Label-Invariant Hessian Regularization Mitigates Grokking in Mathematical Reasoning

Hongyang R. Zhang
Northeastern University, MA
ho.zhang@northeastern.edu

Zhenshuo Zhang
Northeastern University, MA
zhang.zhens@northeastern.edu

Jerry W. Liu
Stanford University, CA
jwl50@stanford.edu

Christopher Ré
Stanford University, CA
chrismre@stanford.edu

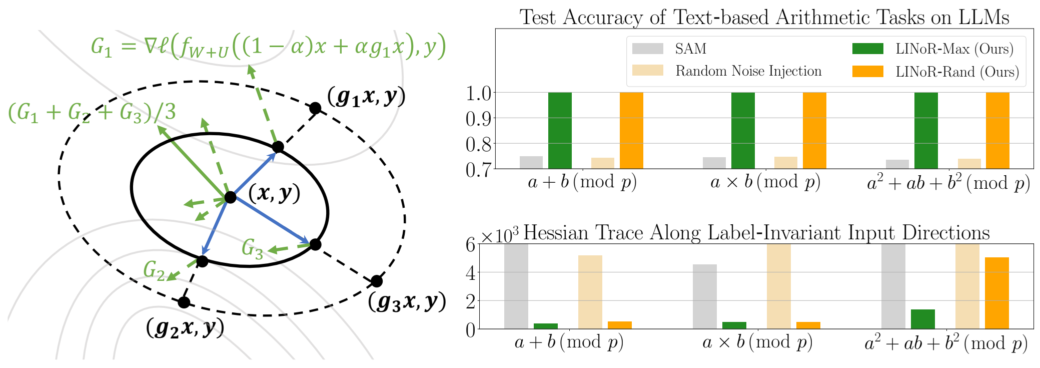
Abstract

Mathematical reasoning is a central aspect in the evaluation of language models. For many modular arithmetic tasks, prior work has observed the phenomenon of grokking, where the training accuracy converges to nearly 100%, whereas the test performance lags behind for an extended number of epochs until finally reaching perfect accuracy. In this paper, we find that by injecting invariant structures into modular arithmetic tasks, we can significantly speed up the number of training steps. Specifically, let g denote a label-invariant transformation and x denote an input. In the case of modular addition, a plus b, if we transform the input into $a + i$ (mod p) and $b - i$ (mod p), the outcome remains the same. Given a math reasoning task and a set of invariant transformation rules, our approach works by applying one of the transformations g to the input x (similar to data augmentation). Then, we interpolate the transformed input $g(x)$ with the original input x. Finally, we also add noise to the weights before computing the gradient to reduce the sharpness of the loss surface. When evaluated on three modular arithmetic tasks, we find that this approach reduces the number of grokking steps by more than 60% compared to existing sharpness-reduction and acceleration methods. In addition, this new approach can also be used for out-of-domain samples. When evaluated on six text-based arithmetic and graph-algorithmic tasks, our approach improves the test accuracy of LLMs by 16.5% and by 69%. Lastly, we provide a generalization bound that depends on a Hessian distance measure for learning invariant function classes to further validate our approach.

1 Introduction

Mathematical reasoning tasks are a central component in the evaluation of LLMs. A phenomenon that has often been observed with training transformer-based LLMs on arithmetic tasks is grokking [25, 34, 33], where models overfit to the training set quickly, but reach perfect generalization after an extended number of epochs. In this paper, we approach grokking by considering label-invariant properties that can commonly be found in math reasoning tasks. We propose regularization methods to add such invariance to the model.

Transformation invariance is central to many math problems, and has been studied before in learning theory [4, 36], but we lack a way to algorithmically engineer them into the model. For instance, graph-algorithmic tasks are invariant under graph-isomorphic transformations of the input graph. Recent work, including evaluation on GraphQA and CLRS benchmarks, shows that LLMs do well



(a) An illustration of our approach (b) Comparison before/after incorporating label-invariance

Figure 1: In this work, we explore the presence of label-invariant transformations in math reasoning tasks and introduce a new regularization method to add such invariance into LLMs. Figure 1a: Given a training data point (x, y), and three label-invariant transformations g_1, g_2, g_3, let $g_i x = g_i^{\textit{L-inv}}(x)$ denote the augmented x using each transformation, for $i = 1, 2, 3$. First, we create an augmented data point, by interpolating $g_i x$ and x with a scale of $1 - \alpha$ and α, for some $\alpha \in (0, 1)$. Second, we insert a random noise of U to the weight W before computing the gradient. In expectation, our algorithm moves along an average of the three individual gradients as the update direction. Figure 1b: Our approach, called label-invariant Hessian regularization (or LIHR in short) can be instantiated either by moving along a random augmentation in Figure 1a, namely LIHR-Rand, or along the max-loss augmentation, namely LIHR-Max. When applied to three arithmetic tasks on LLMs, LIHR substantially improves test accuracy by better capturing sharpness along "label-invariant input directions" compared to prior methods such as sharpness-aware minimization (SAM) [16, 38] and random noise injection [13, 47].

on tasks like degree counting, but their performance suffers for more complex algorithms such as all-pairs shortest path [15, 41, 35]. Even for basic modular arithmetic tasks such as addition and multiplication, achieving perfect generalization is possible but is very slow [34, 43]. Small initialization helps accelerate grokking, but its benefit seems limited beyond modular addition [32].

To formulate the problem, let $\mathcal{G}$ denote a group of label-invariant transformations acting on an input space. For example, in the modular addition task of $a + b \pmod p$, we can define $\mathcal{G}$ as the set of transformations that maps (a, b) to $(a + k, b - k)$, for any $k = 0, 1, \ldots, p - 1$. Then, let $\ell(f_W(x), y)$ denote the loss of a model f_W with parameters W on an input (x, y). A label-invariant direction is any $v = gx - x$, for some $g \in \mathcal{G}$. We define label-invariant sharpness as the Hessian trace of $\ell(f_W(x + \alpha v), y)$ for some $\alpha \in (0, 1)$. While Hessian-based measures are linked to generalization via PAC-Bayes bounds [22, 47], their dynamics during grokking are poorly understood [1]. Our empirical study on modular arithmetic and other algorithmic tasks reveals that our label-invariant sharpness metric is a much more accurate predictor of generalization than existing sharpness measures. Critically, we find that the model landscape is significantly flatter along these label-invariant directions compared to directions toward other random points with different labels. This observation suggests that existing regularizers like SAM [16] or random noise injection [13, 47] are insufficient as they do not specifically target this geometric property.

Building on this insight, we introduce a regularization framework that directly encourages flatness along label-invariant directions by combining data augmentation and Hessian regularization. One algorithm, named label-invariant noise regularization, or LIHR-Rand in short, takes a random transformation $g \sim \mathcal{G}$ to compute an augmented sample $\tilde{x} = x + \alpha(gx - x)$ with α between 0 and 1 (akin to mixup [48, 42, 7]), and then computes the gradient after perturbing W with a random Gaussian noise U. When $\alpha = 1$, this approach generalizes standard data augmentation [**dao2019kernel**, 4, 42]. Another algorithm, named LIHR-Max, solves a min-max objective to tackle the worst-off transformation instead. See Figure 1 for an overall illustration of our approach. The rationale of our approach can be seen as using label-invariant augmentations to improve sample efficiency, and this can be theoretically fleshed out for learning an invariant function class. We show a generalization bound based on a Hessian distance measure along label-invariant input directions

(See Theorem 4.1). Compared to PAC learning under transformation invariances [36], this new result now accounts for the role of data, yielding non-vacuous bounds on LLMs.

We extensively evaluate our approach on transformer models and various LLMs. We consider modular arithmetic tasks, text descriptions of these tasks, and graph-algorithmic tasks. We focus our evaluation on the number of grokking steps required to reach perfect generalization, or the generalization accuracy gap when achieving perfect accuracy is infeasible. We find that for three modular arithmetic tasks, LIHR-MAX reduces the number of grokking steps to reach perfect generalization by over **60%** compared to existing sharpness-regularization [16, 47] and acceleration methods [43]. The approach also applies to training LLMs such as Llama and Qwen3 — by using data augmentation alone, we can improve the test accuracy by **16.5**% over all the widely used regularization methods in practice. Perhaps more strikingly, our approach applies even to out-of-domain examples with unseen value ranges, maintaining **69**% accuracy while all the baseline methods show zero accuracy. We also report detailed ablation studies to show that the two components of our algorithm, namely noise injection and data augmentation, are both essential to its performance.

In summary, this paper makes the following three contributions to understanding and improving generalization in math reasoning: i) Formulating the notion of label-invariance in arithmetic and graph-algorithmic tasks, along with a new label-invariant Hessian trace measure that is also non-vacuous. ii) Developing a new sharpness-regularization method to add invariance using data augmentation. iii) Showing that this approach can accelerate grokking compared with existing methods and, in other cases, reduce the generalization gaps of LLMs.

2 Our Approach

In this section, we present our approach. We first formulate the problem of training language models for math reasoning tasks and motivate the need for better generalization, including the settings that exhibit grokking. Our key idea is to leverage the invariance structure induced by label-invariant transformations in the reasoning tasks. We observe that the trace of the loss Hessian along the directions between label-invariant inputs is closely tied to generalization and remains low once the model generalizes. Motivated by this, we propose an algorithm that regularizes the label-invariant Hessian trace by data augmentation and noise injection. Lastly, we show a generalization bound for learning invariant function classes that can also be measured from data.

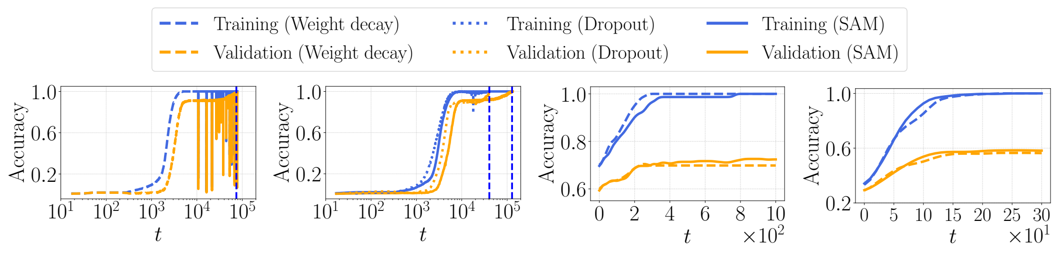
2.1 Problem Setup

We consider training transformer-based language models on math reasoning tasks. We consider a supervised learning setting. Let f_W be a language model parameterized by W, and let $\ell(f_W(x), y)$ denote the loss function. During training time, we have access to a set of training examples $D_{\text{train}} = \{(x_i, y_i)\}_{i=1}^n$ sampled independently from the input-label space of $\mathcal{X} \times \mathcal{Y}$. The training objective is to minimize the empirical risk over the training set D_{train}. Then, the model is evaluated on a test set D_{test} drawn independently from the same data distribution. We give two examples below.

Example 2.1 (Modular arithmetic tasks)**.** *Consider learning a language model to perform modular arithmetic operations over two numbers, in the form of $a \circ b \pmod{p} = c$, where $\circ$ is an arithmetic operation and p is a prime. Inputs are composed of four tokens: a, $\circ$, b, and $=$, with a and b generated between 0 and $p-1$. The label is the result of the arithmetic operation c.*

Example 2.2 (Graph-algorithmic tasks)**.** *Consider training language models to solve graph-related problems, such as finding the shortest distance between two given nodes. In existing graph-algorithmic datasets [15, 41], the input is composed of the description of the graph structure, such as a list of its edges, with a task-specific prompt, such as two node indices. The label is the value of the shortest distance between the two nodes.*

Our focus is on the generalization of language models for math reasoning tasks. We define the generalization error of f_W as the gap between the test loss and the training loss. We aim to design algorithms to close this gap. Moreover, when we can achieve 100% test accuracy, the goal is to reduce the number of optimization steps to reach such a state.



(a) $a^2 + ab + b^2 \pmod p$ (b) $a^2 + ab + b^2 \pmod p$ (c) Shortest Distance (d) Triangle Counting

Figure 2: We illustrate the generalization of models on Examples 2.1 and 2.2. Figure 2a and 2b: We illustrate grokking while training transformers in modular arithmetic tasks with weight decay, dropout, and SAM. High weight decay induces grokking, but causes unstable training curves. Dropout and SAM stabilize the training curve, but SAM increases the optimization steps to achieve generalization. The vertical blue line indicates the step when the model achieves 100% test accuracy. Figure 2c and 2d: We show significant generalization gaps when training Llama models on graph-algorithmic reasoning tasks. SAM closes the gap slightly, and the same gap persists with dropout.

2.2 Measuring Generalization in Math Reasoning

Next, we examine the behavior of transformer-based LLMs on Examples 2.1 and 2.2. We evaluate standard regularization methods, including weight decay, dropout, and sharpness-aware minimization.

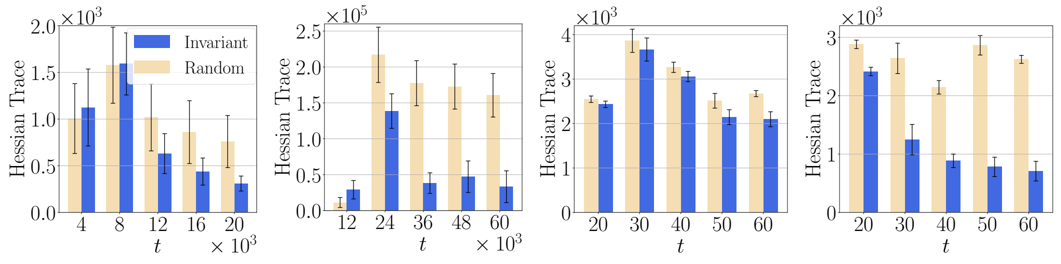
Delayed generalization in modular arithmetic tasks. We study grokking by training a two-layer transformer to predict $a^2 + ab + b^2 \pmod p$. We find that grokking persists even after adding strong regularization. Consistent with prior work [25], Figure 2a shows that using high weight decay with $\lambda = 1$ induces grokking, but it takes 2.3× relatively more optimization steps to achieve generalization than the steps to reach overfitting. Moreover, high weight decay leads to unstable training with oscillations in accuracy. Figure 2b shows that applying dropout and SAM stabilizes the training. Dropout slightly reduces the optimization steps by 8%.

