Attention-based neural network emulators for multiprobe data vectors. III. Modeling the next generation surveys

Yijie Zhu,[1,*] Evan Saraivanov,[1,2] Joshua A. Kable,[2,3] Artemis Sofia Giannakopoulou,[1,2] Amritpal Nijjar,[1] Vivian Miranda,[1,2] Marco Bonici,[4] Tim Eifler,[5,6] and Elisabeth Krause[5,6]

[1]*Department of Physics and Astronomy, Stony Brook University, Stony Brook, New York 11794, USA*
[2]*C. N. Yang Institute for Theoretical Physics, Stony Brook University, Stony Brook, New York 11794, USA*
[3]*Department of Physics and Astronomy, Rutgers, the State University of New Jersey, Piscataway, New Jersey, 08854, USA*
[4]*Waterloo Centre for Astrophysics, Ontario, Canada N2L 3G1*
[5]*Department of Astronomy and Steward Observatory, University of Arizona, 933 N Cherry Ave, Tucson, Arizona 85719, USA*
[6]*Department of Physics, University of Arizona, 1118 E Fourth Str, Tucson, Arizona, 85721-0065, USA*

(Received 11 August 2025; accepted 14 April 2026; published 26 May 2026)

Machine learning can accelerate cosmological inferences that involve many sequential evaluations of computationally expensive data vectors. Previous works in this series have examined how machine learning architectures impact emulator accuracy and training time for optical shear and galaxy clustering two-point function. In this final manuscript, we explore neural network performance when emulating cosmic microwave background (CMB) temperature and polarization power spectra. We maximize the volume of applicability in the parameter space of our emulators within the standard Λ-cold-dark-matter model while ensuring that errors are below cosmic variance. Relative to standard multi-layer perceptron architectures, we find that the dot-product-attention mechanism reduces the number of outliers among testing cosmologies, defined as the fraction of testing points with $\Delta\chi^2 > 0.2$ relative to CAMB outputs, for a wide range of training set sizes. Such precision enables attention-based emulators to be directly applied to real data without requiring any additional correction via importance sampling. Combined with preprocessing techniques and optimized activation and loss functions, attention-based models can meet the precision criteria set by current and future CMB and lensing experiments. For each of Planck, Simons Observatory, CMB-S4, and CMB-HD, we find the fraction of outlier points to be less than 10% with around 2×10^5 to 4×10^5 training data vectors. We further explore the applications of these methods to supernova distance, weak lensing, and galaxy clustering, as well as alternative architectures and preprocessing techniques.

DOI: 10.1103/ks4r-25km

I. INTRODUCTION

Measurements of the Cosmic Microwave Background (CMB) from Simons Observatory [1,2], CMB-S4 [3], and CMB-HD [4] are anticipated to provide unprecedented precision and resolution of both the temperature and polarization anisotropies. This leap in measurement capabilities will require corresponding improvements in the theoretical modeling. As such, the Boltzmann codes CAMB [5] and CLASS [6,7] will have to compute the CMB TT, TE, and EE power spectra with higher accuracy settings. This, in turn, leads to a substantial increase in required computational resources and runtimes.

This slowdown has direct implications for Markov Chain Monte Carlo (MCMC) runs when constraining cosmological parameters, with chains now needing days, sometimes weeks, to achieve appropriate convergence. Such high demand for computational resources will be further worsened in the exploration of models beyond ΛCDM as they introduce additional computational complexity and increase the number of parameters [see, e.g., [8]].

Two main avenues exist to reduce the computational resources required to infer the probability distribution of cosmological parameters using Bayesian techniques, and they both can benefit from machine learning acceleration. First, there are innovative methods that sample the parameter space more efficiently, and they include Hamiltonian Monte Carlo (HMC) [9–12], which is a more efficient method for sampling points during the MCMC process. In this case, deep learning can be used to approximate the probability distribution from a minimum set of MCMC accepted points [13]. Another example is the nested sampling method [14], which transforms the computation of the posterior to a one-dimensional integral over prior mass.

*Contact author: yijie.zhu@stonybrook.edu

Alternatively, one can generate high-fidelity approximations for the outputs of Boltzmann codes, as shown in [15] where the authors propose to solve for the evolution of cosmological perturbations without the Boltzmann hierarchy. In this paper, we will emulate CAMB output for the TT, TE, and EE power spectra with machine-learning-based emulators [16,17]. Emulators offer several advantages over traditional MCMC, including the full parallelization of the computation of the training set.

Previous works have explored using emulators to project from data vector space to parameter space. For instance, the neural density estimator was explored in [18]. This method approximates the posterior by recurrently training emulators that go from data to parameters. In this paper, we do not consider this approach, as we would like to develop techniques that can be flexibly adapted to various analysis procedures more than MCMC and produce products that are reusable in analyses involving varying analysis choices (e.g., scale cuts, prior cuts, fixed parameters, etc.)

The development of emulators for CMB power spectra is well established, and several works have trained machine learning models to emulate the CMB power spectra, including COSMOPOWER, which has been extensively validated on the multipole range and on the precision of both current and future experiments while covering moderate volumes in parameter space [16]. A recent work extended the multipole range in both standard and extended models for an expected allowed cosmological parameter volume from data from a Planck-like or more constraining CMB experiment [19]. Furthermore, previous works have explored the impact of sophisticated pre-processing techniques [10] and alternative machine learning architecture [20,21] on the performance of the emulator.

Two aspects merit further analysis and motivate our work. The first pertains to the validation of machine learning emulators on cosmic-variance-limited experiments, with particular attention not only to the median likelihood error but also to the fraction of outliers with errors that exceed the acceptable threshold for a theoretical approximation to be applied without requiring additional refitting via importance sampling. The second aspect is related to the volume of parameter space where the emulator produces meaningful results in ΛCDM, as this can be a proxy for how the emulator scales with the number of parameters in extended models. The large volume of applicability also allows collaborations to apply blinding methods in their analysis by shifting the data vector by an arbitrary amount without risking generating nonsensical results.

Motivated by these questions, we build a new class of attention-based CMB emulators, expanding two previous analyses that we hereafter refer to as part I [20] and part II [21]. In Parts I and II, the authors applied so-called self-attention mechanisms, typically used in large language models, to emulate optical shear and clustering data vectors. They showed that these architectures reduce emulation error at a fixed training size. The attention mechanism was beneficial for mitigating outliers inconsistent with the maximum error criteria $\Delta\chi^2 = 0.2$ set by the dark energy survey (DES) [22], which allowed for uniform emulator performance over a large parameter volume.

In Sec. II, we summarize the machine learning architectures, attention mechanisms, and activation functions that we explore in this work. In Sec. III, we outline our procedure for training emulators, and in Sec. IV we show the results. In Sec. V, we provide discussions and conclusions. Finally, in the Appendices, we present emulator performance for optical weak lensing, galaxy clustering, and uncalibrated supernovae.

II. MACHINE LEARNING MODELS

A. Architectures

Machine learning (ML) architectures define the data flow and the structural organization of the networks, which can significantly affect the emulator's performance given a set number of training points. The simplest ML architecture we consider in this work is the multi-layer perceptron (MLP), constructed by stacking blocks called dense layers formed by a linear transformation immediately followed by a nonlinear activation function. We assume that each linear transformation in the MLP preserves the dimensionality of the data, which means those linear transformations are always a square matrix. This simplifies the model by reducing the number of hyperparameters that can be tuned when selecting the final network architecture.

MLP designs can be prone to issues such as vanishing or exploding gradients [see, e.g., [23,24]]. One effective strategy to alleviate this failure is the introduction of residual connections between groups of two or more dense layers [25]. A block formed by dense layers and a skip connection is called a ResMLP block, and a pure ResMLP architecture is composed solely of a series of ResMLP blocks. The nomenclature of parts of machine learning models closely follows the in-depth discussion presented in Part II [21], which contains more details about how the ResMLP architecture is defined.

The more advanced transformer architecture differs from MLPs or ResMLPs by incorporating a self-attention layer that computes the similarity between different segments of the internal representation vector, potentially taking advantage of the correlated structure that exists in physically motivated data vectors. Inside the self-attention layer, the vector is subdivided into channels, and then an operation is performed to determine their similarity. Multiple valid options exist for such an operation, the original being a weighted-dot-product [26]. After the data vector passes through this attention layer, each channel is then further processed by residual dense blocks. The original implementation in Ref. [26] assumed identical ResMLP blocks on all channels, but in this work, they will be treated as independent.

B. Attention mechanisms

The original attention mechanism used in [20,21,26] is the dot-product attention. We apply this mechanism by first splitting the input data vector, x, into N data vectors of length d, resulting in a $N \times d$ matrix, X. We transform each of these vectors using three different square weight matrices, W_Q, W_K, and W_V. This results in three separate $N \times d$ matrices, $Q = W_Q X$, $K = W_K X$, and $V = W_V X$. We then perform the following operation:

$$Y(Q, K, V) = \text{SOFTMAX}\left(\frac{QK^T}{\sqrt{d}}\right)V, \tag{1}$$

where QK^T is the $N \times N$ matrix of dot products between each vector in Q and each vector in K. Each of these products acts as a weight to the vectors in V. The factor $\sqrt{d}$ is inserted to prevent the gradient from vanishing when the length of the data vector is too large [26]. Furthermore, the SOFTMAX function is defined as

$$\text{SOFTMAX}(y_j) = \frac{e^{y_j}}{\sum_i e^{y_i}}, \tag{2}$$

where i, j are subscripts of elements in the data vector. We use this dot-product attention mechanism in our baseline emulator and show schematically the architecture in Fig. 1.

In addition to the dot-product attention mechanism, we test three alternative attention mechanisms that could allow more efficient training or better accuracy with the fast-transformer modules [27,28]. The first alternate mechanism we test is called the linear attention model [29], which is motivated by the first-order Taylor approximation of the exponential that appears in the SOFTMAX function. This approximation assumes that the dot product QK^T is small, which would occur if there are few common features in the data vector. In this case, plugging Eq. (1) into Eq. (2) results in exponential terms that to linear order are approximately

$$e^{Q_i K_j^T/\sqrt{d}} \approx 1 + \frac{Q_i K_j^T}{\sqrt{d}}, \tag{3}$$

which reduces the computational resources for evaluating the attention matrix, Y. For the linear attention model, we replace all the exponential functions in Eq. (2) with this approximation [29].

