{\boldsymbol{\theta}} L^m_\mathrm{wts})$
Update architecture: $(\boldsymbol{\theta}_\mathrm{A}, \boldsymbol{\theta}_\mathrm{MLP}) \leftarrow \mathrm{SGD}((\boldsymbol{\theta}_\mathrm{A}, \boldsymbol{\theta}_\mathrm{MLP}), \nabla_{(\boldsymbol{\theta}_\mathrm{A}, \boldsymbol{\theta}_\mathrm{MLP})} L_\mathrm{arch})$

$(\boldsymbol{\theta}^\star_\mathrm{A},\, \boldsymbol{\theta}^\star_\mathrm{MLP}) \;\leftarrow\; (\boldsymbol{\theta}_\mathrm{A},\, \boldsymbol{\theta}_\mathrm{MLP})$.
Output: optimized architecture $\hat{\mathcal{T}}_{\boldsymbol{\theta}^\star_\mathrm{A}\,\boldsymbol{\theta}^\star_\mathrm{MLP}}$
Now $\hat{\mathcal{T}}$ can be used like any other architecture, treating $\boldsymbol{\theta}^\star_\mathrm{A}$ and $\boldsymbol{\theta}^\star_\mathrm{MLP}$ as fixed hyperparameters.

Baseline transformer architecture. Our baseline is a GPT-2-style architecture (Radford et al., 2019). It uses standard multi-head attention, GeLU activation functions in the MLPs, post-norm layers, learned absolute positional embeddings, and a width multiplier of 4 in the MLP hidden layers. All weights are initialized from Gaussians of standard deviation 0.02 truncated at 2 standard deviations.

Parametrization of non-linearities as linear splines. We want a search space free of priors such as the smoothness and monotonicity enforced in similar work on the learning of activation functions (e.g. Apicella et al. (2019); Chelly et al. (2024)). We therefore choose to learn a non-linearity as a linear spline $\phi_{\boldsymbol{\theta}} : \mathbb{R} \to \mathbb{R}$ with control points defined by $\boldsymbol{\theta}$. We define n_c points spread regularly in an interval $[a, b]$, typically $n_\mathrm{c} = 122$ points in $[-20, +20]$ for a spacing of $1/3$ between points (see hyperparameters in Table 4). Then ϕ represents piecewise linear segments interpolating values specified in the learned parameters $\boldsymbol{\theta} := [\,\phi_{\boldsymbol{\theta}}(a), \ldots \phi_{\boldsymbol{\theta}}(b))\,] \in \mathbb{R}^{n_\mathrm{c}}$. The function ϕ can represent simple and complex functions, including smooth curves, periodic functions, sharp transitions, etc.

Datasets for algorithmic tasks. For most tasks, we generated data with code adapted from Zhong & Andreas (2024): `https://github.com/fjzzq2002/random_transformers`. While this prior work generates some of the data on-the-fly, we pre-generate all the data to ensure that the training/validation/test splits are strictly disjoint.

For MANO, we re-implemented the data generation based on the description by Allen-Zhu (2025). Compared to this prior work, we scaled down the task to allow using smaller models. We generated 1e5 training examples, with a number of operations in each sequence in [1,3], a modulus of 7, and without tokens signaling the number of operations.

For all algorithmic tasks, we use a test set of 1e3 examples, strictly disjoint from the training set.

Datasets for language modeling. For datasets tokenized at the character level, every character or symbol in the data simply corresponds to one token. For the TINYSTORIES, SHAKESPEARE, and ENWIK8 datasets tokenized at the subword level, we use the byte-pair encoding (BPE, Gage (1994)) tokenizer from GPT-2 Radford et al. (2019). We consider it a consistent choice suitable to our different datasets since it was originally trained on very diverse data. For the CODESEARCHNET datasets, we use the tokenizer of the CodeGPT model (CodeGPT, 2024). For each dataset, we discard tokens with fewer than 200 training occurrences. This significantly reduces the vocabulary size and training costs. This should not undermine the results of our experiments: if anything, including more rare tokens could reveal larger differences across datasets.

Metrics. For the algorithmic tasks, we measure performance as the token-wise accuracy of the "output" part of the generated sequences (the same part of the sequences as used to compute the training loss). This allows a finer-grained evaluation of partial success than the sequence-wise accuracy.

For the COPY task, we use the sequence-wise accuracy because the token-wise accuracy can remain falsely high when a model fails at length generalization.