Large generalization gaps in graph-algorithmic tasks. Second, we evaluate the generalization gaps of language models on graph-algorithmic tasks. We train Llama-3-1B models following the setup in the GraphQA benchmark [15, 35], to compute the shortest path and the number of triangles in a graph. In Figure 2c and 2d, we observe that using weight decay, the model exhibits an accuracy gap of 29% and 43%, respectively. With weight decay, dropout, or SAM, the gap only reduces slightly.

Injecting label-invariance to math reasoning. Next, we describe our approach to improving generalization in math reasoning tasks. Our key idea is to leverage the invariance structure induced by label-invariant transformations that commonly exist in many math reasoning tasks. Suppose the input space $\mathcal{X}$ admits a group of transformations $\mathcal{G}$ acting on inputs, such that the label remains invariant under group actions. That is, for any $g \in \mathcal{G}$ and a data point (x, y), let $g^{\textit{L-Inv}}(x)$ denote the augmented version of x. When there is no ambiguity, we denote $gx = g^{\textit{L-Inv}}(x)$ for short. That is, the label of gx is equal to the label of x, $y(gx) = y(x)$. $\mathcal{G}$ satisfies that there is an identity element $e \in \mathcal{G}$ such that $ex = x$, and satisfies closure under composition: $g^{\textit{L-Inv}}(h^{\textit{L-Inv}}(x)) = (gh)x$ for all $g, h \in \mathcal{G}$. We refer to the orbit of an input x: $N(x) = \{gx \mid g \in \mathcal{G}\}$ as the label-invariant neighborhood of x. This group-theoretic framework has been introduced to formally study the role of data augmentation [4]. Next, we give examples of label-invariant transformation groups in math reasoning tasks.

Example 2.3. *Consider predicting $a + b \pmod p$. A set of label-invariant transformations is given by $\mathcal{G} := \{g_k(a, b) = (a + k \pmod p), b - k \pmod p)) \mid k = 0, \dots, p - 1\}$. For $a^2 + ab + b^2 \pmod p$ and $a \times b \pmod p$, we give examples of label-invariant transformation groups in Appendix A.2.*

Example 2.4. *Consider counting the number of triangles in a graph. The input consists of a graph $G = (V, E)$ with adjacency matrix $A \in \mathbb{R}^{d \times d}$, which is flattened to an edge list to be the input. This task admits label-invariant transformations under graph isomorphic transformations of G. Each transformation corresponds to a permutation matrix $P \in \mathbb{R}^{d \times d}$ acting on the adjacency matrix. The label-invariant transformations are defined as: $\{g_P(A) = PAP^\top \mid P \in Aut(G)\}$, where $Aut(G)$ denotes the automorphism group of the graph G.*



(a) $a + b \pmod p$ (b) $a^2 + ab + b^2 \pmod p$ (c) Shortest Distance (d) Triangle Counting

Figure 3: We observe that the Hessian trace becomes significantly lower along the label-invariant input direction given by augmented inputs than along the direction given by random inputs with different labels. Given v, we estimate the Hessian trace along $x + \alpha v$. As t increases, the Hessian trace becomes significantly smaller when $v = gx - x$, where g is a random sample of $\mathcal{G}$. To calibrate this result, we compare against $v = x' - x$, where x' is a random sample whose label differs from y. We set $\alpha = 0.4$ and evaluate on Llama checkpoints trained with SAM.

Based on the transformation set, we show that a local sharpness measure, defined on inputs interpolated along label-invariant input directions, is key to generalization on reasoning tasks. To define the sharpness measure, we examine the expected loss with random Gaussian noise injected into the model parameter: $\tilde{\ell}(f_W) = \mathbb{E}_{U \sim \mathcal{N}(0, \sigma^2 \, \mathrm{Id})} \left[\ell\big(f_{W+U}(x), y\big) \right]$. Using Taylor's expansion of $\tilde{\ell}(f_W)$ around W, we have that for small enough σ:

$$\tilde{\ell}(f_W) = \ell\big(f_W(x), y\big) + \frac{\sigma^2}{2} \operatorname{Tr}\left[\nabla^2 \ell(f_W(x), y)\right] + O(\sigma^3).$$

Based on this approximation, we now introduce a Hessian trace measure, which quantifies the curvature of the loss landscape on interpolated inputs between x and $V(x)$.

Definition 2.5. *Let f_W be a model parameterized by weights W, and let $\mathcal{G}$ denote a group of label-invariant transformations. For a data point $x \in \mathcal{X}$, define the set of label-invariant input directions as $V(x) = \{gx - x \mid g \in \mathcal{G}\}$. Let $\alpha \in (0, 1)$. The label-invariant Hessian trace is given by: $\mathcal{H}(x, y) \coloneqq \mathbb{E}_{v \sim V(x)} \left[\operatorname{Tr} \left[\nabla^2 \ell(f_W(x + \alpha v), y) \right] \right]$.*

We observe that $\mathcal{H}(x, y)$ reduces when models generalize, while for the input direction between x and a random input x' whose label differs from y, its Hessian trace remains higher than $\mathcal{H}(x, y)$. To illustrate, for each example (x, y), we estimate the Hessian trace along label-invariant input directions by sampling ten directions from $V(x)$. For random input directions, we estimate by sampling ten random inputs with a different label than y. In Figure 3, we find that initially, the trace remains comparable between label-invariant and random input directions. As t increases, $\mathcal{H}(x, y)$ becomes up to $4.8\times$ lower than that along random inputs. We also evaluate $\tilde{\ell}(f_W)$ and find that it is up to $9.8\times$ lower on label-invariant directions than a random input (See Figure 5, Appendix A).

2.3 Designing Regularization Methods for Injecting Invariance

Having shown that the label-invariant Hessian trace closely correlates with model generalization, we now describe our algorithm to exploit the invariance structure. Our idea is to regularize the label-invariant Hessian trace during training, combining data augmentation and noise injection. We consider the following optimization problem. We are given n training examples, a group of label-invariant transformations $\mathcal{G}$ that act on the input space, and an isotropic Gaussian noise variance of σ^2 (to be added to the model parameters). Our goal is to minimize the average perturbed loss on the interpolated inputs along label-invariant input directions. However, this requires computing the loss along all label-invariant directions, which can be costly. Instead, we take a random sample from $\mathcal{G}$ to augment each sample. For every x_i, let $\tilde{x}_i = x_i + \alpha(g_i x_i - x_i)$, by sampling a transformation g_i from $\mathcal{G}$. In addition, we compute the gradient using model parameters perturbed by random Gaussian noise. We summarize the procedure in Algorithm 1. An alternative approach is to minimize the maximum loss over the augmented inputs. This approach, termed as LIHR-MAX, selects the transformation that aligns most with the input gradient. The detailed algorithm can be found in Appendix A.3.

Algorithm 1 Label-Invariant Noise Regularization with a random augmentation (LIHR-Rand)

```
Input: Training dataset $D_{\text{train}} = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$
Require: A set of label-invariant transformations $\mathcal{G}$, augmentation scale $\alpha$, standard deviation $\sigma$,
    number of epochs $T$, batch size $B$, learning rate $\eta$
Output: Trained model $f_{W^{(T)}}$
 1: $W^{(0)} \leftarrow$ A random (or pretrained) transformer-based LM initialization
 2: for $t = 1, 2, \dots, T$ do
 3:     $(x_1, y_1), (x_2, y_2), \dots, (x_B, y_B) \leftarrow$ A mini-batch of data samples from $D_{\text{train}}$
 4:     for $i = 1, \dots, B$ do
 5:         $g_i \leftarrow$ A uniformly random transformation from $\mathcal{G}$
 6:         $v_i \leftarrow g_i x_i - x_i$                                    // Compute invariant direction
 7:         $\tilde{x}_i \leftarrow x_i + \alpha v_i$                            // Compute augmented input
 8:     end for
 9:     $U^{(t-1)} \leftarrow$ A $d$-dimensional Gaussian noise sampled from $\mathcal{N}(0, \sigma^2 \,\mathrm{Id}_d)$
10:     $W^{(t)} \leftarrow W^{(t-1)} - B^{-1}\eta \sum_{i=1}^{B} \nabla \ell\big(f_{W^{(t-1)}+U^{(t-1)}}(\tilde{x}_i), y_i\big)$
11: end for
```

3 Experiments

We now evaluate our algorithm on a range of math reasoning tasks and transformer-based LLMs. For training transformers on arithmetic tasks, we find that our approach reduces the number of epochs to reach grokking by over **60**% compared to existing regularization methods. Applied to text-based descriptions of modular arithmetic and graph-algorithmic tasks, we find that using data augmentation improves test accuracy by **16.5**% over existing methods, tested on Llama and Qwen3-8B models. When evaluated on out-of-domain samples, our approach shows up to **69**% higher accuracy on integers that fall outside the range of training data. Ablation studies validate the benefit of LIHR compared with either using noise injection or using data augmentation (including mixup) alone.

3.1 Experimental Setup

Datasets and models. We evaluate on modular arithmetic tasks and graph-algorithmic tasks. We use three modular arithmetic tasks, including $a + b \pmod p$, $a \cdot b \pmod p$, and $a^2 + ab + b^2 \pmod p$. We set $p = 97$ and generate all $97^2 = 9409$ input pairs. We use 50% for the training set of the addition and multiplication tasks and 90% for the quadratic task. We use two-layer transformer models, randomly initialized Llama-3-1B, and pretrained Qwen3-8B models. Second, we use graph-algorithmic tasks from the GraphQA benchmark [15, 35], including counting node degrees, computing shortest distances, and triangle counting. We generate data points by sampling label-invariant random Erdős–Rényi graphs with 5 to 10 nodes. We sample 250 graphs for training and 500 graphs for testing. We train randomly initialized Llama-3-1B models on the node degree tasks, and fine-tune Qwen3-8B models on the triangle counting and the shortest distance tasks.

Baselines. We compare our approach with regularization methods, including weight decay and dropout. We also consider sharpness-reduction methods, including sharpness-aware minimization (SAM) [16], noise stability optimization (NSO) [47], and Frob-SAM [38]. Third, we compare with data augmentation with exact label-invariant neighbors [4]. Lastly, we consider another accelerated method called GrokTransfer [43], which trains a small MLP model and then transfers the embedding layer to train a larger transformer.

Metrics and implementations. For modular arithmetic tasks, we compare the number of epochs needed to reach perfect generalization. For Llama and Qwen3-8B, we report test accuracy. For evaluating out-of-domain accuracy, we use data points with at least one of the input numbers between p and $2p - 1$. When implementing LIHR, we vary α between 0.1 and 1.0 and vary σ between 0.01, 0.02, and 0.05. Other details about hyperparameters used are deferred until Appendix B.1.

3.2 Experimental Results

Mitigating grokking of transformers. First, we evaluate LIHR-Rand on modular arithmetic tasks. As shown in Table 1, our algorithm consistently takes fewer epochs than other baselines to achieve

Table 1: We report the number of epochs required for training transformers to perfect generalization (reaching $\approx 100\%$ test accuracy) on modular arithmetic tasks. We normalize the number of epochs with LIHR-MAX as a base for each method (lower is better), thus each value represents the ratio of that method relative to the grokking steps of LIHR-MAX. The mean and standard deviation are measured with three random seeds. The actual number of epochs can be found in Table 4, App. B.2.

Arithmetic Tasks	$a+b \pmod p$	$a \times b \pmod p$	$a^2+ab+b^2 \pmod p$
Weight Decay	$2.8_{\pm 0.2}$	$2.8_{\pm 0.2}$	$1.5_{\pm 0.1}$
Dropout	$2.8_{\pm 0.2}$	$2.8_{\pm 0.2}$	$1.3_{\pm 0.1}$
SAM	$3.0_{\pm 0.4}$	$2.8_{\pm 0.2}$	$2.9_{\pm 0.0}$
Frob-SAM	$2.8_{\pm 0.4}$	$2.6_{\pm 0.3}$	$1.5_{\pm 0.1}$
NSO	$1.6_{\pm 0.1}$	$1.8_{\pm 0.0}$	$1.8_{\pm 0.1}$
GrokTransfer	$1.3_{\pm 0.0}$	$1.7_{\pm 0.0}$	$2.3_{\pm 0.0}$
Data Augmentation	$2.6_{\pm 0.2}$	$2.6_{\pm 0.2}$	$1.8_{\pm 0.5}$
LIHR-RAND (Algorithm 1)	$1.3_{\pm 0.0}$	$1.1_{\pm 0.0}$	$\mathbf{1.0}_{\pm 0.1}$
LIHR-MAX (Algorithm 2)	$\mathbf{1.0}_{\pm 0.1}$	$\mathbf{1.0}_{\pm 0.1}$	$\mathbf{1.0}_{\pm 0.1}$

generalization. Compared to the closest baseline of random noise injection, our algorithm reduces the number of grokking steps by over **60%** on average. In addition, we compare with a transfer learning-based approach, GrokTransfer, which pre-trains the embedding layer in a small MLP model. We find that our algorithm reduces the number of epochs by over **70%** on average. For a concrete example, consider the modular addition task trained with a two-layer transformer. Using weight decay results in a generalization gap of 1.28 and a Hessian distance measure of 0.53. By using LIHR instead, we reduce the generalization gap to 0.01, and the Hessian distance measure is only 0.06, which remains at the same scale as the empirical errors.

Reducing generalization gap of LLMs. Next, we evaluate our approach for training text-based descriptions of arithmetic and graph-algorithmic datasets with LLMs. The results are shown in Table 2. We find that using data augmentation, with or without noise injection, improves test accuracy by **16%** on average, over the closest baseline of SAM. LIHR further improves over data augmentation by **0.5%** on average. We note that the label-invariant transformations can be implemented as a simple function over the input and take less than 1 second. This is negligible (less than 1

Generalization to out-of-domain samples. On arithmetic datasets, we evaluate our approach on inputs where numbers are sampled between p and $2p-1$, larger than the numbers used for training. We find that for all the baseline methods, the test accuracy drops to zero. By contrast, our approach still achieves **69%** average accuracy over the three arithmetic tasks. A detailed comparison is presented in Table 6, Appendix B.2.