Similar to the idea of linear attention, latent attention also aims at reducing the computational cost of attention matrices by breaking the large attention matrices into decompositions of smaller ones [30]. The general idea is to write, for some large matrix A,

$$A^{(n\times n)} \approx A'^{(n\times n)} = A_1^{(n\times k)} A_2^{(k\times n)}. \tag{4}$$

The number of trainable parameters in $A^{(n\times n)}$ is n^2, while it is $2nk$ in the combination $A_1^{(n\times k)}$ and $A_2^{(k\times n)}$. As long as $k \leq n/2$, this approximation reduces the amount of trainable parameters needed. To test this, we replace all K, Q, V matrices with this decomposition and use $k = n/4$ for a factor of 2 reduction of parameter number in the attention layers.

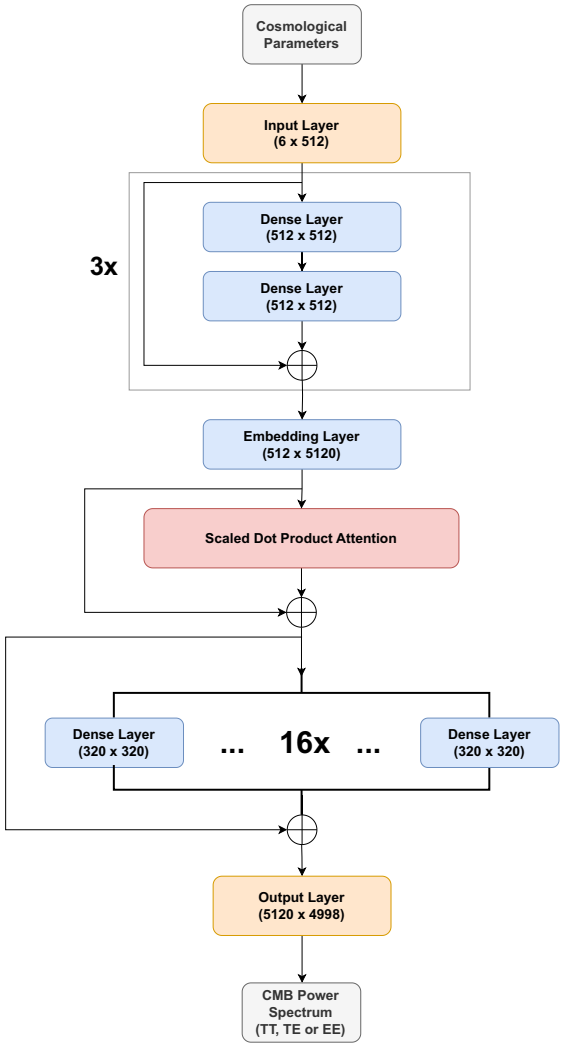

FIG. 1. Baseline transformer architecture used in this work includes three ResBlocks and one 16-channel transformer block. The baseline transformer uses a dot-product attention layer. We train an individual emulator for each of the TT, TE, and EE power spectra; however, we evaluate the accuracy of the emulators by concatenating all three power spectra together.

Alternatively, the data vector may have features that are important for the emulator to learn. The locality sensitive hashing (LSH) attention model [31] is designed to find these most important features while ignoring smaller common features in the data. The LSH model is designed to reduce the computational cost associated with the calculation of the (QK^T) matrix, an intermediary, yet computationally intense, step in computing $\text{SOFTMAX}(QK^T)$ in Eq. (1). Qualitatively, the LSH scheme only evaluates the larger elements of QK^T, reducing the number of rounds of

evaluations for this $\mathcal{O}(d^2)$ computation. The QK^T matrix is recast as a block diagonal matrix, where each block contains the dot product on a subset of the Q and K matrices that are most similar to one another. This similarity is determined by a random projection LSH algorithm that randomly and systematically partitions the space of the K matrix into a number of hashing buckets and then defines parts of the K and Q matrices that fall into the same bucket as similar. The dot product is calculated only for the parts of the K and Q matrices that fall within the same hashing buckets, leading to a block diagonal matrix, so the computation uses fewer resources. Finally, we explore the attention-free transformer (AFT) model [32], which avoids the matrix multiplication in Eq. (1) and instead uses an element-wise product. This reduces the number of computations that need to be performed as information passes through this layer, which could allow it to scale to larger data vectors more efficiently than the standard dot-product mechanism. For the offsets not including the full matrix multiplication, the AFT adds an additional position bias matrix, w, of new trainable parameters that allows the model to learn long-range correlations in the data. In particular, the formula for calculating the attention-weighted output from the AFT model is

$$Y_t = \sigma_q(Q_t) \odot \frac{\sum_{t'=1}^{N} \exp(K_{t'} + w_{t,t'}) \odot V_{t'}}{\sum_{t'=1}^{N} \exp(K_{t'} + w_{t,t'})}, \quad (5)$$

where Q, K, V are defined the same way as in the dot-product attention, $\odot$ is the element-wise multiplication, and σ_q is set to be a SIGMOID function.

C. Activation function

Activation functions enable neural networks to learn nonlinear behavior. Among the most common choices is the rectified linear unit (RELU), which behaves linearly for positive inputs while returning zero for negative inputs. This behavior may not be optimal for normalized inputs, as we normalize both the input and output of the emulator by taking $\tilde{y} = (y - \langle y \rangle)/\sigma_y$ (i.e., both inputs and outputs of the model will be centered around 0 with variance equal to 1). This motivates using the alternative TANH function, as was explored in Part II [21], where the TANH activation function was shown to perform better for weak lensing emulators.

A third activation function, denoted $\mathrm{H}(x)$ and previously used in [16,33,34], is defined as

$$\mathrm{H}(x) = (\gamma + (1 + e^{-\beta \odot x})^{-1} \odot (1 - \gamma)) \odot x, \quad (6)$$

where γ and β are trainable vectors with the same shape as the data vector, x. The vector γ determines the mixing between the linear and nonlinear components, while β controls the sharpness of the function near $x = 0$. According to [33], the flexibility of this activation function allows it to capture both sharp and smooth features that may be present in the data.

III. TRAINING PROCESS

This section outlines the adopted training strategies, sampling, and pre-processing methods used in this work. Their performance will be evaluated using two primary metrics: by the predicted median, $\langle \Delta\chi^2 \rangle_{\mathrm{med}}$, and the fraction of outliers, $f(\Delta\chi^2 > 0.2)$, in the testing sample, with $\Delta\chi^2$ defined as

$$\Delta\chi^2 \equiv (\boldsymbol{y}_{\mathrm{CAMB}} - \boldsymbol{y}_{\mathrm{emu}})^T \Sigma^{-1} (\boldsymbol{y}_{\mathrm{CAMB}} - \boldsymbol{y}_{\mathrm{emu}}). \quad (7)$$

Here, $\boldsymbol{y}_{\mathrm{Z}}$ is the data vector formed by the concatenation $(C_\ell^{\mathrm{TT}}, C_\ell^{\mathrm{TE}}, C_\ell^{\mathrm{EE}})$, $\mathbf{y}_{\mathrm{CAMB}}$ is computed by CAMB, and $\boldsymbol{y}_{\mathrm{emu}}$ is computed by the machine-learning emulator. This data vector is the concatenation of the CMB temperature (TT) and polarization (EE) auto-spectra, as well as their (TE) cross-spectra. In addition, Σ denotes the data covariance for a cosmic-variance-limited experiment,

$$\Sigma = \begin{pmatrix} \Sigma^{\mathrm{TTTT}}_{\ell\ell'} & \Sigma^{\mathrm{TTTE}}_{\ell\ell'} & \Sigma^{\mathrm{TTEE}}_{\ell\ell'} \\ \Sigma^{\mathrm{TTTE}}_{\ell\ell'} & \Sigma^{\mathrm{TETE}}_{\ell\ell'} & \Sigma^{\mathrm{TEEE}}_{\ell\ell'} \\ \Sigma^{\mathrm{TTEE}}_{\ell\ell'} & \Sigma^{\mathrm{TEEE}}_{\ell\ell'} & \Sigma^{\mathrm{EEEE}}_{\ell\ell'} \end{pmatrix}, \quad (8)$$

with

$$\Sigma^{\mathrm{TTTT}}_{\ell\ell'} = \frac{2\delta_{\ell\ell'}}{f_{\mathrm{sky}}(2\ell+1)} (C_\ell^{\mathrm{TT}})^2, \quad (9)$$

$$\Sigma^{\mathrm{TTTE}}_{\ell\ell'} = \frac{2\delta_{\ell\ell'}}{f_{\mathrm{sky}}(2\ell+1)} C_\ell^{\mathrm{TT}} C_\ell^{\mathrm{TE}}, \quad (10)$$

$$\Sigma^{\mathrm{TETE}}_{\ell\ell'} = \frac{\delta_{\ell\ell'}}{f_{\mathrm{sky}}(2\ell+1)} ((C_\ell^{\mathrm{TE}})^2 + C_\ell^{TT} C_\ell^{\mathrm{EE}}), \quad (11)$$

$$\Sigma^{\mathrm{TEEE}}_{\ell\ell'} = \frac{2\delta_{\ell\ell'}}{f_{\mathrm{sky}}(2\ell+1)} C_\ell^{\mathrm{EE}} C_\ell^{\mathrm{TE}}, \quad (12)$$

$$\Sigma^{\mathrm{TTEE}}_{\ell\ell'} = \frac{2\delta_{\ell\ell'}}{f_{\mathrm{sky}}(2\ell+1)} (C_\ell^{\mathrm{TE}})^2, \quad (13)$$

$$\Sigma^{\mathrm{EEEE}}_{\ell\ell'} = \frac{2\delta_{\ell\ell'}}{f_{\mathrm{sky}}(2\ell+1)} (C_\ell^{\mathrm{EE}})^2, \quad (14)$$

where f_{sky} is the fraction of sky observed. In this analysis, we set $f_{\mathrm{sky}} = 1$, corresponding to a full sky measurement. The fiducial point in cosmological parameter space is chosen at a ΛCDM point near the center of the Planck constraints and is specified in Table I. We also perform sampling centered at this point in the Gaussian sampling process of later sections.

