FAST GENERALIZED NEURAL TANGENT KERNEL STATISTICS VIA TRACE ESTIMATION

Anonymous authors
Paper under double-blind review

ABSTRACT

The empirical state-space Neural Tangent Kernel (NTK) describes the local learning geometry of a finite-width neural network, but computing it explicitly quickly becomes impractical. We show that many useful NTK statistics that characterize, for example, the dimensionality of learned updates or how two models or learning rules relate, can instead be computed from matrix-free products using randomized trace estimation. Namely, we use Hutch++ to estimate the NTK trace, Frobenius norm, effective rank, and alignment. Furthermore, we show that the positive-semidefinite structure of the NTK yields *one-sided* estimators that require only forward- or reverse-mode automatic differentiation. We validate these estimators across MLPs, recurrent GRUs, and a natural-language Transformer with up to 410 million parameters. In the latter model, the state is a high-dimensional four-tensor. We demonstrate orders-of-magnitude speedups, with the fastest estimator depending on the ratio of parameter and state dimensions. Equipped with these estimators, we examine rich and lazy RNN training using hidden-state NTK alignment and use NTK alignment as a regularizer for data-scarce knowledge distillation. We find that this regularization can modestly improve generalization, especially in very data-scarce settings. Together, these results make state-space NTK diagnostics practical even at large scales.

1 INTRODUCTION

The Neural Tangent Kernel (NTK) is a linear operator describing the local change in a model's state under gradient descent (GD) (Jacot et al., 2018). Classically, the NTK is defined on model outputs and is primarily studied via infinite-width limits (Mei et al., 2018; Yang & Hu, 2021; Bordelon et al., 2020). Here, we use the more general notion of an NTK defined on a prescribed state space and focus on computational tools that enable its quantification. For example, for an MLP, the state might contain activations at every layer over an empirical batch of inputs (Figure 1). The resulting state-space NTK can describe how *all* selected state variables evolve during training, not just the readout (Hazelden et al., 2026; Liu et al., 2024). State-space NTK statistics of interest include the trace, Frobenius norm, alignment, and effective rank, which respectively quantify overall tangent sensitivity, kernel magnitude and drift, and concentration of learning onto low-dimensional update directions (Xiang et al., 2025; Sharma et al., 2025; Liu et al., 2024; Shan & Bordelon, 2022; Geiger et al., 2019; Atanasov et al., 2021; Kumar et al., 2024; Nikolaou et al., 2025; Hazelden et al., 2026; Murray et al., 2023). These quantities can all be written using traces of the NTK or products of NTKs [§3.1].

The obstacle is computational. Explicitly constructing a finite-width NTK scales poorly with the dimension of the state, becoming especially prohibitive when it includes batches, recurrent timesteps, hidden units, Transformer blocks, or tokens. An RNN with 100 hidden units, evaluated on 100 inputs for 100 timesteps, already has a state with one million entries and an NTK with 10^{12} entries. We instead treat the NTK as a *matrix-free* operator and use randomized trace estimation to recover statistics. In particular, we apply Hutch++ (Meyer et al., 2021) to the NTK and its products, as well as deriving one-sided estimators relying only on forward- or reverse-mode automatic differentiation.

Across an MLP, a recurrent GRU, and the pretrained Pythia natural language Transformer (Biderman et al., 2023), these estimators recover near-exact NTK traces at a small fraction of the cost of exact state-basis evaluation. In the case of Pythia-410M, for example, the fastest estimator reaches 99% accuracy about $13{,}500$ times faster than exact evaluation [§5.2]. The same machinery efficiently

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

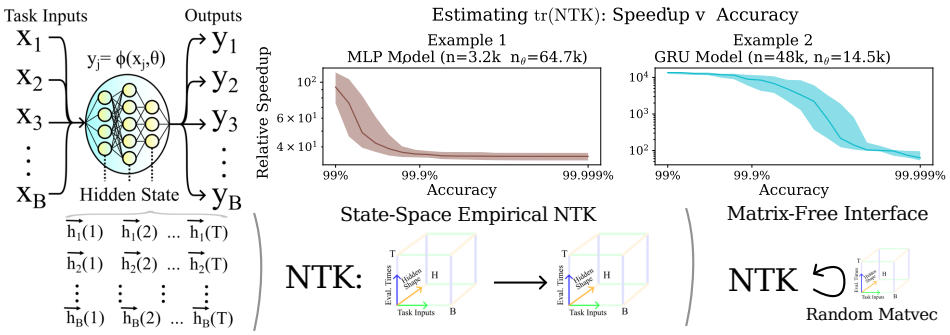

Figure 1: **Trace estimation of the state-space NTK.** The NTK acts on the full model state, which can form a high-dimensional tensor, and is accessed here only through randomized matrix-vector products (matvecs). For representative MLP and GRU examples (top right, see [§5]), the fastest of the three estimators discussed achieves 99% accuracy about 80 and 10,000 times faster than exact trace evaluation, respectively. Curves show medians over 50 runs with interquartile ranges.

estimates NTK norm, alignment, and effective rank, making these quantities practical as training-time diagnostics. We use this capability to track rich versus lazy recurrent learning at high temporal resolution and to regularize data-limited student networks toward a teacher's tangent geometry, improving held-out generalization.

Figure 1 provides a summary. Our *primary contributions* are

1. **Matrix-free NTK statistics.** We formulate fast estimation of finite-width NTK trace, Frobenius norm, alignment, and effective rank using Hutch++ and matrix-free NTK products and quantify their runtime-accuracy performance on MLP, GRU, and Pythia Transformer models up to 410M parameters [§4].
2. **One-sided trace estimation and scaling.** We derive reverse-only and forward-only NTK trace estimators and validate them on the same class of models [§5].
3. **Practical applications.** We use the estimators to study rich and lazy recurrent learning across stages of learning, and to impose teacher-student NTK alignment as a data-scarce regularizer, producing reproducible held-out generalization gains [§6].

2 Related Work

The Neural Tangent Kernel The Neural Tangent Kernel (NTK) was introduced by Jacot et al. Jacot et al. (2018) as the kernel governing the first-order evolution of network outputs under gradient descent. Subsequent work has developed two broad, complementary directions. Infinite-width theory studies regimes in which the NTK converges to a deterministic kernel and training approaches kernel regression, with extensions to general architectures, including recurrent and equilibrium models (Lee et al., 2019; Yang, 2020; Alemohammad et al., 2020; Feng & Kolter, 2023). In parallel, computational work has developed efficient implementations and approximations of infinite- and finite-width NTKs, using analytic recursions, structured Jacobian contractions, output-space approximations, and randomized sketches (Novak et al., 2019; Arora et al., 2019; Novak et al., 2022; Mohamadi et al., 2023; Zandieh et al., 2021). The canonical NTK acts on model outputs. Here, instead, we consider the empirical *state-space NTK*: the state h collects the model's internal variables across examples, layers, or time, and $\mathrm{NTK}_S = J_h J_h^\top$ acts directly on this tensor space. Once a loss specifies the state-space error signal, this operator closes the first-order learning dynamics at the level of the model state, exposing representation dynamics that are absent from an output-only view (Hazelden et al., 2026).

Operators in Deep Learning More broadly, our approach follows a growing view that very large structured matrices are often best represented through their *action* rather than explicitly materialized. Curvlinops applies this principle to Hessian, Fisher/GGN, and structured curvature matrices, exposing them through a common linear-operator interface (Dangel et al., 2025); this perspective

is closely connected to randomized numerical linear algebra, where low-rank structure and matrix statistics are recovered from matrix-vector products Martinsson & Tropp (2020). Related work by Townsend and collaborators studies recovery of infinite-dimensional operators and Green's functions from operator queries, including settings in which the adjoint is unavailable (Boullé et al., 2024). Trace estimation provides a particularly mature example: Hutchinson's estimator (Hutchinson, 1989) has been refined by Hutch++ (Meyer et al., 2021), adaptive and Nyström variants (Persson et al., 2022), XTrace (Epperly et al., 2024), and extensions to continuous implicit operators (Zvonek et al., 2025).

Finally, prior work, in the context of dynamical operator decompositions of the empirical NTK, briefly discussed Hutch++ for estimating the NTK trace (Hazelden et al., 2026). The present paper explains the underlying computational and mathematical structure, derives and analyzes new one-sided estimators, quantifies runtime improvements across varied models including pretrained Transformers, and studies new applications including the gradual emergence of rich vs lazy representations and NTK-informed knowledge distillation.

3 NTK Background

NTK Definition The empirical state-space Neural Tangent Kernel (NTK) is a linear operator defining the first-order dependence of parameterized state variables on their parameters. Crucially, it defines the geometry of gradient descent (GD) learning at each iteration to first order (Jacot et al., 2018). Consider a (typically tensor-valued) state dependent on parameters $\theta \in P$, $h = h(\theta) \in H$.

