

REVERSING LARGE LANGUAGE MODELS FOR EFFICIENT TRAINING AND FINE-TUNING

ESHED GAL*, MOSHE ELIASOF†, JAVIER TUREK‡, URI ASCHER§, ERAN TREISTER¶,
AND ELDAD HABER||

Abstract. Large Language Models (LLMs) are known for their expensive and time-consuming training. Thus, oftentimes, LLMs are fine-tuned to address a specific task, given the pretrained weights of a pre-trained LLM considered a foundation model. In this work, we introduce memory-efficient, reversible architectures for LLMs, inspired by symmetric and symplectic differential equations, and investigate their theoretical properties. Different from standard, baseline architectures that store all intermediate activations, the proposed models use time-reversible dynamics to retrieve hidden states during backpropagation, relieving the need to store activations. This property allows for a drastic reduction in memory consumption, allowing for the processing of larger batch sizes for the same available memory, thereby offering improved throughput. In addition, we propose an efficient method for converting existing, non-reversible LLMs into reversible architectures through fine-tuning, rendering our approach practical for exploiting existing pre-trained models. Our results show comparable or improved performance on several datasets and benchmarks, on several LLMs, building a scalable and efficient path towards reducing the memory and computational costs associated with both training from scratch and fine-tuning of LLMs.

Key words. large language models; reversible architectures; memory-efficient training

MSC codes. 68T05, 65P10, 65Z05

1. Introduction. Large Language Models (LLMs) have achieved remarkable success in natural language processing, powering breakthroughs across diverse applications [6, 20]. However, their rapid scaling has introduced critical computational bottlenecks—chief among them is the excessive memory consumption during training and fine-tuning.

That limitation is common to most Transformer-based architectures, that require storing intermediate activations, to enable gradient computation via backpropagation. As a result, memory usage grows linearly with depth, significantly constraining the training of deep models and their deployment and training on resource-limited hardware. This limitation not only restricts model size, but also forces smaller batch sizes, which increases training time by requiring more iterations and communication to process the same dataset.

To address this limitation, we draw inspiration from time-reversible physical systems such as wave propagation, advection, and Hamiltonian flows [19, 13, 9]. In particular, we generalize and extend recent advances in reversible vision transformers [22, 17] to the language modeling domain. We alleviate the memory limitation by introducing a family of *reversible architectures* for LLMs based on hyperbolic differential equations, implemented through learned reversible dynamics.

A core advantage of our approach is that our architectures are *reversible by design*, allowing the backward pass to reconstruct intermediate activations without storing them during the forward pass. This drastically reduces memory requirements by an

*Department of Computer Science, University of British Columbia (eshedg@cs.ubc.ca).

†Department of Computer Science, Ben Gurion University (eliasof@post.bgu.ac.il).

‡EarthDynamics AI (javier@earthdynamics.ai).

§Department of Computer Science, University of British Columbia (ascher@cs.ubc.ca).

¶Department of Computer Science, Ben Gurion University (erant@bgu.ac.il).

||Department of Earth, Ocean and Atmospheric Sciences, University of British Columbia (ehaber@eoas.ubc.ca).

order of magnitude, enabling larger batch sizes and thereby significantly improving training throughput. Although reversible methods incur a modest increase in floating-point operations, we show that due to the increase in throughput, the memory savings compensate for the additional computations, reducing wall-clock training time under fixed computational budgets. In summary, training a reversible network results in a reduction in computational time. This idea is similar to “Activation Recomputation” [16], the difference is that our reversible network does not require storing *any activations*.

Another distinctive property of our architectures, which are based on hyperbolic PDEs, is their ability to *conserve energy over time*, in contrast to parabolic systems such as diffusion-based networks, which dissipate energy as depth increases [7, 11, 12]. By modeling token dynamics as conservative flows, our architectures preserve the fidelity of representations across depth, making them more scalable. This inductive bias supports stable long-range information propagation and enables the conceptual design of models with *infinite effective depth*.

Lastly, although reversible architectures offer several advantages, there remain many scenarios where fine-tuning a large non-reversible model for a specific task using limited resources is desirable. To address this, we propose a novel method for *retrofitting pre-trained non-reversible models into reversible ones*. By leveraging a structural correspondence between reversible dynamics and standard residual updates, we show that only minimal fine-tuning is required to enable reversible execution. This provides a practical approach to memory-efficient fine-tuning under resource constraints.

Our contributions are summarized as follows:

- We propose a new class of *reversible language model architectures* derived from hyperbolic differential equations, leveraging *conservative discretizations* that preserve information flow and guarantee reversibility. We further study their *theoretical properties*.
- We demonstrate that these models are *invertible by construction*, enabling memory-efficient training that supports larger batch sizes and enhances throughput.
- We present a mechanism for *retrofitting pre-trained non-reversible baseline models* into reversible architectures via fine-tuning compatible with existing LLMs.
- We present an empirical evaluation of our reversible models, including both training from scratch and retrofitting baseline models into reversible architectures. Our results demonstrate competitive or improved performance, highlighting not only the computational efficiency of reversible LLMs but also their effectiveness on downstream tasks.

2. Reversible Large Language Models. In this section, we introduce our reversible LLMs. We start with an overview of existing, non-reversible architectures, followed by a formal definition of reversible LLMs. We then discuss the memory footprint of our models, followed by a discussion of their expressiveness and downstream performance compared with standard architectures.

Common Network Architecture Overview. Modern language models are typically structured in three main stages: (i) an input embedding layer; (ii) a deep core of repeated Transformer layers; (iii) a final projection layer.

Given a tokenized input sequence $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T)$, $\mathbf{x}_i \in \mathbb{N}$, the model first maps each token to a high-dimensional embedding space $\mathbb{R}^d$, via a learned or pre-

defined embedding layer $E : \mathcal{V} \rightarrow \mathbb{R}^d$, where $\mathcal{V}$ is the vocabulary index set. This step yields an embedded sequence:

$$(2.1) \quad \mathbf{p}^{(0)} = E(\mathbf{x}) + P,$$

where $P \in \mathbb{R}^{T \times d}$ represents positional encodings added to the token embeddings. The embedded sequence $\mathbf{p}^{(0)} \in \mathbb{R}^{T \times d}$ is then processed by a stack of L layers composed of multi-head self-attention and feedforward multilayer-perceptrons (MLPs), organized in a residual architecture.

Each layer $\ell = 1, \dots, L$ updates the hidden representation via a two-step transformation:

$$(2.2) \quad \begin{aligned} \mathbf{q}^{(\ell)} &= \mathbf{p}^{(\ell-1)} + \text{Attn}_\ell \left(\text{LN}_1(\mathbf{p}^{(\ell-1)}) \right) \\ \mathbf{p}^{(\ell)} &= \mathbf{q}^{(\ell)} + \text{MLP}_\ell \left(\text{LN}_2(\mathbf{q}^{(\ell)}) \right), \end{aligned}$$

where Attn_ℓ is the multi-head self-attention module, MLP_ℓ is a feedforward network, and LN_1, LN_2 denote layer normalization applied before each sub-block. This structure forms a residual update scheme in which $\mathbf{p}^{(\ell)}$ is incrementally refined across layers by injecting nonlinear attention updates, while maintaining a residual path for stable gradient flow.