In all cases, we use power spectra that cover a multipole range of $2 \leq \ell \leq 5000$. For $\ell \leq 30$, the distribution of power spectra is non-Gaussian [see, e.g., [35–38]], meaning a likelihood function would include additional terms beyond the $\Delta\chi^2$ value. We leave exploration of these non-Gaussian contributions to future work; however, we do find that excluding $\ell \leq 30$ data does not significantly alter our conclusions.

In the data covariance, these spectra are evaluated at the fiducial cosmology described in Table I. Another relevant quantity during training is $\Delta\chi^2_{\rm XY}$, which measures the emulator accuracy of each individual spectra,

$$\Delta\chi^2_{\rm XY} \equiv (C^{\rm XY}_{\ell,\rm CAMB} - C^{\rm XY}_{\ell,\rm emu})^T (\Sigma^{\rm XYXY})^{-1} (C^{\rm XY}_{\ell,\rm CAMB} - C^{\rm XY}_{\ell,\rm emu}), \tag{15}$$

given that we train the temperature and polarization spectra independently.

A. Sampling method

We compare two sampling methods, tempered Gaussian and uniform, for generating the training and testing sets. The parameter spaces sampled by these two methods are shown in the top panels of Figs. 2 and 3, respectively. Gaussian sampling naturally incorporates correlations between parameters constrained by the data, thereby reducing the emulated parameter space volume in the orthogonal directions of the CMB degeneracies. To generate the Gaussian samples, we begin by Fisher forecasting a cosmic variance limited experiment in the multipole range of $30 \leq \ell \leq 400$ (using TT, TE, and EE power spectra) at the fiducial point. Any parameter correlation greater than 0.4 is manually reduced by a factor of 2 to help ensure proper training coverage, as the directions of CMB-induced correlations in the cosmological parameters vary as a function of the maximum adopted multipole. The overall covariance is then evenly extended by a multiplicative factor called the temperature T.

For generating the training and testing sets, we use $T_{\rm train} = 256$ and $T_{\rm test} = 128$, respectively. We additionally place hard prior limits, which are listed in Table II for each of the cosmological parameters. For $\Omega_b h^2$ and $\Omega_c h^2$, the hard prior ranges are the same for both the training and testing sets. These hard boundaries are well beyond the range of values in the $T_{\rm train} = 256$ training set. We find no evidence of hard boundary edge effects in the training of these parameters.

Following the methodology described in Ref. [20], we construct a uniform sampling training set by initially generating a Latin hypercube comprising 10^5 points to efficiently cover the parameter space within the specified prior boundaries defined in Table II [39]. Subsequently, these points are augmented by uniformly sampling additional cosmologies from the parameter space. This strategy guarantees comprehensive initial coverage while enabling an increase in the training set's size without requiring the recomputation of the entire training sample from scratch.[1]

[1]Note that it was also found that merging multiple Latin hypercubes provides worse results [20].

In most cases, the prior range in the testing set is reduced by 5% to mitigate boundary effects caused by the rigid limits of the uniform sampling. However, this is not the case for A_s, where we apply a much more stringent cut of $\log(10^{10}A_s) < 3.5$ in the testing set. This cut is motivated in part by numerical instabilities we find in the range where $\log(10^{10}A_s) > 3.5$ in CAMB outputs when using ACCURACYBOOST $= 1.5$. We explore these numerical instabilities in more depth in Appendix A.

B. Pre-processing

One challenge when emulating the CMB power spectra to small scales arises from the fact that the TT and EE power spectra span several orders of magnitude. Excluding $\ell \leq 30$ polarization data, the overall amplitude of the CMB anisotropy power spectra is approximately proportional to $A_s e^{-2\tau}$ [40]. Consequently, we can reduce the mapping between cosmological parameters and power spectra that the emulator must learn by dividing all vectors by $A_s e^{-2\tau}$; accordingly, the data vector predicted by the emulator must then be multiplied by the same factor to recover the original CMB TT, TE, and EE spectra. In Appendix F, we model the damping tail of the lensed spectra using genetic algorithms to further reduce the data vector dynamical range.

A second pre-processing technique involves decomposing the variance of the CMB power spectra within the training set via a principal components analysis (PCA) transformation. The PCA procedure reduces the dimensionality of the output, thereby facilitating the model training. Here, we assume the training sample contains $N_{\rm train}$ cosmologies $\boldsymbol{\theta}_k$. We compute the correlation matrix of the training set as

$$\mathbb{C}^{\rm XY}_{\ell_i\ell_j} = \sum_{k=1}^{N_{\rm train}} \frac{(C^{\rm XY}_{\ell_i}(\boldsymbol{\theta}_k) - \langle C^{\rm XY}_{\ell_i}\rangle)(C^{\rm XY}_{\ell_j}(\boldsymbol{\theta}_k) - \langle C^{\rm XY}_{\ell_j}\rangle)}{(N_{\rm train}-1)\sigma^{\rm XY}_{\ell_i}\sigma^{\rm XY}_{\ell_k}}, \tag{16}$$

where $\langle C^{\rm XY}_\ell\rangle$ is the average and $\sigma^{\rm XY}_\ell$ is the variance of the power spectra over the entire training set.

We then construct the principal components by computing and ranking the eigenvectors $\mathbf{K}^{\rm XY}$ of the covariance $\mathbb{C}^{\rm XY}$. The dimensionality of the data vector the emulator needs to model can then be reduced from $\ell_{\rm max}$ to $N_{\rm PCA}$ by only selecting the modes that correspond to the $N_{\rm PCA}$ largest eigenvalues, as shown below,

$$C^{\rm XY}_\ell(\vec{\theta}) = \langle C^{\rm XY}_\ell\rangle + \sum_{i=1}^{N_{\rm PCA}} \alpha_{(i)}(\boldsymbol{\theta})\sigma^{\rm XY}_\ell K^{\rm XY}_{(i),\ell}, \tag{17}$$

where α_i corresponds to the PCA amplitudes.

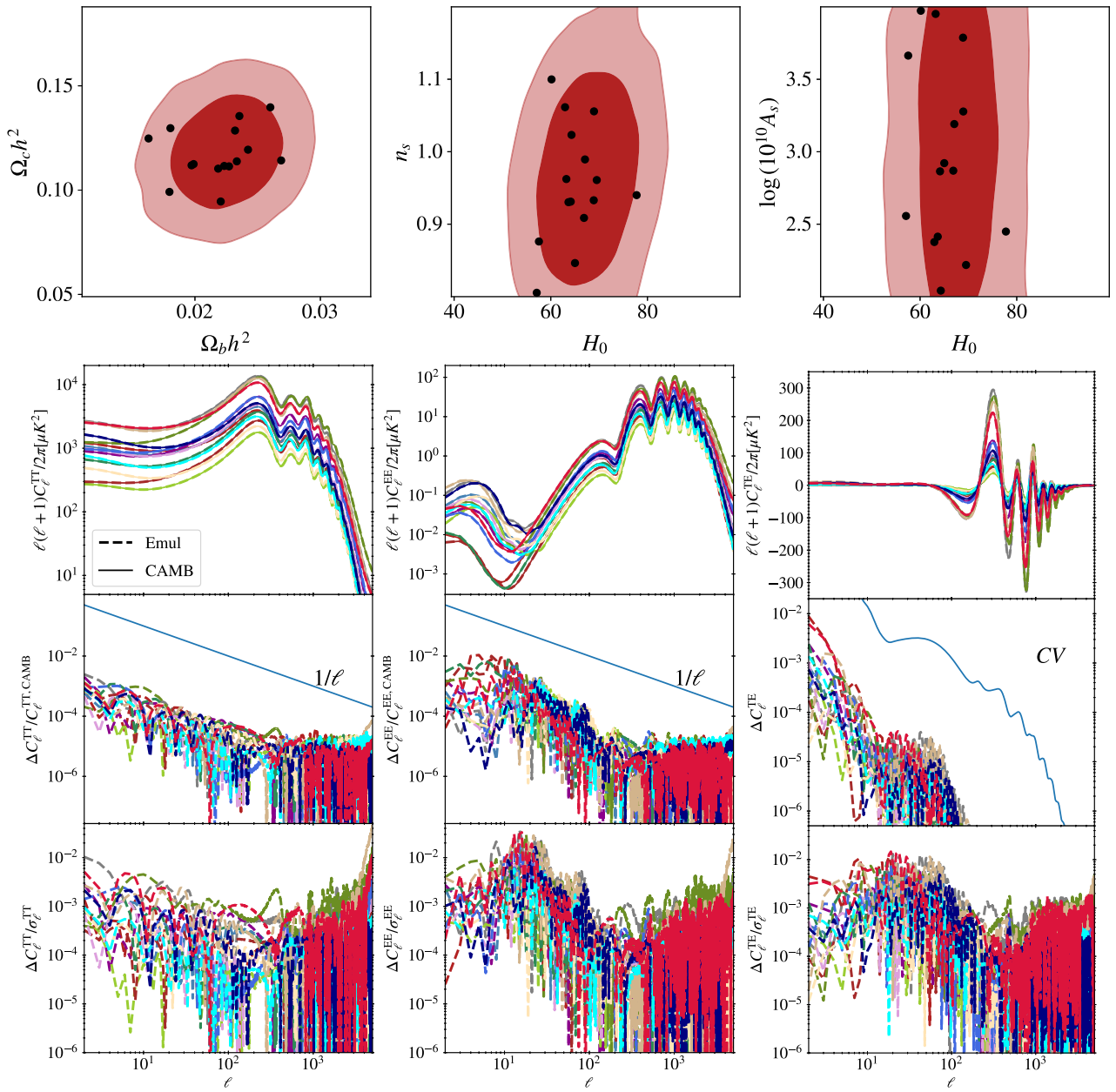

FIG. 2. Comparison between emulator outputs and CAMB outputs for 15 different cosmologies. The model is trained and tested on tempered Gaussian sets. On the top panel, we show the positions of the testing points from the $T = 128$ Gaussian sampling set that are used to calculate the TT, TE, and EE power spectra. For the emulation, we use our baseline transformer model trained on 530 thousand training points, which is described in detail at the start of Sec. IV. From left to right, we show results for each of the TT, EE, and TE spectra. The middle panels show direct comparison between emulator outputs and the true outputs. Lower panels show a comparison between $1/\ell$ scaling and the fractional difference between the emulator and CAMB outputs. We can see that, in general, the emulator errors are below the cosmic variance limits.