3.3 Ablation Studies

Evaluation of the Hessian measure. Next, we evaluate the Hessian distance measure in Theorem 4.1, using a transformer model trained on the modular addition dataset. We compare the measure across different regularization methods. As shown in Figure 4, we find that random noise injection (abbreviated as NSO in the Figure) achieves a relatively low Hessian. Our approach further reduces this measure by **14%**. In addition, we report the evaluation of a compression bound derived from PAC-Bayes analysis [27], which yields a value of 1.38, and is $23\times$ larger than our measure. These results suggest that the Hessian distance provides a non-vacuous measure of sharpness for reasoning tasks.

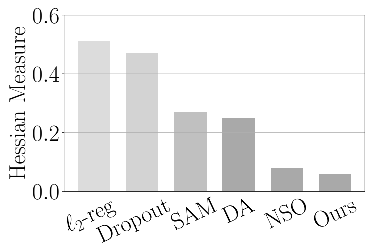


Figure 4: Comparison of the Hessian distance measure (cf. equation (1)) between existing methods and our approach.

Effects of data augmentation. Next, we find that data augmentation is also crucial to the results, especially for training LLMs on text-based reasoning tasks. As shown in Table 2, only using noise injection offers limited gain over weight decay. After adding data augmentation, we can improve over random noise injection by **17%** on average.

Table 2: We report the test accuracy % of training LLMs on text-based modular arithmetic and graph-algorithmic tasks. We compare our approach with existing regularization methods, sharpness-reduction methods, and GrokTransfer. We run each experiment with three random seeds to report the mean and standard deviation.

Text Inputs	Addition	Multiplication	Quadratic	Triangle Counting	Shortest Dist.	Node Deg.
Weight Decay	$74.0_{\pm 0.1}$	$74.1_{\pm 0.9}$	$72.8_{\pm 0.1}$	$89.2_{\pm 0.4}$	$89.1_{\pm 0.5}$	$84.4_{\pm 0.4}$
SAM	$75.0_{\pm 0.1}$	$74.5_{\pm 0.1}$	$73.5_{\pm 0.0}$	$90.3_{\pm 0.8}$	$89.2_{\pm 0.3}$	$83.6_{\pm 0.4}$
NSO	$74.3_{\pm 0.1}$	$74.8_{\pm 0.1}$	$73.9_{\pm 0.1}$	$89.6_{\pm 0.3}$	$89.2_{\pm 0.2}$	$84.5_{\pm 0.2}$
GrokTransfer	$68.4_{\pm 0.2}$	$72.1_{\pm 0.3}$	$71.3_{\pm 0.2}$	/	/	/
DataAugment	$98.8_{\pm 0.1}$	$\mathbf{99.7}_{\pm 0.1}$	$\mathbf{99.7}_{\pm 0.1}$	$95.2_{\pm 0.9}$	$92.2_{\pm 0.7}$	$98.7_{\pm 0.6}$
LIHR-MAX	$\mathbf{100.0}_{\pm 0.0}$	$\mathbf{99.7}_{\pm 0.1}$	$\mathbf{99.7}_{\pm 0.1}$	$95.4_{\pm 0.4}$	$92.4_{\pm 0.5}$	$99.0_{\pm 0.4}$
LIHR-RAND	$\mathbf{100.0}_{\pm 0.0}$	$\mathbf{99.7}_{\pm 0.1}$	$\mathbf{99.7}_{\pm 0.2}$	$\mathbf{95.8}_{\pm 0.3}$	$\mathbf{93.0}_{\pm 0.2}$	$\mathbf{99.1}_{\pm 0.1}$

Additionally, we compare our approach with mixup, the key difference being that LIHR interpolates with an augmented sample. We find that LIHR uses **57**% fewer grokking steps compared to mixup on modular arithmetic. We also vary α and note that using a α between 0 and 1 yields better results, while using $\alpha = 1$ might lead to unstable training or take **43**% longer to finish.

4 Generalization Guarantees for Learning Invariant Functions

We now present a PAC-Bayes generalization bound to better formalize the benefit of capturing label-invariance. Given a data distribution of $\mathcal{D}$, let

$$L_{\mathcal{G}}(f_W) = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}, g\in\mathcal{G}} \left[\ell(f_W(gx), y)\right]$$

denote the expected loss of an input under a random group action. Let $\hat{L}_{\mathcal{G}}(f_W)$ denote the empirical loss given n independent samples $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ taken from $\mathcal{D}$:

$$\hat{L}_{\mathcal{G}}(f_W) = \frac{1}{n}\sum_{i=1}^{n} \mathop{\mathbb{E}}_{g\in\mathcal{G}} \left[\ell(f_W(gx_i), y_i)\right].$$

Let $\mathcal{W} \subseteq \mathbb{R}^d$ denote a weight space of W's. Assume that the weighted Hessian along of both x and gx is bounded by a fixed constant $\mathcal{H}$:

$$\mathcal{H} := \sup_{W\in\mathcal{W}} \sup_{g\in\mathcal{G}} \left(W^\top \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \left[\nabla_+^2 \ell(f_W(gx), y)\right] W \right),$$

where $\nabla_+^2 \ell$ means that we truncate the negative eigen-directions of the Hessian to zero. Then, we have the following generalization error bound on the hypothesis space $\{f_W : W \in \mathcal{W}\}$.

Theorem 4.1. *Assume that the loss function ℓ is bounded between 0 and C for a fixed constant $C > 0$ on the data distribution $\mathcal{D}$. Suppose $\ell(f_W(\cdot), \cdot)$ is twice-differentiable in W and the Hessian matrix $\nabla^2 \ell(f_W(\cdot), \cdot)$ is Lipschitz continuous within the weight space $\mathcal{W}$. Suppose there exists a fixed constant $\mathcal{H}$ such that the above Hessian-based measure holds on any $W \in \mathcal{W}$. Then, for any W in $\mathcal{W}$, any $\epsilon > 0$ small enough, and any small $\delta > 0$, with probability at least $1 - \delta$ over the randomness of n samples, we have:*

$$L_{\mathcal{G}}(f_W) \le (1+\epsilon)\hat{L}_{\mathcal{G}}(f_W) + (1+\epsilon)\sqrt{\frac{C\mathcal{H}}{n}} + O\left(n^{-\frac{3}{4}} \log(\delta^{-1})\right). \tag{1}$$

The key implication of this result is that by adding invariance to the model, we can improve the generalization to unseen data generated via transformations in the group. In particular, we will validate this result by evaluating our approach on out-of-domain data. We will also validate that the measure of $\sqrt{C\mathcal{H}/n}$ is non-vacuous on real-world data. The proof of Theorem 4.1 improves on a trace-norm bound for neural networks [47], and can be found in Appendix A.1.

5 Related Work

Evaluation of LLMs on math reasoning tasks. Besides arithmetic and graph-algorithmic tasks, existing benchmarks evaluate math reasoning with school test problems, such as GSM8K [9] and MATH [18] datasets. These datasets involve multiple-step reasoning and contain natural language problem descriptions. Due to the diverse distribution, directly training large transformers on such benchmarks requires extensive pretraining data and compute [18]. It is plausible to extend our method to define invariant transformation rules for such datasets to improve the efficiency of learning.

Many works have sought to explain the occurrences of grokking. In a theoretical analysis, grokking provably occurs in two-layer ReLU networks on XOR cluster data with partially flipped training labels [44]. For a broad class of neural nets, grokking is attributed to a shift in implicit biases, from kernel predictors early in training to min-norm/max-margin predictors later [28]. Several measures have been defined to track the training dynamics for grokking. For example, alignment with the neural tangent kernel of the network at initialization captures the transition from kernel learning to feature learning [23]. On modular addition tasks, a circuit can be identified within small transformers, and a measure computed on the circuit reveals three phases of training [33]. The training dynamics of grokking have also been analyzed through a local complexity measure of the network based on the density of spline partition regions in input space [19].

Theoretical frameworks for data augmentation. Data augmentation has been widely used to improve generalization across modalities, including images [5], text [3], and graphs [46]. A common strategy involves finding a composition of multiple predefined transformations through a search algorithm, such as AutoAugment [10]. Another approach is consistency regularization, which encourages the model to produce similar predictions between original and augmented inputs [45]. Our work initiates a study to apply data augmentation for math reasoning tasks. Further adopting data augmentation methods for training LLMs on more complex reasoning tasks is left for future work.

The effects of data augmentation for model generalization have been theoretically analyzed in prior works. For example, mixup acts as a data-adaptive regularization that reduces overfitting [49] and finds smoother decision boundaries [7]. In multi-view data settings, mixup can provably recover multiple class-relevant features, where standard ERM fails [8]. An analysis of linear transformations in over-parameterized regression demonstrates that label-invariant augmentations expand the effective data span and reduce estimation error, while label-mixing acts as implicit regularization [42]. A PAC-Bayesian analysis reveals that it is necessary for an algorithm to distinguish between original and transformed inputs to achieve optimal accuracy under data augmentation [36].

Non-vacuous generalization bounds. Generalization upper bounds are a central tool for quantifying generalization in deep learning. Recent work has shown that data-dependent PAC-Bayes bounds are often non-vacuous when evaluated on real data [14]. For instance, the bounds may be derived through compression [2, 26], or via noise sensitivity [22, 47]. More recently, a low-rank parameterization technique enables computing the non-vacuous generalization bounds for large language models [27]. Ju et al. [22, 21] show that Hessian trace yields a non-vacuous measure that closely matches empirical error scales. More broadly, the sharpness of loss surfaces is a crucial technique for understanding generalization. Recent work shows that Gaussian noise injection acts as an implicit ridge regularizer [12]. Another recent framework introduces a universal class of sharpness-aware objectives, capable of expressing any function of the loss Hessian and enabling explicit bias toward minimizing a broad spectrum of sharpness metrics [38]. Compared to prior works, we are the first to design sharpness-regularization methods using label-invariant data augmentation for math reasoning.

6 Conclusion

In this paper, we study the generalization of LLMs on math reasoning tasks. Our key observation is that the Hessian trace remains significantly smaller along label-invariant input directions as the model generalizes. We then introduce an algorithm that combines data augmentation and noise injection to regularize the label-invariant Hessian. Further, we derive a PAC-Bayesian generalization bound involving the Hessian of an augmented loss. We empirically validate our approach across arithmetic and graph-algorithmic tasks. Our approach reduces the steps for grokking in transformers and improves test accuracy and out-of-domain performance in training LLMs. Our work leaves several natural avenues for future work. First, is it possible to achieve perfect generalization on the graph-algorithmic tasks? For example, one might consider a more carefully designed invariance, similar to the development of data augmentation methods. Second, does this invariance structure apply to other types of out-of-distribution settings? More broadly, we hope our work serves as a step towards better understanding the dynamics of Hessians in large neural networks.

References

[1] A. Agarwala and Y. Dauphin (2023). "Sam operates far from home: eigenvalue regularization as a dynamical phenomenon". In: *ICML*. PMLR (2).

[2] S. Arora, R. Ge, B. Neyshabur, and Y. Zhang (2018). "Stronger generalization bounds for deep nets via a compression approach". In: *International conference on machine learning*. PMLR, pp. 254–263 (9).

[3] J. Chen, D. Tam, C. Raffel, M. Bansal, and D. Yang (2023). "An empirical survey of data augmentation for limited data learning in NLP". In: *TACL* (9).

[4] S. Chen, E. Dobriban, and J. H. Lee (2020a). "A group-theoretic framework for data augmentation". In: *Journal of Machine Learning Research* (1, 2, 4, 6).

[5] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton (2020b). "A simple framework for contrastive learning of visual representations". In: *ICML* (9).

[6] T. Chen, T. Trogdon, and S. Ubaru (2021). "Analysis of stochastic Lanczos quadrature for spectrum approximation". In: *International Conference on Machine Learning* (20, 21).

[7] M. Chidambaram, X. Wang, Y. Hu, C. Wu, and R. Ge (2022). "Towards understanding the data dependency of mixup-style training". In: *International Conference on Learning Representations* (2, 9, 20).

[8] M. Chidambaram, X. Wang, C. Wu, and R. Ge (2023). "Provably learning diverse features in multi-view data with midpoint mixup". In: *International Conference on Machine Learning* (9).

[9] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, et al. (2021). "Training verifiers to solve math word problems". In: *arXiv preprint arXiv:2110.14168* (9).

[10] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le (2019). "Autoaugment: Learning augmentation policies from data". In: *CVPR* (9).

[11] Y. N. Dauphin, A. Agarwala, and H. Mobahi (2024). "Neglected hessian component explains mysteries in sharpness regularization". In: *NeurIPS* (25).

[12] O. Dhifallah and Y. Lu (2021). "On the inherent regularization effects of noise injection during training". In: *International Conference on Machine Learning* (9).

[13] J. C. Duchi, P. L. Bartlett, and M. J. Wainwright (2012). "Randomized smoothing for stochastic optimization". In: *SIAM Journal on Optimization* 22.2, pp. 674–701 (2).

[14] G. K. Dziugaite and D. M. Roy (2017). "Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data". In: *arXiv preprint arXiv:1703.11008* (9).

[15] B. Fatemi, J. Halcrow, and B. Perozzi (2023). "Talk like a Graph: Encoding Graphs for Large Language Models". In: *ICLR* (2–4, 6).

[16] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur (2021). "Sharpness-aware minimization for efficiently improving generalization". In: *ICLR* (2, 3, 6).

[17] R. Grosse, J. Bae, C. Anil, N. Elhage, A. Tamkin, A. Tajdini, B. Steiner, D. Li, E. Durmus, E. Perez, et al. (2023). "Studying large language model generalization with influence functions". In: *arXiv preprint arXiv:2308.03296* (25).