After L such updates, the final representation $\mathbf{p}^{(L)}$ is mapped back to the vocabulary logits using a final projection layer that reads:

$$(2.3) \quad \hat{\mathbf{x}} = \text{softmax} \left(\mathbf{W} \mathbf{p}^{(L)} \right),$$

where $\mathbf{W}$ are learned weights. This step yields a probability distribution over next tokens for autoregressive generation or language modeling objectives. While effective, this architecture demands storing all intermediate representations $(\mathbf{p}^{(0)}, \mathbf{q}^{(0)}), \dots, (\mathbf{p}^{(L)}, \mathbf{q}^{(L)})$ for backpropagation, and lacks a principled constraint on how representational energy is preserved or dissipated through the depth of the network — motivating our proposed reversible, energy-conserving LLMS.

2.1. Reversible Architecture. While reversible architectures have been explored in computer vision, their application to language modeling remains limited. Reformer [15] and PaReprop [23] introduced reversible models with Hamiltonian-like dynamics, building on ideas also studied in [13]. In this work, we leverage these foundations and introduce two novel reversible architectures, summarized in Figure 1. We also investigate conditions under which a reversible architecture is both forward and backward stable.

Midpoint Discretization. We begin by introducing a reversible architecture inspired by an explicit midpoint integrator commonly used for time-reversible numerical integration of first-order differential equations. In this formulation, the hidden state evolution is governed by a discretized update rule of the form:

$$(2.4) \quad \mathbf{p}^{(\ell+1)} = \mathbf{p}^{(\ell-1)} + 2h f_{\theta_\ell}(\mathbf{p}^{(\ell)}),$$

where $\mathbf{p}^{(\ell)} \in \mathbb{R}^{T \times d}$ denotes the hidden state at layer ℓ , $h > 0$ is a fixed step size, and f_{θ_ℓ} is a learnable update function applied to the current state. This formulation is reversible by construction: given $\mathbf{p}^{(\ell+1)}$ and $\mathbf{p}^{(\ell)}$, the previous state $\mathbf{p}^{(\ell-1)}$ can be exactly recovered via:

$$\mathbf{p}^{(\ell-1)} = \mathbf{p}^{(\ell+1)} - 2h f_{\theta_\ell}(\mathbf{p}^{(\ell)}).$$



Fig. 1: Reversible architectures: (a) explicit midpoint update and (b) leapfrog update.

The update function f_{θ_ℓ} encapsulates a full transformer block composed of attention and MLP submodules in (2.2). Specifically, we define:

$$(2.5) \quad f_{\theta_\ell}(\mathbf{p}) = \text{Attn}_\ell(\text{LN}_1(\mathbf{p})) + \text{MLP}_\ell(\text{LN}_2(\mathbf{p} + \text{Attn}_\ell(\text{LN}_1(\mathbf{p})))),$$

which mirrors the standard attention–MLP sequence in transformer blocks. In (2.4), this standard block is wrapped in a midpoint-style update that conserves information across layers. In this architecture, h is chosen as a hyper-parameter. Notably, this architecture maintains full reversibility across depth, enabling memory-efficient back-propagation without caching intermediate states, and provides a natural inductive bias for non-decaying representation flow.

Leapfrog Discretization.. While the midpoint-based architecture in (2.4) is both reversible and efficient, it can exhibit sensitivity to real positive eigenvalues in the Jacobian of f_{θ_ℓ} , potentially leading to exponential divergence or instability during training [4]. To improve stability while preserving reversibility and long-range information propagation, we propose an alternative architecture inspired by second-order hyperbolic partial differential equations—specifically, the nonlinear wave equation. Such an architecture has been studied in detail in [19, 8], though not in the context of large language models. In this formulation, the layerwise update is given by:

$$(2.6) \quad \mathbf{p}^{(\ell+1)} = 2\mathbf{p}^{(\ell)} - \mathbf{p}^{(\ell-1)} + h^2 f_{\theta_\ell}(\mathbf{p}^{(\ell)}),$$

which corresponds to a finite-difference discretization of the second-order system:

$$(2.7) \quad \ddot{\mathbf{p}} = f_\theta(\mathbf{p}),$$

where time is indexed by the discrete layer depth ℓ , and $h > 0$ denotes the step size. Like the leapfrog integrator, this update is fully reversible, as previous states can be exactly reconstructed by integrating the update backward.

The second-order formulation in (2.6) yields increased numerical stability and improved handling of oscillatory dynamics, especially when f_{θ_ℓ} has eigenstructure with nonzero real parts. Importantly, the update conserves a discrete analogue of energy over layers, akin to physical systems governed by Hamiltonian or wave dynamics. This makes it well-suited for LLMs tasked with propagating semantic information across long contexts without dissipation.

Hamiltonian Dynamics. We now introduce a third reversible architecture motivated by Hamiltonian dynamics, in which the evolution of the system is governed by coupled first-order differential equations describing the interaction between positions and momenta. A similar system was introduced in [13] and recently explored for vision transformers [17]. To model this structure, we maintain two coupled hidden states: $\mathbf{p}^{(\ell)} \in \mathbb{R}^{T \times d}$, interpreted as the "position", and $\mathbf{q}^{(\ell)} \in \mathbb{R}^{T \times d}$, interpreted as the "momentum". The model evolves these states through a staggered update scheme resembling the symplectic Euler integrator for Hamiltonian systems:

$$(2.8) \quad \mathbf{q}^{(\ell)} = a_{\ell-1} \mathbf{q}^{(\ell-1)} + \text{Attn}_{\ell} \left(\text{LN}_1(\mathbf{p}^{(\ell-1)}) \right)$$

$$(2.9) \quad \mathbf{p}^{(\ell)} = b_{\ell-1} \mathbf{p}^{(\ell-1)} + \text{MLP}_{\ell} \left(\text{LN}_2(\mathbf{q}^{(\ell)}) \right).$$

where we choose $a_{\ell} = b_{\ell} = 1$ (see next section). Each update alternates between applying a self-attention update to the position-like variable $\mathbf{q}$, and an MLP-based update to the momentum-like variable $\mathbf{p}$. This structure mirrors the canonical Hamiltonian equations $\dot{\mathbf{p}} = \nabla_{\mathbf{q}} H$, $\dot{\mathbf{q}} = -\nabla_{\mathbf{p}} H$, and conserves a discrete analogue of total energy, as well as flow volume conservation properties. Notably, the system is reversible, as each state can be recovered from its successor using the inverse of the update function.

