Latent Flow Transformer

Yen-Chen Wu* **Feng-Ting Liao*** **Meng-Hsi Chen**
Pei-Chen Ho **Farhang Nabiei** **Da-shan Shiu**

MediaTek Research†

Abstract

Transformers, the standard implementation for large language models (LLMs), typically consist of tens to hundreds of discrete layers. While more layers can lead to better performance, this approach has been challenged as far from efficient, especially given the superiority of continuous layers demonstrated by diffusion and flow-based models for image generation. We propose the Latent Flow Transformer (LFT), which replaces a block of layers with a single learned transport operator trained via flow matching, offering significant compression while maintaining compatibility with the original architecture. Additionally, we address the limitations of existing flow-based methods in *preserving coupling* by introducing the Flow Walking (FW) algorithm. On the Pythia-410M model, LFT trained with flow matching compresses 6 of 24 layers and outperforms directly skipping 2 layers (KL Divergence of LM logits at 0.407 vs. 0.529), demonstrating the feasibility of this design. When trained with FW, LFT further distills 12 layers into one while reducing the KL to 0.736 surpassing that from skipping 3 layers (0.932), significantly narrowing the gap between autoregressive and flow-based generation paradigms.

1 Introduction

Transformer-based large language models (LLMs) have become foundational to progress in NLP. While their scale enables remarkable capabilities, it also introduces substantial memory and compute demands, limiting their accessibility. Although evidence suggests many layers are overparameterized [41], mainstream compression techniques like pruning and knowledge distillation [26, 40] have yet to deliver the scale of reduction required to fully address these limitations.

In image generation, diffusion and flow matching (FM) models [24] have demonstrated remarkable efficiency, with recent approaches generating high-quality images in as few as one sampling pass [6]. Inspired by this success, several works have explored adapting FM and diffusion models for language modeling [36, 40]. While these approaches offer promising directions, their performance still trails behind SOTA conventional transformer models—likely due to the discrete and compositional nature of language, where token-level generation resists continuous approximation.

Recent studies frame transformer forward passes as discretized continuous dynamical processes [2, 19, 5], and show that latent trajectories across layers can be nonlinear and complex. Concurrently, in image modeling, it has been shown that efficient surrogate models can be trained to approximate the latent mapping of deep networks using straighter transport dynamics [4, 35]. These developments motivate exploring whether flow-based approaches can learn more efficient inter-layer mappings in transformers, offering a new path to structural compression.

Building upon these insights, We propose the **Latent Flow Transformer (LFT)**, a novel transformer variant designed for parameter-and-compute efficient language modeling by replacing a contiguous

*These authors contributed equally.

†Correspondence: yen-chen.wu@mtkresearch.com, ft.liao@mtkresearch.com

Preprint. Under review.

block of transformer layers with a single learned transport operator trained via flow matching [24]. To guide which layers are suitable for replacement, we introduce the *Recoupling Ratio*, an interpretable Optimal Transport-based metric that accurately predicts flow matching quality. Our analysis finds that middle layers are particularly compressible, supporting the intuition that early LLM layers function qualitatively differently from later ones.

In our experiments training the LFT, with only 10^8 training tokens, we show that standard flow matching enables effective LFT training, with perturbations smaller than those caused by removing a single transformer layer [33]. However, performance plateaus as flow path proximity introduces velocity ambiguity [16, 30]—a known limitation in FM models. While reflow techniques are a common strategy to mitigate this issue, they are inherently incompatible with the LFT's objective of preserving the original input-output latent relationship.

To address this limitation, while advanced flow matching techniques offer potential solutions, we introduce **Flow Walking (FW)**, a scalable algorithm for learning latent transport using numerical integration. By training a transformer-like policy to explore paths with clearer view to the target, FW enables the model to navigate and resolve local ambiguities arising from multiple nearby flow paths, thereby significantly improving transport *alignment* and overcoming the performance plateau observed with standard flow matching.

We conduct extensive experiments on the *Pythia-410M* model to validate the proposed framework. First, we find that layer selection is critical: LFTs constructed by replacing the layers identified by the *Recoupling Ratio* consistently outperform those using arbitrary layer selections. Second, we demonstrate that LFT trained via flow matching can compress 12 out of 24 layers into a single transport layer while achieving a lower performance drop (KL divergence of LM logits at 0.407) than simply skipping two layers (KL divergence of LM logits at 0.529), thereby establishing the feasibility of our architecture. Finally, when trained with FW under the same compression setting, the LFT achieves a substantially lower KL divergence of LM logits at 0.736 than that with skipping three layers at 0.932, overcoming the primary obstacle in aligning latent transport across distant transformer layers and effectively bridging autoregressive and flow-based modeling paradigms.

In summary, our contributions are as follows.

- We introduce the Latent Flow Transformer, a novel architecture for efficient language modeling that replaces a block of transformer layers with a single learned transformer-like layer based on flow-matching principles.
- We develop an interpretable predictor *Recoupling Ratio* that estimates the suitability of different transformer layers for replacement. Our analysis with the predictor suggests that the middle layers of transformer models are particularly amenable to this form of compression, supporting hypotheses about the differentiated roles of layers in LLMs.
- We demonstrate that while standard flow matching can learn the LFT's transport operator with a modest data budget, it encounters a performance plateau stemming from challenges such as flow path proximity in the latent space.
- We propose Flow Walking, a scalable explorative algorithm for learning latent transport trajectories, designed to effectively navigate and resolve flow path proximity issues.
- We empirically validate the LFT trained with both FM and FW on the *Pythia*-410M model, demonstrating substantial parameter reduction with limited initial performance degradation and significant capability recovery after fine-tuning.

