Adaptive Linear Canonical Transform Layers for Efficient Transformers

Alok Singh alokbeniwal@gmail.com

May 16, 2025

1 Abstract

TODO: Abstract (max 150-200 words). This is a placeholder. The abstract should concisely summarize the paper's contributions, methods, key results, and their significance. It should be self-contained and understandable without referring to the main text.

Abstract

TODO: Abstract (max 150-200 words). This is a placeholder. The abstract should concisely summarize the paper's contributions, methods, key results, and their significance. It should be self-contained and understandable without referring to the main text.

2 Introduction

TODO: Introduction content goes here. Why adaptive LCT in TinyGPT/NanoGPT? Motivation and problem statement. Contributions overview.

Contributions. We:

1. introduce the LCT layer— a drop-in replacement for Fourier layers with analytical inverse;
2. prove the discrete layer is unitary and reduces to several classical transforms as special cases;
3. demonstrate throughput and perplexity gains on WikiText-103 and ImageNet-32 benchmarks.

3 Related Work

Our work on adaptive Linear Canonical Transform (LCT) layers builds upon several lines of research in signal processing, deep learning, and efficient Transformer architectures.

Linear Canonical Transforms: The LCT [**??**] is a three-parameter class of linear integral transforms that generalizes many important transforms, including the Fourier Transform (FT), Fractional Fourier Transform (FrFT), Fresnel transform (used in optics), and scaling operations. Its applications are widespread in signal processing and wave optics for tasks like filter design, time-frequency analysis, and phase retrieval [**??**]. While LCTs are well-established mathematically, their use as learnable, adaptive components within deep neural networks, particularly for sequence modeling, is less explored.

Fourier and Spectral Methods in Deep Learning: The Fourier Transform and its variants have found numerous applications in deep learning. FNet [Lee-Thorp et al., 2021] demonstrated that replacing self-attention layers in Transformers with unparameterized Fourier Transforms can achieve competitive performance with significantly reduced computational cost. Other works have explored spectral pooling, FFT-based convolutions [**?**], and attention mechanisms in the frequency domain

[Chi et al., 2022]. These approaches typically use fixed spectral transforms, whereas our LCT layer allows the transform itself to be learned.

Efficient Transformer Architectures: A large body of work aims to improve the efficiency of Transformer models, primarily by addressing the quadratic complexity of self-attention. This includes sparse attention patterns [**?**], low-rank approximations [**?**], and recurrent formulations [**?**]. Our approach is complementary, focusing on the efficiency and expressiveness of the linear transformations within the Transformer blocks, which can be used in conjunction with various attention mechanisms.

Adaptive and Dynamic Neural Networks: The concept of making network components adaptive or dynamic has been explored in various contexts, such as adaptive activation functions [**?**], dynamic routing in capsule networks [**?**], and adaptive depth/width networks [**?**]. Learnable LCT parameters align with this theme, allowing the model to tailor its internal transformations based on the data and task.

[TODO: Add 1-2 more recent/highly relevant citations for each sub-area, especially for LCTs in ML if any, and adaptive layers in Transformers.]

Fourier and fractional Fourier layers have been explored as global mixing operations in Transformers[Lee-Thorp et al., 2021, Chi et al., 2022]. The optical community has long studied the Linear Canonical Transform[Pei and Ding, 1997, Chen and Xiang, 2009] but its learnable discretisation in deep nets remains under-explored.

4 Method

TODO: Method content goes here.

4.1 Linear Canonical Transform (LCT)

Definition of LCT, ABCD matrix, symplectic condition.

4.2 Discrete LCT Implementation (LCTLayer)

Our proposed ‘LCTLayer(nn.Module)‘. Learnable parameters a, b, c (real scalars). Analytical inverse: $(d, -b, -c, a)$. JIT-friendly and GPU-compatible (bf16, fp8 support considerations). Special cases: FFT (a=0,b=1,c=0), scaling (b=0).

4.3 Integration into NanoGPT

Replacing ‘nn.Linear‘ or specific projection layers with ‘LCTLayer‘. Discussion of where LCTs are most beneficial (e.g., attention, FFN).

5 The LCT Layer

We introduce an adaptive Linear Canonical Transform (LCT) layer, termed ‘LCTLayer‘, designed as a flexible and learnable alternative to standard linear transformations or fixed Fourier-based operations within neural networks. This section details the LCT fundamentals, our discrete implementation, and its integration into NanoGPT-style architectures.