This staggered formulation not only improves stability compared to direct residual updates, but also introduces a natural division of computational roles: attention governs the flow of global context through momentum-like variables, while MLPs serve as local content-based forces on the positional trajectory. The resulting dynamics support smooth, energy-preserving, volume-preserving propagation of information across depth, further enhancing the model’s capacity to maintain long-range context without diffusion.

2.2. Memory Footprint. Reversible architectures enable substantial memory savings during training by eliminating the need to store intermediate activations for gradient computation. In conventional networks such as transformers, each layer’s output must be retained during the forward pass to compute gradients, resulting in a memory footprint that scales linearly with depth. In contrast, our reversible networks reconstruct intermediate activations on-the-fly during backpropagation using invertible update rules (e.g., (2.4), (2.6), (2.8)), so only inputs and final outputs must be stored. This yields a constant activation memory cost, independent of model depth. We illustrate this behavior in Figure 3.

This memory efficiency comes at a moderate computational cost: during the backward pass, each layer’s forward function must be re-evaluated to reconstruct activations for gradient computation. However, this recomputation does not incur a two-fold increase in compute time. In practice, gradient computation, e.g., via automatic differentiation, is often significantly more expensive than the forward pass, so re-evaluating the layer typically adds only 30–50% overhead. The exact cost depends on layer complexity; for instance, MLPs and attention modules with high arithmetic intensity tend to have costly backward steps, making the marginal cost of recomputing f_{θ} relatively low.

Notably, on GPU hardware, reduced memory usage, even with increased FLOPs, can improve overall runtime, as runtime does not scale linearly with batch size. Algorithms that support larger batches may process more data per unit time, even if individual forward passes are slower. The exact performance gain depends on the architecture and memory footprint. We discuss this further in Section 5.

2.3. Reversibility Impact on Model Quality. A natural question arises when introducing structured, reversible dynamics in place of conventional dynamics: does modifying the architecture’s internal dynamics degrade the model’s expressivity or performance? While reversible updates constrain the form of information flow, our experiments reveal that these constraints do not hinder model quality, at least on the models that can be run with modest resources. In fact, we find that models trained with our energy-conserving reversible architectures not only match the performance of standard transformer-based language models, but often *outperform them*, as we show in Section 5.

This performance gain appears to stem from two sources. First, the structured flow of information, governed by time-reversible dynamics, encourages better global propagation of representations and reduced reliance on shortcut pathways [12]. Second, the conservation of representational energy across layers mitigates the vanishing or diffusion effects common in deep neural network stacks, enabling more stable gradient flow and richer hidden state evolution [7]. Hence, the architecture’s theoretical advantages in memory and stability translate into tangible improvements in generalization and task performance, even when no explicit memory constraints are present.

3. Analysis of Reversible Methods. In this Section, we analyze the stability of reversible network architectures, beginning with constant-coefficient formulations and extending to the practical case of variable coefficients encountered in deep learning.

3.1. Reversibility for Constant Coefficients. While reversible methods have been proposed for deep networks, it is important to note that reversible networks are typically only *marginally stable*, and this can lead to practical difficulties when attempting to train them. To this end, we first study a reversible method of the form

$$(3.1) \quad \mathbf{p}_{j+1} = a\mathbf{p}_{j-1} + b\mathbf{p}_j + hf(\mathbf{p}_j).$$

As commonly done to estimate stability [3], we study the test equation $f(\mathbf{p}) = \lambda\mathbf{p}$, where λ represents the Jacobian of the nonlinear transformation.

This type of equation is solved by assuming a solution of the form $\mathbf{p}_j = r^j$. Substituting in (3.1) we obtain:

$$(3.2) \quad r^2 - (b + h\lambda)r - a = 0.$$

This is a quadratic equation with two, possibly complex, roots. For the stability of the forward pass, we need both solutions to have absolute values equal at most 1. However, we look for stability of both forward and backward processes. To this end we seek solutions where $|r| = 1$, which implies that the solution is:

$$(3.3) \quad r_{12} = \exp(\pm i\theta).$$

Because $r_1 r_2 = a$ we obtain that $|a| = 1$ is a necessary condition for stability.

Substituting the solutions and $a = \pm 1$ in the equation and dividing by $\exp(i\theta)$ we obtain:

$$(3.4) \quad \exp(i\theta) \mp \exp(-i\theta) = \begin{cases} 2i \sin(\theta) \\ 2 \cos(\theta) \end{cases} = b + \lambda h.$$

This equation has two main implications. First, it yields a condition on b , namely:

$$(3.5) \quad |b + \lambda h| \leq 2.$$

Second, it yields a condition on λ . Note that, if λ is complex, then there is no solution to the system, implying that there is no forward-backward stable method for a complex λ . Furthermore, for $a = 1$, λ needs to be purely imaginary while for $a = -1$, λ is required to be purely real and negative. That is, the eigenvalues of the Jacobian should be either purely real (for $a = -1$) or purely imaginary (for $a = 1$) to obtain stability.

The Hamiltonian-type network in (2.8) was also considered as a reversible architecture. A similar analysis, presented below, shows that the product of coefficients $a_{\ell-1}$ and $b_{\ell-1}$ must equal 1 for forward and backward stability. While we focus on the midpoint method and experiment with the leapfrog network, we include the Hamiltonian formulation for *completeness*. Both the Hamiltonian and leapfrog updates represent *second-order dynamics* (i.e., involving second-order time derivatives as in (2.7)), whereas the standard skip connection and midpoint correspond to first-order dynamics. These dynamical systems differ significantly: first-order models describe the velocity of data, while second-order models represent its acceleration. While first-order approaches, such as ResNets [14], have proven highly effective, second-order systems remain less explored.

3.2. Reversibility for Variable Coefficients. In the previous subsection, we explored the reversibility of a constant coefficient method. While such an analysis is common for the analysis of dynamical systems, deep networks may not be fully conservative, indeed, the network may demand that some modes grow moderately while others shrink. We therefore propose estimating the behavior of a network of the form:

$$(3.6) \quad \mathbf{p}_{j+1} = a_{j-1}\mathbf{p}_{j-1} + (1 - a_{j-1})\mathbf{p}_j + hf_j(\mathbf{p}_j).$$

We call this method the midpoint (a) method, similar to the fractional θ methods commonly introduced [4]. Assume that we choose

$$(3.7) \quad a_j \sim [1 + \frac{1}{2}U(-1, 1), -1 + \frac{1}{2}U(-1, 1)].$$

In this case, we cannot consider the stability of a layer in a deterministic form, but rather in expectation. To analyze stability in expectation, we take the expectation and the variance of the recurrence relation. The expectation yields

$$(3.8) \quad \mathbb{E}\mathbf{p}_{j+1} = \mathbb{E}(a_{j-1}\mathbf{p}_{j-1}) + \mathbb{E}((1 - a_{j-1})\mathbf{p}_j) + h\mathbb{E}(\lambda_j\mathbf{p}_j).$$