We release our code for the community to reproduce our results at `https://github.com/mtkresearch/latent-flow-transformer`

2 Preliminary

2.1 Continuous-Time perspective of transformer layers

A neural net with L discrete layers, especially with a residual connection, can be seen as refining its input hidden state h_{l-1} at layer l, $l = 1, 2, ..., L$, to a better state at its output, h_l. A continuous-time counterpart can be expressed by the following dynamics [2, 3, 29]:

$$v_t = \frac{dh_t}{dt} = u_\theta(h_t, t) \tag{1}$$

where θ parameterizes the ordinary differential equation (ODE), and h_t matches with h_l whenever t matches l/L. Learning θ involves conducting expensive simulation [14]:

$$h_{t_2} = h_{t_1} + \int_{t_1}^{t_2} u_\theta(h_t, t)\, dt \tag{2}$$

2.2 Flow Matching for Simulation-Free Training

Flow-based approaches [1, 24, 25] provide a simulation-free solution to the ODE defined in Equation 1. Flow matching moves a particle at initial position x_0 to its target position x_1 via a predefined path in unit time, and learns the corresponding velocity field [24] conditioned on a set of (x_0, x_1) pairs by minimizing the following loss,

$$\mathcal{L}_{FlowMatching} = \mathbb{E}_t \left[\|u_\theta(x_t, t) - v_t\|^2 \right], \tag{3}$$

where x_t and v_t are the position and the velocity of the particle at time t, respectively.

A straight line, constant speed trajectory is a popular and practical choice for flow matching. With such a target trajectory, x_t and v_t are simply $x_t = (1-t) \cdot x_0 + t \cdot x_1$, and $v_t = x_1 - x_0$.

During discrete-time inference, datapoints are transported along the flow in discrete steps. When moving from time points t to $t + d$, a step is:

$$x_{t+d} = x_t + d \cdot u_\theta(x_t, t). \tag{4}$$

To enhance the stability and the accuracy of the flow trajectory, an alternative is to set the velocity at the midpoint of the currently estimated forward step:

$$x_{t+d} = x_t + d \cdot u_\theta(x_t + \frac{d}{2} \cdot u_\theta(x_t, t), t + \frac{d}{2}). \tag{5}$$

2.3 Challenges in Flow Matching for Paired Data

In flow matching with paired samples, it is crucial to preserve the deterministic correspondence between the support of the source and target distributions. Such setting, where interpolation trajectories intersect, challenges standard flow matching methods [24, 25] as these methods tend to average conflicting velocity signals near these intersections, yielding biased estimates and failing to transport source points accurately to their paired targets [30, 16, 6]. To overcome this, Work by [30] introduces an auxiliary acceleration field that regulates the rate of change along each trajectory, thereby sharpening the alignment to the true paired mapping. Alternatively, [17] propose learning latent-space projections that untangle intersecting pairs prior to flow estimation, effectively preventing trajectory crossings before velocity estimation.

3 Latent Flow Transformer

We propose the **Latent Flow Transformer (LFT)**, a novel architecture for language modeling that seeks to leverage the benefits of flow-related concepts, as demonstrated in image generation, for transformer-based large language models (LLMs). The LFT is designed to potentially achieve advantages such as improved parameter efficiency, simulation-free training, and latent trajectory straightening.

The LFT reduces model size by replacing a contiguous block of transformer layers from the teacher with a single learned transport operator, called the **latent flow layer**. This operator is trained using flow concepts [24] to accurately map the input latent at the input of the block to its corresponding output over a desired trajectory.

With a suitable architectural choice, the learned operator can be unrolled over predefined intermediate time steps to effectively recover a transformer-like structure. Furthermore, the number of layers used for a given token can be determined at run time on a per-token basis, giving further flexibility for context dependent performance-computation trade-off.

3.1 Velocity field estimating network

Specifying an LFT requires defining the network architecture for the velocity field estimator $u_\theta(x_t, t)$; following the approach in [31], our method augments a teacher transformer layer with additional scale and shift operators and an MLP predicting their factors, as shown in Figure 1(a). We derive the velocity estimate by subtracting the input latent from this augmented network's output (Figure 1(b)).