[18] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt (2021). "Measuring mathematical problem solving with the math dataset". In: *NeurIPS* (9).

[19] A. I. Humayun, R. Balestriero, and R. Baraniuk (2024). "Deep networks always grok and here is why". In: *International Conference on Machine Learning* (9).

[20] A. Jacot, F. Gabriel, and C. Hongler (2018). "Neural tangent kernel: Convergence and generalization in neural networks". In: *NeurIPS* (25).

[21] H. Ju, D. Li, A. Sharma, and H. R. Zhang (2023). "Generalization in Graph Neural Networks: Improved PAC-Bayesian Bounds on Graph Diffusion". In: *International Conference on Artificial Intelligence and Statistics* (9).

[22] H. Ju, D. Li, and H. R. Zhang (2022). "Robust Fine-Tuning of Deep Neural Networks with Hessian-based Generalization Guarantees". In: *International Conference on Machine Learning* (2, 9).

[23] T. Kumar, B. Bordelon, S. J. Gershman, and C. Pehlevan (2024). "Grokking as the transition from lazy to rich training dynamics". In: *International Conference on Learning Representations* (9).

[24] Y. LeCun (1993). "Efficient learning and second-order methods". In: *A tutorial at NIPS* 93, p. 61 (25).

[25] Z. Liu, E. J. Michaud, and M. Tegmark (2022). "Omnigrok: Grokking beyond algorithmic data". In: *International Conference on Learning Representations* (1, 4).

[26] S. Lotfi, M. Finzi, S. Kapoor, A. Potapczynski, M. Goldblum, and A. G. Wilson (2022). "PAC-Bayes compression bounds so tight that they can explain generalization". In: *Conference on Neural Information Processing Systems* (9).

[27] S. Lotfi, M. A. Finzi, Y. Kuang, T. G. Rudner, M. Goldblum, and A. G. Wilson (2024). "Non-Vacuous Generalization Bounds for Large Language Models". In: *International Conference on Machine Learning* (7, 9).

[28] K. Lyu, J. Jin, Z. Li, S. S. Du, J. D. Lee, and W. Hu (2024). "Dichotomy of early and late phase implicit biases can provably induce grokking". In: *International Conference on Learning Representations* (9, 25).

[29] N. Mallinar, D. Beaglehole, L. Zhu, A. Radhakrishnan, P. Pandit, and M. Belkin (2024). "Emergence in non-neural models: grokking modular arithmetic via average gradient outer product". In: *arXiv preprint arXiv:2407.20199* (26).

[30] D. McAllester (2013). "A PAC-Bayesian tutorial with a dropout bound". In: *arXiv preprint arXiv:1307.2118* (13).

[31] R. A. Meyer, C. Musco, C. Musco, and D. P. Woodruff (2021). "Hutch++: Optimal stochastic trace estimation". In: *Symposium on Simplicity in Algorithms (SOSA)*. SIAM, pp. 142–155 (22).

[32] M. A. Mohamadi, Z. Li, L. Wu, and D. J. Sutherland (2024). "Why Do You Grok? A Theoretical Analysis on Grokking Modular Addition". In: *International Conference on Machine Learning* (2, 26).

[33] N. Nanda, L. Chan, T. Lieberum, J. Smith, and J. Steinhardt (2023). "Progress measures for grokking via mechanistic interpretability". In: *International Conference on Learning Representations* (1, 9).

[34] A. Power, Y. Burda, H. Edwards, I. Babuschkin, and V. Misra (2022). "Grokking: Generalization beyond overfitting on small algorithmic datasets". In: *arXiv preprint arXiv:2201.02177* (1, 2).

[35] C. Sanford, B. Fatemi, E. Hall, A. Tsitsulin, M. Kazemi, J. Halcrow, B. Perozzi, and V. Mirrokni (2024). "Understanding transformer reasoning capabilities via graph algorithms". In: *NeurIPS* (2, 4, 6).

[36] H. Shao, O. Montasser, and A. Blum (2022). "A theory of pac learnability under transformation invariances". In: *NeurIPS* (1, 3, 9).

[37] A. Soleymani, B. Tahmasebi, S. Jegelka, and P. Jaillet (2025). "Learning with Exact Invariances in Polynomial Time". In: *International Conference on Machine Learning* (26).

[38] B. Tahmasebi, A. Soleymani, D. Bahri, S. Jegelka, and P. Jaillet (2024). "A universal class of sharpness-aware minimization algorithms". In: *International Conference on Machine Learning* (2, 6, 9, 26).

[39] R. Vershynin (2018). *High-dimensional probability: An introduction with applications in data science*. Vol. 47. Cambridge university press (17).

[40] M. J. Wainwright (2019). *High-Dimensional Statistics: A Non-Symptotic Viewpoint*. Cambridge University Press (17).

[41] H. Wang, S. Feng, T. He, Z. Tan, X. Han, and Y. Tsvetkov (2023). "Can language models solve graph problems in natural language?" In: *NeurIPS* (2, 3).

[42] S. Wu, H. Zhang, G. Valiant, and C. Ré (2020). "On the generalization effects of linear transformations in data augmentation". In: *International Conference on Machine Learning* (2, 9).

[43] Z. Xu, Z. Ni, Y. Wang, and W. Hu (2025). "Let Me Grok for You: Accelerating Grokking via Embedding Transfer from a Weaker Model". In: *International Conference on Learning Representations* (2, 3, 6, 26).

[44] Z. Xu, Y. Wang, S. Frei, G. Vardi, and W. Hu (2024). "Benign Overfitting and Grokking in ReLU Networks for XOR Cluster Data". In: *International Conference on Learning Representations* (9).

[45] S. Yang, Y. Dong, R. Ward, I. S. Dhillon, S. Sanghavi, and Q. Lei (2023). "Sample efficiency of data augmentation consistency regularization". In: *AISTATS* (9).

[46] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen (2020). "Graph contrastive learning with augmentations". In: *Advances in neural information processing systems* (9).

[47] H. R. Zhang, D. Li, and H. Ju (2024a). "Noise Stability Optimization for Finding Flat Minima: A Hessian-based Regularization Approach". In: *Transactions on Machine Learning Research* (2, 3, 6, 8, 9).

[48] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz (2018). "mixup: Beyond empirical risk minimization". In: *ICLR* (2, 20).

[49] L. Zhang, Z. Deng, K. Kawaguchi, A. Ghorbani, and J. Zou (2021). "How Does Mixup Help With Robustness and Generalization?" In: *International Conference on Learning Representations*. URL: `https://openreview.net/forum?id=8yKEo06dKNo` (9).

[50] Y. Zhang, C. Chen, T. Ding, Z. Li, R. Sun, and Z.-Q. Luo (2024b). "Why Transformers Need Adam: A Hessian Perspective". In: *NeurIPS* (25).

A Omitted Materials from Section 2

A.1 Complete Proofs

We provide a high-level illustration of the proof of Theorem 4.1. There are two main steps in the proof. First, we would like to show that on the original input distribution, the gap between test and training losses remains bounded by the Hessian distance measure. Second, we show that such a statement holds for every transformed distribution of the input distribution. Thus, we reduce the proof of equation (1) to the first step.

Proof sketch. We start with the first step. Let

$$\hat{L}(f_W) = \frac{1}{n}\sum_{i=1}^{n} \ell(f_W(x_i), y_i), \text{ and}$$
$$L(f_W) = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \left[\ell(f_W(x), y)\right],$$

denote the training (and test) losses of f_W, given n independent samples from data distribution $\mathcal{D}$. Let $\mathcal{Q}$ denote the *posterior* distribution. Specifically, we consider $\mathcal{Q}$ as being centered at the learned hypothesis W (which could be anywhere within the hypothesis space), given by a Gaussian distribution $\mathcal{N}(W, \Sigma)$, where Σ is a p by p covariance matrix. Given a sample $U \sim \mathcal{N}(0, \Sigma)$, let the perturbed loss be given by

$$\ell_{\mathcal{Q}}(f_W(x), y) = \mathop{\mathbb{E}}_{U}\left[\ell(f_{W+U}(x), y)\right]. \tag{2}$$

Then, let $\hat{L}_{\mathcal{Q}}(W)$ be the averaged value of $\ell_{\mathcal{Q}}(f_W(\cdot), \cdot)$, taken over n empirical samples from the training dataset. Likewise, let $L_{\mathcal{Q}}(W)$ be the population average of $\ell_{\mathcal{Q}}(f_W(\cdot), \cdot)$, in expectation over an unseen data sample from the underlying data distribution.

Having introduced the notations, we start with the PAC-Bayes bound [30], stated as follows.

Theorem A.1. *Suppose the loss $\ell(f_W(x), y)$ lies in a bounded range $[0, C]$ given any $x \in \mathcal{X}$ with label y. Let $\mathcal{P}$ and $\mathcal{Q}$ be prior and posterior distributions on the weights W. For any $\beta \in (0,1)$ and $\delta \in (0,1)$, with probability at least $1-\delta$, we hhave:*

$$L_{\mathcal{Q}}(W) \le \frac{1}{\beta}\hat{L}_{\mathcal{Q}}(W) + \frac{C\big(KL(\mathcal{Q}||\mathcal{P}) + \log\frac{1}{\delta}\big)}{2\beta(1-\beta)n}. \tag{3}$$

This result provides flexibility in setting β. Our results will set β to balance the two terms. We will need the KL divergence between the prior $\mathcal{P}$ and the posterior $\mathcal{Q}$ in the PAC-Bayesian analysis. This is stated in the following classical result.

Proposition A.2. *Suppose $\mathcal{P} = N(X, \Sigma)$ and $\mathcal{Q} = N(Y, \Sigma)$ are both Gaussian distributions with mean vectors given by $X \in \mathbb{R}^p, Y \in \mathbb{R}^p$, and population covariance matrix $\Sigma \in \mathbb{R}^{p\times p}$. The KL divergence between $\mathcal{P}$ and $\mathcal{Q}$ is equal to*

$$KL(\mathcal{Q}||\mathcal{P}) = \frac{1}{2}(X-Y)^\top\Sigma^{-1}(X-Y).$$

We are interested in the perturbed loss, $\ell_{\mathcal{Q}}(f_W(x), y)$, which is the expectation of $\ell(f_{W+U}(x), y)$ over U. Using a Taylor expansion, we find that

$$\ell_{\mathcal{Q}}(f_W(x), y) - \ell(f_W(x), y) \le \left\langle \Sigma, \nabla^2 \ell(f_W(x), y)\right\rangle + \epsilon,$$

where Σ is the population covariance matrix of the perturbation, ∇^2 is the Hessian matrix with respect to the weights of f_W, and $\epsilon = C_1(\mathrm{Tr}\,[\Sigma])^{3/2}$. See Lemma A.4 for the complete statement of this result.

Based on the above expansion, next, we apply the PAC-Bayes bound from equation (3) to an L-layer transformer neural network f_W parameterized by W. We note that the KL divergence between the prior and posterior distributions, which are both Gaussian, is equal to $\left\langle \Sigma^{-1}, vv^\top\right\rangle$, where v is the difference between the initialized and the trained weights.

Next, we combine the above Hessian-based bound and KL divergence in the PAC-Bayes bound. Let ∇_+^2 denote the truncated Hessian matrix where we set the negative eigenvalues of ∇^2 to zero. We have that

$$\left\langle \Sigma, \nabla_W^2[\ell(f_W(x), y)] \right\rangle \leq \left\langle \Sigma, \nabla_+^2[\ell(f_W(x), y)] \right\rangle. \tag{4}$$

Substituting into (3), and minimizing over β and Σ, we will derive an upper bound on the generalization error (between $L(f_W)$ and $\hat{L}(f_W)$) equal to

$$\alpha := \sup_{W \in \mathcal{W}} \sup_{(x,y) \sim \mathcal{D}} \frac{\sqrt{W^\top \left[\tilde{\nabla}_+^2 \ell(f_W(x), y)\right] W}}{\sqrt{n}}, \tag{5}$$

where $\mathcal{W}$ is the support of the distribution from which the data is drawn.

We will use a Taylor expansion to bound the perturbed loss, as follows.

Claim A.3. *Let f_W be twice-differentiable, parameterized by weight vector $W \in \mathbb{R}^p$. Let $U \in \mathbb{R}^p$ be another vector with dimension p. For any W and U, the following identity holds*

$$\ell(f_{W+U}(x), y) = \ell(f_W(x), y) + U^\top \nabla \ell(f_W(x), y) + \frac{1}{2} U^\top [\nabla^2 \ell(f_W(x), y)] U + R_2(\ell(f_W(x), y)),$$

where $R_2(\ell(f_W(x), y)))$ is a second-order error term in a Taylor expansion of $\ell \circ f_W$ around W.

Proof. The proof follows from the fact that $\ell \circ f_W$ is twice-differentiable. By the mean value theorem, there must exist η between W and $U + W$ such that

$$R_2(\ell(f_W(x), y)) = U^\top \Big(\nabla^2[\ell(f_\eta(x), y)] - \nabla^2[\ell(f_W(x), y)]\Big) U.$$

This completes the proof of the claim.

Based on the above result, we provide a Taylor's expansion for $\ell_{\mathcal{Q}} - \ell$.