Note that due to the linearity of the expectation and the choice of distribution of a_{j-1} having a zero mean, the first term vanishes and we obtain that

$$(3.9) \quad \mathbb{E}\mathbf{p}_{j+1} = \mathbb{E}(a_{j-1}\mathbf{p}_j) + (1 + h\lambda_j)\mathbb{E}\mathbf{p}_j = (1 + h\lambda_j)\mathbb{E}\mathbf{p}_j.$$

To compute the variance of a single step, we rearrange the recurrence relation to group terms involving a_{j-1} :

$$\mathbf{p}_{j+1} = a_{j-1}(\mathbf{p}_{j-1} - \mathbf{p}_j) + (1 + h\lambda_j)\mathbf{p}_j.$$

For the choice of a_{j-1} in (3.7), we have that $\text{Var}(a_{j-1}) = 1$, which implies that

$$(3.10) \quad \text{Var}(\mathbf{p}_{j+1}) = (\mathbf{p}_{j-1} - \mathbf{p}_j)^2.$$

Note that this *reversible* method behaves in expectation just like the forward Euler equation in terms of stability. As we see in Section 4, this can be used to convert a non-reversible method to a reversible one.

3.3. Stability of reversible Hamiltonian methods. A more general reversible method is a Hamiltonian-style method of the form

$$(3.11a) \quad \mathbf{p}_{j+1} = a\mathbf{p}_j + f(\mathbf{q}_j)$$

$$(3.11b) \quad \mathbf{q}_{j+1} = b\mathbf{q}_j + g(\mathbf{p}_{j+1})$$

To understand the stability property we study the linear case

$$(3.12a) \quad \mathbf{p}_{j+1} = a\mathbf{p}_j + \alpha\mathbf{q}_j$$

$$(3.12b) \quad \mathbf{q}_{j+1} = b\mathbf{q}_j + \beta\mathbf{p}_{j+1}$$

Rearranging we have that

$$(3.13) \quad \begin{aligned} \mathbf{p}_{j+1} &= a\mathbf{p}_j + \alpha\mathbf{q}_j \\ \mathbf{q}_{j+1} &= b\mathbf{q}_j + \beta(a\mathbf{p}_j + \alpha\mathbf{q}_j) = a\beta\mathbf{p}_j + (b + \alpha\beta)\mathbf{q}_j \end{aligned}$$

or

$$(3.14) \quad \begin{pmatrix} \mathbf{p}_{j+1} \\ \mathbf{q}_{j+1} \end{pmatrix} = \begin{pmatrix} a & \alpha \\ a\beta & b + \alpha\beta \end{pmatrix} \begin{pmatrix} \mathbf{p}_j \\ \mathbf{q}_j \end{pmatrix}$$

For forward-backward stability we need that $|\lambda_1| = |\lambda_2| = 1$ which implies that

$$(3.15) \quad |\det(\mathbf{A})| = |\lambda_1\lambda_2| = |ab| = 1$$

$$(3.16) \quad \text{trace}(\mathbf{A}) = |a + b + \alpha\beta| \leq 2$$

4. From Baseline to Reversible LLMs. While reversible models can be trained from scratch, it is often advantageous to leverage the weights of pre-trained models and convert them into reversible networks. In this section, we present a method to retrofit standard residual architectures into approximately reversible ones, which can optionally be refined through fine-tuning. We further analyze this approximation by introducing higher-order estimators and studying their accuracy.

4.1. Approximating Reversibility. Consider a residual network of the form:

$$(4.1) \quad \mathbf{p}_{j+1} = \mathbf{p}_j + f_j(\mathbf{p}_j),$$

where f_j denotes the j th layer with its own parameters. The update for the previous layer is:

$$(4.2) \quad \mathbf{p}_j = \mathbf{p}_{j-1} + f_{j-1}(\mathbf{p}_{j-1}).$$

Multiplying (4.2) by a_{j-1} and substituting $a_{j-1}\mathbf{p}_j$ into (4.1), we obtain:

$$(4.3) \quad \begin{aligned} \mathbf{p}_{j+1} &= a_{j-1}\mathbf{p}_{j-1} + (1 - a_{j-1})\mathbf{p}_j + \\ &\quad (f_j(\mathbf{p}_j) + a_{j-1}f_{j-1}(\mathbf{p}_{j-1})). \end{aligned}$$

This update recovers the residual structure in (4.1) but resembles the reversible form in (3.6). However, since the right-hand side depends on both $\mathbf{p}_j$ and $\mathbf{p}_{j-1}$, the network remains non-reversible.

To restore reversibility, we approximate $\mathbf{p}_{j-1}$ using a backward estimate from (4.2):

$$(4.4) \quad \mathbf{p}_{j-1} = \mathbf{p}_j - f_{j-1}(\mathbf{p}_{j-1}) \approx \mathbf{p}_j - f_{j-1}(\mathbf{p}_j).$$

Substituting this estimate into (4.3) yields the reversible update:

$$(4.5) \quad \begin{aligned} \hat{\mathbf{p}}_{j-1} &= \mathbf{p}_j - f_{j-1}(\mathbf{p}_j), \\ \mathbf{p}_{j+1} &= a_{j-1}\hat{\mathbf{p}}_{j-1} + (1 - a_{j-1})\mathbf{p}_j + \\ &\quad (f_j(\mathbf{p}_j) + a_{j-1}f_{j-1}(\hat{\mathbf{p}}_{j-1})). \end{aligned}$$

This approximation yields a fully reversible update that mimics the dynamics of the original residual network. A detailed analysis of the approximation error is provided in the appendix.

While (4.5) can be used as a zero-shot approximation, it may also be refined via fine-tuning. Let $\mathbf{Y}_{\text{fe}} = [\mathbf{y}_1, \dots, \mathbf{y}_N]$ denote the output probabilities from the original residual model in (4.1), and let $\mathbf{Y}_{\text{rev}}$ be those from the reversible formulation in (4.5). We fine-tune the model by solving:

$$(4.6) \quad \min \mathbb{E} [KL(\mathbf{Y}_{\text{rev}}, \mathbf{Y}_{\text{fe}})],$$

where the optimization is over the parameters generating $\mathbf{Y}_{\text{rev}}$, while keeping those of $\mathbf{Y}_{\text{fe}}$ fixed.

4.2. Higher Order Estimators of $\hat{\mathbf{p}}_{j-1}$ and their analysis. The quality of the approximation of our network in Equation (4.5) to the original network in Equation (4.1) relies on the quality of the approximation to $\mathbf{p}_{j-1}$. In the above, we have used a single-step approximation. A more general idea is to use a fixed-point iteration of the form

$$(4.7) \quad \mathbf{p}_{j-1}^{(k+1)} = \mathbf{p}_j - f_{j-1}(\mathbf{p}_{j-1}^{(k)})$$

initializing $\mathbf{p}_{j-1}^{(0)} = \mathbf{p}_j$. The first iteration yields our previous update; however, using more iterations, we obtain a more accurate estimate of the true $\mathbf{p}_{j-1}$, which implies a better reversible approximation of the original system.