For the language modeling tasks, we measure performance using the training perplexity (exponential of cross-entropy loss) as well as token-wise accuracy on validation data as a more intuitive measure of performance. For the accuracy, we measure it on the latter half of the context window to ensure that we evaluate predictions with enough conditioning.

For the compatibility across algorithmic tasks (Figure 5), we plot the difference in test accuracy with the baseline after a fixed number of steps. We adapt the number of steps to each task to capture improvements in generalization and/or training speed depending on the task. This is because both the baseline and optimized architectures saturate at perfect accuracy for multiple tasks, hence the *final* accuracy alone is not informative.

- MEMORIZE: 150 steps.
- PARENTHESES: 300 steps.
- ADDMOD: 300 steps.
- HAYSTACK: 400 steps.
- ADD: 700 steps.
- ADDREVERSED: 350 steps.
- COPY: 2,000 steps.
- MANO: 3,000 steps.

Hyperparameters. We tuned the hyperparameters in Table 4 for a standard transformer on each task, to make sure that our optimized architectures are compared against strong baselines. For example, we use the Canon layers proposed by Allen-Zhu (2025) for many tasks (sequence-wise 1D convolutions) because they clearly improve the performance of the baseline.

Table 4: Hyperparameters used for each task.

	MEMORIZE	PARENTHESES	ADDMOD	HAYSTACK	ADD	ADDREVERSED	COPY	MANO	Language datasets
Num. layers	2								4
Num. att. heads	2	2	2	2	2	2	8	4	4
Width	32	32	32	128	128	128	128	128	512
Tied embeddings	No								Yes
Canon layers	No						Yes		
Num. tr. steps	500	500	1,000	1,000	1,000	1,000	2,000	5,000	3,000
Peak LR	.005	.001	.02	.001	.001	.001	.004	.001	.001
Batch size	512								64
Optimizer	Adam								
Adam (β_1,β_2)	(0.9, 0.999)						(0.92, 0.98)		(0.9, 0.999)
LR schedule	5% linear warm-up, 50% cosine cool-down (not necessary for algorithmic tasks; used on all tasks for consistency)								
Weight decay	0 (better on all tasks than using any weight decay)								
Dropout rate	0								
Parallel models M	8								3
Spline range $[a,b]$	$[-20, 20]$								
Spline spacing n_c	1/9						1/3		

C ADDITIONAL RESULTS ON ALGORITHMIC TASKS

Training curves. Figure 10 shows that the optimized architectures (2nd and 3rd columns) always converge significantly faster than a baseline transformer (1st column) and show less variance across seeds. There is little difference between the 2nd and 3rd columns, which means that most of the benefits come from optimizing the non-linearity within the MLP layers rather than the attention.