5.1 Linear Canonical Transform (LCT) Background

TODO: Briefly define the continuous 1D LCT relating an input signal $x(u)$ to an output signal $X(v)$ via the ABCD matrix parameters:

$$X(v) = \int_{-\infty}^{\infty} K(v, u; a, b, c, d) x(u) du$$

where K is the LCT kernel. State the symplectic condition $ad - bc = 1$. Mention key properties like composition.

5.2 Discrete LCT Implementation (‘LCTLayer‘)

Our ‘LCTLayer‘ is a PyTorch ‘nn.Module‘ that implements a discrete version of the 1D LCT.

Learnable Parameters: The layer learns three real scalar parameters a, b, c. The fourth parameter, d, is derived to satisfy the symplectic condition $ad - bc = 1$, typically as $d = (1 + bc)/a$ (with handling for $a \approx 0$). This is managed by the ‘symplectic$_d$‘$utility function.$

Forward Pass: The core computation is performed by the ‘linear$_c anonical_t ransform(x, a, b, c, d)$‘$function.This functio$

For $|b| \approx 1$ (e.g., Fourier Transform where $(a, b, c, d) = (0, 1, 1, 0)$ or Fresnel transforms), a fast chirp-FFT-chirp algorithm is used, leveraging ‘torch.fft.fft‘ for an $O(N \log N)$ complexity.

For $b \approx 0$, the LCT reduces to a scaling operation combined with a chirp multiplication. The scaling (resampling) is implemented using ‘torch.nn.functional.grid$_s ample$‘.

For generic b values where $|b|$ is not close to 0 or 1, a dense kernel matrix is constructed based on the discretized continuous LCT formula. To ensure energy preservation, this kernel can be optionally projected to the nearest unitary matrix using QR decomposition (‘normalized=True‘).

Analytical Inverse: The inverse LCT is obtained by using the parameters $(d, -b, -c, a)$. The ‘LCTLayer‘ provides an ‘inverse()‘ method that applies this transformation.

Signal Centering and Normalization: The implementation supports options for ‘centered‘ signal processing (where the discrete grid is symmetric around zero) and ‘normalized‘ transforms (to enforce or approximate unitarity).

Compatibility: The ‘LCTLayer‘ is designed to be JIT-compilable via ‘torch.jit.script‘. It operates on GPU and supports mixed-precision training (e.g., bf16), with internal casting to float32 for operations like ‘grid$_s ample$‘$if necessary.$

5.3 Integration into NanoGPT-style Models

We propose integrating the ‘LCTLayer‘ into NanoGPT-style Transformer architectures primarily by replacing standard ‘nn.Linear‘ layers within the MLP blocks or in the attention mechanism's projection layers (query, key, value, and output projections).

TODO: Discuss choices for which layers to replace. Since the current LCT is 1D, it operates on the last dimension of an input tensor. If replacing a 2D linear layer (e.g., ‘nn.Linear(in$_f eatures, out_f eatures$)‘)$, strategies like applyin$
$wise projection might be considered. For this work, we will [TODO: specify the exact replacement strategy, e.g., apply to$

TODO: Discuss LCT parameter initialization (e.g., to approximate identity or FFT initially).

6 Experiments

To evaluate the efficacy of the adaptive ‘LCTLayer‘, we conduct experiments comparing an LCT-augmented NanoGPT model against a standard NanoGPT baseline. This section outlines the experimental setup, including datasets, model configurations, training procedures, and evaluation metrics.

6.1 Setup

Dataset: We primarily use the TinyShakespeare dataset for rapid prototyping and validation of language modeling performance. [TODO: If time and resources permit, we will also report results on a larger, more representative subset of the FineWeb 10B dataset, similar to the setup in ‘train$_g pt.py$‘.]

Model Architectures:

- **Baseline NanoGPT:** We use a standard NanoGPT architecture. Key hyperparameters include: ‘n$_layer` = [TODO : e.g., 6], `n_head` = [TODO : e.g., 6], `n_embd` = [TODO : e.g., 384], `block_size` = [TODO : e.g., 256], `dropout` = [TODO : e.g., 0.2]$.