C. Loss function

We evaluate five candidate loss functions that differ along two critical dimensions: the weighting assigned to outliers and the rescaling of the data vectors. During training, they are all evaluated as a function of $\Delta\chi^2_{\mathrm{XY}}$ as each power spectrum is trained with an independent emulator. During testing, however, our metrics, $\langle\Delta\chi^2\rangle_{\mathrm{med}}$ and $\int(\Delta\chi^2 > 0.2)$ (outlier fraction), for evaluating the final joint emulator performance act on the data vector that concatenates all three CMB spectra and whose data covariance accounts for cross correlations between the spectra. The first loss function candidate is

$$\mathcal{L}_1(\Delta\chi^2_{\mathrm{XY}}) = \langle\Delta\chi^2_{\mathrm{XY}}\rangle, \tag{18}$$

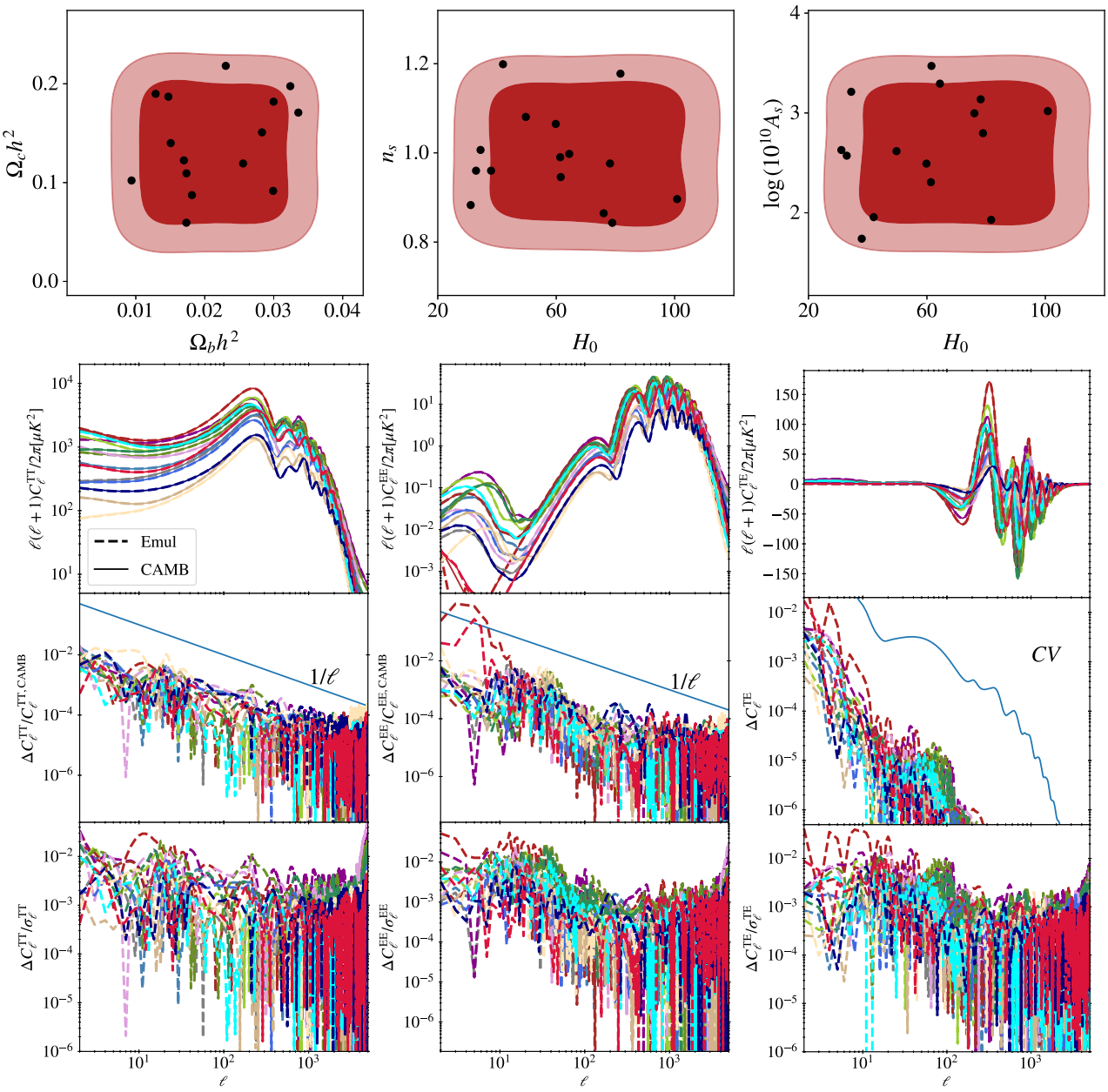

FIG. 3. Comparison between emulator outputs and CAMB outputs for 15 different cosmologies, similar to Fig. 2. However, in this case, the model is trained and tested on uniformly sampled sets instead of tempered Gaussian sets. On the top panel, we show the positions of the testing points from the uniform sampling set that are used to calculate the TT, TE, and EE power spectra. The emulator model here is the baseline transformer model, which is described in detail at the start of Sec. IV, trained with 530 thousand training points. From left to right, we show results for each of the TT, EE, and TE spectra. Upper panels show direct comparison between emulator outputs and the true outputs. Lower panels show the comparison between $1/\ell$ scaling and the fractional difference between the emulator and CAMB outputs. When trained and tested on the uniform sets, the emulators are able to produce approximated data vectors to high precision under cosmic variance limit.

which scales quadratically in the limit $\Delta\chi^2_{\rm XY} \gg 1$. In contrast, both

$$\mathcal{L}_2(\Delta\chi^2_{\rm XY}) = \langle(\Delta\chi^2_{\rm XY})^{1/2}\rangle \tag{19}$$

and

$$\mathcal{L}_3(\Delta\chi^2_{\rm XY}) = \langle(1+2\Delta\chi^2_{\rm XY})^{1/2}\rangle \tag{20}$$

weight outliers linearly.

The loss $\mathcal{L}_1$ was adopted in Part I and Part II and provided outstanding results when applied to an optical weak lensing data vector. However, $\mathcal{L}_1$ overemphasizes outlier points to the extent that the model sacrifices

TABLE I. The fiducial cosmological parameters in flat ΛCDM we adopt throughout the manuscript. These values are chosen to be close to the Planck best-fit cosmology [40].

Parameter	Value
$\Omega_b h^2$	0.02239
$\Omega_c h^2$	0.1178
H_0(km/s/Mpc)	67.5
τ	0.06
$\log(10^{10}A_s)$	3.064
n_s	0.965
Σm_ν (eV)	0.06

TABLE II. Parameter priors for the training and testing parameter sets for the uniform and Gaussian sampling methods.

	Uniform	Uniform	Gaussian	Gaussian
Parameter	Train	Test	Train	Test
$100\Omega_b h^2$	[0.6, 3.8]	[0.8, 3.5]	[0, 4]	[0, 4]
$10\Omega_c h^2$	[0.30, 2.35]	[0.40, 2.2]	[0, 3]	[0, 3]
H_0	[25, 114]	[30, 110]	[25, 114]	[30, 110]
10τ	[0.07, 1.5]	[0.1, 1.4]	[0.07, 1.5]	[0.1, 1.4]
$\log(10^{10}A_s)$	[1.61, 3.6]	[1.7, 3.5]	[1.61, 4.5]	[1.7, 4.0]
n_s	[0.7, 1.3]	[0.8, 1.2]	[0.7, 1.3]	[0.8, 1.2]

accuracy in a large number of training models in an effort to better mitigate them. While both $\mathcal{L}_2$ and $\mathcal{L}_3$ exhibit similar behavior in handling outliers, they differ in the opposite limit, with $\mathcal{L}_3$ having the potential advantage of allowing the training to disregard further optimizations on cosmologies that are already approximated below the desired threshold $\Delta\chi^2_{\rm XY} \lesssim 0.2$.

The remaining proposed loss functions differ from $\mathcal{L}_{1,2,3}$ by being computed directly from the rescaled $\tilde{C}^{\rm XY}_\ell \equiv C^{\rm XY}_\ell / A_s e^{-2\tau}$ instead of the original spectra. Although the emulator outputs $\tilde{C}^{\rm XY}_\ell$, we undo the rescaling prior to compute $\Delta\chi^2_{\rm XY}$ and $\Delta\chi^2$, as they depend on the original $C^{\rm XY}_\ell$. If the data covariance rescaling varied inside the likelihood as a function of the cosmological parameters, such scaling would be irrelevant as it would leave $\Delta\chi^2$ invariant. However, this is not the case as Σ is computed only at a fiducial cosmology. We then define both the rescaled $\Delta\tilde{\chi}^2_{\rm XY}$ that act on $\tilde{C}^{\rm XY}_\ell$ and on the rescaled covariance matrix

$$\tilde{\Sigma}_{\rm XYXY} = \Sigma_{\rm XYXY} / (A_s^{\rm fid} e^{-2\tau^{\rm fid}})^2. \tag{21}$$

The two remaining loss functions are then defined as $\mathcal{L}_4 \equiv \mathcal{L}_2(\Delta\tilde{\chi}^2_{\rm XY})$ and $\mathcal{L}_5 \equiv \mathcal{L}_3(\Delta\tilde{\chi}^2_{\rm XY})$.

D. Training hyperparameters

The LEARNING-RATE controls the extent to which additional information influences the model's trainable parameters. In our setup, the initial LEARNING-RATE is set to 10^{-3}. A learning rate scheduler then reduces this rate by a factor of 2 whenever the validation loss plateaus for 15 epochs (this tunable number of epochs is the so-called PATIENCE) until the LEARNING-RATE reaches approximately 10^{-8}. At this point, no further reductions are applied. By progressively decreasing the learning rate, the algorithm avoids trapping the model in local minima during the early training stages while allowing minor parameter adjustments at later epochs.