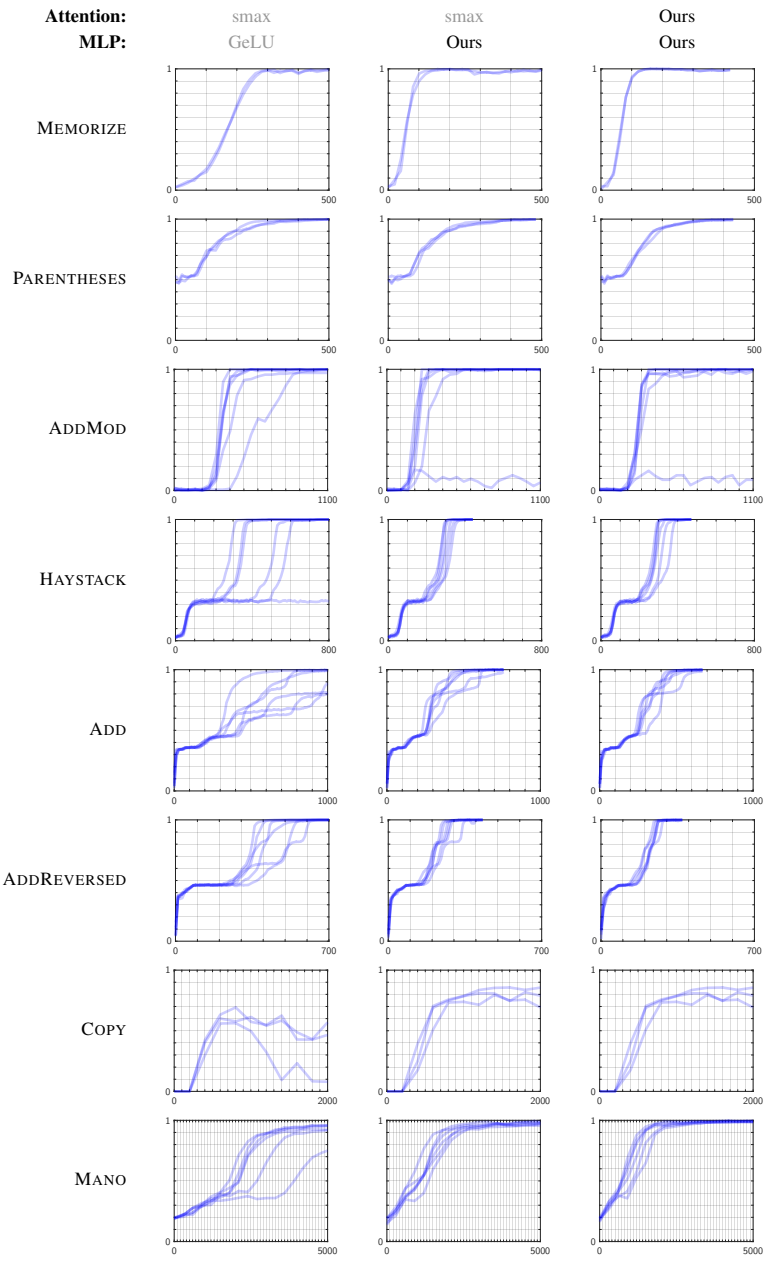


Figure 10: Training curves (test accuracy vs. training steps, one curve per seed) of models trained on algorithmic tasks with a baseline transformer (first column) or optimized architectures (second and third columns).

Compatibility of architectures across algorithmic tasks. We present below the full results following the format of Figure 5. We show the effect when optimizing the non-linearities in MLP or attention layers, or both. Optimizing the non-linearities in the attention proves to be really challenging, and the best results are usually obtained by optimizing only the MLPs.

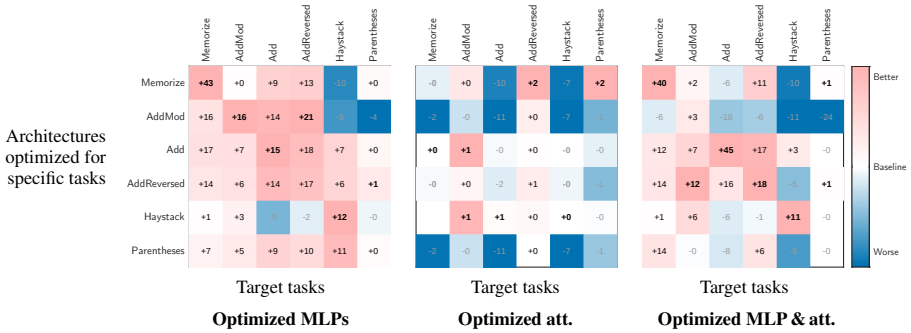


Figure 11: Compatibility of architectures across algorithmic tasks (difference in test accuracy with the baseline after a fixed number of steps).

D ADDITIONAL RESULTS ON LANGUAGE MODELING

Manipulating optimized non-linearities. In these experiments, we slightly modify the optimized MLP non-linearities to understand the importance of their fine details. Since they often look like sinusoidal wavelets, perhaps an even more regular version of them could perform better. We automate a "cleaning" process of the optimized non-linearities as follows. We take the optimized spline, reverse it along the X and/or Y axis (yielding three different versions), then align it with the original one by maximizing their cross-correlation. We then keep the average of the two. Among the three versions, we retain the one with the highest cross-correlation (i.e. similarity) with the original spline. The result is symmetric or anti-symmetric with fewer irregularities than the original one. We visualize this effect in Figure 12 on MLP non-linearities optimized for TINYSTORIES and various model sizes. We train models with these, but in almost every case, they perform worse than the original ones. This shows that fine details in the original optimized non-linearities matter.