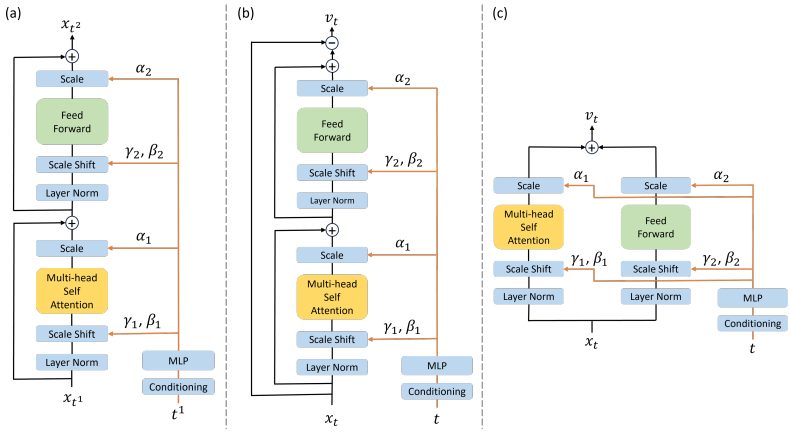


Figure 1: **Velocity field estimator.** (a) The DiT block, mapping an input hidden state to an output hidden state conditioned on time t. (b) Velocity field estimator derived from the DiT block, mapping an input state x_t and time t to a velocity v_t. (c) Velocity field estimator for *Pythia*.

Figure 1(c) details the specific case for *Pythia* models. We remark that these networks are used solely for learning the velocity field; during LFT inference, they are augmented with skip connections to produce latents.

There are a lot of considerations and consequently room for innovation regarding how these estimators causally attend to prior tokens' latents. It is particularly interesting when evaluating $u_\theta(x_t, t)$ at different values of t from token to token. We leave the details to the experiment section.

3.2 Predict flow matching quality by the Recoupling ratio

Selecting the optimal block of layers for replacement is critical for LFT performance. Learnability via flow matching is bottlenecked by flow path crossing [32, 24], a challenge exacerbated in LFT as preserving original input-output pairings precludes the use of reflow methods [25, 4, 16]. To guide layer selection, we introduce an estimator which we refer to as the *Recoupling ratio*. Given sample latent pairs from layers m to n, this estimator quantifies the deviation between their original pairing and the pairing dictated by Optimal Transport (OT). Optimal Transport identifies the minimal cost mapping between two distributions. In Equation 6, the OT matrix M represents the ideal pairing between latents at layer m and layer n that minimizes a transport cost, such as Euclidean distance:

$$M = \arg\min_{\gamma} \sum_{i,j} \gamma_{i,j}\, d\big(h_m^{(i)},\, h_n^{(j)}\big) \tag{6}$$

We define the Recoupling Ratio R as the percentage of pairing relationship that disagrees with M. If M is a square matrix of order O_M:

$$R := 1 - \mathbf{E}\big[\tfrac{\mathrm{Tr}(M)}{O_M}\big]. \tag{7}$$

Because the Recoupling Ratio quantifies the misalignment with original pairings, therefore, controlling for O_M, lower R indicates better alignment and thus predicting fewer flow-crossing issues and higher feasibility for learning the LFT. In our experiments, a practically small O_M is sufficient for the prediction to aid decision making on the selection of layers.

3.3 Learning the velocity field

The velocity estimators shall learn the velocity field from a collection of x_0, x_1 pairs. For an LFT to replacing all layers from layer m to layer n (inclusive), omitting the details of preserving the shared context, we take the input latent of layer m for a given token as x_0, and the output latent of layer n as the corresponding x_1.

To learn the velocity field via standard flow matching [24], Algorithm 1 is applied. We remark that LFT can be trained by many other compatible flow-related algorithms.

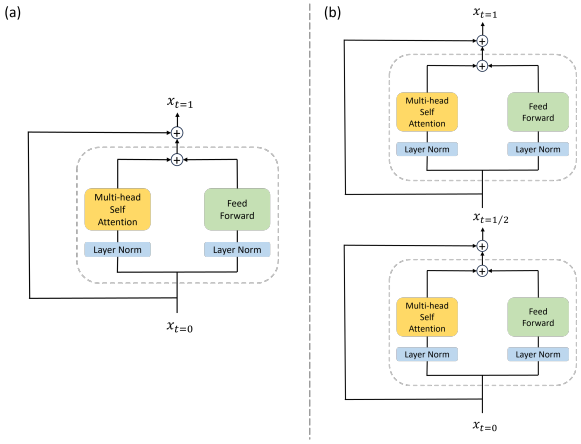


Figure 2: **Static structure of an unrolled LFT.** We highlight only the latent flow layer. The simple reconstruction rule of Equation 4 is assumed. (a) LFT based on *Pythia* with single step reconstruction. (b) LFT based on *Pythia* with two step reconstruction.

Algorithm 1 Learning the velocity field via standard flow matching [24]

$\mathcal{D}$ is a set of training data, $\text{LLM}_{\text{teacher}}(d, m)$ is the latent of LLM at layer m over data d, $AdamW$ is the AdamW optimizer.
while termination condition not met **do**
  $d \sim \mathcal{D}$
  $(x_0, x_1) \leftarrow (\text{LLM}_{\text{teacher}}(d, m), \text{LLM}_{\text{teacher}}(d, n))$
  $t \sim U$
  $x_t \leftarrow (1-t)x_0 + tx_1$
  $\theta \leftarrow AdamW(\theta, \nabla_\theta \|u_\theta(x_t, t) - (x_1 - x_0)\|^2)$
end while

3.4 Unrolled LFT has a Transformer structure

At inference, unrolling the latent flow layer over a fixed set of time points $t_0 = 0 < t_1 < t_2 ... < 1$ hardens the latent evolution process into a static processing graph, beneficial for visualizing data flow and optimizing hardware implementation. For the specific case of a single step flow matching combined with the simple reconstruction approximation rule of Equation 4, the latent flow layer becomes equivalent to a single standard transformer layer (see Figure 2). If more than one step is used, the latent flow layer is equivalent to a stack of transformer layers with cross-layer attention [21].