Neural network training adjusts the model parameters by comparing the model predictions against smaller batches of data vectors. The size of these batches can affect the time that the training takes to achieve convergence and the model's overall performance. However, limitations on the available RAM in GPUs restricted our choices. We fixed our batch size to 512, which resulted in a training duration of around ten hours on an NVIDIA RTX 3060 GPU. If we set our batch size to 256, the time for training to finish will be twice as long as our current setting because the model will be trained on twice as many batches. We tested using a batch size of 1024 with the same transformer model with the same amount of training data vectors, and the $\langle\Delta\chi^2\rangle_{\rm med}$ increases by a factor of 4 and ($f(\Delta\chi^2 > 0.2)$ increases by a factor of 1.5.

A challenge that neural networks can encounter is overfitting, in which the model becomes overly specialized to the training data and fails to generalize to unseen data. One approach to reducing the probability of such a training failure involves the introduction of penalties in the loss function that are proportional to the numerical amplitudes of the model weights. Another similar mitigation strategy forces the decay of such amplitudes. Numerically, this penalty is controlled with a parameter called the WEIGHT-DECAY. However, we do not run training with a nonzero weight decay in this work, because our baseline model, defined in Sec. IV, includes trainable parameters in the activation functions that would be negatively influenced by a nonzero weight decay. Instead, we check for overfitting by visually inspecting loss values on a validation dataset.

IV. RESULTS

In this section, we present results that quantify the performance of various emulator architectures, activation functions, loss functions, and methods for sampling parameter values for the training and testing sets. Our baseline emulator is shown schematically in Fig. 1. It includes the $N_{\rm TRF} = 1$ transformer block, and within the transformer block, the data vector is subdivided into $N_{\rm channel} = 16$ individual channels to compute the self-attention, which is implemented via the dot-product attention defined in

Eq. (1). We fix the internal dimension of each transformer layer to be INT-DIM-TRF = 5120, which is approximately the same size as our data vector[2] and is divisible by powers of 2.

Our baseline architecture precedes the transformer block with $N_{\rm ResMLP} = 3$ ResMLP blocks, each with an internal dimension of INT-DIM-RESMLP = 512. Further, we adopt the H(x) function defined in Eq. (6) as the baseline activation function, which is one of the primary differences in our emulator compared to the emulator design presented in Part II. Unless otherwise specified, we train our emulators with parameters sampled from a tempered Gaussian distribution with temperature $T_{\rm train} = 256$, and we test them with points sampled at $T_{\rm test} = 128$. Finally, we employ the $\mathcal{L}_4$ loss function, which acts on the rescaled $\Delta\tilde{\chi}^2_{\rm XY}$ (see Sec. III C for details on how the $\mathcal{L}_4$ loss function and the rescaled $\Delta\tilde{\chi}^2_{\rm XY}$ are defined).

We trained the baseline emulator using $N_{\rm train} = 5.3 \times 10^5$ training points. As illustrated in Fig. 2, our emulator replicates CAMB outputs. Quantitatively, the fractional difference between the baseline emulator and CAMB remains within cosmic variance limits for the TT, TE, and EE power spectra. This result is consistent with expectations, given that the emulator was trained against a loss function that is essentially the square root of the rescaled cosmic variance likelihood. We repeated this test using a uniformly sampled parameter space instead of a tempered Gaussian. To train this emulator, we used 1.2 million training points. We summarize the performance of the emulator in Fig. 3, which shows that our conclusions are robust to the choice of sampling method. The runtime difference against CAMB is massive, with the Boltzmann code taking over a minute and with nine OpenMP cores, while the emulator runtime is only $\mathcal{O}(0.01)$–$\mathcal{O}(0.1)$ seconds with a single core. In Table III, we summarize the major comparisons between a ResMLP + PCA model and a transformer model in terms of precision, run time, and training time. The transformer model shows higher emulation precision, while the ResMLP model requires less time to train and is faster to run during analysis. However, the difference in time consumption is not as significant as the difference in precision.

TABLE III. A general comparison between the ResMLP + PCA architecture and transformer architecture trained on the $T_{\rm train} = 256$ Gaussian set with 530 thousand points and tested on the $T_{\rm test} = 128$ Gaussian set. In the first column, the number outside of the parenthesis is the $\langle\Delta\chi^2\rangle_{\rm med}$ and the number inside the parenthesis is the fraction of outlier $f(\Delta\chi^2 > 0.2)$. The second column demonstrates the approximate run time of the emulator on one cosmology on one CPU core when running in analysis. The third column shows the approximate time needed for training for each architecture under 530 thousand training points on an NVIDIA 3060 GPU. The ResMLP model only uses half of the training time of a transformer model and a third of the run time of the transformer. However, the difference in precision is significant, as the transformer is able to achieve $f(\Delta\chi^2 > 0.2) < 10\%$ with around 500k training points, while the ResMLP model has around 20% of outliers.

Architecture	$T_{\rm train} = 256$	Run Time (s)	Train Time (hrs)
ResMLP + PCA	0.05 (0.19)	0.08	8
TRF	0.006 (0.06)	0.30	18

A. Effect of pre-processing

We test how pre-processing via rescaling the CMB power spectra by $A_s e^{-2\tau}$, which was discussed in Sec. III B, affects the emulator accuracy at fixed architecture, i.e., we do not independently re-optimize the emulator hyperparameters when comparing the loss with and without the pre-processing rescaling. In Table IV, we show that for a training set with $N_{\rm train} = 2 \times 10^5$ points drawn from a tempered Gaussian distribution, the data vector rescaling reduces the emulator errors to the level of $\langle\Delta\chi^2\rangle_{\rm med} = 0.04$ and $f(\Delta\chi^2 > 0.2) = 0.14$, representing an approximate order of magnitude improvement over the training without rescaling. In Fig. 4, we show that the error of emulation decreases consistently with rescaling compared to without rescaling.

With enough cosmologies in the training set, the neural network eventually achieves similar accuracy without pre-processing. This is evident by the second column in Table IV, where for 5.3×10^5 training points, both the rescaled and nonrescaled cases show similar performance. However, only with rescaling do we reach the desired threshold with $N_{\rm train} \lesssim 5.0 \times 10^5$. In Appendix F, we explore an additional pre-processing of the CMB power spectra with a more sophisticated rescaling of the CMB damping tail. In particular, we based our damping tail rescaling on the analytic formulas developed in [41]. Lensing transfers power to small scales, resulting in a transition from pure exponential damping, with the diffusion scale defined in Eq. (F3), to a power-law regime. We employed a genetic algorithm to determine the optimal cosmological dependence of the improved scaling defined in Eq. (F14), valid in a small region in the parameter space around a fiducial cosmology.

For the case where we use only a ResMLP architecture (i.e., where we do not include a self-attention layer), we employ PCA to reduce the output dimension. This step decomposes the data into the most relevant pieces, which can reduce the number of trainable parameters in the model and the number of features that need to be learned, as the emulator neglects the least relevant principal components. This helps control RAM usage and reduces training difficulty with a smaller model that requires less training time

[2]Recall that we emulate the TT, TE, and EE power spectra individually, so INT-DIM-TRF = 5120 is approximately the same length as one of these power spectra with a multipole range of $2 \leq \ell \leq 5000$.

TABLE IV. Comparison between the same baseline transformer architectures trained with and without the rescaled power spectra. The number outside the parenthesis is the $\langle\Delta\chi^2\rangle_{\rm med}$, and the number inside the parenthesis is the fraction of outlier $f(\Delta\chi^2 > 0.2)$.

TRF	$N_{\rm train} = 200$ k	$N_{\rm train} = 530$ k
Not rescaled	4.30 (1.00)	0.05 (0.16)
Rescaled	0.04 (0.14)	0.02 (0.07)

and is easier to converge. The number of principal components needed is selected by looking at which component is the first to appear dominated by noise. We also test how much we deviate from the actual data vector by performing a PCA and then an inverse PCA. We find that keeping the first 96 principal components is sufficient for the emulator result to obtain $\langle\Delta\chi^2\rangle_{\rm med} \sim \mathcal{O}(0.1)$ and that our results are robust to including more principal components. Higher-order principal components are mostly noisy, so they do not help generate better ML models [16,17].

We show the results of including or not including the PCA pre-processing in Tables V and VI for the transformer and ResMLP architectures, respectively. We find that including the PCA pre-processing step causes the transformer-based model to lose accuracy relative to the transformer model that does not include a PCA decomposition, so we do not include the PCA pre-processing for the baseline transformer model. On the other hand, we find that the ResMLP accuracy improves by almost three orders of magnitude when using the PCA, so we do include this step when using a ResMLP-only-based architecture. We attribute the worse performance by the transformer architecture to the fact that PCA removes features that aid the attention mechanism in finding correlations in the data.

TABLE V. Comparison between the baseline transformer architecture trained with and without first performing a PCA on the CMB power spectra. The number outside of the parenthesis is the $\langle\Delta\chi^2\rangle_{\rm med}$, and the number inside the parenthesis is the fraction of outlier $f(\Delta\chi^2 > 0.2)$.

TRF	$N_{\rm train} = 200$ k	$N_{\rm train} = 530$ k
PCA	0.550 (0.77)	0.031 (0.13)
No PCA	0.040 (0.17)	0.006 (0.06)

TABLE VI. Comparison between the same ResMLP architecture trained with and without PCA. The models are trained on the $T_{\rm train} = 256$ Gaussian set and tested on the $T_{\rm test} = 128$ Gaussian set. The number outside of the parenthesis is the $\langle\Delta\chi^2\rangle_{\rm med}$ and the number inside the parenthesis is the fraction of outlier $f(\Delta\chi^2 > 0.2)$.