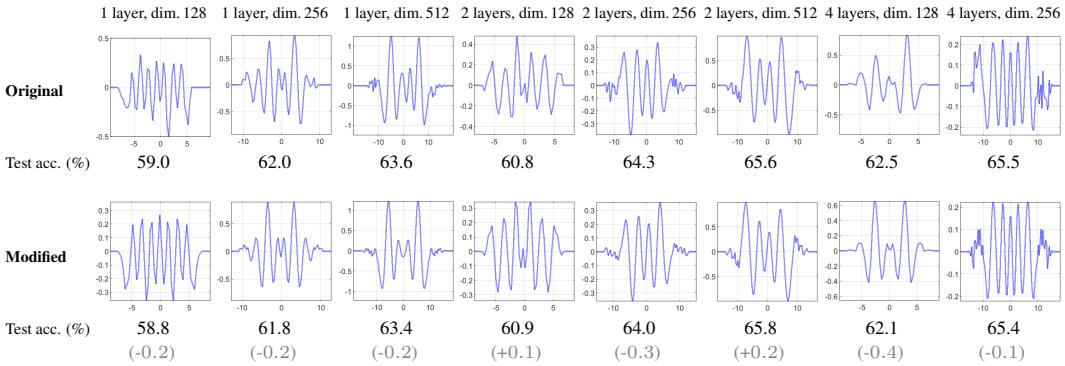


Figure 12: MLP non-linearities optimized for TINYSTORIES and versions modified to enforce symmetry. Almost all of these perform worse than the original ones, whose fine details therefore matter.

Multi-model training. We compare in Table 5 architectures for TINYSTORIES obtained with the proposed method and $M=1$ or $M=6$ models in parallel. The latter are slightly better, and the optimized non-linearities look slightly more regular.

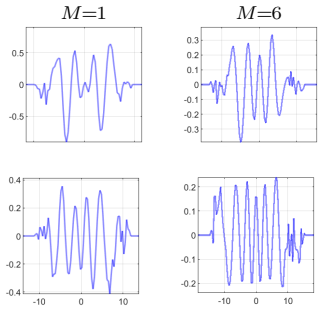
Table 5: Models for TINYSTORIES with architectures optimized with M=1 or 6 parallel models.

(Models with 2 layers, width 256)

Attention **MLP**	smax Linear	smax GeLU	smax Ours, M=1	smax Ours, M=6
Tr. perplexity	1.78	1.58	1.59	**1.57**
Val. acc. (%)	59.9	63.7	63.8	**64.3**

(Models with 4 layers, width 256)

Attention **MLP**	smax Linear	smax GeLU	smax Ours, N=1	smax Ours, N=6
Tr. perplexity	1.73	1.53	1.53	**1.52**
Val. acc. (%)	60.8	65.1	65.3	**65.4**



Existing methods. Below are references for the attention and MLP designs evaluated in Table 3.

- **Adaptive softmax**: Veličković et al. (2024).
- **NormSoftmax**: Jiang et al. (2023).
- **Polynomial attention P1**: $(\mathbf{Q}^\top\mathbf{K})/\sqrt{\text{seqLength}}$: Saratchandran et al. (2024b).
- **Polynomial attention P3**: $(\mathbf{Q}^\top\mathbf{K})^3/\sqrt{\text{seqLength}}$: Saratchandran et al. (2024b).
- **GLU**: Shazeer (2020).
- **ReLU^2**: So et al. (2021b).
- **Sinc**: Saratchandran et al. (2024a).
- **Gaussian**: Saragadam et al. (2023).

Full results on Shakespeare. We present below results on the SHAKESPEARE dataset for various model sizes, in the same format as Figure 7. The best configuration is to optimize the MLP non-linearities while keeping the original softmax attention (second panels from the left).

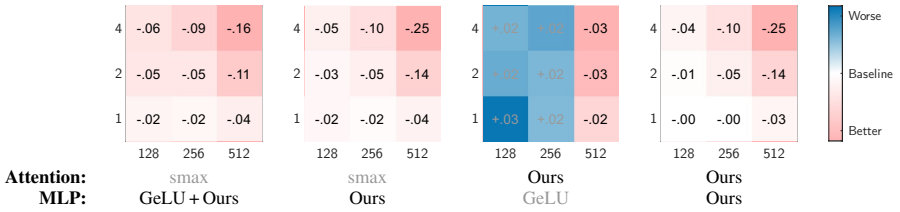


Figure 13: Absolute improvements in training perplexity on character-level SHAKESPEARE for models of different sizes (number of layers $\times$ width).

Figure 14: Same as Figure 13 with subword-level tokenization.

Training curves on language datasets. Figure 15 shows that the optimized architectures (■) show a larger improvement over a baseline transformer early during training, which then diminishes.