4.2.1. Linear analysis. Above, we have discussed a methodology that approximates a non-reversible architecture with a reversible one. To understand the approximation we analyze the simple linear case. We consider a network of the form

$$(4.8) \quad \mathbf{p}_{j+1} = a_{j-1}\mathbf{p}_{j-1} + (1 - a_{j-1})\mathbf{p}_j + \mathbf{A}_j\mathbf{p}_j + a_j\mathbf{A}_{j-1}\mathbf{p}_{j-1}$$

This network corresponds to the standard single recurrence network. The corresponding approximate architecture reads

$$(4.9) \quad \begin{aligned} \hat{\mathbf{p}}_{j+1} &= a_{j-1}\mathbf{p}_{j-1} + (1 - a_{j-1})\mathbf{p}_j + \mathbf{A}_j\mathbf{p}_j + \\ &\quad a_{j-1}\mathbf{A}_{j-1}(\mathbf{p}_j - \mathbf{A}_{j-1}\mathbf{p}_j) \end{aligned}$$

Subtracting (4.8) from (4.9) we obtain that

$$(4.10) \quad \begin{aligned} \hat{\mathbf{p}}_{j+1} - \mathbf{p}_{j+1} &= \mathbf{A}_{j-1}\mathbf{p}_{j-1} - a_{j-1}(\mathbf{A}_{j-1} - \mathbf{A}_{j-1}^2)\mathbf{p}_j \\ &= \mathbf{A}_{j-1}\mathbf{p}_{j-1} - a_{j-1}(\mathbf{A}_{j-1} - \mathbf{A}_{j-1}^2)(\mathbf{I} + \mathbf{A}_{j-1})\mathbf{p}_{j-1} \\ &= \mathbf{A}_{j-1}(\mathbf{I} - a_{j-1}(\mathbf{I} - \mathbf{A}_{j-1}^2))\mathbf{p}_{j-1} \end{aligned}$$

Thus we obtain that

$$(4.11) \quad \begin{aligned} \|\hat{\mathbf{p}}_{j+1} - \mathbf{p}_{j+1}\|^2 &\leq \\ &\|\mathbf{A}_{j-1}\|^2 \|(1 - a_{j-1})\mathbf{I} - a_{j-1}\mathbf{A}_{j-1}^2\|^2 \|\mathbf{p}_{j-1}\|^2 \end{aligned}$$

Network	GPT Small (124M)		GPT Large (772M)	
	Training	Validation	Training	Validation
Baseline	2.8810	2.9022	2.6544	2.6988
Midpoint	2.8681	2.9148	2.5243	2.6183
Leapfrog	2.9245	2.9432	2.5750	2.6091

Table 1: Training and validation cross-entropy loss($\downarrow$) of baseline and our reversible Midpoint and Leapfrog architectures.

Benchmark	GPT-2 Small (124M)			GPT-2 Large (772M)		
	Baseline	Midpoint	Leapfrog	Baseline	Midpoint	Leapfrog
piqa	51.14%	53.05%	52.83%	53.16%	52.72%	55.11%
ARC-e	27.02%	26.14%	28.95%	23.86%	26.84%	26.49%
ARC-c	22.74%	22.73%	24.08%	22.41%	28.09%	26.09%
Obqa	24.20%	24.40%	24.20%	24.00%	23.80%	19.80%
WinoGrande	49.57%	51.70%	51.85%	50.28%	50.20%	48.30%

Table 2: Zero-shot accuracy (% , $\uparrow$) for GPT-2 models under baseline, midpoint, and leapfrog architectures.

This approximation can be very good when the norm of $\mathbf{A}_{j-1}$ is small. The approximation falls apart when $\|\mathbf{A}_{j-1}\| > 1$.

Residual networks often exhibit a behavior where the norm of the update is small and therefore, our proposed approximation works well for many networks in practice.

5. Numerical Experiments. Demonstrating that a new LLM architecture outperforms state-of-the-art models typically requires substantial computational and data resources. Here, we use modest resources to explore the following two research questions:

1. Given the same number of parameters and compute budget, can a reversible architecture achieve comparable performance while using significantly less memory?
2. Given a pre-trained model, can it be converted into a reversible one using only fine-tuning, and how does the resulting model perform on standard benchmarks?

We address these research questions through experiments and evaluations on small and large GPT-2 [18] architectures, as well as TinyLlama [21] and SmolLM2 [1], both of which contain over a billion parameters.

5.1. Training a Reversible Model. We train two reversible architectures: Midpoint (3.6) and Leapfrog (2.6), on the OpenWebText dataset, and evaluate their performance against the standard residual skip-connection denoted Baseline, using both GPT-2 Small and GPT-2 Large (740M parameters). We primarily focus on the Midpoint with θ formulation, as it generalizes the Midpoint method from (2.4). An ablation with the Midpoint method is provided in the Appendix. All models are trained using identical hyperparameters and procedures to ensure a fair comparison. Results for both model sizes are shown in Figure 2 and summarized in Table 1. As

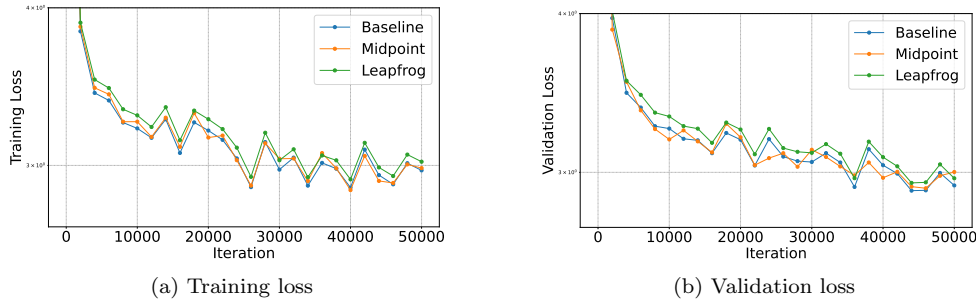


Fig. 2: Training and validation loss curves of GPT-2 using baseline, midpoint, and leapfrog architectures.

seen in the training curves and final validation metrics, the reversible architectures behave similarly to their non-reversible counterparts during training and exhibit slightly better validation loss. This demonstrates that reversible models not only are memory efficient, but also match or outperform standard residual models, without added training complexity.

Evaluation.. To evaluate the generalization performance of our reversible Midpoint and Leapfrog models beyond the training distribution, we conduct zero-shot evaluation on five downstream benchmarks: piqa, ARC-easy, ARC-challenge, Obqa, and WinoGrande. These experiments are performed using models trained on the OpenWebText dataset, as described in Section 4. Results are reported in Table 2. Across tasks, the reversible architectures perform on par with their baseline, standard residual counterparts and, in several cases, outperform them. These findings suggest that reversible models not only provide substantial memory savings during training but also maintain competitive generalization performance without added training complexity.