The structural similarity between the unrolled LFT and a standard transformer is of significant practical importance, enabling researchers and practitioners to leverage the extensive ecosystem and highly optimized infrastructure developed for transformer-based LLMs.

4 Flow Matching with Paired Data

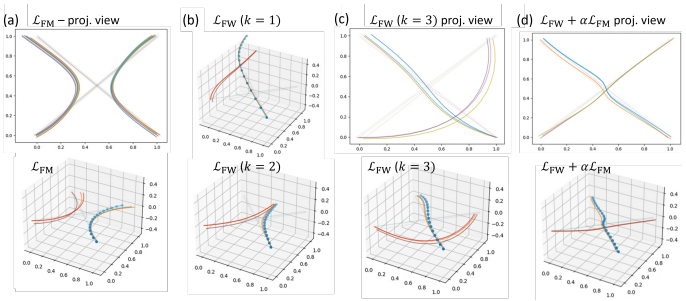
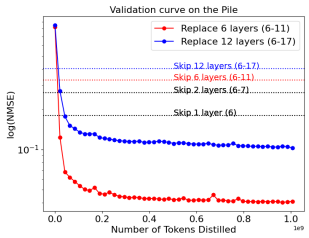
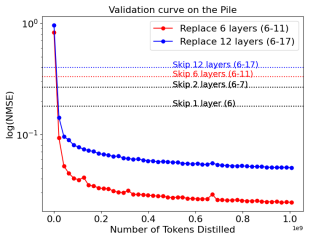


Figure 3: **Toy trajectories for paired data.** Faded lines show the ground-truth trajectories of paired points, while solid curves depict predictions from various flow-matching algorithms. (a) Standard flow matching fails to maintain pairwise correspondence. (b) FW results for $k = 1$ (top) and $k = 2$ (bottom): using fewer integration steps leads to poor trajectory estimates. (c) FW with $k = 3$: projected view (top) and full trajectories (bottom) demonstrate how the learned velocity field generates smooth, curved paths that avoid intersections. (d) A hybrid of FW and standard flow matching with $\alpha = 0.001$ preserves the constant velocity of paired data while allowing curvature at intersections to prevent flow crossings.



(a) LFT-SFM



(b) LFT-FW

Figure 4: **Latent distillation with flow matching vs layer-skipping baselines.** Validation NMSE between predicted and original hidden states vs. training tokens. The dotted lines show Pythia layer-skipping baselines. Flow-matching layers approximate original representations with lower reconstruction error than layer-skipping baselines across both setting.

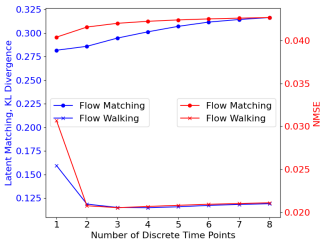


Figure 5: **Inference performance ($\mathrm{KL}_{x||\hat{x}}$ and NMSE) vs number of discrete points** k. We report results of LFT-SFM and LFT-FW with latent flow layer replacing layer 6-18 of Pythia-410m.

To address the challenge of crossing trajectory in 2.3 while to enable the unrolling of latent flow layer back to a Transformer structure in 3.4, we introduce the Flow Walking (FW) algorithm for training and inference the latent flow layer of LFT.

FW approximates x_0 to x_1 using numerical integration with discrete time points. We define a step as $s_\theta(x_t, t, t') = x_{t+d}$, where $d = t' - t$ and x_{t+d} can be determined by Equation 4 or Equation 5. Our key intuition is that we can learn non-crossing trajectories by slightly separate trajectories around intersecting points. We define the learning of velocity fields with:

$$\mathcal{L}_{\text{FlowWalking}(k)} = \mathbb{E}_{t_1,\ldots,t_{k-1}} \left\| x_0 + \sum_{i=1}^{k} \Delta_{\theta,t_i} - x_1 \right\|^2, \tag{8}$$

where $\Delta_{\theta,t_i} = s_\theta\big(x_{t_{i-1}}, t_{i-1}, t_i\big) - x_{t_{i-1}}$, $t_0 = 0$, $t_k = 1$, and $t_i \sim [0, 1]$ for $i \in [1, k-1]$. Unlike linear inference paths (2.2) that randomly samples different numbers of points k [7], we would like to choose the minimum k with randomly sampled t to avoid expensive simulation. We empirically choose an efficient and generalized training loss using $k = 3$ (Figure 3) as choosing $k = 2$ and $t_1 \sim [0, 1]$ cannot generalize to inference time points $k > 2$.

We compare FW with alternative training schemes on toy paired-data experiments (Figure 3. In panel (a), standard flow matching - unstable for paired data - exhibits trajectory "recoupling," though in the absence of crossings it can recover a straight, constant speed path. Panels (b) and (c) show FW with $k = 1$, and $k = 2$ (top and bottom, respectively), which fail to learn a generalizable velocity field, and FW with $k = 3$ preserves the original coupling at the expense of straightness. Finally, in panel (d), we add a straightness regularizer via standard flow matching, i.e.,

$$\mathcal{L} = \mathcal{L}_{\text{FlowWalking}} + \alpha \mathcal{L}_{\text{FlowMatching}}, \tag{9}$$

which yields a non-crossing, straight interpolation of the paired data. While this stability-regularized variant shows promise for paired-data application, we leave its comprehensive study to future work and here focus on demonstrating FW's feasibility to LFT.