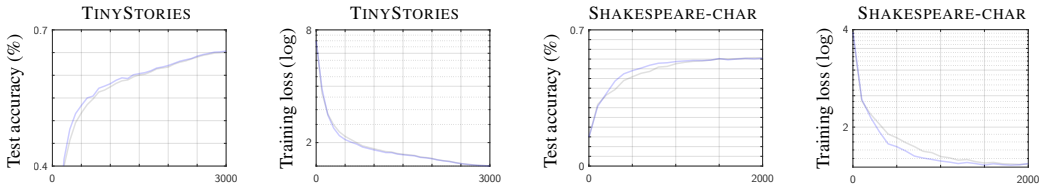


Figure 15: Training curves on language datasets with baseline (■) and optimized (■) architectures.

E RESULTS WITH LARGER LANGUAGE MODELS

On the suggestions of reviewers, we perform additional experiments to evaluate the improvements from the optimized non-linearities at various scales. We repeat experiments on language modeling as in Section 4 with the following differences.

- We use the **FINEWEB dataset** (Penedo et al., 2024), a popular high-quality dataset of cleaned and deduplicated English text from CommonCrawl.
- We implement our method on top of a very strong baseline, the **NanoGPT Speedrun** (Jordan et al., 2024). This is a competitive repository where contributors specifically push the implementation and data efficiency of the model on the FINEWEB dataset. We specifically build on top of record #16, which includes rotary embeddings, QK normalization, the Muon optimizer, sliding-window attention, mixed-precision training, etc. The code was designed for 8 H100 GPUs but we adapted it to enable experiments with a single Nvidia RTX 4090 laptop GPU. Our results are therefore not directly comparable with the official Speedrun competition. See our code for details: `https://github.com/anonymized/anonymized`.
- We first run stage I of our method to optimize the MLP non-linearities of a small model, since this stage is computationally more expensive (2 layers, width 256, 4 attention heads). We then re-use the optimized non-linearity to run stage II (i.e. standard training) with models of **various sizes from 2 to 12 layers**. This setup therefore evaluates how the optimized non-linearities transfer across models of different depths.
- We train similar models (with 2 to 12 layers) with a ReLU, which is the best baseline for this codebase. We always use a standard attention with a softmax since we found in Section 4 that it was difficult to improve upon.

Results. The results in Table 6 show that our optimized non-linearities perform similarly or better than the baselines. There is little improvement at the smallest scale (probably because the model is very weak overall) but we get a consistent improvements at all other scales up to 12 layers, surpassing both the ReLU and GeLU baselines in most cases.

Regarding the computational cost of the optimized non-linearities, our implementation (Listing 1) is as fast or faster than a ReLU in very small models. In larger models however, they become much more expensive. We propose in Appendix F a polynomial approximation. Table 6 shows that this approximation performs about as well as the original spline and about as fast as a ReLU.

Table 6: Evaluation of models of various depths trained on FINEWEB (average over 3 seeds).

Validation loss (FINEWEB)

Number of layers	2	4	8	10	12
Number of parameters (M)	91	105	133	148	162
Linear	4.21	4.05	3.93	3.90	3.88
ReLU	4.01	3.87	3.78	3.75	3.73
GeLU	**4.00**	3.89	**3.72**	3.75	3.72
Ours: linear spline	**4.00**	**3.82**	**3.72**	**3.68**	3.69
Ours: polynomial approx. ($n = 18$)	4.01	**3.82**	**3.72**	3.70	**3.68**

Training time (sec)

Number of layers	2	4	8	10	12
Linear	**1,440**	**1,920**	**2,940**	**3,540**	**19,680**
ReLU	1,500	1,980	3,120	13,080	28,020
GeLU	**1,440**	**1,920**	3,090	20,580	34,020
Ours: linear spline	1,500	2,070	8,520	26,700	81,720
Ours: polynomial approx. ($n = 18$)	**1,440**	2,040	3,180	14,070	29,100