Lemma A.4. *In the setting of Theorem 4.1, suppose each parameter is perturbed by an independent random variable drawn from $N(0, \Sigma)$. Let $\ell_{\mathcal{Q}}(f_W(x), y)$ be the loss averaged over the noise. There is a value C_1 that does not depend on n and $1/\delta$ such that*

$$\ell_{\mathcal{Q}}(f_W(x), y) - \ell(f_W(x), y) \leq \frac{1}{2} \left\langle \Sigma, \nabla^2[\ell(f_W(x), y)] \right\rangle + C_1 (\mathrm{Tr}\,[\Sigma])^{\frac{3}{2}}. \tag{6}$$

Proof. We take the expectations over U of both sides of the equation in Claim A.3. The result becomes

$$\begin{aligned}
&\mathop{\mathbb{E}}_{U}[\ell(f_{W+U}(x), y)] \\
=& \mathop{\mathbb{E}}_{U}\left[\ell(f_W(x), y) + U^\top \nabla \ell(f_W(x), y) + \frac{1}{2} U^\top \nabla^2[\ell(f_W(x), y)] U + R_2(\ell(f_W(x), y))\right].
\end{aligned}$$

Then, we use the perturbation distribution to calculate

$$\begin{aligned}
\ell_{\mathcal{Q}}(f_W(x), y) =& \mathop{\mathbb{E}}_{U}[\ell(f_W(x), y)] + \mathop{\mathbb{E}}_{U}\left[U^\top \nabla \ell(f_W(x), y)\right] + \frac{1}{2} \mathop{\mathbb{E}}_{U}\left[U^\top \nabla^2[\ell(f_W(x), y)] U\right] \\
&+ \mathop{\mathbb{E}}_{U}[R_2(\ell(f_W(x), y))].
\end{aligned}$$

Since $\mathbb{E}[U] = 0$, the first-order term vanishes. The second-order term becomes

$$\mathop{\mathbb{E}}_{U}\left[U^\top [\nabla^2 \ell(f_W(x), y)] U\right] = \left\langle \Sigma, \nabla^2[\ell(f_W(x), y)] \right\rangle. \tag{7}$$

The expectation of the error term $R_2(\ell(f_W(x), y))$ be

$$\begin{aligned}
\mathop{\mathbb{E}}_{U}[R_2(\ell(f_W(x), y))] &= \mathop{\mathbb{E}}_{U}\left[U^\top \left(\nabla^2[\ell(f_\eta(x), y)] - \nabla^2[\ell(f_W(x), y)]\right) U\right] \\
&\leq \mathop{\mathbb{E}}_{U}\left[\|U\|^2 \cdot \left\|\nabla^2[\ell(f_\eta(x), y)] - \nabla^2[\ell(f_W(x), y)]\right\|_F\right] \\
&\lesssim \mathop{\mathbb{E}}_{U}\left[\|U\|^2 \cdot C_1 \|U\|\right] \\
&\lesssim C_1 \left(\mathbb{E}\left[U^\top U\right]\right)^{\frac{3}{2}} = C_1 (\mathrm{Tr}\,[\Sigma])^{\frac{3}{2}}.
\end{aligned}$$

To obtain the first inequality in the last line, we have used that η lies on the line between W and $W + U$, so that $\|\eta\| \leq \|U\|$, and that the Hessian is Lipschitz. Thus, the proof is complete.

The last piece we will need is the uniform convergence of the Hessian, which uses the fact that the Hessian is Lipschitz continuous.

Lemma A.5. *In the setting of Theorem 4.1, there exist values C_2, C_3 that do not grow with n and $1/\delta$, such that for any $\delta > 0$, with probability at least $1-\delta$ over the randomness of the n training examples, we have*

$$\left\| \frac{1}{n}\sum_{i=1}^{n} \nabla^2[\ell(f_W(x_i), y_i)] - \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\nabla^2[\ell(f_W(x),y)]\right] \right\|_F \le \frac{C_2\sqrt{\log(C_3 n/\delta)}}{\sqrt{n}}. \tag{8}$$

The proof will be deferred to Section A.1.2. With these results ready, we will provide proof of the Hessian-based generalization bound.

A.1.1 Proof of Theorem 4.1

Proof of Theorem 4.1. First, we separate the gap between $L(W)$ and $\frac{1}{\beta}\hat{L}(W)$ into three parts:

$$L(W) - \frac{1}{\beta}\hat{L}(W) = L(W) - L_Q(W) + L_Q(W) - \frac{1}{\beta}\hat{L}_Q(W) + \frac{1}{\beta}\hat{L}_Q(W) - \frac{1}{\beta}\hat{L}(W).$$

By Lemma A.4, we can bound the difference between $L(W)$ and $L_Q(W)$ by the Hessian trace plus an error:

$$\begin{aligned} L(W) - \frac{1}{\beta}\hat{L}(W) \le & - \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\frac{1}{2}\left\langle \Sigma, \nabla^2[\ell(f_W(x),y)]\right\rangle\right] + C_1(\mathrm{Tr}\,[\Sigma])^{\frac{3}{2}} + \left(L_Q(W) - \frac{1}{\beta}\hat{L}_Q(W)\right) \\ & + \frac{1}{\beta}\left(\frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}\left\langle \Sigma, \mathrm{Tr}\left[\nabla^2[\ell(f_W(x_i),y_i)]\right]\right\rangle + C_1(\mathrm{Tr}\,[\Sigma])^{\frac{3}{2}}\right). \end{aligned}$$

After rearranging the terms, we find

$$\begin{aligned} L(W) - \frac{1}{\beta}\hat{L}(W) \le & \underbrace{- \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\frac{1}{2}\left\langle \Sigma, \nabla^2[\ell(f_W(x),y)]\right\rangle\right] + \frac{1}{n\beta}\sum_{i=1}^{n}\frac{1}{2}\left\langle \Sigma, \nabla^2[\ell(f_W(x_i),y_i)]\right\rangle}_{E_1} \\ & + \frac{1+\beta}{\beta}C_1\mathrm{Tr}\,[\Sigma]^{\frac{3}{2}} + \underbrace{L_Q(W) - \frac{1}{\beta}\hat{L}_Q(W)}_{E_2}. \end{aligned} \tag{9}$$

We analyze E_1 by separating it into two parts:

$$E_1 = \frac{1}{\beta}\left(\frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}\left\langle \Sigma, \nabla^2[\ell(f_{\hat{W}}(x_i),y_i)]\right\rangle - \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\frac{1}{2}\left\langle \Sigma, \nabla^2[\ell(f_W(x),y)]\right\rangle\right]\right) \tag{10}$$

$$+ \frac{1-\beta}{2\beta}\mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\left\langle \Sigma, \nabla^2\ell(f_W(x),y)\right\rangle\right]. \tag{11}$$

We can use the uniform convergence result of Lemma A.5 to bound the term in (10), leading to:

$$\begin{aligned} & \frac{1}{2\beta}\left(\frac{1}{n}\sum_{i=1}^{n}\left\langle \Sigma, \nabla^2\ell(f_W(x_i),y_i)\right\rangle - \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\left\langle \Sigma, \nabla^2\ell(f_W(x),y))\right\rangle\right]\right) \\ \le & \frac{\sigma^2}{2\beta}\cdot\sqrt{p}\cdot\left\| \frac{1}{n}\sum_{i=1}^{n}\nabla^2[\ell(f_W(x_i),y_i)] - \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\nabla^2[\ell(f_W(x),y)]\right]\right\|_F \qquad \text{(by Cauchy-Schwarz)} \\ \le & \frac{\sigma^2\sqrt{p}\cdot C_2\sqrt{\log(C_3 n/\delta)}}{2\beta\sqrt{n}}. \end{aligned} \tag{12}$$

In particular, the second step also uses the fact that the Hessian is a symmetric p by p matrix. As for equation (11), we have that

$$\frac{1-\beta}{2\beta}\mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\left\langle \Sigma, \nabla^2\ell(f_W(x),y)\right\rangle\right] \le \frac{1-\beta}{2\beta}\left\langle \Sigma, \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\nabla_+^2\ell(f_W(x),y)\right]\right\rangle,$$

Combined with equation (12), we have shown that

$$E_1 \leq \frac{\sigma^2 \sqrt{p} \cdot C_2 \sqrt{\log(C_3 n/\delta)}}{2\beta\sqrt{n}} + \frac{1-\beta}{2\beta} \left\langle \Sigma, \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \left[\nabla_+^2 \ell(f_W(x), y)\right] \right\rangle. \tag{13}$$

Next, for E_2, we use the PAC-Bayes bound of Theorem A.1. In particular, we set the prior distribution $\mathcal{P}$ as the distribution of U and the posterior distribution $\mathcal{Q}$ as the distribution of $W + U$. Thus,

$$E_2 \leq \frac{C\big(KL(\mathcal{Q}||\mathcal{P}) + \log(\delta^{-1})\big)}{2\beta(1-\beta)n} \leq \frac{C\Big(\frac{1}{2}W^\top \Sigma^{-1} W + \log(\delta^{-1})\Big)}{2\beta(1-\beta)n}. \tag{14}$$

Combining equations (9), (13), (14), we claim that with probability at least $1-2\delta$, we must have:

$$L(W) - \frac{1}{\beta}\hat{L}(W) \leq \frac{\sigma^2\sqrt{p}\cdot C_2\sqrt{\log(C_3 n/\delta)}}{2\beta\sqrt{n}} + \frac{1+\beta}{\beta} C_1 \mathrm{Tr}\left[\Sigma\right]^{3/2} + \frac{C\log\frac{1}{\delta}}{2\beta(1-\beta)n} \tag{15}$$

$$+ \frac{CW^\top\Sigma^{-1}W}{4\beta(1-\beta)n} + \frac{1-\beta}{2\beta}\left\langle \Sigma, \mathbf{H}_W \right\rangle, \tag{16}$$

where $\mathbf{H}_W = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\nabla_+^2 \ell(f_W(x), y)\right]$. We next choose Σ and $\beta \in (0,1)$ to minimize the above bound. In particular, we set

$$\Sigma = \sqrt{\frac{C}{2(1-\beta)^2 n \left\|W\right\|^2}} \mathbf{H}_W{}^{-\frac{1}{2}} W W^\top \tag{17}$$

so that the two terms in equation (16) are equal to each other:

$$\begin{aligned}
\frac{1-\beta}{\beta}\cdot\sqrt{\frac{C}{2(1-\beta)^2 n\left\|W\right\|^2}}\left\langle \mathbf{H}_W{}^{\frac{1}{2}} W, W\right\rangle &\leq \frac{1}{\beta}\sqrt{\frac{C}{2n\left\|W\right\|^2}}\left\|\mathbf{H}_W{}^{\frac{1}{2}}W\right\|\left\|W\right\| \\
&= \frac{1}{\beta}\sqrt{\frac{C}{2n}}\left\|W^\top\mathbf{H}_W W\right\| \\
&\leq \frac{1}{\beta}\sqrt{\frac{C\mathcal{H}}{2n}}.
\end{aligned}$$

By plugging in σ to equation (15) and re-arranging terms, the gap between $L(W)$ and $\beta^{-1}\hat{L}(W)$ can be bounded as:

$$L(W) - \frac{1}{\beta}\hat{L}(W) \leq \frac{1}{\beta}\sqrt{\frac{C\mathcal{H}}{n}} + \frac{C_2\sqrt{2p\log(C_3 n/\delta)}}{2\beta\sqrt{n}}\sigma^2 + \frac{1+\beta}{\beta}C_1\mathrm{Tr}\left[\Sigma\right]^{3/2} + \frac{C}{2\beta(1-\beta)n}\log\frac{1}{\delta}.$$

Let $\beta = 1/(1+\epsilon)$ and so that $\epsilon = (1-\beta)/\beta$. We find that

$$\begin{aligned}
&L(W) \leq (1+\epsilon)\hat{L}(W) + (1+\epsilon)\sqrt{\frac{C\mathcal{H}}{n}} + \xi, \text{ where} \\
&\qquad \xi = \frac{C_2\sqrt{2p\log(C_3 n/\delta)}}{2\beta\sqrt{n}}\sigma^2 + \Big(1+\frac{1}{\beta}\Big)C_1\mathrm{Tr}\left[\Sigma\right]^{3/2} + \frac{C}{2\beta(1-\beta)n}\log\frac{1}{\delta}.
\end{aligned}$$

Notice that ξ is of order $O(n^{-\frac{3}{4}} + n^{-\frac{3}{4}} + \log(\delta^{-1})n^{-1}) = O(\log(\delta^{-1})n^{-\frac{3}{4}})$. Therefore, we have shown that with probability at least $1-\delta$,

$$L(f_W) \leq (1+\epsilon)\hat{L}(f_W) + (1+\epsilon)\sqrt{\frac{C\mathcal{H}}{n}} + O(n^{-3/4}). \tag{18}$$

To finish the proof, we want to generalize the argument to the entire group-transformed data distribution under $\mathcal{G}$. For any $g \in \mathcal{G}$, we consider the following transformed training and test losses:

$$\begin{aligned}
\hat{L}_g(f_W) &= \frac{1}{n}\sum_{i=1}^{n}\ell(f_W(gx_i), y_i) \\
L_g(f_W) &= \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\ell(f_W(gx), y)\right].
\end{aligned}$$

By following the same argument as above, and using the bound on $\mathcal{H}$, which now applies to gx, we get that

$$L_g(f_W) \leq \hat{L}_g(f_W) + \sqrt{\frac{C\mathcal{H}}{n}} + O(n^{-3/4}).$$

By taking expectation over g, we conclude that equation (1) is true. The proof is now complete.

A.1.2 Proof of Lemma A.5

In this section, we provide the proof of Lemma A.5, which shows the uniform convergence of the loss Hessian.