Algorithm 2 LFT Training (FW Algorithm)

$\mathcal{D}$ is a set of training data, $\text{LLM}_{\text{teacher}}(d, m)$ is the latent of LLM at layer m over data d, $AdamW$ is the AdamW optimizer.
while termination condition not met **do**
  $d \sim \mathcal{D}$
  $(x_0, x_1) \leftarrow (\text{LLM}_{\text{teacher}}(d, m), \text{LLM}_{\text{teacher}}(d, n))$
  $t_1, t_2 \sim U$ ▷ Sorted $t_2 > t_1$
  $x_{t_1} \leftarrow s_\theta(x_0, 0, t_1)$ ▷ 1st step
  $x_{t_2} \leftarrow s_\theta(x_{t_1}, t_1, t_2)$ ▷ 2nd step
  $\hat{x}_1 \leftarrow s_\theta(x_{t_2}, t_2, 1)$ ▷ 3rd step
  $\theta \leftarrow AdamW(\theta, \nabla_\theta \|\hat{x}_1 - x_1\|^2)$
end while

Algorithm 3 LFT Inference

$x \leftarrow \text{EmbedIn}(token)$
$d \leftarrow 1/k$
$t \leftarrow 0$
$t' \leftarrow d$
for $k \in [0, \ldots, k-1]$ **do**
  $x \leftarrow s_\theta(x, t, t')$
  $t \leftarrow t + d$
  $t' \leftarrow t' + d$
end for
logits = $\text{EmbedOut}(x)$
return logits

5 Experiments

We implemented the LFT based on `Pythia-410m` and present a series of experiments evaluating the choice of distilled layers, the convergence of flow matching distillations, and inference as an integrated transformer. We implement Algorithm 1 by following the reference code of [23], and to ensure a fair comparison between methods - we use the same recursive update rule Equation 5 to compute x_{t+d} for FW.

5.1 Evaluation metric

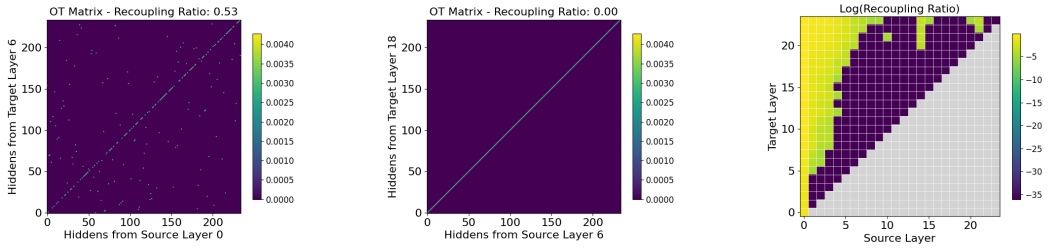
To assess the local fidelity of a latent flow layer, we measure (i) the **normalized mean squared error** (NMSE) and (ii) the **KL divergence** between its output $\hat{x}_1$ and the target latent x_1, denoted as $\mathbf{KL}_{x_1||\hat{x}_1}$. The NMSE is defined as $\text{NMSE} = \frac{\mathbb{E}\,\|\hat{x}_1 - x_1\|^2}{\mathbb{E}\,\|x_1\|^2}$.

For an end-to-end language-modeling evaluation, we compute the empirical KL divergence between the teacher distribution P and the LFT output distribution Q, denoted as $\mathbf{KL}_{P||Q}$, and we also report **perplexity** (PPL) based on the LFT's output logits over a held-out subset of The Pile.

As a reference, Table 1 reports NMSE, KL divergence, and perplexity for the cases where we replace (a) layer 6, or (b) layers 6–7 of Pythia-410M with identity skip connections—conditions under which negligible degradation was previously observed [9].

5.2 Can Recoupling Ratio rank the feasibility of learning latent flow layers?

Sampling just 256 tokens from the first portion of the Pile, we computed the recoupling matrix over different choices of starting layer m and last layer n. The results are given in Figure 6.



(a) OT Matrix of paired hiddens between layer 0 and layer 6.

(b) OT Matrix of paired hiddens between layer 6 and layer 18.

(c) Recoupling Ratios of different source target layers.

Figure 6: **Optimal Transport Matrices and Recoupling Ratio.**

Figure 6(a) is the Optimal Transport matrix M for $(n, m) = (0, 6)$. A low recoupling ratio of 0.53 signals that one might encounter a lot of issues related to flow crossing. In comparison, in Figure 6(b), which gives the OT matrix M for $(n, m) = (6, 18)$, with a zero recoupling ratio, the existing pairings for this batch of data are already optimal. Therefore, the recoupling ratio predicts that the quality of the latent flow layer when replacing layers 6 to 18 should be better than replacing layers 0 to 6, even though the flow over twice as many layers must be learned.