Throughout, we assume $\mathrm{vec}(h) \in \mathbb{R}^n$, the state, reduced to a vector, has n entries. Next, define the parameter Jacobian of this mapping $J_\theta := \frac{\mathrm{d}h}{\mathrm{d}\theta}$, which may be tensor-valued in general. Then, we can define two operations $\mathtt{vjp} : H \to P$, $\mathtt{jvp} : P \to H$ (vector-Jacobian and Jacobian-vector product, respectively). Specifically, given any tangent tensor $v \in H$ or $p \in P$ we define $\mathtt{vjp}(v) := J_\theta^T v$; $\mathtt{jvp}(p) := J_\theta p$. From this, $\mathtt{NTK} : H \to H$ is a linear operator defined by

$$\mathtt{NTK}(v) := \mathtt{jvp}(\mathtt{vjp}(v)) = J_\theta J_\theta^T v \quad \text{for any } v \in H \tag{1}$$

Importantly, J_θ is never explicitly constructed since this matrix can be prohibitively large. Performing $\mathtt{vjp}$ uses *reverse-mode* automatic differentiation (AD), and $\mathtt{jvp}$ uses *forward-mode* AD (Paszke et al., 2019). We note that sometimes the left and right Jacobian above differ (the cross-NTK), $J_\theta(x)J_\theta(x')^\top$, for different inputs x, x' (Lee et al., 2019). For simplicity, we assume the structure above, but Hutch++ (Algorithm 1) applies to this non-symmetric case immediately and the one-sided estimator (Algorithm 2) can be tweaked to apply, as detailed in Proposition 1 in the Appendix B.

3.1 Statistics Derived From the Trace

Here, we discuss a few statistics derived from the trace and motivated applications: the trace, alignment, effective rank and Frobenius norm. Each can be estimated with Hutch++, as well as with one-sided estimators, as discussed in Appendix B.

Trace The trace itself is of interest, since $\mathrm{tr}(\mathtt{NTK}) = \|J_\theta\|_F^2$, describing the overall sensitivity of the state to the parameters. Such a quantity, capturing total sensitivity, could be used for example to detect sudden changes in learning such as bifurcations, known as *critical slowing down*, or detecting emergent stiff parameters (Kramer & Ross, 1985; Anderson & Transtrum, 2023; Chen et al., 2026).

Alignment Between NTKs After a model is trained on a task, how much the NTK changes can quantify rich and lazy GD training regimes (Liu et al., 2024; Shan & Bordelon, 2022; Geiger et al., 2019; Atanasov et al., 2021; Kumar et al., 2024). This can be measured by how much spans overlap, specifically, given two NTKs with the same underlying state-space, $\mathtt{NTK}_1, \mathtt{NTK}_2 : H \to H$, their alignment is

$$\cos(\mathtt{NTK}_1, \mathtt{NTK}_2) := \frac{\mathrm{tr}(\mathtt{NTK}_1^T \mathtt{NTK}_2)}{\|\mathtt{NTK}_1\|_F \|\mathtt{NTK}_2\|_F} \in [0, 1] \tag{2}$$

Effective Rank The NTK effective rank can quantify how constrained and potentially low-dimensional GD updates are (Nikolaou et al., 2025; Hazelden et al., 2026; Murray et al., 2023).

This can be expressed through the *participation ratio* (PR),

$$r_{\text{eff}}(\texttt{NTK}) := \frac{\text{tr}(\texttt{NTK})^2}{\text{tr}(\texttt{NTK} \cdot \texttt{NTK}^T)} = n \cdot \cos^2(\texttt{NTK}, \text{Id}_H) \in [0, n] \tag{3}$$

Frobenius Norm The Frobenius norm of the NTK can be used to measure how well two operators align, e.g., through $\|\texttt{NTK}_1 - \texttt{NTK}_2\|_F$. Comparing such operators can also be used to measure how much two learning rules, corresponding to modified operators, align (Hazelden et al., 2026). The Frobenius norm of an NTK operator is simply $\|\texttt{NTK}\|_F^2 = \text{tr}(\texttt{NTK} \cdot \texttt{NTK}^\top)$.

4 METHODS

4.1 THE HUTCH++ TRACE ESTIMATOR

Throughout, we let $G : H \to H$ denote a generic linear operator on the state-space H. For any orthonormal basis $Q = (q_i)_{i=1}^n$ of the state-space H, the trace is $\text{tr}(G) := \sum_{i=1}^n q_i^T G q_i = \text{tr}(Q^T G Q)$. This expression provides the simplest brute-force direct approach to computing the exact trace, via performing n matvec evaluations with G. However, prior work has shown that far fewer matvecs can often be used to estimate $\text{tr}(G)$. A fast and simple approach for estimating this expression is Hutch++ (Meyer et al., 2021). Briefly, Hutch++ uses a randomized sketch of G to identify a low-dimensional subspace capturing much of the operator's action, then estimates the remaining trace contribution stochastically (see (Meyer et al., 2021) and (Epperly et al., 2024) for a detailed introduction). Crucially, modern analyses have shown that Hutch++ estimates $\text{tr}(G)$ for any G (not necessarily PSD) with variance $O(1/m^2)$ as the sample count, $m \to n$ (Epperly et al., 2024). Below is a summary of the Hutch++ algorithm for estimating an NTK trace.

Algorithm 1 Hutch++ NTK Trace Estimator

Input: Access to $\texttt{NTK}(v)$ matvec for all $v \in H$. Number of samples m
Output: Approximation to $\text{tr}(\texttt{NTK})$

1: Draw $S, T \in \mathbb{R}^{n \times \frac{m}{3}}$ with i.i.d. $\{+1, -1\}$ entries // Or $\mathcal{N}(0,1)$ entries
2: $Q \leftarrow \texttt{QR}(\texttt{NTK}(S))$ // Basis for range of $\texttt{NTK}$
3: $B \leftarrow (I - QQ^\top)T$ // Nullspace part of $\texttt{NTK}$ range
4: **return** $\text{tr}\left(\texttt{NTK}(Q)^\top Q\right) + \frac{3}{m}\text{tr}(\texttt{NTK}(B)^T B)$ // Low rank + residual estimates

4.2 ONE-SIDED NTK TRACE ESTIMATORS (RHUTCH / FHUTCH)

As we show below, we find Hutch++ to be a robust and efficient estimator for traces of the NTK and related quantities. However, evaluating the NTK requires both `jvp` and `vjp` access (forward- and backward-mode AD). Sometimes one of these AD modes is not implemented or may bottleneck the trace estimation, as discussed below. To overcome this, we introduce two alternative Hutchinson-style estimators of the trace that rely on a single, specified AD direction (forward or backward) (Hutchinson, 1989). Specifically, since the NTK has the form $JJ^\top$, the trace can be estimated as $\text{tr}(JJ^\top) = \mathbb{E}_v\left[\|J^\top v\|_2^2\right] = \mathbb{E}_p\left[\|Jp\|_2^2\right]$, where $v \in H, p \in P$ are isotropic with entries from $\mathcal{N}(0,1)$ in the state- or parameter-space, respectively. Thus, $\text{tr}(\texttt{NTK})$ can be estimated using only reverse-mode AD, ($G(v) = \texttt{vjp}(v)$, ***RHutch***) or forward-mode AD, ($G(p) = \texttt{jvp}(p)$, ***FHutch***). The algorithm below summarizes these estimators.

Algorithm 2 One-Sided NTK Trace Estimators

Input: Either define $G(v) = \texttt{vjp}(v)$ for any $v \in H$ and $d = \dim(H)$, or $G(p) := \texttt{jvp}(p)$ for any $p \in P$ and $d = \dim(P)$. Number of samples m.
Output: Approximation to $\text{tr}(\texttt{NTK})$

1: Draw $S, T \in \mathbb{R}^{d \times \frac{m}{2}}$ with i.i.d. $\{+1, -1\}$ entries // Or $\mathcal{N}(0,1)$
2: $Q \leftarrow \texttt{QR}(S)$ // Optional orthogonalization
3: $B \leftarrow (I - QQ^\top)T$ // Probes in the orthogonal complement
4: **return** $\|G(Q)\|_F^2 + \frac{2}{m}\|G(B)\|_F^2$ // Low rank + residual estimates

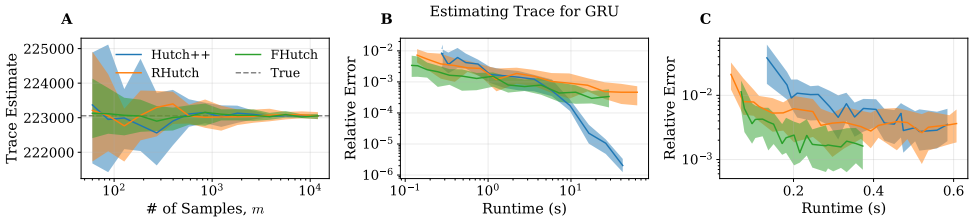

Figure 2: **Estimating tr(NTK) when $h(\theta)$ tracks the hidden unit activations of a GRU neural network at every evaluation time.** Consistent with Fig. 7, **A** illustrates the trace estimates for Hutch++ and the one-sided Hutch estimator (Algorithms 1 and 2). **B** illustrates relative error $\frac{|\text{tr}(\text{NTK})-t_m|}{|\text{tr}(\text{NTK})|}$ of each estimate, t_m versus runtime in seconds. In this case, exactly computing the trace with n matvecs took 1557.72 seconds (around 26 minutes). **C** shows the low-runtime region of B in more detail with linearly scaled runtime. In contrast to Fig. 7, FHutch performed fastest and both one-sided estimators outperformed Hutch++ for error above $2 \cdot 10^{-3}$. Hutch++ was fastest for accuracy above 99.9%.