Proof of Lemma A.5. Let C, $\epsilon > 0$, and let $S = \{W \in \mathbb{R}^p : \|W\| \leq C\}$. There exists an ϵ-cover of S with respect to the ℓ_2-norm with at most $\max\left(\left(\frac{3C}{\epsilon}\right)^p, 1\right)$ elements; see, e.g., Example 5.8 [40]. Let $T \subseteq S$ denote this cover. Recall that the Hessian $\nabla^2[\ell(f_W(x), y)]$ is C_1-Lipschitz for all $(W + U) \in S, W \in S$. Then we have

$$\left\|\nabla^2[\ell(f_{W+U}(x), y)] - \nabla^2[\ell(f_W(x), y)]\right\|_F \leq C_1 \|U\|.$$

For $\delta, \epsilon > 0$, define the event

$$E = \left\{\forall\, W \in T, \left\|\frac{1}{n}\sum_{i=1}^{n} \nabla^2[\ell(f_W(x_i), y_i)] - \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\nabla^2[\ell(f_W(x), y)]\right]\right\|_F \leq \delta\right\}.$$

By the matrix Bernstein inequality [39], we have

$$\Pr[E] \geq 1 - 4 \cdot |\mathcal{N}| \cdot p \cdot \exp\left(-\frac{n\delta^2}{2\alpha^2}\right). \tag{19}$$

Next, for any $W \in S$, we can pick some $W + U \in T$ such that $\|U\| \leq \epsilon$. We have

$$\left\|\mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\nabla^2[\ell(f_{W+U}(x), y)]\right] - \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\nabla^2[\ell(f_W(x), y)]\right]\right\|_F \leq C_1 \|U\| \leq C_1\epsilon$$
$$\left\|\frac{1}{n}\sum_{j=1}^{n} \nabla^2[\ell(f_{W+U}(x_j), y_j)] - \frac{1}{n}\sum_{j=1}^{n} \nabla^2[\ell(f_W(x_j), y_j)]\right\|_F \leq C_1 \|U\| \leq C_1\epsilon.$$

Therefore, for any $W \in S$, we obtain:

$$\left\|\frac{1}{n}\sum_{j=1}^{n} \nabla^2[\ell(f_W(x_j), y_j)] - \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\nabla^2[\ell(f_W(x), y)]\right]\right\|_F \leq 2C_1\epsilon + \delta.$$

Set $\epsilon = \delta/(2C_1)$ so that on the event E,

$$\left\|\frac{1}{n}\sum_{j=1}^{n} \nabla^2[\ell(f_W(x_j), y_j)] - \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\nabla^2[\ell(f_W(x), y)]\right]\right\|_F \leq 2\delta.$$

The event E happens with a probability of at least:

$$1 - 4|T|p \cdot \exp\left(-\frac{n\delta^2}{2\alpha^2}\right) = 1 - 4p \cdot \exp\left(\log|T| - \frac{n\delta^2}{2\alpha^2}\right).$$

Now, we have $\log|T| \leq p\log(3B/\epsilon) = p\log(6CC_1/\delta)$. If we set

$$\delta = \sqrt{\frac{4p\alpha^2 \log(3\tau CC_1 n/\alpha)}{n}}, \tag{20}$$

so that $\log(3\tau CC_1 n/\alpha) \geq 1$ (because $n \geq \frac{e\alpha}{3C_1}$ and $\tau \geq 1$), then we find

$$\begin{aligned}
p\log(6CC_1/\delta) - n\delta^2/(2\alpha^2) =& p\log\left(\frac{6CC_1\sqrt{n}}{\sqrt{4p\alpha^2\log(3\tau CC_1 n/\alpha)}}\right) - 2p\log(3\tau CC_1 n/\alpha) \\
=& p\log\left(\frac{3CC_1\sqrt{n}}{\alpha\sqrt{p\log(3\tau CC_1 n/\alpha)}}\right) - 2p\log(3\tau CC_1 n/\alpha) \\
\leq& p\log(3\tau CC_1 n/\alpha) - 2p\log(3\tau CC_1 n/\alpha) \\
& \qquad\qquad (\tau \geq 1, \log(3\tau CC_1 n/\alpha) \geq 1) \\
=& -p\log(3\tau CC_1 n/\alpha) \leq -p\log(e\tau). \qquad (3CC_1 n/\alpha \geq e)
\end{aligned}$$

Therefore, with a probability at least

$$1 - 4|\mathcal{N}|p \cdot \exp(-n\delta^2/(2\alpha^2)) \geq 1 - 4p(e\tau)^{-p}, \tag{21}$$

we have

$$\left\| \frac{1}{n}\sum_{j=1}^{n} \nabla^2[\ell(f_W(x_j), y_j)] - \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\nabla^2[\ell(f_W(x), y)]\right] \right\|_F \leq \sqrt{\frac{16p\alpha^2\log(3\tau CC_1 n/\alpha)}{n}}.$$

Denote $\delta' = 4p(e\tau)^{-p}$, $C_2 = 4\alpha\sqrt{p}$, and $C_3 = 12pCC_1/(e\alpha)$. With probability at least $1-\delta'$, we have

$$\left\| \frac{1}{n}\sum_{i=1}^{n} \nabla^2[\ell(f_W(x_i), y_i)] - \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\nabla^2[\ell(f_W(x), y)]\right] \right\|_F \leq C_2\sqrt{\frac{\log(C_3 n/\delta')}{n}}.$$

This completes the proof of Lemma A.5.

A.2 Examples of Label-Invariant Transformation Groups

We now describe the group of label-invariant transformations that we have used for the modular arithmetic tasks.

Example A.6 (Extensions of Example 2.1)**.** *Consider the modular subtraction task where inputs are integer pairs (a, b), and the label is given by $a - b \pmod p$. We define the label-invariant transformation group as:*

$$\mathcal{G} := \{g_k(a,b) = (a + k \pmod p, b + k \pmod p) \mid k = 0, \ldots, p-1\}.$$

In the modular multiplication task where inputs are integer pairs (a, b), and the label is given by $a \times b \pmod p$. This expression is symmetric in a and b, and remains invariant under transformations that scale one input and inversely scale the other. Thus, we define the label-invariant transformation group as:

$$\mathcal{G} := \{g_k(a,b) = (ka \pmod p, k^{-1}b \pmod p) \mid k = 1, \ldots, p-1\}.$$

For the modular division task where inputs are integer pairs (a, b), and the label is given by $a/b \pmod p$. We define the label-invariant transformation group as:

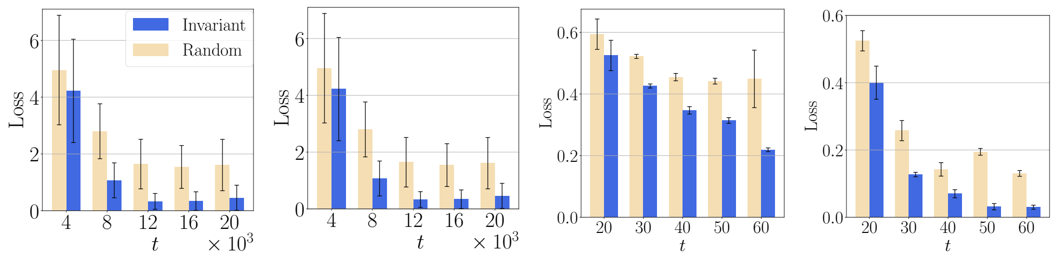
$$\mathcal{G} := \{g_k(a,b) = (ka \pmod p, kb \pmod p) \mid k = 1, \ldots, p-1\}.$$

For the symmetric quadratic task where inputs are integer pairs (a, b), and the label is given by $a^2 + ab + b^2 \pmod p$. This expression is also symmetric in a and b. We define the label-invariant transformation group as:

$$\mathcal{G} := \left\{ \begin{array}{ll}
g_0 = (a, b), & g_1 = (b, a), \\
g_2 = (p - a, p - b), & g_3 = (p - b, p - a), \\
g_4 = (a, -a - b \pmod p), & g_5 = (b, -a - b \pmod p), \\
g_6 = (-a - b \pmod p, a), & g_7 = (-a - b \pmod p, b), \\
g_8 = (p - a, a + b \pmod p), & g_9 = (p - b, a + b \pmod p), \\
g_{10} = (a + b \pmod p, p - a), & g_{11} = (a + b \pmod p, p - b)
\end{array} \right\}$$

For the asymmetric quadratic task where inputs are integer pairs (a, b), and the label is given by $a^2 + ab + b^2 + a \pmod p$. This expression is also symmetric in a and b. We define the label-invariant transformation group as:

$$\mathcal{G} := \left\{ \begin{array}{ll}
g_0 = (a, b), & g_1 = (b - 1(\bmod p), a + 1(\bmod p)), \\
g_2 = (-a - b - 1(\bmod p), b(\bmod p)), & g_3 = (b - 1(\bmod p), -a - b(\bmod p)), \\
g_4 = (-a - b - 1(\bmod p), a + 1(\bmod p)), & g_5 = (a(\bmod p), -a - b \pmod p)),
\end{array} \right\}$$



(a) $a + b \pmod p$ (b) $a^2 + ab + b^2 \pmod p$ (c) Shortest Distance (d) Triangle Counting

Figure 5: We observe that the augmented loss along invariant input directions is significantly lower than along random directions. Specifically, given a direction v, we compute the average perturbed loss $\ell(f_{W+U}(x + \alpha v), y)$ over the training set, where $U \sim \mathcal{N}(0, \sigma^2 \, \mathrm{Id})$. As training progresses, the loss decreases notably along invariant directions $v = gx - x$ for $g \in \mathcal{G}$. For comparison, we also compute the loss along random directions $v = x' - x$, where x' is sampled such that $y' \neq y$. We use $\alpha = 0.4$, $\sigma = 0.01$, and evaluate using Llama checkpoints trained with SAM.

Evaluations of the augmented losses in the label-invariant input directions. Recall from Section 2 that the Hessian trace along label-invariant input directions decreases as models begin to generalize, while it remains high along directions between inputs with different labels. We now show a similar trend in the augmented loss: the loss on inputs interpolated along label-invariant directions decreases, whereas it remains large when interpolating between inputs with different labels.

For each example (x, y), we compute the augmented loss $\ell(f_{W+U}(x + \alpha v), y)$, where v is sampled from the label-invariant direction set $V(x)$, and $U \sim \mathcal{N}(0, \sigma^2 \, \mathrm{Id})$. For comparison, we sample ten random inputs x' with $y' \neq y$ and use $v = x' - x$. As shown in Figure 5, the augmented loss is up to $9.8\times$ lower along label-invariant directions than along random directions.

Remark A.7. *The label-invariant transformation group $\mathcal{G}$ can be specified based on domain knowledge of a reasoning task. For instance, one can define a heuristic set of candidate transformations and apply automated augmentation methods to learn compositions of them that preserve the label. While we focus on leveraging the label-invariance to improve efficiency, designing automated methods to discover and compose such transformations is an interesting question for future work.*

A.3 Details of Our Algorithms

Sharpness reduction along a random augmentation. We write the formal objective corresponding to LIHR-RAND as follows.

$$\tilde{L}_{\mathrm{AVG}}(f_W) = \mathop{\mathbb{E}}_{U \sim \mathcal{N}(0, \sigma^2 \, \mathrm{Id})} \left[\frac{1}{n} \sum_{i=1}^{n} \mathop{\mathbb{E}}_{v \sim V(x_i)} \left[\ell\Big(f_{W+U}(x_i + \alpha v), y_i \Big) \right] \right].$$

To compute the augmented loss, we sample a random transformation $g \sim \mathcal{G}$ and transform the inputs as $x + \alpha(gx - x)$ at each iteration.

Sharpness reduction along the maximum-loss augmentation. An alternative is to consider optimizing the maximum loss of the interpolated inputs among the label-invariant transformations:

$$L_{\max}(f_W) = \mathop{\mathbb{E}}_{U \sim \mathcal{N}(0, \sigma^2 \, \mathrm{Id})} \left[\frac{1}{n} \sum_{i=1}^{n} \max_{v \in V(x_i)} \ell(f_{W+U}(x_i + \alpha v), y_i) \right].$$

Since computing the maximum over all directions is expensive, we select a transformation that leads to the maximum loss over m randomly sampled transformations in each iteration. For the m randomly sampled transformations, we first compute the loss on the interpolated input by each transformation, and then use the transformation with the largest loss to compute the gradient.

We then generate the interpolated input and compute the gradient update based on the perturbed model loss at the interpolated input instead. This algorithm takes m times the computation cost of performing SGD. The overall procedure is described in Algorithm 2. Our ablation studies show that minimizing the average loss over interpolated inputs tends to yield better performance in practice.

Algorithm 2 Label-Invariant Noise Regularization with the max-loss augmentation (LIHR-Max)

```
Input: Training dataset $D_{\text{train}} = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$
Require: A group of label-invariant transformations $\mathcal{G}$, augmentation scale $\alpha$, standard deviation $\sigma$,
    number of epochs $T$, batch size $B$, learning rate $\eta$, number of sub-samples $m$
Output: Trained model $f_{W^{(T)}}$
 1: $W^{(0)} \leftarrow$ A random (or pretrained LLM) initialization
 2: for $t = 1, 2, \dots, T$ do
 3:     $(x_1, y_1), (x_2, y_2), \dots, (x_B, y_B) \leftarrow$ Sample a mini-batch of data from $D_{\text{train}}$
 4:     $U^{(t-1)} \leftarrow$ A $d$-dimensional Gaussian noise sampled from $\mathcal{N}(0, \sigma^2 \, \mathrm{Id}_d)$
 5:     for $i = 1, \dots, B$ do
 6:         $\tilde{\mathcal{G}}_m \leftarrow$ Randomly sample $m$ transformations from $\mathcal{G}$
 7:         for $g \in \tilde{\mathcal{G}}_m$ do
 8:             $\ell_g(x_i, y_i) \leftarrow \ell\Big(f_{W^{(t-1)}+U^{(t-1)}}(x_i + \alpha(gx_i - x_i)),\ y_i\Big)$
 9:         end for
10:         $\tilde{g}_i \leftarrow \arg\max_{g \in \tilde{\mathcal{G}}_m} \ell_g(x_i, y_i)$
11:         $\tilde{x}_i \leftarrow (1-\alpha)x_i + \alpha \tilde{g}_i x_i$                    // Compute augmented sample
12:     end for
13:     $W^{(t)} \leftarrow W^{(t-1)} - B^{-1}\eta \sum_{i=1}^{B} \nabla \ell\Big(f_{W^{(t-1)}+U^{(t-1)}}(\tilde{x}_i), y_i\Big)$
14: end for
```

Remark A.8. *A special case of our algorithm corresponds to data augmentation when $\alpha = 1$, which samples a label-invariant neighbor, i.e., $\tilde{x} = gx$ for some $g \in \mathcal{G}$. By interpolating between x and gx, we thus leverage the strength of data augmentation while improving the robustness of training. This approach is related to mixup [**zhang2021does**, 48, 7], the difference being that we interpolate with another label-invariant augmented sample.*

A.4 Details of Hessian Computation

Next, we describe our method for computing the trace of the loss Hessian, based on an algorithm to estimate the Hessian spectrum. Given an n by n symmetric matrix A, the cumulative empirical spectral distribution $\Phi_A(t) : \mathbb{R} \to [0, 1]$ gives the fraction of eigenvalues less than or equal to a given threshold t:

$$\Phi_A(t) := \sum_{i=1}^{n} \frac{1}{n} \mathbb{1}_{\lambda_i(A) \le t},$$

where $\mathbb{1}_E$ is the indicator function of whether a probabilistic event E will happen or not.