Prior studies have shown that in a typical transformer LLM, the first and the last few layers often apply transformations to the hidden state in a way qualitatively different from those by the middle layers. In Figure 6(c), the recoupling ratio over all possible choices of m and n is given. We can see that the recoupling ratio from our experiments agrees with the past observation for the early layers. By exhibiting quite serious flow crossing issues, the first few layers in particular look very different from other layers.

5.3 Distillation Quality on The Pile

In this section, we first empirically demonstrate the convergence behavior of LFT when trained via Standard Flow Matching (LFT-SFM) in Algorithm 1 and Flow Walking (LFT-FW) in Algorithm 2. We then compare these methods against two baselines: (i) layer-skipping, which directly treats the source hidden state as the target, and (ii) a regression model [9], which uses a single transformer layer (initialized from the teacher's source layer) trained with an MSE loss to map source to target hidden states.

Method	Latent Matching		Language Modeling	
	$\mathrm{KL}_{x\|\|\hat{x}}$	NMSE	$\mathrm{KL}_{P\|\|Q}$	PPL
Pythia-410m	-	-	-	9.02
Skip one layer (layer 6)	0.052	0.180	0.257	11.43
Skip two layer (layer 6, 7)	0.307	0.266	0.529	15.27
Skip three layer (layer 6, 7, 8)	1.157	0.307	0.932	22.59
Replacing Layer 6-12				
Regression	0.042	0.010	0.305	12.45
LFT - SFM ($k = 1$)	0.076	0.015	0.409	14.04
LFT - SFM ($k = 3$)	0.076	0.015	0.407	13.98
LFT - FW ($k = 1$)	0.050	0.012	0.300	12.42
LFT - FW ($k = 3$)	**0.032**	**0.007**	**0.254**	**11.86**
Replacing Layer 6-18				
Regression	0.156	0.024	0.864	21.97
LFT - SFM ($k = 1$)	0.282	0.040	1.216	31.35
LFT - SFM ($k = 3$)	0.295	0.042	1.232	31.91
LFT - FW ($k = 1$)	0.160	0.031	0.838	21.82
LFT - FW ($k = 3$)	**0.115**	**0.021**	**0.736**	**19.92**

Table 1: **Comparison of transformer-layer reduction methods.** We evaluate LFT at multiple discrete time points k; **boldface** highlights the top-performing method within each group of layer-replacement experiments.

We distill on 2.6 billion tokens from The Pile by replacing either layers 6–12 (25 % of parameters) or layers 6–18 (50 % of parameters) of Pythia-410m with a single flow-matching layer. As shown in Figure 4, both LFT-SFM and LFT-FW converge rapidly and substantially outperform naive layer-skipping. We then perform inference at multiple discrete time points and compare to the regression baseline in Table 1 (see 5.4 for analysis on time-point selection). Although LFT-SFM performance exceeds layer-skipping, it fails to match the regression model's performance—likely because Standard Flow Matching struggles to learn fine-grained velocity fields when paired trajectories are closely clustered in latent space (4). In contrast, Flow Walking consistently outperforms both baselines in terms of latent-state matching and downstream language modeling. Notably, LFT-FW with $k = 1$ achieves parity with the regression model and outperforms LFT-SFM with $k = 8$, suggesting that Flow Walking's implicit velocity estimates more accurately guide the model toward its target hidden state.

5.4 Effect of discrete time points on inference performance

The number of discrete time points k is a key hyperparameter of the LFT at inference time (see Algorithm 3). While minimizing the total number of learnable parameters is crucial, it is equally important to choose a small k, since each additional time point incurs extra FLOPs during inference. In Figure 5, we report both the $\mathrm{KL}_{x||\hat{x}}$ and NMSE of LFT as functions of k. For LFT-SFM, both KL and NMSE decrease as k is reduced, implying that early velocity estimates steer more correctly the hidden state toward its target hidden state. As to LFT-FW, it attains its best performance at $k = 3$, matching the three-step integration in Equation 8. As k approaches 1, KL divergence rises sharply, indicating that the implicit velocity estimation at $t = 0$ is inaccurate and must be refined through multi-step corrections. Interestingly, despite never training with more than three integration steps, LFT-FW extrapolates robustly to $k = 8$, showing only minor degradation.

6 Related Work

Theory of Transformer. Transformer architectures [37] are the foundation of LLMs, enabling strong performance across various NLP tasks. As model scale increases, understanding their internal structure becomes essential. Recent works explore the evolution of representations across layers, attention distributions, and layer-level modularity. For example, "Frankenstein" models recombine layers from different checkpoints to study knowledge localization and compatibility [28, 12, 15]. Other research investigates latent trajectory and attention alignment across layers [38], offering insights into the compositional behavior of LLMs.

Efficient Transformer. Improving transformer efficiency has been approached through pruning, parameter sharing, and distillation. Magnitude-based pruning [8] and structured approaches [41] have shown that substantial parameter reduction is possible with limited performance drop [27]. Parameter-sharing architectures like ALBERT [18] compress models by reusing weights across layers. Distillation methods such as DistilBERT [34] and TinyBERT [13] transfer knowledge from larger models to smaller ones through intermediate supervision and multi-stage training.