Memory Consumption.. While reversible architectures may yield modest improvements in accuracy, their key advantage lies in significantly reduced memory requirements during training. Table 3 reports the maximum batch size that fits in GPU memory for both baseline and reversible (Midpoint) models across a range of GPU architectures. The reversible architecture enables batch sizes nearly an order of magnitude larger than the baseline, specifically around $10\times$ on all tested hardware. This highlights its practical benefit for memory-constrained training settings, enabling more efficient hardware utilization with no degradation in model quality. To further illustrate this advantage, Figure 3 shows memory usage as a function of model depth. While the memory footprint of standard residual networks grows linearly with depth, that of reversible models remains constant.

In Table 4, we report the throughput improvements achieved using reversible architectures. For a model with a hidden dimension of 512 and 8 attention heads, we observe that the benefit of our reversible updates increases with depth. In particular, deeper models support substantially larger batch sizes, leading to significant throughput gains, reaching up to 101% improvement for the 96-layer model, and thereby enabling faster and more memory-efficient training.

5.2. Converting Baseline to Reversible LLMs. While training a reversible model from scratch offers substantial advantages, retrofitting a pre-trained non-

Method	GPU Memory	RTX6000 24GB	A10 24GB	A100 40GB	A6000 48GB	H100 80GB
Baseline	Max batch	6	6	12	15	26
Midpoint	Max batch	58	58	119	140	257
	Enhancement	9.66	9.66	9.91	9.33	9.88

Table 3: Maximum batch size that fits in memory for baseline and reversible (Midpoint) networks across various GPU architectures. Enhancement indicates the relative improvement in batch size made possible by the reversible formulation.

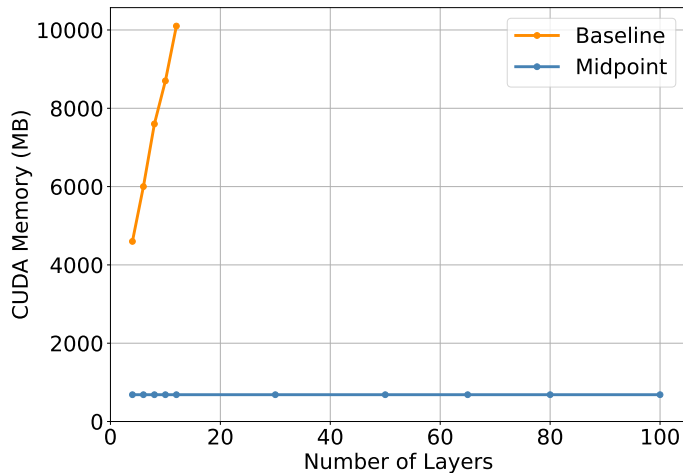


Fig. 3: Training GPU memory usage vs. network depth for baseline and reversible (Midpoint) models. The baseline memory grows linearly and fails beyond 12 layers, while the reversible model remains constant.

reversible model can be equally valuable. To demonstrate this, we convert TinyLlama-v1.0 using the reversible dynamics in (4.5). We align the output distributions of the original and converted models on the WikiText dataset via the objective in (4.6). The conversion uses only 80,000 examples and yields a close match on the validation set. As shown in Figure 4, the reversible model closely follows the training and validation loss of the original in next-token prediction. These results confirm that the conversion preserves model behavior and quality while enabling reversibility.

Since the reversible model is designed to approximate the behavior of its non-reversible counterpart, we expect its performance to closely match, but not necessarily exceed, that of the original model. To evaluate the generalization capability of the converted reversible network, we assess its performance on a suite of zero-shot commonsense benchmarks. Results are reported in Table 5. *Additionally*, we include results for a converted SmoLM2 model with 1.7B parameters on the MMLU benchmark, provided in the Appendix. This model was retrofitted using two epochs of WikiText data and achieves an accuracy of 49%, closely matching the original model’s 50.36%. Overall, the converted and original models exhibit highly similar per-

Method		$L = 16$	$L = 32$	$L = 64$	$L = 96$
Baseline	Batch Size	326	176	88	52
	Time (s)	0.667	0.669	0.774	0.913
	Throughput	488.8	263.1	113.7	56.96
Midpoint	Batch Size	1550	1374	1076	874
	Time (s)	2.627	4.207	6.362	7.634
	Throughput	590.0	326.6	169.1	114.49
Throughput Gain (%)		20.7%	24.1%	48.8%	101.0%

Table 4: Maximum batch size, per-step training time (s), and resulting throughput (samples/s) for different model depths, as a function of number of layers L .

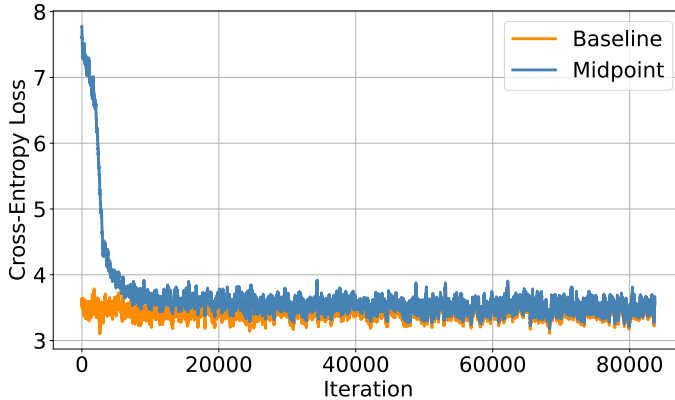


Fig. 4: Next token prediction cross-entropy loss during the conversion of TinyLlama-v1.0 to a reversible Midpoint architecture. The reversible model (Midpoint) closely matches the baseline in next-token prediction, demonstrating successful functional alignment.

formance. These results demonstrate that reversible architectures can be effectively obtained from pre-trained models and are well-suited for fine-tuning with minimal overhead.

5.3. Ablation Study. We conduct an ablation study comparing the midpoint methods described in (2.4) and (3.6), and show our results in Table 6. While the results are often comparable, the method introduced in (3.6) may demonstrate potential for significant improvement.

5.4. Experimental Details. For training the GPT-2 models, we utilized 8 NVIDIA L40S GPUs. Model conversion was performed on an RTX6000 GPU. We adopted standard hyperparameters following the configurations provided in <https://github.com/karpathy/nanoGPT>. During conversion, we used a learning rate of 10^{-5} .

5.5. Additional Results: MMLU. The MMLU (Massive Multitask Language Understanding) dataset is a comprehensive benchmark for assessing language models’

Benchmark	Baseline	Baseline $\rightarrow$ Midpoint
piqa	72.09%	70.24%
ARC-e	50.88%	50.35%
ARC-c	27.76%	26.75%
Obqa	38.20%	35.60%
WinoGrande	50.36%	51.62%

Table 5: Zero-shot accuracy (% , $\uparrow$) on commonsense reasoning benchmarks. We compare the original TinyLlama-v1.0 model to its reversible counterpart obtained via conversion with the Midpoint method. The reversible model maintains comparable performance across all tasks.