Viability RHutch/FHutch sample different spaces and rely on a single AD mode, thus they can be beneficial when $\dim P$ is much larger than $\dim H$, or vice-versa. Also, for some architectures, GPU-accelerated implementations may only exist for one AD mode. The variance of both estimators scales as $O(1/m)$ (Hutchinson, 1989). For the higher $O(1/m^2)$ accuracy, we recommend Hutch++, but show RHutch/FHutch can be faster in the small m regime, based on the architecture (see [§5]). Technically, the estimator in Algorithm 2 builds upon standard Hutchinson, instead using Haar-orthogonal probes and a residual control-variate, similar to Hutch++ but without sketching the NTK (Epperly et al., 2024). We used this form, since in practice it typically outperformed plain Hutchinson in our numerical experiments (details in Appendix B).

	Pros	Cons
Hutch++	$O(1/m^2)$ variance guarantee and robust	May be bottlenecked by either forward- or backward-mode AD
One-Sided	Only either forward- or backward-AD	Worse $O(1/m)$ variance guarantee

4.2.1 One-Sided Variants of NTK Statistics

For consistency and enabling future directions, in Appendix [§B] we prove a Proposition that facilitates estimation of NTK products of the form $J_1 J_2^T J_3 J_4^T$ relying solely on either forward-mode or reverse-mode AD. We show that this enables one-sided estimation of the NTK norm, alignment, effective rank and cross-trace in the style of Algorithm 2. However, for the examples here, for which we are able to perform both forward-mode and reverse-mode AD, we find Hutch++ is simpler and generally outperforms these one-sided product estimators. Thus, the reported numerical experiments for these expressions below relied solely on Hutch++ [§3.1]. Variance reduction for these one-sided product estimators is an interesting direction for future work (discussed in [§7]).

5 Numerical Validation and Quantification of Speedup

Here, we validate the estimators, measuring runtime versus accuracy for a GRU and a language model. Additionally, we validate on the classical output-space NTK of an MLP in Appendix C.

5.1 NTK for the Hidden State of a Recurrent GRU

Motivation The NTK for recurrent models has been used to classify rich and lazy training regimes of recurrent models (Liu et al., 2024), characterize GD learning for linear RNNs (Bordelon et al., 2025) and explain low-dimensional learning of latent dynamics (Pellegrino et al., 2023; Brandon et al., 2026; Hazelden et al., 2026). However, the state dimension grows as $n = B \cdot T \cdot H$ over batches, time, and hidden or output units, so explicitly forming the NTK requires an $n \times n$ matrix. Faster matrix-free methods are therefore needed in the recurrent setting.

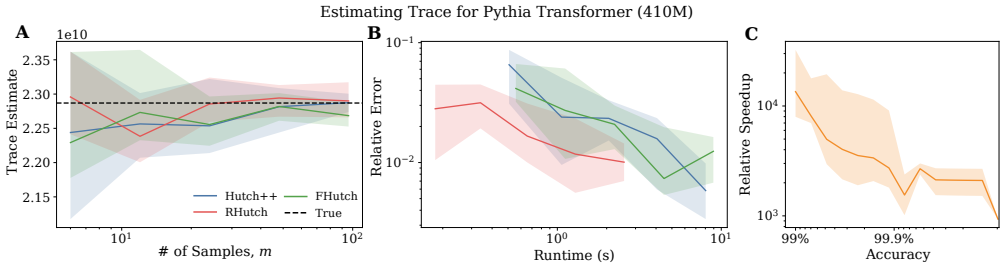

Figure 3: **Matrix-free estimation of tr(NTK) for the hidden-state representations of released Pythia Transformers.** For each model, $h(\theta)$ contains the hidden representations of all 16 tokens at the output of every Transformer block. **A** shows trace estimates for Hutch++, RHutch, and FHutch against randomized sample count m. **B** shows relative error $\frac{|t_m - \mathrm{tr}(\mathrm{NTK})|}{|\mathrm{tr}(\mathrm{NTK})|}$ versus runtime. **C** shows the corresponding speedup relative to exact state-basis trace evaluation. Curves show medians over 20 estimates, with shaded regions indicating the 25th and 75th percentiles.

Setup We let $h(\theta)$ track the hidden state of a GRU recurrent network (Cho et al., 2014). Similar to the MLP, we chose 64 hidden units, evaluated for $T = 15$ timesteps on each input. Also, we provided the network with 50 random normal 10-dimensional inputs at every timestep. Unlike the output of the MLP, tracking the hidden state here means $h(\theta)$ is a 3-tensor in $\mathbb{R}^{64\times15\times50}$, over *all* timesteps, not just the final layer, so it has $48,000$ entries. By contrast, since the GRU re-uses the weights for each layer, the parameter count is $14,592$. Hence, in this case $\dim(P) < \dim(H)$.

Results The results are summarized in Fig. 2. For reference, the full trace computation was very slow, taking about 26 minutes, or 1557.72 seconds. By contrast, 99% accuracy could be obtained in less than 0.1 seconds with FHutch. An accuracy of 99.99% (relative error 10^{-4}) took about 10 seconds with the Hutch++ estimator. Consistent with the prior findings, Hutch++ was fastest for high accuracy estimates, but the one-sided estimators could compete for low accuracy estimates. In contrast to the MLP output, however, here FHutch (using `jvp`) was faster than Hutch++ for errors above around 10^{-3}. This is consistent with our expectation, since the number of entries in $h(\theta)$ exceeds the parameter count in this case and RHutch and FHutch sample from the state and parameter spaces, respectively.

5.2 NTK for the Hidden States of Transformer Language Models

Setup We next test whether the estimators remain practical for large Transformer state spaces. We use released Pythia-14M and Pythia-410M-deduped checkpoints, two GPT-NeoX language models from the Pythia suite (Biderman et al., 2023). Pythia provides publicly available models from the same family across a wide range of scales, making it a convenient controlled benchmark. For a fixed 16-token input, we define the state h as the hidden representations of all tokens at the output of every Transformer block. This gives state dimensions $n = 12{,}288$ and $n = 393{,}216$ for Pythia-14M and Pythia-410M, respectively. Full architecture and implementation details are given in Appendix D.

Results Figure 3 shows that the advantage of randomized trace estimation grows sharply with the Transformer state dimension. Exact state-basis evaluation takes 101.06 seconds for Pythia-14M and 8,328.58 seconds for Pythia-410M. At the 99% accuracy target, the fastest randomized estimator is approximately 347 times faster for Pythia-14M and $13{,}499$ times faster for Pythia-410M. For the larger model, percent-level estimates require only seconds: at $m = 24$, RHutch reaches a median relative error of 1.66% in 0.66 seconds, while at $m = 96$, Hutch++ reaches 0.58% median error in 8.03 seconds. Thus, hidden-state NTK statistics remain practical even when the state spans hundreds of thousands of coordinates and hundreds of millions of differentiated parameters.

5.3 NTK Norm, Effective Rank and Alignment for MLP Trained on MNIST

Setup We next apply the Hutch++ method to estimate the three trace-related quantities in Section 3.1: Frobenius norm, alignment and effective rank. Specifically, we trained an MLP model on MNIST with a single hidden layer of 256 neurons and 10 output units, yielding about 97% test-set

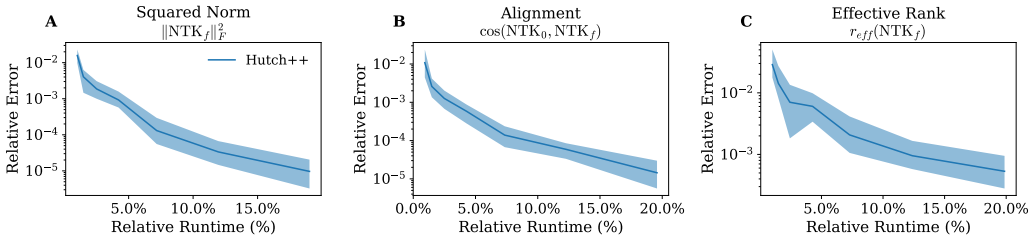

Figure 4: **Relative runtime versus error for an MLP trained on MNIST, estimating the Frobenius norm, kernel alignment and the effective rank, respectively.** Here, $\mathtt{NTK}_f$ denotes the NTK of the model post-training while $\mathtt{NTK}_0$ is the NTK pre-training. **A–C** plot the relative error versus the exact value, computed with exact trace evaluation. We plot these versus the percentage of total runtime required to compute the exact expression in each case (range 0% to 100%). Curves correspond to the median over 50 re-evaluations with Hutch++ and the shaded regions illustrate the 25th and 75th percentiles.

accuracy. We then evaluated the norm and effective rank of the NTK after training, $\mathtt{NTK}_f$, and alignment between the NTK before and after training ($\cos(\mathtt{NTK}_0, \mathtt{NTK}_f)$). We used a testing set of 512 MNIST examples, yielding $n = 5,120$.

Results We evaluated the three expressions with Hutch++ with results summarized in Fig. 4, sweeping the sample count m. There, we compare the estimation error to the relative runtime percentage (in the range 0-100), taken as the ratio of the median runtime over 50 re-runs of the estimator, divided by the time for evaluating the exact expression. In all three cases, we found Hutch++ attained 99% accuracy (relative error 10^{-2}) with $m = 60$ samples and in about 1% of the total runtime for the exact trace calculation, i.e., a 100 times speedup. A relative error of 10^{-4}, i.e., 99.99% accuracy, was attained 7.5% of the total runtime for the norm and alignment and around 25% of the runtime for the effective rank. In total, we found that Hutch++ can attain high accuracies for all three expressions in much shorter runtime.