We use the stochastic Lanczos quadrature (SLQ) algorithm [6] in Algorithm 3, which we use to estimate the above spectral distribution of the loss Hessian, denoted as $\nabla^2 \hat{L}_W$ where $\hat{L}_W$ is the empirical risk of the network with parameters W.

We orthogonalize the vectors $[\nabla^2 \hat{L}_W], [\nabla^2 \hat{L}_W]v, \dots, [\nabla^2 \hat{L}_W]v^M$ for some vector v and some parameter M. In particular, we will obtain a tridiagonal Jacobi matrix $J^{(j)}$ because $\nabla^2 \hat{L}_W$ is symmetric. The tridiagonal Jacobi matrix $J^{(j)}$ has real and simple eigenvalues. They are known as Ritz values and are approximations of the eigenvalues of $\nabla^2 \hat{L}_W$ given by the Lanczos algorithm. Under the Gauss quadrature rule, the eigenvalues of the Jacobi matrix $J^{(j)}$ are the nodes of the quadrature, and the weights $\alpha_i^{(j)}$ are the squares of the first elements of the normalized eigenvectors of $J^{(j)}$.

Performance analysis: Given two probability distribution functions p and q, the Wasserstein (earth mover) distance between p and q is

$$d_W(p, q) := \int |p(x) - q(x)| dx.$$

Based on the analysis of the stochastic Lanczos quadrature algorithm [6], we can derive the following guarantee regarding the performance of Algorithm 3. In particular, we define $\mathbb{1}[A \le x]$ as the

Algorithm 3 The stochastic Lanczos quadrature (SLQ) procedure for estimating the spectral density of a Hessian matrix

```
Input: The empirical loss function $\hat{L}_W$ of a trained model with parameters in dimension $p$
Require: Hessian matrix-vector product function $[\nabla^2 \hat{L}_W]v$ for any given vector $v \in \mathbb{R}^p$
Parameters: Degree of the Krylov subspace $M$; number of stochastic vector samples $n$
Output: An approximation of the spectral density of $\nabla^2 \hat{L}_W$
 1: $\hat{\mathbb{P}} \leftarrow \varnothing$: Initialize an (empty) empirical probability density of the spectrum of $\nabla^2 \hat{L}_W$
 2: for $j = 1, 2, \dots, n$ do
 3:     Sample a random Gaussian vector $v_1^{(j)} \in \mathbb{R}^p$, where $p$ is the dimension of the Hessian, and
        normalize $v_1^{(j)} \leftarrow \frac{v_1^{(j)}}{\|v_1^{(j)}\|}$
 4:     $\alpha_0 \leftarrow 0$, $\beta_0 \leftarrow 0$
 5:     for $i = 2, \dots, M$ do
 6:         $\alpha_i^{(j)} \leftarrow {v_{i-1}^{(j)}}^\top [\nabla^2 \hat{L}_W] v_{i-1}^{(j)}$
 7:         $w_i^{(j)} \leftarrow [\nabla^2 \hat{L}_W] v_{i-1}^{(j)} - \alpha_i^{(j)} v_{i-1}^{(j)} - \beta_{i-1}^{(j)} v_{i-1}^{(j)}$
 8:         $\beta_i^{(j)} \leftarrow \left\| w_i^{(j)} \right\|$
 9:         $v_i^{(j)} \leftarrow \frac{w_i^{(j)}}{\beta_i^{(j)}}$
10:     end for
11:     Let $J^{(j)}$ be a tri-diagonal matrix defined from $\alpha_1^{(j)}, \alpha_2^{(j)}, \dots, \alpha_M^{(j)}$ and $\beta_1^{(j)}, \beta_2^{(j)}, \dots, \beta_{M-1}^{(j)}$
        as follows
```

$$J^{(j)} = \begin{bmatrix} \alpha_1^{(j)} & \beta_1^{(j)} & 0 & \cdots & 0 \\ \beta_1^{(j)} & \alpha_2^{(j)} & \beta_2^{(j)} & \ddots & \vdots \\ 0 & \beta_2^{(j)} & \alpha_3^{(j)} & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \beta_{M-1}^{(j)} \\ 0 & \cdots & 0 & \beta_{M-1}^{(j)} & \alpha_M^{(j)} \end{bmatrix} \tag{22}$$

```
12:     $\left\{\lambda_1^{(j)}, \lambda_2^{(j)}, \dots, \lambda_M^{(j)}\right\}, \left\{\mu_1^{(j)}, \mu_2^{(j)}, \dots, \mu_M^{(j)}\right\} \leftarrow$ Eigenvalues and eigenvectors of $T^{(j)}$
13:     $\hat{\mathbb{P}} \leftarrow \hat{\mathbb{P}} \cup \left\{\left(\lambda_i^{(j)}, \left(\mu_i^{(j)}[1]\right)^2\right) : \text{for all } i = 1, 2, \dots, M\right\}$, where $\mu_i^{(j)}[1]$ represents the first
        entry of the $i$-th eigenvector
14: end for
15: Return normalized $\hat{\mathbb{P}}$
```

matrix with the same eigenvectors as A, but whose eigenvalues are 0 or 1 depending on whether the corresponding eigenvalue of A is below or above x. That is, $\mathbb{1}[A \le x]$ is the orthogonal projection onto the eigenspace associated with eigenvalues $\lambda_i(A)$ such that $\lambda_i(A) \le x$. Thus, we have that $\Phi_A(x) = n^{-1}\operatorname{Tr}[\mathbb{1}_{A \le x}]$. With this definition in place, we can define the following Gaussian quadrature rule:

$$\Phi_{A,v}(x) := v^\top \mathbb{1}[A \le x] v.$$

Proposition A.9 (Theorem 1 [6])**.** *Let $A \in \mathbb{R}^{p \times p}$ be a real-valued, symmetric matrix. Suppose v is sampled from a Gaussian distribution with its length normalized to one. Then, we have that*

$$\Pr\left[|\Phi_A(x) - \Phi_{A,v}(x)| > t\right] \le 2\exp\left(-n(p+2)t^2\right). \tag{23}$$

Further, when $n \ge \Theta\big(\ln(n\eta^{-1})/(pt^2)\big)$, the Wasserstein distance between Φ_A and the result of SLQ is bounded by $t(\lambda_{\max}(A) - \lambda_{\min}(A))$ with probability at least $1 - \delta$.

In Figure 6, we illustrate the histograms of eigenvalue spectrum estimated by the SLQ algorithms on a GPT-2 and Llama-3B model. We can observe that the SLQ algorithm yields an accurate approximation to the distribution of true eigenvalues (which are illustrated in the gray bars).

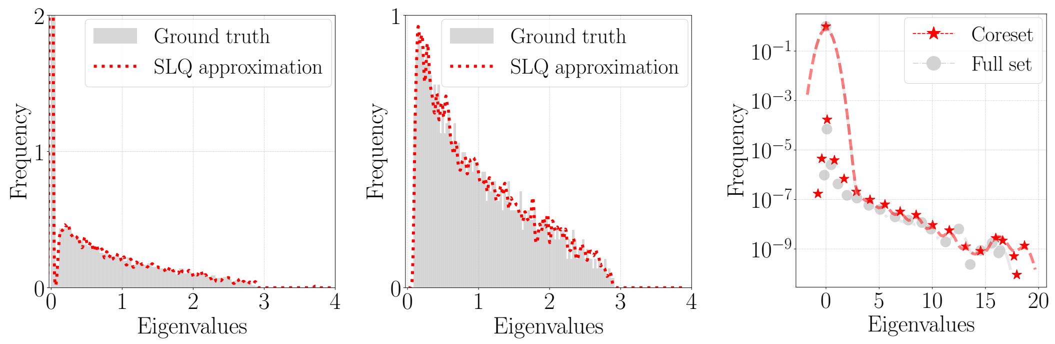


Figure 6: Comparison of the histograms of the true eigenvalues and the approximated spectrum computed using stochastic Lanczos quadrature. For the first example, the matrix is $A = 5e_1e_1^\top + 4e_2e_2^\top + 3e_3e_3^\top + \frac{1}{p}ZZ^\top$, where $e_1 = [1, 0, \ldots, 0], e_2 = [0, 1, \ldots, 0], e_3 = [0, 0, 1, \ldots]$ are the basis vectors, and Z is a p by p Gaussian random matrix whose entries are drawn with mean zero and variance one. For the second example, the matrix is $A = p^{-1}ZZ^\top$. For the third example, the matrices are the Hessians computed by full data and coreset, using Llama-3-1B.

Estimation error of SLQ vs. Hutchingson's estimator. We evaluate the accuracy of various Trace approximation algorithms on a synthetic experiment. Specifically, we consider a matrix $A = 5e_1e_1^\top + 4e_2e_2^\top + 3e_3e_3^\top + p^{-1}ZZ^\top$, where $e_1 = [1, 0, \ldots, 0], e_2 = [0, 1, \ldots, 0], e_3 = [0, 0, 1, \ldots]$ are the basis vectors, and Z is a p by p Gaussian random matrix whose entries are drawn with mean zero and variance one. We compare the SLQ algorithm with two other methods that only compute the trace of the Hessian matrix, including Hutchinson's estimation and the Hutch++ algorithm [31]. For all the methods, we iterate 100 times. We observe that the trace estimation error for SLQ is 2%, for the Hutchinson method is 0.1%, and for the Hutch++ method is 0.2%. The results suggest that all three methods give accurate estimations of Hessian trace, while SLQ provides more information about the Hessian spectrum.

Accelerating Hessian computation using coresets. Furthermore, in LLM training scenarios, the primary computational bottleneck in Hessian approximation algorithms lies in computing Hessian-vector products, which requires computing the model's second-order derivatives. Assuming the Hessian function is smooth, we explore reducing the number of backward passes by applying a coreset-based downsampling strategy to estimate the Hessian spectrum using fewer batches.

We compute the Hessian spectrum using only 10% of the batches in the arithmetic addition task using the LLaMA-3-1B model. To construct this coreset, we first compute the loss value for each batch in the dataset. We then apply K-means clustering to the loss values and select the batch closest to each cluster centroid. These batches are then used to compute the down-sampled Hessian spectrum. We iterate 20 times to get the SLQ approximation. The resulting spectrum is shown in Figure 6. Compared to SLQ on the full dataset, the trace estimation error under this coreset approach is only 2%. However, we now get a $10\times$ speedup due to downsampling.

B Supplementary Materials for Section 3

B.1 Omitted Experimental Setup

Datasets. For modular arithmetic datasets, we consider the following functions, with $p = 97$: $a + b \pmod{p} = c$, $a \times b \pmod{p} = c$, $a/b \pmod{p} = c$, $a^2 + ab + b^2 \pmod{p} = c$, and $a^2 + ab + b^2 + a \pmod{p} = c$. For the transformer experiments, we construct a dataset of equations of the form "a," "$\circ$," "b," "$=$," "c," where "x" represents the encoding corresponding to a token x. For the LLM experiments, we construct the text dataset of the form "a and $b = c$."

For graph-algorithmic datasets, the input text consists of a graph and task-specific descriptions, with no additional prompts. The graph consists of a sequence of edges, where each edge is undirected such as (a, b), indicating an edge between node a and node b. The task description specifies the prediction target. For node degree, the task description is a single node index for which we want to return the node degree. For the shortest distance, the task description consists of two node indices, indicating a pair of query nodes. For triangle counting, the task is to count the total number of triangles in the graph. For maximal flow, the task-specific description consists of two node indices, specifying the source and sink nodes for computing the maximum flow. For the cycle check, the task is to check if there is a cycle in the graph.

Implementations and hyperparameters: For the two-layer transformer experiments, we vary the learning rate η from 1×10^{-4} to 1×10^{-3}, the weight decay regularization λ from 0.1 to 5, and the interpolation parameter α from 0.1 to 0.9. Based on cross-validation, we select $\eta = 1 \times 10^{-4}$, $\lambda = 1$, and $\alpha = 0.2$. For the LLM experiments, we vary the learning rate η from 1×10^{-5} to 1×10^{-4}, the weight decay λ from 0 to 20, and the interpolation parameter α from 0.1 to 0.9. Based on cross-validation, we select $\eta = 5 \times 10^{-5}$ and $\lambda = 10$. We set $\alpha = 0.5$ for the arithmetic tasks, and for graph reasoning tasks, we set $\alpha = 0.8$.

We use a hidden dimension of 128 and 4 attention heads for the two-layer Transformer experiments. We use Llama-3.2-1B for training from scratch and Qwen3-8B for LoRA fine-tuning for the LLM experiments. We report the hyperparameters for training transformers and LLMs on modular arithmetic datasets in Table 3.

Table 3: We report the hyperparameters for training transformers on arithmetic tasks (top) and LLMs (bottom) on text-based arithmetic and graph tasks. In the bottom table, we train Llama-3-1B from random initialization for the first eight tasks, and we fine-tune Qwen3-8B for the last three tasks using LoRA (to speed up training). The rank parameter of LoRA corresponds to the rank of the low-rank adapter used in LoRA, and the weight parameter of LoRA is the weighting factor for the adapter module.