Benchmark	GPT-2 Small (123.59M)		
	Midpoint ((2.4))	Midpoint ((3.6))	Baseline
piqa	50.98%	53.05%	51.14%
ARC-e	26.49%	26.14%	27.02%
ARC-c	23.08%	22.73%	22.74%
Obqa	20.60%	24.40%	24.20%
WinoGrande	50.75%	51.70%	49.57%

Table 6: Zero-Shot Accuracy (%) on Various Benchmarks for GPT-2 Small Model.

reasoning and domain knowledge across a wide range of tasks. Our results, shown in Table 7 demonstrate that the proposed methods achieve performance comparable to, and in some cases exceeding, that of standard approaches.

Table 7: Full MMLU subject-level accuracy for SmolLM2 1.7B before and after reversible conversion.

Subject	Samples	Original Acc	Original Std. Error	Converted Acc	Converted Std. Error
all	14042	0.5036	0.0042	0.4900	0.0042
abstract algebra	100	0.2900	0.0454	0.3100	0.0462
anatomy	135	0.4370	0.0427	0.4148	0.0424
astronomy	152	0.6118	0.0395	0.5987	0.0398
business ethics	100	0.6000	0.0490	0.5600	0.0496
clinical knowledge	265	0.5660	0.0304	0.5509	0.0306
college biology	144	0.5556	0.0414	0.5208	0.0416
college chemistry	100	0.3800	0.0485	0.3500	0.0477
college computer science	100	0.4500	0.0497	0.4300	0.0495
college mathematics	100	0.3400	0.0474	0.3800	0.0485
college medicine	173	0.4971	0.0380	0.4509	0.0378
college physics	102	0.3431	0.0470	0.3431	0.0470

Subject	Samples	Original Acc	Original Std. Error	Converted Acc	Converted Std. Error
computer security	100	0.6200	0.0485	0.6200	0.0485
conceptual	235	0.4809	0.0326	0.4638	0.0325
physics					
econometrics	114	0.3596	0.0449	0.3947	0.0458
electrical	145	0.5241	0.0415	0.5379	0.0414
engineering					
elementary	378	0.3783	0.0249	0.3889	0.0251
mathematics					
formal logic	126	0.3254	0.0417	0.3175	0.0415
global facts	100	0.3100	0.0462	0.3000	0.0458
high school	310	0.5323	0.0283	0.5516	0.0282
biology					
high school	203	0.4631	0.0350	0.4286	0.0347
chemistry					
high school	100	0.5000	0.0500	0.4600	0.0498
computer science					
high school	165	0.5636	0.0386	0.5636	0.0386
european history					
high school	198	0.6162	0.0346	0.5909	0.0349
geography					
high school	193	0.7254	0.0321	0.7098	0.0327
government and politics					
high school	390	0.4923	0.0253	0.4615	0.0252
macroeconomics					
high school	270	0.3296	0.0286	0.3222	0.0284
mathematics					
high school	238	0.5042	0.0324	0.4622	0.0323
microeconomics					
high school	151	0.3576	0.0390	0.3444	0.0387
physics					
high school	545	0.6936	0.0197	0.6514	0.0204
psychology					
high school	216	0.4074	0.0334	0.3611	0.0327
statistics					
high school US	204	0.5588	0.0348	0.5637	0.0347
history					
high school world	237	0.6076	0.0317	0.5823	0.0320
history					
human aging	223	0.6054	0.0327	0.6009	0.0328
human sexuality	131	0.6031	0.0427	0.6031	0.0427
international law	121	0.6529	0.0433	0.6529	0.0433
jurisprudence	108	0.5833	0.0474	0.5833	0.0474
logical fallacies	163	0.6503	0.0374	0.6380	0.0376
machine learning	112	0.3482	0.0450	0.3214	0.0441
management	103	0.7087	0.0448	0.6893	0.0456
marketing	234	0.7222	0.0293	0.7222	0.0293
medical genetics	100	0.5900	0.0492	0.5700	0.0495
miscellaneous	783	0.6986	0.0164	0.6858	0.0166
moral disputes	346	0.5520	0.0267	0.5376	0.0268
moral scenarios	895	0.2380	0.0142	0.2369	0.0142
nutrition	306	0.5523	0.0284	0.5523	0.0284

Subject	Samples	Original Acc	Original Std. Error	Converted Acc	Converted Std. Error
philosophy	311	0.6141	0.0276	0.6431	0.0272
prehistory	324	0.5957	0.0273	0.5895	0.0273
professional	282	0.3369	0.0281	0.3262	0.0279
accounting					
professional law	1534	0.3683	0.0123	0.3449	0.0121
professional	272	0.4118	0.0298	0.3934	0.0296
medicine					
professional	612	0.5245	0.0202	0.4951	0.0202
psychology					
public relations	110	0.5091	0.0477	0.5636	0.0473
security studies	245	0.5878	0.0314	0.5755	0.0316
sociology	201	0.6667	0.0333	0.6318	0.0340
US foreign policy	100	0.7500	0.0433	0.7700	0.0421
virology	166	0.4880	0.0388	0.4639	0.0387
world religions	171	0.7193	0.0344	0.6901	0.0354

6. Summary and Outlook. We presented a new class of *reversible large language models* derived from first and second order dynamics. Conventional transformers must cache every intermediate activation for back-propagation, creating a severe memory bottleneck. Our Leapfrog, Midpoint, and Hamiltonian updates are *reversible by construction*; hidden states are reconstructed during the backward pass instead of stored, yielding up to a $10\times$ reduction in activation memory and enabling much larger batch sizes.

Numerical PDE Methods in the Machine Learning Context. It is important to keep in mind that our employment of numerical methods for PDEs does not mean that we are interested in numerically solving PDE systems in this paper. Specifically, numerical methods that may fail in the PDE context (especially when a highly accurate numerical solution is required) may not necessarily be inadequate in our present context. Noise can have a therapeutic effect!

The clearest example for this effect is the explicit midpoint method 2.4. As a PDE method it is not even compact, as it spans over two mesh intervals in “time” while related to a 1st order PDE. Thus, fast waves of the form

$$p^\ell = \begin{cases} m, & \text{if } \ell \text{ is odd,} \\ n, & \text{if } \ell \text{ is even.} \end{cases}$$

will produce zero in the left hand side of 2.4 for any real values m and n . However, in the ML domain (unlike for predicting a tsunami) this method can be successful, which is a side effect of dealing with many noisy observations.

In contrast, the methods presented in 2.6 and 2.8, which approximate the 2nd order PDE (2.7), are compact. And yet, they too could suffer from a catastrophic accumulation of roundoff errors when applied in the traditional numerical mathematics context of approximating an exact solution for a nonlinear PDE system over a long time interval; see for instance [10, 5, 2]. However, again in the present context, delicate usage of symmetric methods, which occasionally gives the nod to symplectic methods in the context of accurately solving models such as Korteweg de Vries and nonlinear Schrodinger, does not buy much leverage here.