F EFFICIENT IMPLEMENTATION OF SPLINES

Exact implementation. Our non-linearities are parametrized as linear splines. We first provide an exact efficient implementation (Listing 1) that we find to be as fast as standard activations such as GeLUs for small models. However, depending on the architecture and GPU used, this function can quickly get bandwidth-constrained and become significantly slower. Therefore we propose a faster approximation with polynomials to be used when the spline has already been optimized and is used as a frozen non-linearity (i.e. for standard training, as in stage II of our experiments).

```
# Evaluate, at points x (typically in bfloat16), a 1D function defined as the
# linear interpolation of knots, of coordinates 'knotPos' and values 'knotVals'
# (both typically in float32).
@torch.compile(dynamic=False)
def eval_spline(x, knotPos, knotVals):
    idx = torch.bucketize(x, knotPos) - 1 # Find the interval each x falls into
    idx = idx.clamp(0, len(knotPos) - 2)

    stepSize = knotPos[1] - knotPos[0]
    x0 = knotPos[0] + idx * stepSize
    frac = (x - x0) / stepSize
    frac = frac.clamp(0.0, 1.0) # Constant extrapolation beyond the knots

    y0 = knotVals[idx]
    y1 = knotVals[idx + 1]
    out = y0 + frac * (y1 - y0) # Linear interpolation
    return out.to(x.dtype) # Back to bfloat16; knotPos/frac/out were float32
```

Listing 1: Exact evaluation of a linear spline, used for stages I and II of most of our experiments.

Approximation with polynomials. The splines learned in our experiments with language models are quite smooth (unlike with algorithmic tasks in Section 3). It is therefore reasonable to approximate them with polynomials, which are much simpler and faster to evaluate. Concretely, given a linear spline optimized in stage I of our method, we determine an approximation through a least-squares fit of a polynomial of chosen degree n on the spline values at its knots, on its support that has non-zero values. We choose a high degree ($n = 18$ typically) to ensure high fidelity with the original spline and to avoid ringing artifacts near the support boundaries. Beyond the boundaries, the polynomial is clamped to 0. For efficiency, we evaluate the polynomial with Horner's method, and implement it in a compiled function using `TorchsSscript` (see Listing 2).

```
@torch.jit.script
def eval_polynomial(x: torch.Tensor) -> torch.Tensor:
    x = x.clamp(-79.52, 71.65) # Clamp for constant extrapolation
    x = x / 79.52 # Normalize to get values within [-1,1] for numerical stability
    return ((((((((((((((((((29327.20)*x + 18324.92)*x - 41591.43)*x - 12376.90)*x -
    14822.88)*x - 29015.27)*x + 33452.63)*x + 10354.57)*x + 21105.54)*x + 45592.25)*x -
    47565.33)*x - 47925.56)*x + 26296.37)*x + 18216.14)*x - 6145.61)*x - 2660.53)*x +
    522.13)*x + 66.86)*x - 0.63 # Evaluate polynomial with Horner's method
```

Listing 2: Example of polynomial approximation of a spline (best one from Table 7). It uses Horner's method with hard-coded coefficients and is compiled with `TorchScript` for efficiency.

Importance of *high degree* polynomials. We tried reducing the maximum degree of the polynomials. This creates smoother functions that look appealing, but they perform systematically worse than high-degree polynomials or than the original spline. This shows the importance of fine details in the optimized splines. We also tried to suppress noise and artifacts near the support boundaries, by analytically enforcing null derivatives (up to 4th derivatives) of the polynomial at the boundaries. The functions are again visually appealing but they do not necessarily work better when training models with them. The data-driven optimization is clearly superior to our hand-crafted tweaks. One possible improvement that we have not implemented is an approximation with Chebyshev polynomials. These are known to provide better approximations of functions with finite supports, with less artifacts and better numerical stability.

Do we need splines at all? We tried to do away with splines entirely and directly optimize coefficients of a polynomial in stage I of our method. This completely fails however. Even though splines and polynomials can represent similar sets of functions, the different parametrization apply different inductive biases on the learned non-linearities. As discussed in Section 2, splines are particularly effective because they correspond to the most uniform prior on the space of functions.

Evaluation of polynomial approximations. We train small language models on FINEWEB with a different non-linearity for the MLP layers. We keep all hyperparameters identical and similar to Section E. Here, we use 6 layers, a width of 256, 4 attention heads, ∼20M parameters, and ∼80M training tokens. The results in Table 7 show that our spline performs best and slightly better than a ReLU. As expected, **the polynomial approximations are increasingly effective as we increase the degree**. The approximation then becomes very close to the exact spline. Low-degree polynomials yield smoother functions that are visually appealing but do not work as well. This shows that the parametrization as a spline is important to capture subtle important details.

Table 7: Models trained on FINEWEB with various MLP non-linearities. Our optimized spline works best. Approximations with high-degree polynomials are effective as they faithfully approximate the spline.