6 Applications Enabled by Fast NTK Trace Estimation

6.1 Rich and Lazy Representation-Space Quantification

Motivation Rich and lazy learning differ in how strongly training changes a model's representation and NTK geometry (Jacot et al., 2018). The lazy regime is desirable from a theoretical perspective because negligible NTK drift makes gradient flow tractable to analyze, yielding kernel gradient descent (Chizat et al., 2019). More recent work generalized these notions to recurrent networks and showed that the rank of the initial recurrent weights, $\mathrm{rank}(W_0)$, modulates a transition from richer (low-rank) to lazier (high-rank) learning (Liu et al., 2024). Their hidden-state NTK analysis was limited by computation time, resulting in sparse measurements. Recent work has also emphasized that a simple rich-versus-lazy dichotomy can hide multiple phases of feature learning (Chou et al., 2025). Here, as a proof of utility, we extend Liu et al. (2024) with a denser sweep over training, more initialization ranks, and additional NTK statistics.

Setup We trained 300-unit RNNs on sequential MNIST while varying $\mathrm{rank}(W_0)$ over 20 values and 3 random seeds per rank. Each model was trained with the same SGD setup as in Liu et al. (2024). Rather than comparing only initialization and the end of training, we evaluated the hidden-state NTK at 101 checkpoints per run, giving 6,060 checkpoint-level measurements. At each checkpoint, the same matrix-free estimator bundle provided NTK alignment and effective rank, with three estimator repeats for each reported value. The full trace-dependent statistics (involving $54{,}900$ trace evaluations) were computed in 18.3 hours on a GTX Titan GPU. To separate geometric reorganization from differences in optimization speed, Figure 5 compares networks at matched levels of task performance rather than matched training time.

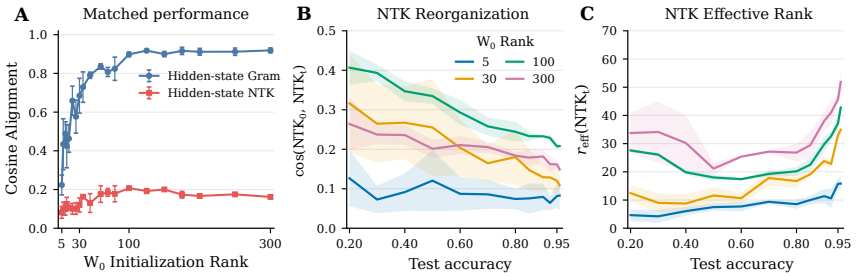

Figure 5: **Initialization-dependent rich and lazy reorganization of recurrent representations and NTK during learning on sequential MNIST.** As in (Liu et al., 2024), we swept the rank of the recurrent weights W_0 before training, with all models achieving above 95% testing accuracy. **A** Quantifying drift (rich learning) in the RNN hidden state and hidden-state NTK before and after training per run. Each dot represents a distinct training run. **B** Hidden-state NTK alignment over training as test accuracy increases for four representative RNN initializations. **C** Corresponding NTK effective rank, showing continued expansion of tangent geometry during learning. Curves and error bars correspond to the standard error of the mean (SEM) over 3 seeds per run.

Results Figure 5A recovers the broad rank-dependent rich-to-lazy trend reported in Liu et al. (2024), but at substantially higher resolution. Hidden-state representations become markedly more stable as initialization rank increases, while hidden-state NTK alignment shows a weaker dependence, with the strongest tangent reorganization concentrated at low rank. Panels **B–C** show why repeated measurements are useful: as task performance improves, the NTK continues to drift from its initialization and generally expands in effective rank. A single initial-to-final alignment therefore captures the overall rich-to-lazy trend but misses structure in the trajectory of tangent geometry.

6.2 Teacher NTK Alignment as a Data-Scarce Regularizer

Motivation The matrix-free construction developed above makes it inexpensive to compare tangent geometry during training. In particular, the runtime results for MLPs in Appendix C suggest that stochastic estimates of operator alignment can be incorporated directly into optimization at modest additional cost. As a second motivating example, we examine using the NTK as a regularizer during GD training, specifically as a source of additional geometric information in a data-limited knowledge distillation setting. This is motivated by recent work of Xiang et al. (2025) which used the output-space NTK as a regularizer for improving training of an optical neural network but did not examine generalization and was limited computationally, relying on estimating the NTK matrix (as in Wang et al. (2023)) to compute its statistics.

Setup We first train a teacher network on the full Fashion-MNIST training set. This is a single-hidden-layer MLP with 512 hidden units. We then independently initialize student networks of the same architecture using only $N \in \{250, 500, 1000\}$ training examples. Students minimize $\mathcal{L} = (1-\lambda)\mathcal{L}_{\text{CE}} + \lambda\,(1 - \cos(\texttt{NTK}_S, \texttt{NTK}_T))$, where $\texttt{NTK}_S$ and $\texttt{NTK}_T$ denote student and teacher NTK operators evaluated on the student training data. Alignment is estimated entirely matrix-free using Hutch++ with 12 probes. We consider both output-logit and hidden-state NTKs, and compare against otherwise identical students trained with cross-entropy alone ($\lambda = 0$). All students reach essentially 100% training accuracy, allowing us to isolate differences in generalization among interpolating solutions.

Results The Table below and Figure 6 summarize the results. NTK matching yields modest generalization gains, strongest in the very data-scarce setting, suggesting that it could be used in future directions to impart additional geometrical structure in knowledge distillation.

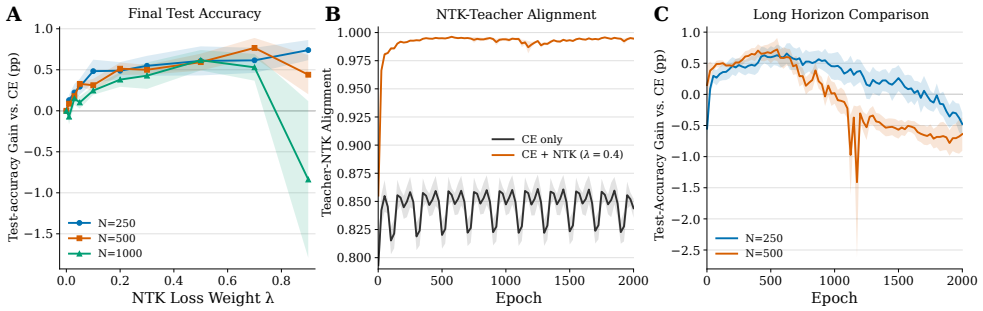

Figure 6: **Teacher-student NTK alignment as a regularizer for data-limited Fashion-MNIST students.** A student is trained on a pretrained teacher's output with limited training data and an NTK regularization term with factor λ. **A** Final test-accuracy gain relative to matched cross-entropy-only students as a function for training set size $N \in \{250, 500, 1000\}$. Positive values indicate improved generalization. **B** Teacher-NTK alignment during training for cross-entropy-only and NTK-regularized students ($\lambda = 0.4$), showing rapid and sustained alignment with the teacher geometry. **C** Test-accuracy gain relative to cross-entropy-only training over a longer training horizon for $N = 250$ and $N = 500$. Shaded regions in A and C correspond to SEM over reruns.

NTK Space	λ	N	Δ test acc. (pp)	Paired wins
Logits	0.7	250	$+0.62 \pm 0.15$	$5/6$
Logits	0.7	500	$+0.77 \pm 0.12$	$6/6$
Logits	0.7	1000	$+0.53 \pm 0.16$	$6/6$
Hidden	0.5	250	$+0.98 \pm 0.22$	$3/3$
Hidden	0.5	500	$+0.49 \pm 0.23$	$3/3$
Hidden	0.5	1000	-0.21 ± 0.16	$1/3$

7 Discussion

Finite-width NTK analysis can be prohibitively expensive, especially for state-space NTKs spanning batches, hidden units, timesteps, layers, or tokens. We show that useful quantities including the trace, Frobenius norm, alignment, and effective rank can instead be estimated from matrix-free NTK products using randomized trace estimation. We also introduce one-sided estimators using only forward- or reverse-mode AD, which can be advantageous at coarse tolerances or when one AD direction is cheaper. Across MLPs, recurrent networks, and Transformers, these methods make finite-width NTK statistics practical at scales where direct evaluation is expensive.

This enables repeated NTK measurements during training rather than only isolated endpoint calculations. We use this to track tangent-geometry changes during rich versus lazy recurrent learning and to include teacher-student NTK alignment directly in a training objective, where it modestly improves generalization in data-scarce settings.

Limitations and Future Directions The one-sided estimators do not exploit the NTK's low-rank output space as effectively as Hutch++, so their advantage is largely confined to coarse accuracy regimes. Output-informed one-sided estimators are a natural future direction. Partial trace estimation is another promising extension, allowing tensor-valued NTKs to be reduced while retaining temporal, spatial, or feature structure (Chen et al., 2024; Xu & Zhu, 2024; Hazelden et al., 2026).

Code Availability The NTK and related operators are implemented in a matrix-free way in our package `https://anonymous.4open.science/r/gentk-47B8`. Timing figures can be reproduced by running `experiments/trace_paper/main.py`.

LLM Usage Statement Large language models were used for assistance with experimental code (specifically, the two applications in Section 6) and manuscript editing. The methodology, experimental design, core implementation and testing, and validation were developed by the authors without LLMs.