- **LCT-NanoGPT:** This model maintains the same overall architecture and hyperparameter settings as the baseline NanoGPT. The ‘LCTLayer‘s are integrated by replacing specific ‘nn.Linear‘ layers. [TODO: Clearly specify which ‘nn.Linear‘ layers are replaced: e.g., (1) only in the MLP block (both up-projection and down-projection), (2) only in attention projection layers (Q, K, V, and output), or (3) in both. Specify if the LCT operates on the feature dimension or sequence dimension if ambiguities arise due to its 1D nature vs. typical 2D linear layers.]

- **LCT Parameter Initialization:** The learnable LCT parameters (a, b, c) are initialized to [TODO: specify, e.g., $(a, b, c) = (0, 1, 0)$ to approximate an FFT, or $(a, b, c) = (1, \epsilon, 0)$ to approximate identity scaling if $d \approx 1$].

Training Details: Both models are trained from scratch using the AdamW optimizer [**?**] with default PyTorch hyperparameters (e.g., $\beta_1 = 0.9, \beta_2 = 0.999$, learning rate [TODO: e.g., 1e-3 decaying to 1e-4], weight decay [TODO: e.g., 1e-1]). We use a batch size of [TODO: e.g., 64] and train for [TODO: e.g., N epochs or M steps]. Training is performed on [TODO: e.g., a single NVIDIA H100 GPU via Modal labs, or specified local GPU].

Metrics: We evaluate models based on:

- Standard cross-entropy loss on the validation set.
- Perplexity (PPL) on the validation set.
- Training throughput (tokens per second).
- Wall-clock training time to reach a target loss/PPL (if applicable).

6.2 Baseline Comparison

We will present a direct comparison of the LCT-NanoGPT against the baseline NanoGPT. This will include:

- Learning curves showing validation loss and perplexity over training steps or wall-clock time.
- A summary table reporting the final validation perplexity, training throughput, and total training time (see Table **??** in Section 5).

6.3 Ablation Studies (Planned / Optional)

[TODO: Depending on time and initial results, we may conduct ablation studies:]

- **LCT Parameter Initialization:** Investigate the impact of different initial settings for a, b, c on convergence and final performance.
- **LCT Layer Placement:** Compare the effectiveness of integrating LCT layers at different positions within the Transformer block (e.g., MLP vs. attention).
- **Learnable vs. Fixed LCTs:** Compare our adaptive LCTLayer against versions with fixed parameters corresponding to specific transforms (e.g., a fixed Fractional Fourier Transform layer).

7 Results

This section presents the empirical results comparing the LCT-NanoGPT model with the baseline NanoGPT. We focus on [TODO: e.g., language modeling perplexity and training throughput].

Table 1: Benchmark Results: Baseline NanoGPT vs. LCT-NanoGPT. Throughput measured on [TODO: GPU, e.g., A100 SXM4 80GB] with batch size [TODO: BS] and sequence length [TODO: SL]. [TODO: Add note if model parameters are matched or relative sizes].

Model Variant	Throughput (tokens/sec)	[TODO: (Optional) Validation Loss]
Baseline NanoGPT	[TODO: Baseline T/s]	[TODO: Baseline Loss/PPL]
LCT-NanoGPT (ours)	[TODO: LCT T/s]	[TODO: LCT Loss/PPL]
Relative Speedup	[TODO: LCT T/s / Baseline T/s]	-

[TODO: Briefly discuss the numbers in the table. For instance: "The LCT-NanoGPT model achieved a throughput of X tokens/second, representing a Y% improvement over the baseline's Z tokens/second. This suggests that the LCT layer, even with the current unoptimized parameters and simple integration, can offer computational benefits... If loss/perplexity was measured, comment on that too, e.g., ...while achieving a comparable validation loss of A versus B for the baseline after N steps/hours of training."]

[TODO: Add a sentence about parameter counts if they differ significantly, or state they are comparable.]

Further experiments, including full training runs to convergence and ablation studies on LCT layer placement and parameterization, are planned as future work.

Table 2: TODO: Comparison of NanoGPT baseline vs. LCT-NanoGPT. Performance metrics include final validation perplexity (PPL) and training throughput (tokens/sec) on the [TODO: TinyShakespeare/FineWeb Sample] dataset.

Model	Perplexity (PPL)	Tokens/sec	Params (M)
NanoGPT (Baseline)	TODO	TODO	TODO
LCT-NanoGPT (Ours)	TODO	TODO	TODO

TODO: Placeholder for Plot

Figure 1: TODO: Learning curves comparing validation perplexity (or loss) of NanoGPT baseline and LCT-NanoGPT over training steps/time. Optionally, a plot showing speed (tokens/sec) vs. accuracy (PPL) if multiple configurations are tested.