Transformers	Training dataset %	Batch size	Learning rate	Weight decay	# transformer layers	Model dimension	Attention heads
$a+b$	0.5	512	1e-4	1	2	128	4
$a \times b$	0.5	512	1e-4	1	2	128	4
a^2+ab+b^2	0.9	512	1e-4	1	2	128	4
$a-b$	0.5	512	1e-4	1	2	128	4
$\frac{a}{b}$	0.5	512	1e-4	1	2	128	4
a^2+ab+b^2+a	0.9	512	1e-4	1	2	128	4

LLMs	Training dataset %	Batch size	Learning rate	Weight decay	Pretrained LLM	LoRA rank	LoRA weight
$a+b$	0.5	512	5e-5	10	Llama-3-1B	/	/
$a \times b$	0.5	512	5e-5	10	Llama-3-1B	/	/
a^2+ab+b^2	0.5	512	5e-5	10	Llama-3-1B	/	/
$a-b$	0.5	512	5e-5	10	Llama-3-1B	/	/
$\frac{a}{b}$	0.5	512	5e-5	10	Llama-3-1B	/	/
a^2+ab+b^2+a	0.5	512	5e-5	10	Llama-3-1B	/	/
Node degree	0.5	16	5e-5	10	Llama-3-1B	/	/
Cycle Check	0.3	16	5e-5	10	Llama-3-1B	/	/
Shortest Distance	0.5	32	5e-5	10	Qwen3-8B	16	128
Triangle Counting	0.5	32	5e-5	10	Qwen3-8B	16	128
Maximul Flow	0.5	32	5e-5	10	Qwen3-8B	16	128

B.2 Complete Experimental Results

Mitigating grokking of transformers. We report the absolute number of training steps for transformers to achieve grokking on three modular arithmetic datasets: $a + b \pmod p$, $a \times b \pmod p$, $a^2 + ab + b^2 \pmod p$, in Table 4.

In addition, we present the results of grokking steps on two-layer transformers on three asymmetric arithmetic tasks, including $a - b \pmod p$, $a/b \pmod p$, and $a^2 + ab + b^2 + a \pmod p$. Consistently, we observe that our algorithm reduces the number of steps to achieve grokking by 40% compared to existing methods.

Table 4: We report the number of epochs for training transformers to generalize (reaching 100% test accuracy) on modular arithmetic datasets, comparing our algorithm to existing regularization and sharpness-reducing methods. We run each experiment with three random seeds to report the mean and standard deviation.

Task	$a+b \pmod p$	$a\times b \pmod p$	$a^2+ab+b^2 \pmod p$
Weight Decay	$1.5_{\pm 0.1}\times 10^4$	$1.5_{\pm 0.1}\times 10^4$	$4.5_{\pm 0.2}\times 10^4$
Dropout	$1.5_{\pm 0.1}\times 10^4$	$1.5_{\pm 0.1}\times 10^4$	$4.0_{\pm 0.2}\times 10^4$
SAM	$1.6_{\pm 0.2}\times 10^4$	$1.5_{\pm 0.1}\times 10^4$	$8.8_{\pm 0.1}\times 10^4$
Frob-SAM	$1.5_{\pm 0.2}\times 10^4$	$1.4_{\pm 0.2}\times 10^4$	$4.6_{\pm 0.4}\times 10^4$
NSO	$8.3_{\pm 0.3}\times 10^3$	$9.5_{\pm 0.1}\times 10^3$	$5.4_{\pm 0.4}\times 10^4$
Data Augmentation	$1.4_{\pm 0.1}\times 10^4$	$1.4_{\pm 0.1}\times 10^4$	$5.3_{\pm 1.6}\times 10^4$
GrokTransfer	$6.9_{\pm 0.0}\times 10^3$	$9.0_{\pm 0.1}\times 10^3$	$6.8_{\pm 0.1}\times 10^4$
LIHR-MAX	$\mathbf{5.3_{\pm 0.5}\times 10^3}$	$\mathbf{5.3_{\pm 0.7}\times 10^3}$	$3.0_{\pm 0.2}\times 10^4$
LIHR-RAND	$6.7_{\pm 0.1}\times 10^3$	$5.9_{\pm 0.1}\times 10^3$	$\mathbf{2.9_{\pm 0.2}\times 10^4}$

Task	$a-b \pmod p$	$\frac{a}{b} \pmod p$	$a^2+ab+b^2+a \pmod p$
Weight Decay	$2.5_{\pm 0.3}\times 10^4$	$1.6_{\pm 0.2}\times 10^4$	$4.8_{\pm 0.5}\times 10^4$
SAM	$2.1_{\pm 0.1}\times 10^4$	$2.1_{\pm 0.2}\times 10^4$	$1.1_{\pm 0.2}\times 10^5$
NSO	$1.3_{\pm 0.1}\times 10^4$	$1.5_{\pm 0.1}\times 10^4$	$7.7_{\pm 0.3}\times 10^4$
Data Augmentation	$2.4_{\pm 0.3}\times 10^4$	$2.2_{\pm 0.3}\times 10^4$	$1.2_{\pm 0.1}\times 10^5$
GrokTransfer	$\mathbf{1.2}_{\pm 0.2}\times 10^4$	$1.3_{\pm 0.2}\times 10^4$	$5.8_{\pm 0.3}\times 10^4$
LIHR-MAX	$1.4_{\pm 0.5}\times 10^4$	$1.3_{\pm 0.2}\times 10^4$	$7.6_{\pm 0.4}\times 10^4$
LIHR-RAND	$\mathbf{1.2}_{\pm 0.1}\times 10^4$	$\mathbf{9.2}_{\pm 0.1}\times 10^3$	$\mathbf{4.7}_{\pm 0.2}\times 10^4$

Additional results of training LLMs on text-based math reasoning tasks. Further, we evaluate our approach for training text-based descriptions of additional arithmetic and graph-algorithmic datasets with LLMs. The results are shown in Table 5. Using data augmentation, with or without noise injection, improves test accuracy by 28% on average over the closest baseline, SAM. Our algorithm further improves over data augmentation by 1% on average.

Table 5: We report the test accuracy % of training LLMs on three text-based modular arithmetic and two graph-algorithmic tasks. We compare our approach with existing regularization methods and sharpness-reduction methods. We run each experiment with three random seeds to report the mean and standard deviation.

Text Inputs	$a-b$	$\frac{a}{b}$	a^2+ab+b^2+a	Max Flow	Cycle Check
Weight Decay	$49.9_{\pm 0.1}$	$51.0_{\pm 0.1}$	$50.1_{\pm 0.1}$	$93.3_{\pm 0.2}$	$95.3_{\pm 0.2}$
SAM	$49.9_{\pm 0.1}$	$51.0_{\pm 0.1}$	$50.1_{\pm 0.4}$	$96.0_{\pm 0.3}$	$93.4_{\pm 0.2}$
NSO	$49.9_{\pm 0.1}$	$51.0_{\pm 0.1}$	$50.1_{\pm 0.1}$	$93.2_{\pm 0.2}$	$93.4_{\pm 0.2}$
Data Augmentation	$\mathbf{99.9}_{\pm 0.1}$	$\mathbf{99.9}_{\pm 0.1}$	$85.5_{\pm 0.7}$	$98.1_{\pm 0.1}$	$97.4_{\pm 0.2}$
LIHR-MAX	$\mathbf{99.9}_{\pm 0.1}$	$\mathbf{99.9}_{\pm 0.1}$	$78.6_{\pm 0.7}$	$98.1_{\pm 0.1}$	$94.6_{\pm 0.2}$
LIHR-RAND	$99.2_{\pm 0.1}$	$\mathbf{99.9}_{\pm 0.1}$	$\mathbf{90.5}_{\pm 0.3}$	$\mathbf{98.6}_{\pm 0.2}$	$\mathbf{98.8}_{\pm 0.2}$

Generalization to out-of-domain samples. We report the results of evaluating our approach on inputs where numbers are sampled between p and $2p-1$, larger than the numbers used for training on arithmetic datasets, in Table 6. We note that the test accuracy drops to zero for the baselines. Our approach still achieves **69**% average accuracy over the arithmetic datasets.

B.3 Ablation Analysis of Data Augmentation

We report the results of varying the augmentation scale α in Table 7. We find that using a α between 0 and 1 yields better results, while using $\alpha = 1$ (exact data augmentation, excluding test data) might lead to unstable training or take **43**% longer steps to grokking.

Table 6: We compare the test accuracy on out-of-domain samples of modular arithmetic datasets. We generate the out-of-domain samples by sampling inputs between p and $2p-1$, which is a different value range compared with the training data samples.

Task	$a+b \pmod p$	$a \times b \pmod p$	$a^2+ab+b^2 \pmod p$
Weight decay	$\leq 1.0_{\pm 0.1}$	$\leq 1.0_{\pm 0.1}$	$\leq 1.0_{\pm 0.1}$
SAM	$\leq 1.0_{\pm 0.1}$	$\leq 1.0_{\pm 0.1}$	$\leq 1.0_{\pm 0.1}$
NSO	$\leq 1.0_{\pm 0.1}$	$\leq 1.0_{\pm 0.1}$	$\leq 1.0_{\pm 0.1}$
LIHR-RAND	**86.2**$_{\pm 0.3}$	**90.6**$_{\pm 0.3}$	**32.7**$_{\pm 0.7}$

Table 7: We vary the augmentation size α in our algorithm, where $\alpha = 1$ corresponds to standard invariant data augmentation. A small step size less than 1 generally leads to the best results, since $\alpha = 1$ generally results in longer training until perfect generalization. "/" indicates the model does not reach perfect generalization.

α	0.2	0.4	0.6	0.8	1
$a+b$	$8.5_{\pm 0.2} \times 10^3$	$\mathbf{6.9_{\pm 0.3} \times 10^3}$	$7.3_{\pm 0.7} \times 10^3$	$9.8_{\pm 0.5} \times 10^3$	$1.4_{\pm 0.1} \times 10^4$
$a \times b$	$8.3_{\pm 0.8} \times 10^3$	$\mathbf{6.8_{\pm 0.4} \times 10^3}$	$7.9_{\pm 0.4} \times 10^3$	$9.9_{\pm 0.8} \times 10^3$	$1.4_{\pm 0.1} \times 10^4$
a^2+ab+b^2	$\mathbf{2.9_{\pm 0.2} \times 10^4}$	$5.9_{\pm 0.3} \times 10^4$	/	/	$4.2_{\pm 0.5} \times 10^4$

Lastly, we discuss the comparison between LIHR-RAND and LIHR-MAX, which chooses either a random augmentation or the maximum-loss augmentation, respectively. We find that on the modular arithmetic tasks, LIHR-RAND further reduces the number of grokking steps compared with LIHR-MAX by 7%.

C Extended Related Work Discussion

C.1 Hessian Dynamics of Neural Networks

The study of the Hessian of neural networks dates back to early work on second-order optimization methods, such as Newton or quasi-Newton methods, which utilize the Hessian matrix in approximating the optimization landscape [**lecun1990second**, **becker1988improving**, 24]. Despite extensive studies about the loss surface of neural networks, the dynamics of the Hessian throughout training are not well-understood in the context of large models. In the regime of neural tangent kernels [20], the Hessian matrix remains close to the kernel throughout training [**arora2019fine**]. Several recent studies have sought to observe the spectral structure of the Hessian with various optimizers. For instance, **ghorbani2019investigation** use the SLQ algorithm to track the full Hessian spectrum during the training of ResNets and show that large isolated eigenvalues emerge in un-normalized networks, concentrating gradients in sharp directions and leading to slow optimization. **wang2022analyzing** carefully examine gradient descent dynamics and introduce a four-phase decomposition of the gradient descent trajectory and empirically characterize the "edge of stability" regime, where sharpness grows to and oscillates near the critical threshold of $2/\eta$, where η denotes the step size.

Recent work has also sought to study the Hessian spectrum of transformer models, which gives insight into optimization behaviors of SGD vs. Adam [50]. Besides, the Hessian is related to the sharpness of loss surfaces and thus connects to generalization. For instance, recent work has decomposed the Hessian of the loss function into a positive semi-definite matrix and another matrix termed the nonlinear modeling error matrix [11]. The Hessian also appears in the evaluation of influence functions, which have revealed internal structures of LLMs [17].

C.2 Grokking Analysis on Arithmetic Tasks

Recent studies have sought to understand the phenomenon of grokking by carefully analyzing training dynamics from different perspectives. **xu2023benign** show that on an XOR cluster data distribution, a two-layer neural network fits the training data in one step, while the test error is close to that of a random guess. For an extended period, the network continues to overfit the training data, until the test error drops to exponentially small rates. Alternatively, Lyu et al. [28] examine modular addition tasks and consider the theory on homogeneous neural networks with large initialization and small weight decay. Mohamadi et al. [32] develop rigorous theoretical analyses of learning dmoular addition with gradient descent on two-layer MLPs. One insight from their work includes showing that starting from a small initialization can mitigate grokking on modular addition tasks. Our own experiments coincide with their findings, while further showing that small initialization does not suffice to mitigate grokking on more complicated arithmetic tasks. Finally, Xu et al. [43] expand on the XOR cluster data analysis of earlier results and develop a provable guarantee for GrokTransfer, which is a transfer learning based procedure. This approach is validated on modular addition, multiplication and parity tasks. All of these results focus on learning neural networks. Mallinar et al. [29] argue that such "emergence" is not specific to neural networks and demonstrate the presence with learning modular arithmetic with recursive feature machines, an iterative algorithm that uses the average gradient outer product (AGOP).

C.3 Invariant Sharpness Measures

The quest towards the "right" notion of sharpness has received intense studies. Beyond the largest eigenvalue or the trace of the Hessian, Tahmasebi et al. [38] argue that for non-convex loss surfaces, a parameterized family of sharpness measures is universal in the sense that it captures loss surface sensitivity for arbitrary functions of the Hessian. A very recent paper by Soleymani et al. [37] provides the first step towards capturing learning with exact invariance (or symmetries) in the setting of kernel regression. A polynomial-time algorithm is proposed based on constraining the ERM objective with spectral constraints from the group transformations. Our proposed algorithm of LIHR-RAND can be related to this procedure, the difference being that we use this constraint as a soft penalty through regularization rather than hard constraints. Another difference is that we focus on learning such invariance in arithmetic and, more generally, math reasoning tasks, which allow us to empirically evaluate the algorithms compared to prior methods in the grokking literature. Finally, the design of LIHR-MAX is new relative to their work. Further studying the theoretical properties of LIHR-RAND (and also LIHR-MAX) is an interesting question for future work.