REFERENCES

Saman Alemohammad, Zhuang Wang, Remi Balestriero, and Richard Baraniuk. The recurrent neural tangent kernel. *Advances in Neural Information Processing Systems*, 33, 2020.

Christian N. K. Anderson and Mark K. Transtrum. Sloppy model analysis identifies bifurcation parameters without normal form analysis. *Physical Review E*, 108(6):064215, 2023. doi: 10.1103/PhysRevE.108.064215.

Sanjeev Arora, Simon S. Du, Wei Hu, Zhiyuan Li, Ruslan Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://arxiv.org/abs/1904.11955.

Alexander Atanasov, Blake Bordelon, and Cengiz Pehlevan. Neural networks as kernel learners: The silent alignment effect, 2021. URL https://arxiv.org/abs/2111.00034.

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. Pythia: A suite for analyzing large language models across training and scaling, 2023. URL https://arxiv.org/abs/2304.01373.

Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. Gpt-neox-20b: An open-source autoregressive language model, 2022. URL https://arxiv.org/abs/2204.06745.

Blake Bordelon, Abdulkadir Canatar, and Cengiz Pehlevan. Spectrum dependent learning curves in kernel regression and wide neural networks. In *International Conference on Machine Learning*, pp. 1024–1034. PMLR, 2020.

Blake Bordelon, Jordan Cotler, Cengiz Pehlevan, and Jacob A. Zavatone-Veth. Dynamically learning to integrate in recurrent neural networks, 2025. URL https://arxiv.org/abs/2503.18754.

Nicolas Boullé, Diana Halikias, Samuel E. Otto, and Alex Townsend. Operator learning without the adjoint. *Journal of Machine Learning Research*, 25(364):1–54, 2024. URL https://www.jmlr.org/papers/v25/24-0162.html.

Julian Brandon, Bruno Loureiro, N Alex Cayco Gajic, and Arthur Pellegrino. Mode collapse emerges from low-rank biases in the learning dynamics of generative models. In *High-dimensional Learning Dynamics 2026*, 2026. URL https://openreview.net/forum?id=sp206XLwHr.

Sida Chen, Siqi Yang, Zhao Chang, Taro Toyoizumi, Werner Sommer, Lianchun Yu, Qian-Yuan Tang, and Changsong Zhou. Beyond variance: sensitivity-based dimensions in brain networks underlie individual differences in cognitive ability, 2026. URL https://arxiv.org/abs/2501.19106.

Tyler Chen, Robert Chen, Kevin Li, Skai Nzeuton, Yilu Pan, and Yixin Wang. Faster randomized partial trace estimation. *SIAM Journal on Scientific Computing*, 46(6):A3427–A3447, November 2024. ISSN 1095-7197. doi: 10.1137/23M1620399. URL http://dx.doi.org/10.1137/23M1620399.

Lénaïc Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Kyunghyun Cho, Bart Van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Chi-Ning Chou, Hang Le, Yichen Wang, and SueYeon Chung. Feature learning beyond the lazy-rich dichotomy: Insights from representational geometry, 2025. URL https://arxiv.org/abs/2503.18114.

Felix Dangel, Runa Eschenhagen, Weronika Ormaniec, Andres Fernandez, Lukas Tatzel, and Agustinus Kristiadi. Position: Curvature matrices should be democratized via linear operators, 2025. URL https://arxiv.org/abs/2501.19183.

Ethan N. Epperly, Joel A. Tropp, and Robert J. Webber. Xtrace: Making the most of every sample in stochastic trace estimation. *SIAM Journal on Matrix Analysis and Applications*, 45(1):1–23, 2024. doi: 10.1137/23M1548323. URL https://doi.org/10.1137/23M1548323.

Zhili Feng and J. Zico Kolter. On the neural tangent kernel of equilibrium models. *arXiv preprint arXiv:2310.14062*, 2023.

Mario Geiger, Stefano Spigler, Arthur Jacot, and Matthieu Wyart. Disentangling feature and lazy learning in deep neural networks: an empirical study. *CoRR*, abs/1906.08034, 2019. URL http://arxiv.org/abs/1906.08034.

James Hazelden, Laura Driscoll, Eli Shlizerman, and Eric Shea-Brown. The global empirical ntk: Self-referential bias and dimensionality of gradient descent learning, 2026. URL https://arxiv.org/abs/2605.08746.

M. F. Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, 18(3):1059–1076, 1989. doi: 10.1080/03610918908812806.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in Neural Information Processing Systems*, 31, 2018.

Jesse Kramer and John Ross. Stabilization of unstable states, relaxation, and critical slowing down in a bistable system. *The Journal of Chemical Physics*, 83(12):6234–6241, 1985.

Tanishq Kumar, Blake Bordelon, Samuel J. Gershman, and Cengiz Pehlevan. Grokking as the transition from lazy to rich training dynamics, 2024. URL https://arxiv.org/abs/2310.06110.

Jaehoon Lee, Lechao Xiao, Samuel S. Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *Advances in Neural Information Processing Systems*, 2019.

Yuhan Helena Liu, Aristide Baratin, Jonathan Cornford, Stefan Mihalas, Eric Todd Shea-Brown, and Guillaume Lajoie. How connectivity structure shapes rich and lazy learning in neural circuits. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=slSmYGc8ee.

Per-Gunnar Martinsson and Joel A. Tropp. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numerica*, 29:403–572, 2020. doi: 10.1017/S0962492920000021.

Song Mei, Tomasz Misiakiewicz, and Andrea Montanari. Mean-field theory of two-layer neural networks: dimension-free bounds and kernel limit. *Advances in Neural Information Processing Systems*, 31, 2018.

Raphael A. Meyer, Cameron Musco, Christopher Musco, and David P. Woodruff. Hutch++: Optimal stochastic trace estimation. In *Symposium on Simplicity in Algorithms (SOSA)*, pp. 142–155. SIAM, 2021. doi: 10.1137/1.9781611976496.16. URL https://arxiv.org/abs/2010.09649.

Mohamad Amin Mohamadi, Wonho Bae, and Danica J. Sutherland. A fast, well-founded approximation to the empirical neural tangent kernel. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 25011–25035, 2023.

Michael Murray, Hui Jin, Benjamin Bowman, and Guido Montúfar. Characterizing the spectrum of the ntk via a power series expansion. In *International Conference on Learning Representations (ICLR) 2023*, 2023. URL https://openreview.net/forum?id=Tvms8xrZHyR. Poster.

Konstantin Nikolaou, Sven Krippendorf, Samuel Tovey, and Christian Holm. Beyond scaling curves: Internal dynamics of neural networks through the ntk lens. *arXiv preprint arXiv:2507.05035*, 2025. URL https://arxiv.org/abs/2507.05035.

Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A. Alemi, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. Neural tangents: Fast and easy infinite neural networks in python, 2019. URL https://arxiv.org/abs/1912.02803.

Roman Novak, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. Fast finite width neural tangent kernel. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 17018–17044. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/novak22a.html.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. URL https://arxiv.org/abs/1912.01703.

Arthur Pellegrino, N. Alex Cayco Gajic, and Angus Chadwick. Low tensor rank learning of neural dynamics. In *Advances in Neural Information Processing Systems*, volume 36, pp. 11674–11702, 2023.

David Persson, Alice Cortinovis, and Daniel Kressner. Improved variants of the Hutch++ algorithm for trace estimation. *SIAM Journal on Matrix Analysis and Applications*, 43(3):1162–1185, 2022. doi: 10.1137/21M1447623.

Haozhe Shan and Blake Bordelon. A theory of neural tangent kernel alignment and its influence on training, 2022. URL https://arxiv.org/abs/2105.14301.

Arnav Sharma, Ahmed Wez, and Karthik Srikumar. On the relationship between neural tangent kernel frobenius distance and distillation sample complexity. In *Workshop on Lock-LLM at Neural Information Processing Systems (NeurIPS 2025)*, 2025. URL https://openreview.net/pdf/89e812786af493b2305487f7a6a1689d41b324ec.pdf. Poster.

Yite Wang, Dawei Li, and Ruoyu Sun. Ntk-sap: Improving neural network pruning by aligning training dynamics, 2023. URL https://arxiv.org/abs/2304.02840.

Jinlin Xiang, Minho Choi, Yubo Zhang, Zhihao Zhou, Arka Majumdar, and Eli Shlizerman. Neural tangent knowledge distillation for optical convolutional networks, 2025. URL https://arxiv.org/abs/2508.08421.

Jiaming Xu and Hanjing Zhu. Overparametrized multi-layer neural networks: Uniform concentration of neural tangent kernel and convergence of stochastic gradient descent. *Journal of Machine Learning Research*, 25(1):1–83, 2024. URL https://jmlr.org/papers/v25/23-0740.html.

Greg Yang. Tensor programs ii: Neural tangent kernel for any architecture. *arXiv preprint arXiv:2006.14548*, 2020.

Greg Yang and Edward J. Hu. Tensor programs iv: Feature learning in infinite-width neural networks. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 11727–11737. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/yang21c.html.

Amir Zandieh, Insu Han, Haim Avron, Neta Shoham, Chaewon Kim, and Jinwoo Shin. Scaling neural tangent kernels via sketching and random features, 2021. URL https://arxiv.org/abs/2106.07880.