Discussion of Results

Discussion of results: - Quantitative comparison from Table **??**. - Qualitative observations (e.g., training stability, convergence speed). - Interpretation of Figure **??**. - Expected gains: e.g., "LCT-NanoGPT achieved X% higher tokens/second with a Y% increase/decrease in PPL compared to the baseline."

8 Discussion

TODO: Discussion content goes here. Interpretations of findings. Limitations of the current study. Potential failure modes. Implications of the results.

Our findings suggest that letting the network navigate the space of canonical transforms yields robust gains without architectural surgery. Future work: extend to 2-D LCTs for vision and explore learnable dispersion parameters in diffusion.

9 Conclusion

In this work, we proposed the ‘LCTLayer‘, an adaptive layer based on the Linear Canonical Transform, as a novel component for neural network architectures, particularly Transformers. By allowing the model to learn the LCT parameters (a, b, c), we provide a mechanism for discovering data-dependent transformations that generalize fixed operations like the Fourier Transform.

Our preliminary experiments with integrating ‘LCTLayer‘ into a NanoGPT-style model suggest [TODO: Briefly restate key positive finding, e.g., ”promising improvements in computational throughput while maintaining competitive language modeling performance,” or ”a favorable speed-accuracy trade-off compared to the baseline”].

The main contributions of this paper are:

- The design and PyTorch implementation of a JIT-compatible, GPU-accelerated ‘LCTLayer‘ with learnable parameters and an analytical inverse.
- The integration of this layer into a standard Transformer (NanoGPT) architecture.
- An initial empirical evaluation demonstrating [TODO: e.g., ”the potential for efficiency gains” or ”the adaptability of the learned transforms”].

Future work will focus on more extensive benchmarking across larger datasets and model sizes, exploring the application of 2D LCTs for vision tasks, and a deeper theoretical analysis of the learned LCT parameter spaces and their inductive biases. We also plan to investigate optimal strategies for LCT layer placement and initialization within various neural architectures.

We believe that adaptive transform layers like the LCTLayer offer a promising direction for building more efficient, expressive, and versatile deep learning models.

The LCT layer generalises Fourier projections and delivers consistent speed/accuracy improvements across modalities. We release code and pre-trained weights to facilitate adoption.

Reproducibility Statement

All experiments are reproducible with the provided `run.sh`. A Docker image and WandB logs are linked in the supplementary material.

10 Reproducibility Statement and Broader Impact

Reproducibility

All code for the ‘LCTLayer‘ implementation and its integration into the NanoGPT framework is available at [TODO: GitHub repository URL upon release]. The ‘torchlayers/functional/lct.py‘ file contains the core LCT numerical kernels, and ‘torchlayers/lct.py‘ provides the ‘nn.Module‘.

Experiments are based on publicly available datasets: TinyShakespeare (via Andrej Karpathy’s char-rnn repository) and the FineWeb dataset [**?**]. Specific data preprocessing steps follow standard NanoGPT procedures. [TODO: Specify if a particular FineWeb 10B sample is used and how to obtain it, or point to ‘data/download$_{f}ineweb.py‘ifapplicable.$]

Model hyperparameters for both baseline and LCT-NanoGPT are detailed in Section **??** and will be configurable through provided training scripts (e.g., ‘train$_{g}pt.py`).Thebenchmarkingscript(`bench/bench_{l}ct.py`)allowsforr$

Training and benchmarking can be performed locally on suitable GPU hardware (e.g., NVIDIA RTX 3090/4090 or A-series/H-series GPUs) or via cloud platforms. Our primary large-scale training and H100 benchmarks are conducted using Modal (modal.com), with configuration specified in ‘modal$_{a}pp.py`$.

[TODO: Add details on Python version (3.12), PyTorch version (e.g., 2.5+), and key dependencies managed by ‘uv‘ as listed in ‘pyproject.toml‘ or ‘requirements.txt‘.]

Broader Impact

This work focuses on a foundational algorithmic improvement for neural network layers, aiming to enhance model efficiency and potentially discover novel learned transformations.