ResMLP	$N_{\rm train} = 200$ k	$N_{\rm train} = 530$ k
PCA	0.10 (0.38)	0.05 (0.19)
No PCA	75.20 (1.00)	75.38 (1.00)

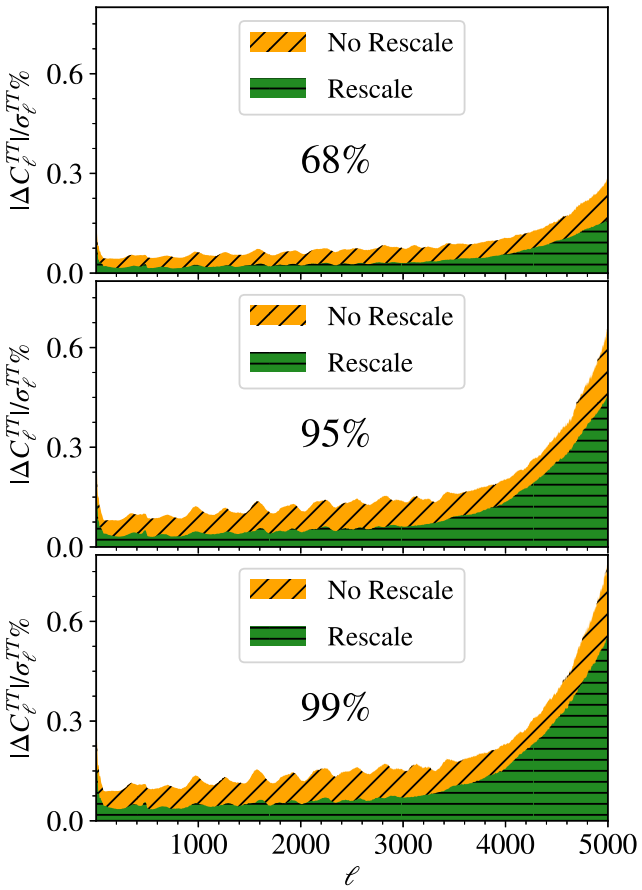

FIG. 4. Distributions of the ratio between absolute error of the baseline transformer-based emulator with CAMB and the cosmic variance standard deviation in percentage, for the TT spectrum, with and without our rescaling scheme. The models are trained on a $T_{\rm train} = 256$ Gaussian and tested on a $T_{\rm test} = 128$ Gaussian set. The emulator trained on the rescaled power spectra performs better than the one trained on the nonrescaled power spectra at every ℓ. This overall improvement results in a reduction in the value of $f(\Delta\chi^2 > 0.2)$ by a factor of 2. In both cases, the emulation error increases as ℓ increases, which we attribute to modeling the effects of the lensed CMB damping scale. In Appendix F, we test how including further pre-processing of the CMB damping scale can provide further improvement.

B. Transformer vs. ResMLP

In this subsection, we compare emulator performance between the baseline transformer architecture and a ResMLP architecture with PCA pre-processing. The ResMLP-only architecture has four ResMLP blocks, each with **Int-dim-ResMLP** = 512. We vary the number of sample points in the training set and compare the resulting $\langle\Delta\chi^2\rangle_{\rm med}$ and $f(\Delta\chi^2 > 0.2)$ values for both the baseline transformer and ResMLP-only architectures and show the results in Figs. 5 and 6.

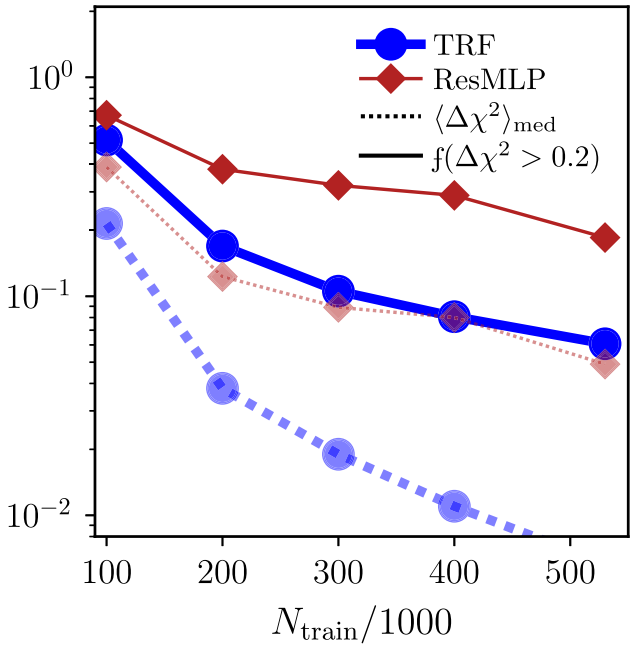

FIG. 5. Median $\Delta\chi^2$ (dashed lines) and fraction of testing points with $\Delta\chi^2 > 0.2$, $f(\Delta\chi^2 > 0.2)$ (solid lines) for different architectures. Thick blue lines are for our baseline transformer (3 ResBlocks appended with 1 transformer block), and thin red lines are for the baseline ResMLP with PCA pre-processing (4 ResBlocks). Each model was trained using a tempered Gaussian sampling set with $T_{\rm train} = 256$ and tested with a Gaussian sampling set with $T_{\rm test} = 256$. Adding additional training points results in a reduction in both accuracy metrics for both the transformer-based and ResMLP-only architectures; however, the transformer-based architecture has lower $\langle\Delta\chi^2\rangle_{\rm med}$ and $f(\Delta\chi^2 > 0.2)$ values for every number of training points tested. While the transformer-based architecture can reduce the $f(\Delta\chi^2 > 0.2) < 0.1$ with around 400 thousand training points, at even 530 thousand, the ResMLP-only-based architecture cannot achieve this same threshold.

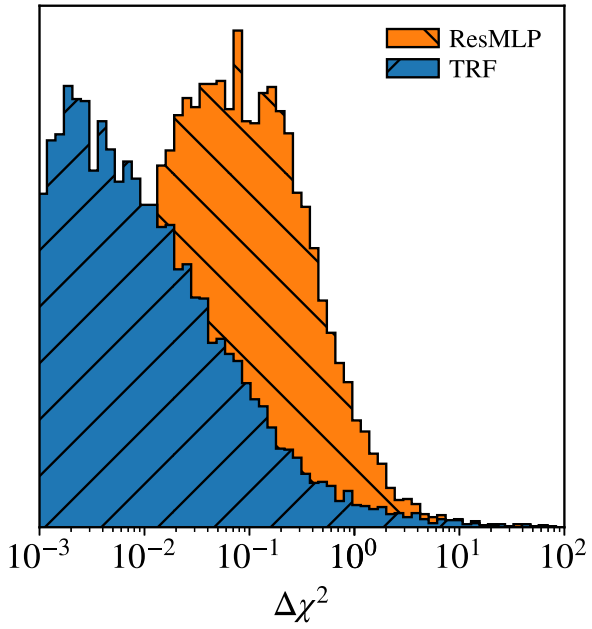

FIG. 6. Comparison between the $\Delta\chi^2$ distribution of the ResMLP and transformer models trained with 530 thousand data vectors drawn from a Gaussian sample with temperature $T_{\rm train} = 256$ and then tested on a Gaussian sampling set with temperature $T_{\rm test} = 128$. Overall, the transformer-based emulator shifts the $\Delta\chi^2$ values toward smaller values relative to the ResMLP-based architecture. Because the entire distribution shifts toward smaller values, the median value also shifts to smaller values, and the fraction of points with $\Delta\chi^2 > 0.2$ also decreases. While we choose to use $f(\Delta\chi^2 > 0.2)$ as our accuracy metric for outlier points, this plot illustrates that the transformer also has a lower fraction of outlier points than the ResMLP case up to $\Delta\chi^2 \approx 4$.

Increasing the number of training points generally improves the performance of both emulator architectures; however, it comes at the cost of having to compute more data vectors. For every number of training points in the sample, the baseline transformer architecture performs better than the ResMLP-only architecture for both the $\langle\Delta\chi^2\rangle_{\rm med}$ and $f(\Delta\chi^2 > 0.2)$ metrics. For comparison, the emulator that includes a transformer block needs approximately 2×10^5 training points to achieve lower $\langle\Delta\chi^2\rangle_{\rm med}$ and $f(\Delta\chi^2 > 0.2)$ values than the ResMLP-only-based emulator achieves with 5×10^5 training points.

Extrapolating Fig. 5 suggests that the ResMLP-only architecture could potentially achieve $f(\Delta\chi^2 > 0.2)$ less than 10%, the threshold we set in this analysis, with a few million data vectors. However, including a transformer block achieves this performance threshold by roughly 4×10^5 points, albeit at the cost of adding more trainable parameters, which increases the training time. For reference, our ResMLP model takes only 25% of the training time of the transformer model.

In general, we find that the largest computational bottleneck in training an emulator for CMB anisotropy power spectra, both in runtime and in use of computational resources, comes from computing the data vectors and not the training times, even for models that add transformer blocks. With our accuracy settings, 9 Intel Sapphire Rapids CPU cores need 100 seconds to compute one data vector using CAMB. This highlights a key advantage of using transformer blocks for emulating CMB anisotropy power spectra.

Figure 6 shows the distribution of $\Delta\chi^2$ values between the CAMB and an emulator that either includes or excludes a self-attention layer for all of the points in the testing set. The $\Delta\chi^2$ distribution for the transformer-based architecture is shifted toward smaller $\Delta\chi^2$ values than the ResMLP-only architecture. This lowers both the median $\Delta\chi^2$ value as well as suppresses the fraction of outlier points where $\Delta\chi^2 > 0.2$, as was seen in Fig. 5. Importantly, Fig. 6 illustrates that the full distribution has shifted toward smaller $\Delta\chi^2$ values, which highlights that the

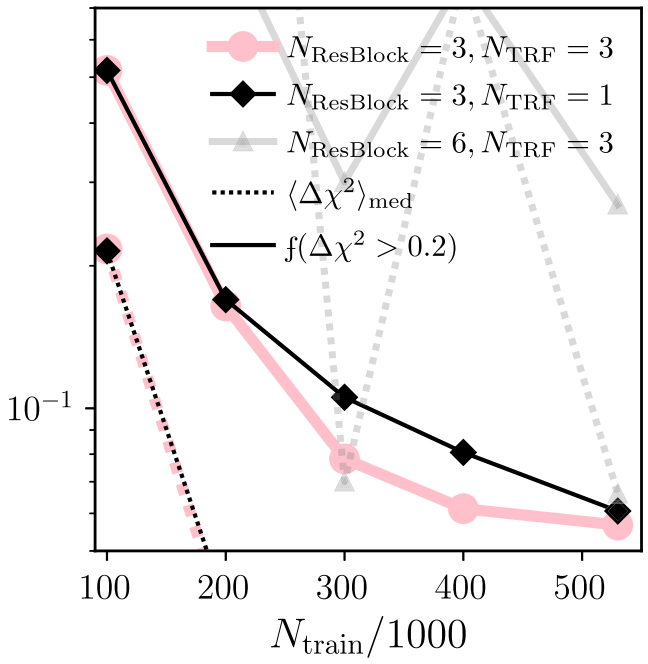

FIG. 7. Comparison of the fraction of outliers, $f(\Delta\chi^2 > 0.2)$, with a varying number of transformer and ResMLP blocks. The black line is for a model with one transformer block and three ResMLP blocks, the pink line is for the model with three transformer blocks and three ResBlocks blocks, and the gray line is for the model with three transformer blocks and six ResBlocks. Increasing the number of transformer blocks improves emulator accuracy at 300 and 400 thousand training vectors. However, when additional ResMLP blocks are added, the training becomes more unstable. The model does not necessarily benefit from having more data points fed during training, as we can see when the number of training data vectors increases to 400 thousand and more. This is due to overparametrization of the model, introducing numerical difficulties for training.

transformer-based emulator should perform better than the ResMLP-only for alternative potential accuracy metrics such as $f(\Delta\chi^2 > 0.1)$ or $f(\Delta\chi^2 > 1)$.