Jennifer Zvonek, Andrew J. Horning, and Alex Townsend. ContHutch++: Stochastic trace estimation for implicit integral operators. *SIAM Journal on Numerical Analysis*, 63(1):334–359, 2025. doi: 10.1137/23M1614365.

APPENDIX

A IMPLEMENTATION DETAILS

All numerical experiments were implemented in PyTorch (Paszke et al., 2019). Unless stated otherwise, the smaller MLP and recurrent validation experiments were run on a CPU. The Pythia experiments were run on an NVIDIA L40S GPU, as described in [§5.2]. We compute the exact trace by applying the NTK to the n standard basis vectors $e_i \in H$, without explicitly storing the full NTK. Where batching is used, these products are accelerated with `torch.vmap`. As a runtime sanity check, Hutch++ with $m = n$ matvec evaluations typically took 80–120% of the direct estimator runtime, giving a speedup close to $1\times$. This suggests that the reported speedups are not artifacts of substantially different implementations.

For the Pythia experiment, products were vectorized with chunks of size 8 on the NVIDIA L40S GPU. We compared Hutch++, RHutch, and FHutch with $m \in \{6, 12, 24, 48, 96\}$ using 20 independent repeats for each case.

A.1 HUTCH++ IMPLEMENTATION DETAILS

The allocation of the total probe budget between the range sketch and residual estimator is a tunable hyperparameter. Algorithm 1 uses the simple equal allocation $q = r = m/3$, so the total number of NTK products is $2q + r = m$. The numerical experiments use the experiment-specific allocations stated below; these do not change the Hutch++ decomposition.

B ONE-SIDED NTK PRODUCT ESTIMATOR

In the main text, we show in Algorithm 2 that the NTK trace can be approximated with a Hutchinson estimator using only one AD mode (reverse, `vjp`, or forward, `jvp`). Here, we show this extends to products of NTKs, yielding one-sided estimators for the norm, alignment, effective rank, as well as the case where $\texttt{NTK} = J_1 J_2^T$ with distinct Jacobians on the left and right side (e.g., evaluated on the testing and training set, respectively). This result is summarized in the following Proposition.

Proposition 1. Let $J_1, J_2, J_3, J_4 : P \to S$ be operators mapping the parameter space to the state space. Then, letting $u, v \in H$ be independent random vectors such that $\mathbb{E}[u] = \mathbb{E}[v] = 0$ and $\mathbb{E}[uu^\top] = \mathbb{E}[vv^\top] = I$ (e.g., uniform $\{+1, -1\}$ or $\mathcal{N}(0, 1)$ entries). Then,

$$\operatorname{tr}\big(J_1 J_2^\top J_4 J_3^\top\big) = \mathbb{E}\big[\big\langle J_1^\top u,\, J_2^\top v\big\rangle \cdot \big\langle J_4^\top u,\, J_3^\top v\big\rangle\big]\,,$$

yielding a reverse-mode only estimator. Likewise, probing in the parameter space: if $p, q \in P$ denote random vectors with $\mathbb{E}[p] = \mathbb{E}[q] = 0$, $\mathbb{E}[pp^\top] = \mathbb{E}[qq^\top] = I$, then

$$\operatorname{tr}\big(J_1 J_2^\top J_3 J_4^\top\big) = \mathbb{E}[\langle J_4 p,\, J_1 q\rangle \cdot \langle J_3 p,\, J_2 q\rangle]\,,$$

yielding a forward-mode only estimator.

Proof:
Let $A := J_1 J_2^T$, $B := J_3 J_4^T$, so we want to estimate $\operatorname{tr}(AB)$. By the standard Hutchinson estimator (Hutchinson, 1989), assuming $u, v \sim H$ denote random variables as in the proposition and expand

$$\begin{aligned}
\mathbb{E}\big[\big\langle J_1^\top u,\, J_2^\top v\big\rangle \cdot \big\langle J_4^\top u,\, J_3^\top v\big\rangle\big] &= \mathbb{E}\big[\big\langle J_1^\top u,\, J_2^\top v\big\rangle \cdot \big\langle J_3^\top v,\, J_4^\top u\big\rangle\big] \\
&= \mathbb{E}[u^T J_1 J_2^T v v^T J_3 J_4^T u] \\
&= \mathbb{E}[u^T A v v^T B u]
\end{aligned}$$

Marginalizing over u, we have

$$\mathbb{E}[u^T A v v^T B u | u] = u^T A \mathbb{E}[v v^T] B u,$$

using the assumptions thus yields

$$\mathbb{E}[u^T A v v^T J_4 J_3^T u | u] = u^T A B u$$

So,

$$\mathbb{E}[u^T A v v^T B^T u] = \mathbb{E}[u^T A B^T u] = \text{tr}(AB).$$

This proves the reverse-mode estimator. For the forward mode, we have

$$\text{tr}(J_1 J_2^T J_3 J_4^T) = \text{tr}(J_4^T J_1 J_2^2 J_3) \tag{4}$$

and apply the same theorem to the matrices $\hat{J}_1 = J_4^T, \hat{J}_2 = J_1^T, \hat{J}_3 = J_2^T, \hat{J}_4 = J_3^T$, yielding

$$\begin{aligned}
\text{tr}(J_1 J_2^T J_3 J_4^T) &= \text{tr}(\hat{J}_1 \hat{J}_2^T \hat{J}_3 \hat{J}_4^T) && (5)\\
&= \mathbb{E}\Big[\Big\langle \hat{J}_1^\top p,\ \hat{J}_2^\top q \Big\rangle \cdot \Big\langle \hat{J}_4^\top p,\ \hat{J}_3^\top q \Big\rangle\Big] && (6)\\
&= \mathbb{E}[\langle J_4 p,\ J_1 q\rangle \cdot \langle J_3 p,\ J_2 q\rangle] && (7)
\end{aligned}$$

yielding the forward-mode estimator. Note implicitly here that the domains are different: p, q are samples from the parameter space P while u, v are samples from the H space $\square$

From this, we can estimate the NTK norm,

$$\begin{aligned}
\|\texttt{NTK}\|_F^2 &= \text{tr}(\texttt{NTK} \cdot \texttt{NTK}^T) && (8)\\
&= \mathbb{E}\big[\langle \texttt{vjp}(u), \texttt{vjp}(v)\rangle^2\big] = \mathbb{E}\big[\langle \texttt{jvp}(p), \texttt{jvp}(q)\rangle^2\big], && (9)
\end{aligned}$$

where the u, v form is the reverse-mode estimate and the p, q form is the forward-mode estimate.

Likewise, the alignment. Using the notation $\texttt{NTK}_i(v) = \texttt{jvp}_i(\texttt{vjp}_i(v))$ for $i = 1, 2$, Proposition 1 yields the following expression

$$\begin{aligned}
\cos(\texttt{NTK}_1, \texttt{NTK}_2) &= \frac{\mathbb{E}[\langle \texttt{vjp}_1(u), \texttt{vjp}_1(v)\rangle \langle \texttt{vjp}_2(u), \texttt{vjp}_2(v)\rangle]}{\sqrt{\mathbb{E}[\langle \texttt{vjp}_1(u), \texttt{vjp}_1(v)\rangle^2]\,\mathbb{E}[\langle \texttt{vjp}_2(u), \texttt{vjp}_2(v)\rangle^2]}} && (10)\\
&= \frac{\mathbb{E}\big[\langle \texttt{jvp}_2(p), \texttt{jvp}_1(q)\rangle^2\big]}{\sqrt{\mathbb{E}[\langle \texttt{jvp}_1(p), \texttt{jvp}_1(q)\rangle^2]\,\mathbb{E}[\langle \texttt{jvp}_2(p), \texttt{jvp}_2(q)\rangle^2]}} && (11)
\end{aligned}$$

Finally, the effective rank is given by

$$r_{eff}(\texttt{NTK}) = \frac{\mathbb{E}[\|\texttt{vjp}(u)\|_2^2]}{\mathbb{E}[\langle \texttt{vjp}(u), \texttt{vjp}(v)\rangle^2]} = \frac{\mathbb{E}[\|\texttt{jvp}(p)\|_2^2]}{\mathbb{E}[\langle \texttt{jvp}(p), \texttt{jvp}(q)\rangle^2]} \tag{12}$$

We found that the one-sided trace estimator could be faster than Hutch++ for trace predictions with low accuracy (around 99%). However, we find for now that the one-sided product estimators detailed here are typically slower than Hutch++ in all cases. Indeed, they rely on sampling independently $u, v \sim H$ or $p, q \sim P$ without any information about the NTK column space. Incorporating information about the NTK into these estimates, as in Hutch++, could be a future direction for improving the runtime of the one-sided estimators. We provide these here for future reference and directions.

C NTK FOR THE OUTPUT OF A DEEP MLP

Setup While the main focus of our paper is on the generalized NTK describing updates in a model's state space, here we give an example in the classical NTK framework, in which the NTK is computed for updates in a model's outputs (Jacot et al., 2018). Here, we use a deep Multi-Layer Perceptron (MLP) neural network. We use 15 layers with 64 hidden units per layer. As input to the network, we feed 50 distinct Gaussian 100-dimensional inputs into the first layer of the MLP, $x \sim \mathcal{N}(0, 1)^{100}$. This simple setup is chosen primarily to measure the validity of the trace estimators as runtime versus error, compared to the direct approach of computing the exact NTK trace with n matvecs. In this MLP case, the state $h(\theta)$ reflects all activations of the output layer of the network, hence $n = 3,200 = 50 \cdot 64$. The parameter count, $\dim P$, is $64,704$ in this case.