Flow matching for image generation. Flow-based models have emerged as efficient alternatives to diffusion in generative modeling. Flow Matching (FM) [24] learns continuous velocity fields for optimal transport and has been extended with Rectified Flow [25] and Conditional Flow Matching [20] to improve training stability and sample quality. Recent works [6] demonstrate high-quality one-step generation by learning direct endpoint mappings, highlighting FM's potential in reducing generation steps.

Diffusion and flow matching models for language modeling. Recent efforts have adapted diffusion and FM frameworks for language generation. Diffusion-based models such as GENIE [22] apply denoising processes to continuous token embeddings, while AR-Diffusion [39] introduces autoregressive properties for better sequence modeling. FM-based approaches include FlowSeq [11], which improves sampling efficiency, and CaLMFlow [10], which applies Volterra integral formulations to better align continuous and discrete modeling in language.

7 Discussions and open problems

Speculative decoding refers to the use of a small draft model to accelerate the inference of a full-sized model. As the size of a straight flow transformer can be order-of-magnitude smaller than its teacher model, in our opinion it is natural to use the straight flow transformer as the draft model in a speculative decoding set up. Given the KL-distance result that was presented in our work, the performance of such a configuration is quite promising.

A transformer has a fixed computational cost with a fixed generation quality. In contrast, a straight flow transformer allows for dynamically varying the number of steps, i.e. the computational investment, on a sentence-by-sentence, or even a token-by-token basis. Thus one might contemplate whether and how to reach an optimal trade-off between the number of steps used and the corresponding quality. However, how to self- or cross-estimate the quality of the output logits of a given token at the output of a transformer is still an open problem.

Lately, state-space models and recurrent models have emerged as candidates to substitute for transformers due to their constant computational complexity of handling context. We suspect that if the method in our work is applied, one can create straight flow RWKV, straight flow MAMBA, straight flow xLSTM, etc. Unfortunately, at the moment we are not properly setup to train SOTA state-space or recurrent models.

8 Future Work

Flow Untangle. In our current approach, we have retained the input and output layers of the source transformer unchanged. It is well established that arranging flow matching pairs to minimize flow crossing significantly enhances performance. We propose optimizing the input and output layers to minimize flow crossing, conditioned on the input embedding and output logit. Although we are actively working on this optimization, results are not yet available at the time of writing. Successfully implementing this strategy could reduce the number of layers by an order of magnitude or more, thereby improving inference efficiency.

Architectural Search and Initialization Method. In this paper, we have distilled the interior layers of the source transformer into straight flow layers, presenting a principled pruning method. Post-pruning, we aim to investigate the training dynamics if pretraining is continued. This approach has the potential to outperform its parameter class, offering significant benefits for training by reducing the number of layers by an order of magnitude or more.

Scale Up to Larger Models Due to limited computational resources, our experiments have been constrained to the Pythia model. In our future work, we plan to scale up our experiments, but achieving this requires collaboration with researchers who have access to larger computational power. Extending our approach to state-of-the-art models, such as the larger OLMO, is a crucial next step in validating and enhancing our findings.

Training Flow-Replaced Transformers from Scratch. An open question is whether the resulting shallow transformer, once flow-replaced and structurally simplified, can be trained from scratch without first pretraining a full-depth model. Prior observations in diffusion literature suggest that one-step generators trained from scratch rarely match the performance of distilled ones. A similar pattern may hold in language modeling. This motivates further investigation into curriculum training, staged flow replacement, or leveraging lightweight teacher guidance during early training stages.

References

[1] Michael S Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. *arXiv preprint arXiv:2209.15571*, 2022.

[2] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[3] Yuqi Chen, Kan Ren, Yansen Wang, Yuchen Fang, Weiwei Sun, and Dongsheng Li. Contiformer: Continuous-time transformer for irregular time series modeling. *Advances in Neural Information Processing Systems*, 36:47143–47175, 2023.

[4] Ayan Das, Stathi Fotiadis, Anil Batra, Farhang Nabiei, FengTing Liao, Sattar Vakili, Da shan Shiu, and Alberto Bernacchia. Image generation with shortest path diffusion, 2023.

[5] Jacob Fein-Ashley. Flowing through layers: A continuous dynamical systems perspective on transformers, 2025.

[6] Kevin Frans, Danijar Hafner, Sergey Levine, and Pieter Abbeel. One Step Diffusion via Shortcut Models, October 2024. arXiv:2410.12557 [cs].

[7] Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv:2502.05171*, 2025.

[8] Mitch Gordon, Kevin Duh, and Nicholas Andrews. Compressing bert: Studying the effects of weight pruning on transfer learning. In *Proceedings of the 5th Workshop on Representation Learning for NLP*, 2020.

[9] Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A. Roberts. The unreasonable ineffectiveness of the deeper layers, 2025.

[10] Sizhuang He, Daniel Levine, Ivan Vrkic, Marco Francesco Bressana, David Zhang, Syed Asad Rizvi, Yangtian Zhang, Emanuele Zappala, and David van Dijk. Calmflow: Volterra flow matching using causal language models, 2024.

[11] Vincent Hu, Di Wu, Yuki Asano, Pascal Mettes, Basura Fernando, Björn Ommer, and Cees Snoek. Flow matching for conditional text generation in a few sampling steps. In Yvette Graham and Matthew Purver, editors, *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 380–392, St. Julian's, Malta, March 2024. Association for Computational Linguistics.