In Fig. 7, we vary the number of points in the training sample and compare the performance of the baseline emulator, which has one transformer block and three ResMLP blocks, to emulators that have three transformer blocks and either three or six ResMLP blocks. Increasing the number of transformer blocks from one to three results in an improvement in emulator accuracy for both the median and outlier points. The model with three transformer blocks has less than 10% of points with $f(\Delta\chi^2 > 0.2)$ when trained with $\lesssim 3 \times 10^5$ points, whereas the model that uses only one transformer block requires $\lesssim 4 \times 10^5$ points. However, it takes around 50% more time to train on an NVIDIA 3060 GPU on the same amount of data vectors for the three transformer models due to the larger number of trainable parameters.

In contrast, Fig. 7 also shows that increasing the number of ResMLP blocks from three to six overall worsens the performance of the emulator for both the median and outlier points, despite including more trainable parameters. We attribute this worse performance to over-parametrization of the network, which leads to training instability.

C. Types of attention mechanisms

We have demonstrated that including a dot-product attention mechanism can reduce the number of sampled training points required to achieve a set threshold accuracy for the emulator relative to ResMLP-only emulators. However, the dot-product attention is not the only type of attention mechanism in the literature. Here, we compare the four alternative attention mechanisms (linear, LSH, AFT, and latent), which are discussed in more detail in Sec. II A and compare them to the baseline dot-product attention model.[3] We compare the values of $f(\Delta\chi^2 > 0.2)$ for each of the four different potential attention mechanisms as a function of the number of training points and show the results in Fig. 8. Overall, we find that the baseline dot-product attention mechanism that was previously used in [20,21] is at least approximately as accurate for every given number of training points as each of the alternative attention mechanisms.

The linear, latent, and LSH attention mechanisms provide alternative approaches to computing the attention matrix QK^T in Eq. (1); they all involve approximations to reduce the time required for attention evaluation in large models. For example, the LSH attention mechanism only computes the largest products in QK^T. The latent attention, on the other hand, attempts to reduce the size of the Q, K, and T matrices by decomposing them into smaller matrices.

For all five numbers of training points explored, Fig. 8 shows that these approximations have worse accuracy relative to the baseline dot-product attention mechanisms. Quantitatively, the linear and latent attention models are roughly 4 times less accurate for outlier points than the dot-product attention models when using 5.3×10^5 training data vectors. Meanwhile, the LSH attention model is around 1.5 times less accurate for the same number of training data vectors.

The AFT model avoids computing the attention matrix entirely and has nearly the same accuracy level as the dot-product attention model, both reaching around 6% fraction of outliers when trained on $N_{\rm train} = 5.3 \times 10^5$ data vectors. However, the pairwise position biases in the AFT model introduce more learnable weights compared with the dot-product attention model. Since the dot-product attention

[3]We optimize the hyperparameters for the dot-product attention mechanism and use the same values for the alternative mechanisms. We find that including these optimizations lead to a small improvement in emulator performance that does not qualitatively alter our conclusions.

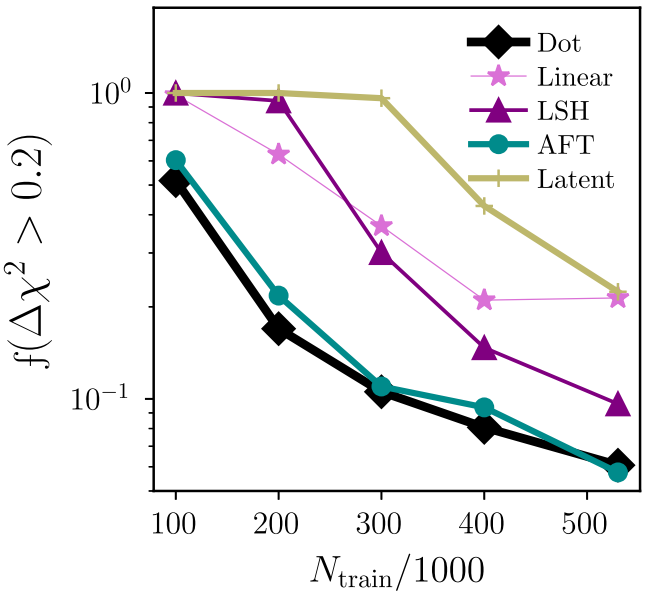

FIG. 8. Comparison of the fraction of outliers, $f(\Delta\chi^2 > 0.2)$, when using different attention mechanisms outlined in Sec. II B. We find that the dot-product attention and the attention-free transformer (AFT) perform about equally well. While AFT performs equally as well as the dot-product attention, we find that it takes about 50% more time to train the dot-product attention because of the added trainable parameters. Hence, we choose to use the dot-product attention mechanism in our baseline model. The other attention mechanisms, linear, latent, and LSH attention, all of which rely on an approximation, perform about 1.5–5 times worse than the dot-product attention.

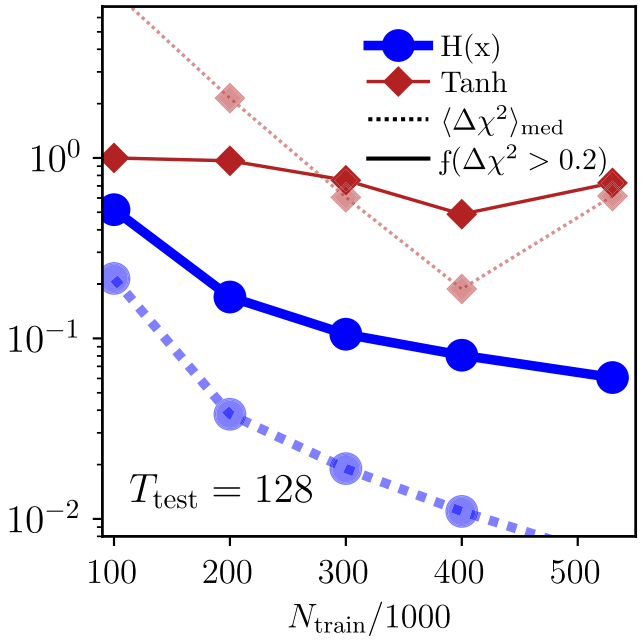

FIG. 9. Median $\Delta\chi^2$ (dashed lines) and fraction of testing points with $\Delta\chi^2 > 0.2$, $f(\Delta\chi^2 > 0.2)$ (solid lines) for the $\mathrm{H}(x)$ (blue) and $\mathrm{TANH}(x)$ (red) activation functions. The emulators trained using the $\mathrm{TANH}(x)$ activation function perform worse than those trained with the $\mathrm{H}(x)$ activation function for every number of training points tested and for both accuracy metrics. For the $\langle\Delta\chi^2\rangle_{\rm med}$, this difference is about an order of magnitude better for the $\mathrm{H}(x)$ activation function. While increasing the number of training points generally results in a lowering of the number of outlier points for the emulator using the $\mathrm{H}(x)$ activation function, the emulator using the $\mathrm{TANH}(x)$ activation function does not reduce the number of outliers as the number of training points increases. These results highlight the benefit of using a dynamical activation function.

uses fewer computational resources while achieving the same accuracy, we consider it the optimal attention mechanism.

We test both the positional embedding technique used in [42] and the rotary positional embedding technique used in [43]. Both methods achieve the same level of precision as the dot-product attention method with 530 thousand training data vectors and the same training procedures as above. Specifically, we find $f(\Delta\chi^2 > 0.2) = 0.07$ and $f(\Delta\chi^2 > 0.2) = 0.06$, respectively, for the two positional embedding methods. We see no significant changes from applying this technique for our purpose.

D. Choice of activation functions

In our baseline emulator, we use the $\mathrm{H}(x)$ activation function used previously in [16,33]. In this subsection, we compare the performance of the emulator using this activation function to the TANH function that was used in Parts I and II. In Fig. 9, we show this comparison for the median $\Delta\chi^2$ value and the $f(\Delta\chi^2 > 0.2)$ for varying numbers of training points. We find that the $\mathrm{H}(x)$ activation function generally performs better for both accuracy metrics than TANH. Increasing the number of training points reduces the median $\Delta\chi^2$ value for both activation functions, though there is a notable uptick in the median $\Delta\chi^2$ value when increasing from 4×10^5 to 5.3×10^5 training points. Nevertheless, the $\mathrm{H}(x)$ activation function performs about an order of magnitude better at each number of training points. For the $f(\Delta\chi^2 > 0.2)$ value, the TANH activation function shows negligible improvement when increasing the number of training points from 10^5 to 5.3×10^5. Over the same range, the $\mathrm{H}(x)$ activation goes from being 1.5 times smaller than the corresponding TANH outlier fraction to being 10 times smaller.