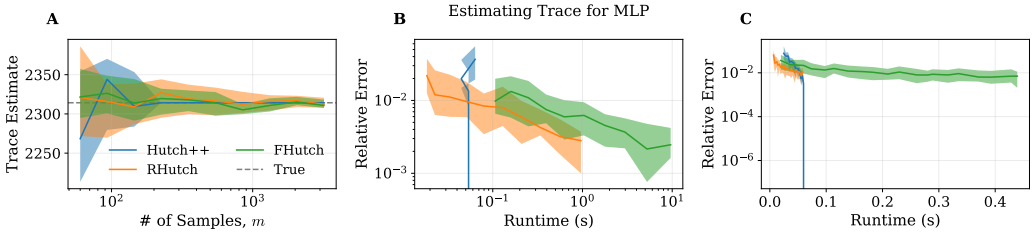

Figure 7: **Estimating $\mathrm{tr}(\texttt{NTK})$ when $h(\theta)$ tracks the output of an MLP neural network.** **A** illustrates the trace estimates for the three approaches in [§4]: Hutch++ (Algorithm 1) and the one-sided estimator (Algorithm 2) with forward- and backward-AD (FHutch and RHutch, respectively). **B** shows the relative error $\frac{|\mathrm{tr}(\texttt{NTK}) - t_m|}{|\mathrm{tr}(\texttt{NTK})|}$ of each estimate, t_m versus runtime in seconds. For reference, exactly computing the trace took 1.96 seconds. **C** zooms into the low-runtime region of B and displays results with linearly scaled runtime, showing that FHutch is the slowest for this setup. Note Hutch++ obtained relative error below 10^{-6} in about 0.06 seconds (about $33\times$ faster than the exact trace).

Results Fig. 7 summarizes our results for this architecture. RHutch and FHutch denote the one-sided trace estimator in Algorithm 2 using $G = \texttt{vjp}$ and $G = \texttt{jvp}$, respectively. Evaluating the full trace with $3,200$ matvecs required around 1.96 seconds. By contrast, the Hutch++ estimator achieved relative error below 10^{-6} in about 0.06 seconds, about $33\times$ faster, using about 600 matvecs. Interestingly, for small sample counts, $m < 100$, the RHutch estimator using reverse-mode AD achieved the best accuracy in the fastest time. In this typical deep-learning setup, the parameter count is much higher than the state count. Thus, in this case RHutch samples a smaller space (H) than $\texttt{jvp}$ (P). In many applications of the NTK, estimates of the trace with relative error 10^{-2} (99% accurate) are acceptable, so using the one-sided estimator could provide a fast way to compute the NTK trace in this circumstance. However, if high accuracy is desired, we find the Hutch++ estimator always performs best. Finally, note in this case that the forward-mode estimator was slowest because it samples in the full parameter space, P, not the MLP output space H.

Summary To summarize, if high accuracy is required, Hutch++ is fastest. RHutch can be faster in the low-accuracy regime provided $\dim(H) < \dim(P)$. By contrast, FHutch can be faster in this regime when $\dim(P) < \dim(H)$. The next experiment illustrates the latter case.

D Pythia Experiments and the Transformer State-Space NTK

D.1 Models and Background

Here, we provide further details and background on the Transformer experiments in Section 5.2. The goal is to compare the present randomized NTK estimators to direct trace computation on realistic Transformer state spaces. We use the released models `EleutherAI/pythia-14m` and `EleutherAI/pythia-410m-deduped`, both at step 143000, from the Pythia suite (Biderman et al., 2023). Using two checkpoints from the same released model family provides a controlled comparison across substantially different state and parameter dimensions. Both models use GPT-NeoX Transformer blocks (Black et al., 2022).

Each Transformer block contains causal self-attention and a feed-forward network, with normalization and residual connections. Table 1 lists the architecture dimensions. The models are evaluated in `eval` mode with key/value caching disabled. Float32 is used for both the parameters and automatic differentiation.

We utilize the statement "Neural tangent kernels describe how parameter updates change the output of a neural network." as the fixed prompt for these Pythia Transformer experiments. We tokenize without special tokens and use the first 16 token IDs. We repeat the tokenized prompt if a larger sequence of tokens is requested. An all-ones attention mask is used, and GPT-NeoX supplies the causal mask. All tokens enter one forward pass: this experiment does not autoregressively generate

16 new tokens. No labels, train-test splits, or task loss are required to evaluate the trace at this fixed input.

	Pythia-14M	Pythia-410M-deduped
Blocks L	6	24
Hidden width d	128	1,024
Attention heads	4	16
Feed-forward width	512	4,096
Sequences B / token positions T	$1 \,/\, 16$	$1 \,/\, 16$
State dimension $n = LBTd$	12,288	393,216
Differentiated parameters p	7,628,800	353,822,720

Table 1: Dimensions of the numerical validation experiment where all blocks and 16 tokens are used. The parameter count excludes the vocabulary output projection.

D.2 The tensor-valued state, the Jacobian, and the NTK

Let $H^{(\ell)}(\theta; x) \in \mathbb{R}^{B\times T\times d}$ denote the hidden representation returned for Transformer block ℓ. In our code, by default, we stack the returned representations, excluding the initial embedding representation. Thus we have

$$\mathcal{H}(\theta; x) = \operatorname{stack}_{\ell=1}^{L} H^{(\ell)}(\theta; x) \in \mathbb{R}^{L\times B\times T\times d}, \qquad h(\theta; x) = \operatorname{vec} \mathcal{H}(\theta; x) \in \mathbb{R}^{n}. \tag{13}$$

Before vectorization, the state is a four-dimensional tensor. We can also use a different ordering for the axes. The intermediate returned hidden states are block outputs. The final returned hidden state includes the final layer normalization. "All blocks" refers to all the hidden states at the end of each transformer block and not every attention layer or feed-forward output. The saved benchmark stores all 16 token positions. We do not measure vocabulary logits or token probabilities.

The input x is held fixed across all estimator repeats for a given model. With parameter scope `all`, θ contains the input embedding table, all the Transformer blocks, and the final layer-normalization parameters. The vocabulary projection is excluded because the chosen hidden state does not depend on it. This is why p is smaller than the nominal model size. Including a parameter in this scope does not guarantee a non-zero derivative on the chosen sentence. We can change the selected state blocks and the differentiated parameter scope. After flattening the selected parameter tensors, we obtain

$$J(\theta; x) = \frac{\partial h(\theta; x)}{\partial \theta} \in \mathbb{R}^{n\times p}, \qquad \mathtt{NTK}(\theta; x) = JJ^{\top}. \tag{14}$$

The unflattened Jacobian has four axes and one flattened parameter axis. For two state coordinates $a = (\ell, b, t, c)$ and $a' = (\ell', b', t', c')$, we have

$$\mathtt{NTK}_{a,a'} = \sum_{j=1}^{p} \frac{\partial \mathcal{H}_a}{\partial \theta_j} \frac{\partial \mathcal{H}_{a'}}{\partial \theta_j}. \tag{15}$$

The NTK includes cross-token and cross-layer blocks. The trace measures the total squared parameter sensitivity of the selected hidden states, but does not compute the alignment between different state coordinates' parameter gradients.

D.3 Trace computation

This subsection details the exact trace computation and randomized estimates. The NTK is detailed in Section 3. The exact trace used the following formula and did not require constructing the Jacobian or NTK.

$$\operatorname{tr}(\mathtt{NTK}) = \sum_{i=1}^{n} e_i^T \mathtt{NTK} e_i = ||J||_F^2$$

The implementation used VJPs and JVPs to compute each summand in the summation above. There is one complete evaluation per model. The saved runs use chunks of 8 basis vectors and

`torch.vmap` within each chunk. While the products are computed in float32, the summation occurs in float64.
The estimators are detailed in Section 4.

For Hutch++ (Meyer et al., 2021), the more general notation is $S \in \mathbb{R}^{n\times q}$ and $G \in \mathbb{R}^{n\times r}$. Define

$$Q = \mathrm{orth}(\mathtt{NTK}S), \quad V = (I - QQ^\top)G, \quad \widehat{t}_{++} = \mathrm{tr}(Q^\top \mathtt{NTK} Q) + \frac{1}{r}\mathrm{tr}(V^\top \mathtt{NTK} V). \tag{16}$$

For the Pythia runs, the command-line budget m is split as $q = \lfloor m/6 \rfloor$ and $r = \lfloor (m-q)/2 \rfloor$. Thus $m \in \{6, 12, 24, 48, 96\}$ corresponds to $2q + r \in \{4, 9, 18, 36, 72\}$ actual NTK actions, each comprising a VJP and a JVP.

Here q is the number of sketch columns, r the number of residual probes, and Q the resulting orthonormal basis matrix. They are distinct quantities. The pseudocode 1 uses the valid equal allocation $q = r = m/3$, where we assume m is divisible by 3. It uses $2q + r = m$ NTK products and a residual multiplier $1/r = 3/m$. Equal allocation is not required for the Hutch++ decomposition. For example, if $m = 24$, we use $q = r = 8$ in the pseudocode and $q = 4, r = 10$ in the code. The floor operations allow arbitrary command-line budgets; we do not claim that this split is optimal.