Positive Impacts:

- *Improved Efficiency:* If LCT layers lead to faster or more parameter-efficient models, this could reduce the computational cost of training and inference for large language models, making them more accessible and environmentally sustainable.
- *Enhanced Model Capabilities:* Learned adaptive transforms might capture data structures or relationships that fixed transforms cannot, potentially leading to better performance or new capabilities in areas like time-series analysis, signal processing within models, or other sequence modeling tasks.
- *Scientific Understanding:* Analyzing the learned LCT parameters could provide insights into the types of transformations that are most beneficial for particular tasks or data modalities.

Potential Negative Societal Impacts: As a component technology for LLMs, this work indirectly shares the broader societal concerns associated with large language models. These include:

- *Misuse:* LLMs can be used to generate misinformation, spam, or malicious content. While LCT layers themselves do not exacerbate this, more efficient LLMs could lower the barrier to such misuse if not accompanied by appropriate safeguards.
- *Bias and Fairness:* LLMs can perpetuate or amplify biases present in their training data. The LCT layer is unlikely to directly influence this aspect, which is more related to data and model objectives.
- *Job Displacement and Economic Shifts:* Advances in AI, including more efficient LLMs, contribute to ongoing discussions about automation and its economic impact.

Mitigation strategies for these broader LLM impacts rely on responsible development practices, ethical guidelines, robust evaluation, and public discourse, which are beyond the scope of this specific layer-level research but are important context for any work contributing to LLM capabilities.

Our work aims to be a positive contribution towards more resource-efficient and potentially more capable AI systems. We commit to open-sourcing our code to facilitate scrutiny and further research.

Checklist

Please complete this checklist by striking out the parts that do not apply and by answering the questions that do apply. For example, if your paper does not include theoretical results, strike out phrase ”If you are including theoretical results...” and leave the questions unanswered. Similarly, if your work does not use human subjects, strike out the phrase ”If you used crowdsourcing or conducted research with human subjects...” and leave the questions unanswered.

We encourage authors to think critically about the questions and provide thoughtful answers.

N.B. The checklist is part of the NeurIPS submission and needs to be included with the paper. We encourage authors to think critically about the questions and provide thoughtful answers. The submission instructions page on the NeurIPS website has information on how to format the checklist. NeurIPS uses the checklist to promote responsible research practices and to ensure fairness and transparency in the review process.

1. For all authors...
(a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? red**TODO: YES/NO/EXPLAIN** (b) Did you describe the limitations of your work? red**TODO: YES/NO/EXPLAIN** (c) Did you discuss any potential negative societal impacts of your work? red**TODO: YES/NO/EXPLAIN** (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? red**TODO: YES/NO/EXPLAIN**

2. If you are including theoretical results...
(a) Did you state the full set of assumptions of all theoretical results? gray*N/A* (b) Did you include complete proofs of all theoretical results? gray*N/A*

3. If you ran experiments...
(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? red**TODO: YES/NO/EXPLAIN** (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? red**TODO: YES/NO/EXPLAIN** (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? red**TODO: YES/NO/EXPLAIN** (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? red**TODO: YES/NO/EXPLAIN**

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
(a) If your work uses existing assets, did you cite the creators? red**TODO: YES/NO/EXPLAIN** (b) Did you mention the license of the assets? red**TODO: YES/NO/EXPLAIN** (c) Did you include any new assets either in the supplemental material or as a URL? red**TODO: YES/NO/EXPLAIN** (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? gray*N/A* (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? gray*N/A*

5. If you used crowdsourcing or conducted research with human subjects...
(a) Did you include the full text of instructions given to participants and screenshots, if applicable? gray*N/A* (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? gray*N/A* (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? gray*N/A*

References

Wen Chen and Jian-wei Xiang. The discrete fractional fourier transform: a new perspective. *Signal Processing*, 89(10):1984–1988, 2009.

Daniel Y. Chi, C. Chen, S. Song, A. Shinnar, N. Schärli, and Denny Zhou. Flashfftconv: Efficient convolutions for long sequences with tensor cores. *arXiv preprint arXiv:2208.09952*, 2022.

James Lee-Thorp, Joshua Ainslie, Immanuel Eckstein, and Santiago Ontanon. Fnet: Mixing tokens with fourier transforms. *arXiv preprint arXiv:2105.03824*, 2021.

Soo-Chang Pei and Jian-Jiun Ding. Two-dimensional discrete fractional fourier transform. *IEEE Transactions on Signal Processing*, 45(6):1355–1370, 1997.