Theoretical and empirical contributions. We provide a stability analysis that pinpoints conditions for forward- and backward-stable training. On GPT-2

(124M/772M), TinyLlama (1.1B), and SmolLM2 (1.7B), R-LLMs match or slightly surpass the perplexity and zero-shot accuracy of non-reversible baselines on PIQA, ARC-easy/-challenge, OBQA, WinoGrande, and MMLU, demonstrating that memory efficiency need not compromise quality. In addition, we introduce a lightweight fine-tuning procedure that converts existing checkpoints into fully reversible models using only two epochs of auxiliary data, making our approach immediately applicable across today’s LLM ecosystem.

Future directions. We identify the following research paths:

1. **Frontier models.** Extend reversible updates and retrofit to hundred-billion-parameter foundation models.
2. **Richer conservative dynamics.** Explore variational, symplectic, or other Hamiltonian integrators as inductive biases.
3. **Long-context training.** Exploit constant-memory back-propagation to handle document-scale or code-base sequences.
4. **Faster retrofit objectives.** Design loss functions and parameter-freezing schemes that further cut conversion cost.

By decoupling memory from depth, reversible architectures offer a practical, scalable path toward the next generation of efficient and extensible language models.

REFERENCES

- [1] L. B. ALLAL, A. LOZHKOVA, E. BAKOUCH, G. M. BLÁZQUEZ, G. PENEDO, L. TUNSTALL, A. MARAFIOTI, H. KYDLÍČEK, A. P. LAJARÍN, V. SRIVASTAV, J. LOCHNER, C. FAHLGREN, X.-S. NGUYEN, C. FOURRIER, B. BURTENSHAW, H. LARCHER, H. ZHAO, C. ZAKKA, M. MORLON, C. RAFFEL, L. VON WERRA, AND T. WOLF, *Smollm2: When smol goes big – data-centric training of a small language model*, 2025, <https://arxiv.org/abs/2502.02737>, <https://arxiv.org/abs/2502.02737>.
- [2] U. ASCHER AND R. I. McLACHLAN, *On symplectic and multisymplectic schemes for the KdV equation*, J. Sci. Computing, 25 (2005), pp. 83–104.
- [3] U. ASCHER AND L. PETZOLD, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*, SIAM, Philadelphia, PA, 1998.
- [4] U. M. ASCHER, *Numerical methods for evolutionary differential equations*, SIAM, 2008.
- [5] T. BRIDGES AND F. LAINE-PEARSON, *Multi-symplectic relative equilibria, multi-phase wave-trains and coupled NLS equations*, Studies in Applied Math.
- [6] T. BROWN, B. MANN, N. RYDER, M. SUBBIAH, J. KAPLAN, P. DHARIWAL, A. NEELAKANTAN, P. SHYAM, G. SASTRY, A. ASKELL, ET AL., *Language models are few-shot learners*, Advances in neural information processing systems, 33 (2020), pp. 1877–1901.
- [7] B. CHANG, M. CHEN, E. HABER, AND E. H. CHI, *Antisymmetricrnn: A dynamical system view on recurrent neural networks*, 2019, <https://arxiv.org/abs/1902.09689>, <https://arxiv.org/abs/1902.09689>.
- [8] M. ELIASOF, E. HABER, AND E. TREISTER, *PDE-GCN: Novel architectures for graph neural networks motivated by partial differential equations*, Advances in Neural Information Processing Systems, 34 (2021), pp. 3836–3849.
- [9] M. ELIASOF AND E. TREISTER, *DiffgcN: Graph convolutional networks via differential operators and algebraic multigrid pooling*, 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada., (2020).
- [10] E. FAOU, *Geometric Numerical Integration and Schrödinger Equations*, European Mathematical Society, 2012. Zurich Lectures in Advanced Mathematics.
- [11] A. GRAVINA, D. BACCIU, AND C. GALLICCHIO, *Anti-symmetric dgn: a stable architecture for deep graph networks*, arXiv preprint arXiv:2210.09789, (2022).
- [12] A. GRAVINA, M. ELIASOF, C. GALLICCHIO, D. BACCIU, AND C.-B. SCHÖNLIEB, *On oversquashing in graph neural networks through the lens of dynamical systems*, in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 39, 2025, pp. 16906–16914.
- [13] E. HABER AND L. RUTHOTTO, *Stable architectures for deep neural networks*, Inverse Problems, 34 (2017).
- [14] K. HE, X. ZHANG, S. REN, AND J. SUN, *Deep residual learning for image recognition*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
- [15] N. KITAEV, L. KAISER, AND A. LEVSKAYA, *Reformer: The efficient transformer*, in International Conference on Learning Representations (ICLR), 2020, <https://openreview.net/forum?id=rkgNKkHtvB>.
- [16] V. A. KORTHIKANTI, J. CASPER, S. LYM, L. MCAFEE, M. ANDERSCH, M. SHOEYBI, AND B. CATANZARO, *Reducing activation recomputation in large transformer models*, in MLSys, 2023, https://proceedings.mlsys.org/paper_files/paper/2023/hash/80083951326cf5b35e5100260d64ed81-Abstract-mlsys2023.html.
- [17] K. MANGALAM, H. FAN, Y. LI, C.-Y. WU, B. XIONG, C. FEICHTENHOFER, AND J. MALIK, *Reversible vision transformers*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 10830–10840.
- [18] A. RADFORD, J. WU, R. CHILD, D. LUAN, D. AMODEI, I. SUTSKEVER, ET AL., *Language models are unsupervised multitask learners*, OpenAI blog, 1 (2019), p. 9.
- [19] L. RUTHOTTO AND E. HABER, *Deep neural networks motivated by partial differential equations*, Journal of Mathematical Imaging and Vision, 62 (2020), pp. 352–364.
- [20] H. TOUVRON, T. LAVRIL, G. IZACARD, X. MARTINET, M.-A. LACHAUX, T. LACROIX, B. ROZIÈRE, N. GOYAL, E. HAMBRO, F. AZHAR, ET AL., *Llama: Open and efficient foundation language models*, arXiv preprint arXiv:2302.13971, (2023).
- [21] P. ZHANG, G. ZENG, T. WANG, AND W. LU, *Tinyllama: An open-source small language model*, 2024, <https://arxiv.org/abs/2401.02385>.
- [22] C. ZHAO, S. LIU, K. MANGALAM, G. QIAN, F. ZOHRA, A. ALGHANNAM, J. MALIK, AND B. GHANEM, *Dr2net: Dynamic reversible dual-residual networks for memory-efficient fine-tuning*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2024, pp. 15835–15844.

- [23] T. ZHU AND K. MANGALAM, *Pareprop: Fast parallelized reversible backpropagation*, in T4V Workshop at the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR-W), 2023.