One-sided estimators and their sample counts. The Pythia one-sided estimators use independent Rademacher probes defined as

$$\widehat{t}_R = \frac{1}{m}\sum_{i=1}^{m} \|J^\top z_i\|_2^2, \qquad z_i \in \mathbb{R}^n, \tag{17}$$

$$\widehat{t}_F = \frac{1}{m}\sum_{i=1}^{m} \|J w_i\|_2^2, \qquad w_i \in \mathbb{R}^p. \tag{18}$$

The identities $\mathbb{E}[zz^\top] = I_n$ and $\mathbb{E}[ww^\top] = I_p$ give expectation $\mathrm{tr}(JJ^\top)$ in both cases. Here m is the actual number of independent directions. RHutch uses m VJPs, and FHutch uses m JVPs. These saved runs use plain one-sided estimators without QR, a basis Q, or a split into q and r directions. Probes are processed sequentially.

For comparison to the pseudocode 2, q denotes the number of orthonormal basis columns in Q and r the number of independent residual probes. The total budget is $m = q + r$ for the derivative calls, not $2q + r$. Computing $Q = \mathrm{orth}(S)$ requires no Jacobian. The pseudocode provided uses $q = r = m/2$ (m is assumed to be even) and averages the residual with $1/r = 2/m$. The code estimators above average all m squared norms with $1/m$. Thus, the equal-split one-sided pseudocode is not the exact implementation used for these Pythia measurements.

D.4 Implementation details and Additional Results

Each method/budget pair is run 20 times, giving 300 estimates per model. The model, prompt, state, and parameter scope stay fixed. Only the probes change. Seeds are offsets from base seed 20260825 and are stored. Model loading, tokenization, and similar things are excluded from timings. Hutch++ includes QR and residual projection. Initial state-space probe generation and exact-basis construction are not timed. FHutch's streamed parameter-probe generation is counted as part of the time. The randomized methods use sequential probe actions, unlike the chunk-vmapped exact reference.

We plot the relative error as $|t_m - t_\star|/t_\star$ for an estimate t_m. Trace/error curves show medians and interquartile intervals across probe repeats. For each repeat, we check the sample budgets from smallest to largest. We find where the error first reaches the chosen tolerance. We estimate the time needed by drawing a straight line between the two nearby measurements. If the smallest budget already meets the tolerance, we use $(\text{time}, \text{error}) = (0, 1)$ as the starting point. We then choose the fastest method that meets the tolerance in that repeat. Repeats where no method succeeds are left out. The speedup is the exact computation time divided by this estimated time. This comparison is done after the measurements are collected. Here, accuracy refers to the estimated NTK trace.

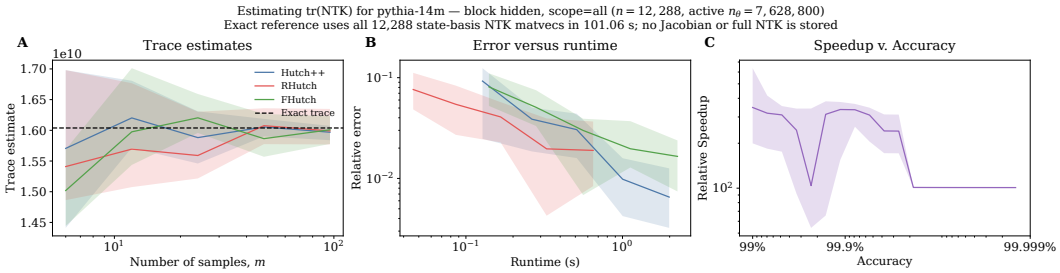

Figure 8: **Matrix-free estimation of tr(NTK) for the hidden-state representations of released Pythia Transformers.** For each model, $h(\theta)$ contains the hidden representations of all 16 tokens at the output of every Transformer block. **A** shows trace estimates against randomized sample count, m. **B** shows relative error $|t_m - \mathrm{tr}(\mathtt{NTK})|/|\mathrm{tr}(\mathtt{NTK})|$ against runtime. **C** shows speedup of randomized trace estimation relative to the exact state-basis evaluation. At each target accuracy and repeat, we record the fastest measured Hutch++, RHutch, or FHutch estimate satisfying the target tolerance and divide the exact-computation runtime by the estimator runtime. The most stringent-accuracy tail points should be interpreted cautiously when only a small number of estimates satisfy the target tolerance. Curves show medians over 20 estimates, and shaded regions show the 25th and 75th percentiles.

Numerical results The exact state-basis computation gives $\mathrm{tr}(\mathtt{NTK}) = 1.6038 \times 10^{10}$ in 101.06 seconds for Pythia-14M and $\mathrm{tr}(\mathtt{NTK}) = 2.2869 \times 10^{10}$ in 8,328.58 seconds for Pythia-410M-deduped. For Pythia-14M, Hutch++ has median relative errors of 3.05% at $m = 24$ and 0.65% at $m = 96$, with median runtimes of 0.51 and 1.99 seconds, respectively; the median fastest-method speedup at the 99% accuracy target is $347\times$. For Pythia-410M, at $m = 24$ the median relative errors are 2.34%, 1.66%, and 2.08% for Hutch++, RHutch, and FHutch, with runtimes of 2.06, 0.66, and 2.23 seconds. At $m = 96$, Hutch++ reaches 0.58% median relative error in 8.03 seconds. The median fastest-method speedup at 99% accuracy is $13{,}499\times$.

E Rich and Lazy Recurrent Learning Additional Details

Following Liu et al. (2024), we trained bias-free 300-unit RNNs on row-wise sequential MNIST, where each image is presented as 28 vectors $x_t \in \mathbb{R}^{28}$:

$$h_0 = 0, \qquad h_t = e^{-1} h_{t-1} + (1 - e^{-1}) \left[W_{\mathrm{rec}} \, \mathrm{ReLU}(h_{t-1}) + W_{\mathrm{in}} x_t \right], \tag{19}$$

$$y = W_{\mathrm{out}} \, \mathrm{ReLU}(h_{28}). \tag{20}$$

We used 55,000 training and 5,000 validation examples and trained for 10,000 updates with cross-entropy and SGD (learning rate 0.003, momentum 0.9, batch size 200, gradient clipping at 10). We evaluated 20 initialization ranks,

$$r \in \{5, 7, 10, 12, 15, 20, 25, 30, 35, 45, 60, 70, 80, 100, 125, 150, 175, 200, 250, 300\},$$

with three seeds (8021–8023), giving 60 runs and 101 checkpoints per run. Matched-performance measurements use the first checkpoint reaching 95% accuracy on a fixed 1,024-example test subset.

For each seed, rank conditions share the same underlying Gaussian initialization. We sample

$$A_{ij} \sim \mathcal{N}(0, 1.5^2/N), \qquad A = U \Sigma V^\top, \qquad W_{\mathrm{rec},0} = \frac{\|A\|_F}{\|A_r\|_F} A_r,$$

where $A_r = U_{:,1:r} \Sigma_{1:r,1:r} V_{:,1:r}^\top$. Thus $W_{\mathrm{rec},0}$ has rank r while preserving the Frobenius norm of the full Gaussian matrix; its rank is unconstrained during training. Input and output weights are Gaussian with standard deviations $1/\sqrt{28}$ and $1/\sqrt{300}$, respectively.

We measure the NTK of the final pre-readout hidden state h_{28} with respect to W_{rec}. Using 20 fixed class-balanced validation examples,

$$J_s = \frac{\partial \, \mathrm{vec}(H_s)}{\partial \, \mathrm{vec}(W_{\mathrm{rec}})}, \qquad K_s = J_s J_s^\top,$$

where $H_s \in \mathbb{R}^{20\times 300}$, giving a 6000×6000 matrix-free NTK. We report

$$\mathrm{align}(K_0, K_s) = \frac{\mathrm{tr}(K_0 K_s)}{\sqrt{\mathrm{tr}(K_0^2)\,\mathrm{tr}(K_s^2)}}, \qquad r_{\mathrm{eff}}(K_s) = \frac{\mathrm{tr}(K_s)^2}{\mathrm{tr}(K_s^2)}.$$

Traces use Hutch++ with 24 probes and three repeats. Anchor examples and estimator randomness are fixed across checkpoints within each run, and error bars show mean $\pm$ SEM over the three model seeds.

F Data-Scare Knowledge Distillation Additional Details

We trained MLP teachers on all $60{,}000$ Fashion-MNIST training examples and students on fixed subsets of $N \in \{250, 500, 1000\}$. All models used cross-entropy classification loss. The regularized student minimized

$$\mathcal{L} = (1-\lambda)\mathcal{L}_{\mathrm{CE}} + \lambda\big(1 - \mathrm{align}(K_s, K_t)\big),$$

where K_s and K_t are the student and teacher empirical NTK operators evaluated on the same 16-example minibatch. Alignment was estimated matrix-free using 12 Hutch++ probes; no Gram matrix was materialized.

For output-space NTKs, we used one-hidden-layer MLPs with widths 512 and 128 for the teacher and student, respectively. The teacher reached 90.05% test accuracy. Students were trained for 200 epochs over six paired seeds, using identical subsets and initializations for CE-only and regularized runs. The best fixed coefficient across subset sizes was $\lambda = 0.7$. For hidden-space NTKs, we used two-hidden-layer MLPs with the same widths and compared the stacked layer activations after a matrix-free partial trace over the hidden-unit axis. The teacher reached 90.44% test accuracy. These students were trained for 150 epochs over three paired seeds, with $\lambda = 0.5$.