[12] Gabriel Ilharco, Mitchell Wortsman, et al. Editing models with task arithmetic. In *ICML*, 2023.

[13] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. In *Findings of EMNLP*, 2019.

[14] Patrick Kidger. On neural differential equations. *arXiv preprint arXiv:2202.02435*, 2022.

[15] Dahyun Kim, Chanjun Park, Sanghoon Kim, Wonsung Lee, Wonho Song, Yunsu Kim, Hyeonwoo Kim, Yungi Kim, Hyeonju Lee, Jihoo Kim, et al. Solar 10.7 b: Scaling large language models with simple yet effective depth up-scaling. *arXiv preprint arXiv:2312.15166*, 2023.

[16] Semin Kim, Jaehoon Yoo, Jinwoo Kim, Yeonwoo Cha, Saehoon Kim, and Seunghoon Hong. Simulation-Free Training of Neural ODEs on Paired Data, October 2024. arXiv:2410.22918 [cs].

[17] Semin Kim, Jaehoon Yoo, Jinwoo Kim, Yeonwoo Cha, Saehoon Kim, and Seunghoon Hong. Simulation-free training of neural odes on paired data. *Advances in Neural Information Processing Systems*, 37:60212–60236, 2024.

[18] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations, February 2020. arXiv:1909.11942.

[19] Bei Li, Quan Du, Tao Zhou, Yi Jing, Shuhan Zhou, Xin Zeng, Tong Xiao, JingBo Zhu, Xuebo Liu, and Min Zhang. ODE transformer: An ordinary differential equation-inspired model for sequence generation. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8335–8351, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[20] Xiangning Li, Shuang Li, Zhengyang Zhou, et al. Diffusion via conditional transport. *arXiv preprint arXiv:2305.08891*, 2023.

[21] Yangyang Li, Qin Huang, Xuan Pei, Yanqiao Chen, Licheng Jiao, and Ronghua Shang. Cross-layer attention network for small object detection in remote sensing imagery. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 14:2148–2161, 2020.

[22] Zhenghao Lin, Yeyun Gong, Yelong Shen, Tong Wu, Zhihao Fan, Chen Lin, Nan Duan, and Weizhu Chen. Text generation with diffusion language models: A pre-training approach with continuous paragraph denoise. In *International Conference on Machine Learning*, pages 21051–21064. PMLR, 2023.

[23] Yaron Lipman, Marton Havasi, Peter Holderrieth, Neta Shaul, Matt Le, Brian Karrer, Ricky T. Q. Chen, David Lopez-Paz, Heli Ben-Hamu, and Itai Gat. Flow matching guide and code, 2024.

[24] Yotam Lipman, Talia Ringer Cohen, Yujia Zhang, Ehsan Hajiramezanali, et al. Flow matching for generative modeling. In *NeurIPS*, 2022.

[25] Minghao Liu, Belinda Tzen, Maxim Raginsky, et al. Rectified flow: Learning deep generative models using optimal transport. *arXiv preprint arXiv:2209.14687*, 2022.

[26] Yao Lu, Hao Cheng, Yujie Fang, Zeyu Wang, Jiaheng Wei, Dongwei Xu, Qi Xuan, Xiaoniu Yang, and Zhaowei Zhu. Reassessing Layer Pruning in LLMs: New Insights and Methods, November 2024. arXiv:2411.15558.

[27] Yao Lu, Hao Cheng, Yujie Fang, Zeyu Wang, Jiaheng Wei, Dongwei Xu, Qi Xuan, Xiaoniu Yang, and Zhaowei Zhu. Reassessing layer pruning in llms: New insights and methods. *arXiv preprint arXiv:2411.15558*, 2024.

[28] Michael Matena and Colin Raffel. Merging models with fisher-weighted averaging. In *arXiv preprint arXiv:2206.14841*, 2022.

[29] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021.

[30] Dogyun Park, Sojin Lee, Sihyeon Kim, Taehoon Lee, Youngjoon Hong, and Hyunwoo J. Kim. Constant acceleration flow, 2024.

[31] William Peebles and Saining Xie. Scalable diffusion models with transformers, 2023.

[32] Gabriel Peyré, Marco Cuturi, et al. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

[33] Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. On the effect of dropping layers of pre-trained transformer models. *Computer Speech & Language*, 77:101429, January 2023.

[34] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. In *arXiv preprint arXiv:1910.01108*, 2019.

[35] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models, 2023.

[36] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient knowledge distillation for bert model compression, 2019.

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

[38] Yiming Wang, Pei Zhang, Baosong Yang, Derek F Wong, and Rui Wang. Latent space chain-of-embedding enables output-free llm self-evaluation. *arXiv preprint arXiv:2410.13640*, 2024.

[39] Tong Wu, Zhihao Fan, Xiao Liu, Hai-Tao Zheng, Yeyun Gong, Yelong Shen, Jian Jiao, Juntao Li, Zhongyu Wei, Jian Guo, Nan Duan, and Weizhu Chen. Ar-diffusion: auto-regressive diffusion model for text generation. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

[40] Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. A survey on knowledge distillation of large language models, 2024.

[41] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Prune once for all: Sparse pre-trained language models. In *arXiv preprint arXiv:2102.07436*, 2021.