E. Choice of loss functions

An optimal choice of loss function can help improve the accuracy of the emulator for some set number of training points and reduce the amount of time necessary to train the emulator. Previous works, such as [16], have used the mean square error (MSE) as the loss function for training CMB emulators. We trained our baseline emulator using the MSE loss function and 530 thousand training points. In Fig. 10, we show the difference between the outputs of the emulator

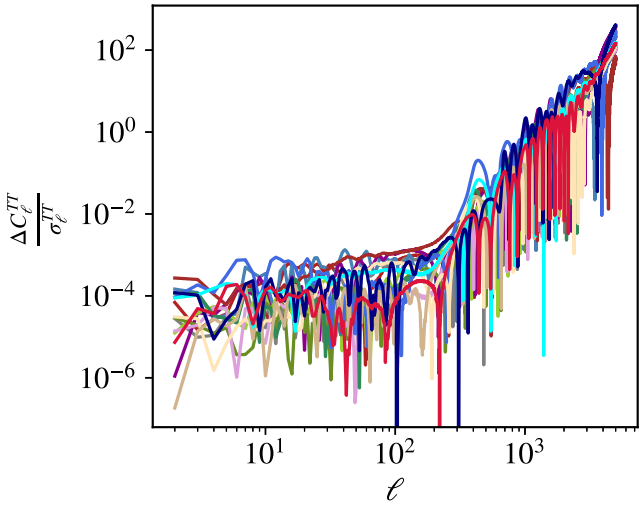

FIG. 10. Difference between the TT power spectrum outputs of the baseline emulator trained with the mean square error (MSE) loss function and CAMB divided by the square root of the diagonal of the cosmic variance covariance matrix for the 15 cosmologies from Fig. 2. This difference is shown per ℓ mode. As the multipole moment increases, the MSE-trained emulator results in larger errors relative to cosmic variance. We attribute this larger relative error to MSE equating differences in power on large scales, where the overall power is much larger, to differences in power on small scales. This causes the emulator to be biased towards fitting the larger scales more optimally. This motivates us to use the loss functions based on the cosmic variance $\Delta\chi^2_{\rm XY}$, which are defined in Sec. III C.

trained with the MSE loss function and CAMB divided by the square root of the diagonal component of the cosmic variance covariance matrix for the 15 randomly chosen cosmologies in Fig. 2. We find that the emulation error relative to cosmic variance increases as the multipole increases, with the emulation error exceeding cosmic variance for $\ell \gtrapprox 2000$ and approaching 100 times larger than cosmic variance by $\ell = 5000$.

The baseline emulator is trained using the $\mathcal{L}_4$ loss function defined in Sec. III C. One simple alternative is the $\mathcal{L}_1$ loss function of Eq. (18), which is the average $\Delta\chi^2_{\rm XY}$[4] value between the emulator and CAMB. However, we find that this loss function performs orders of magnitude worse for both the median $\Delta\chi^2$ value and the fraction of outlier points, $f(\Delta\chi^2 > 0.2)$, than the baseline case for every number of training points tested. The difficulty of training with the $\mathcal{L}_1$ function relative to alternative loss functions that we explore is that it is highly influenced by outliers with very large $\Delta\chi^2$ values. These outliers tend to be located in highly nonphysical points in parameter space. Thus, the model cannot easily learn the relationship between parameters and output data vectors in the region of best interest.

Figure 11 shows the accuracy metrics, $\langle\Delta\chi^2\rangle_{\rm med}$ and $f(\Delta\chi^2 > 0.2)$, for each of the $\mathcal{L}_2$, $\mathcal{L}_3$, $\mathcal{L}_4$, and $\mathcal{L}_5$ loss functions given some number of training points. In general, all of the loss functions show improvement in accuracy as the number of training points increases. However, emulators trained using the $\mathcal{L}_2$ loss function hit a plateau for both accuracy metrics between 2×10^5 and 4×10^5 training points. Using the hyperbolic loss functions, $\mathcal{L}_3$ and $\mathcal{L}_5$, results in generally less accurate emulators than using the square root of the $\Delta\chi^2_{\rm XY}$ values as is done in $\mathcal{L}_2$ and $\mathcal{L}_4$. In Appendix C, we show results updating the optical weak lensing 3×2 pt emulator from Part II with a hyperbolic loss function similar to $\mathcal{L}_3$, where we find that this loss function reduces emulation error compared to the loss function in Part II.

The $\mathcal{L}_5$ loss function performs approximately as well as the $\mathcal{L}_2$ on the $\langle\Delta\chi^2\rangle_{\rm med}$ values, highlighting that rescaling the covariance matrix and CAMB outputs can help improve the accuracy of the emulator. This is further supported by the performance of the $\mathcal{L}_4$ loss function relative to $\mathcal{L}_2$. However, an emulator trained with the $\mathcal{L}_5$ loss function tends to perform worse than emulators trained with either $\mathcal{L}_2$ or $\mathcal{L}_4$ for outlier points in the tails of the distribution. This latter point is corroborated by the baseline $\mathcal{L}_4$ loss function performing as well or better than the $\mathcal{L}_2$. When trained with 5.3×10^5 data vectors, the model trained with $\mathcal{L}_4$ can eliminate outliers to 6%, and the one with $\mathcal{L}_2$ has 7% of outliers. Notably, even without the rescaling that is unique to CMB, a similar loss function, $\mathcal{L}_2$, can achieve performance on par with $\mathcal{L}_4$. Hence, this result of $\mathcal{L}_2$ can likely be generalized to other emulation scenarios. On the other hand, $\mathcal{L}_4$ is only physically motivated for CMB.

F. Choice of training sampling methods

In previous sections, we have performed all tests using the $T_{\rm train} = 256$ Gaussian sampled training set, where we can achieve our criteria of $f(\Delta\chi^2 > 0.2) < 0.1$ for points in the testing set with around 4×10^5 training points. The training and testing points are generated through Gaussian sampling centered at the fiducial cosmology, with a covariance matrix that is tempered, or scaled, by a temperature parameter T. For simplicity, we choose the temperature to be the powers of 2, and we go up to $T_{\rm train} = 256$ and $T_{\rm test} = 128$. For higher temperature samplings, we place hard boundary cuts on certain parameters, which are listed in Table II, to avoid going into highly nonphysical regions. In comparison, we find that training on the uniformly sampled training set makes it more difficult to control this fraction of outliers. To explore this further, we test the same transformer architecture used in the previous

[4]Recall that we train the emulators for individual power spectra, so the subscript *XY* simply denotes the relevant power spectra that is being emulated (e.g., TT). We adopt this notation to differentiate the $\Delta\chi^2_{\rm XY}$ used in the loss function with the $\Delta\chi^2$, which includes all three concatenated power spectra that we use to assess the accuracy of our emulators primarily through $\langle\Delta\chi^2\rangle_{\rm med}$ and $f(\Delta\chi^2 > 0.2)$.

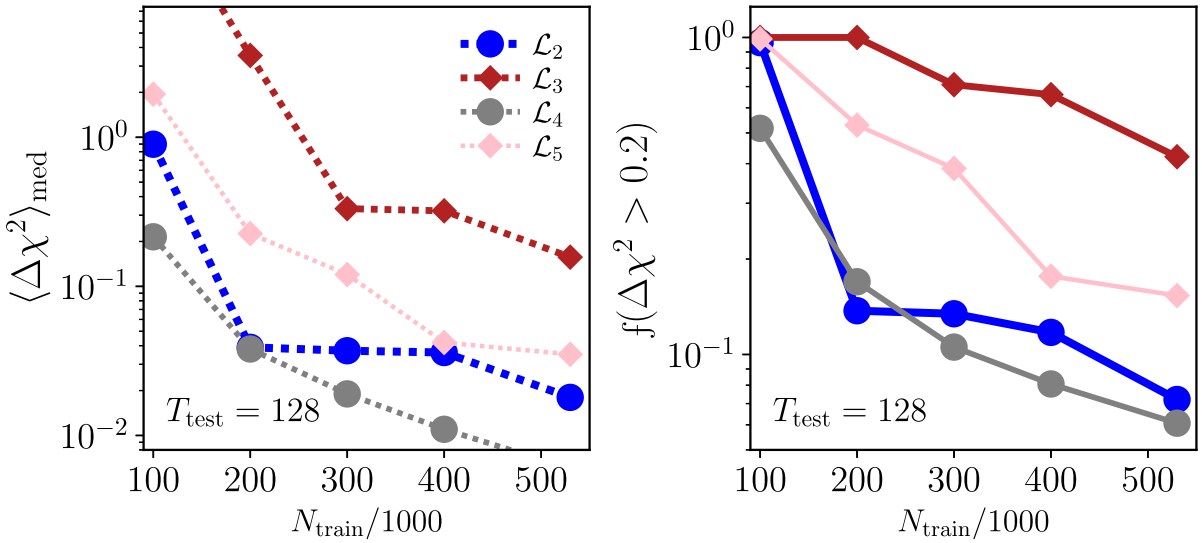

FIG. 11. Left panel: Comparison between the median $\Delta\chi^2$ for each loss function outlined in Sec. III C. We exclude plotting $\mathcal{L}_1$ because the lowest median $\Delta\chi^2$ we observed was about 10^2. We find that the function $\mathcal{L}_4$ scales best with the addition of more training data while also providing the smallest median $\Delta\chi^2$. The next best function, $\mathcal{L}_2$, has a median $\Delta\chi^2$ that is higher than $\mathcal{L}_4$ for every number of training points except 200 thousand. Both $\mathcal{L}_3$ and $\mathcal{L}_5$, improve with more training points; they never exceed the performance of $\mathcal{L}_4$. Right panel: Comparison between the $f(\Delta\chi^2 > 0.2)$ for each loss functions. In agreement with the left panel, the functions $\mathcal{L}_2$ and $\mathcal{L}_4$ perform approximately equally well to each other while performing significantly better than the other loss functions. However, between $\mathcal{L}_3$ and $\mathcal{L}_5$, we find that the rescaled option $\mathcal{L}_5$ is significantly better than the nonrescaled version. From these plots, we conclude that $\mathcal{L}_4$ is the best loss function to use for our CMB emulator.

tests but trained and tested on the uniform samples (see Table II for the parameter ranges).

As we can see in Fig. 12, even with 6×10^5 training data vectors, the uniform sampling trained model still has $f(\Delta\chi^2 > 0.2) = 31\%$. Nevertheless, there is a downward trend in the accuracy metrics of the emulator trained and tested on the uniform sets. Thus, we further apply up to 1.2×10^6 uniformly sampled training data vectors, where we are able to achieve $f(\Delta\chi^2 > 0.2) = 10\%$ on the uniform testing set. This shows that, given enough training data, it is possible to train on parameters uniformly sampled from the prior, although it requires a tighter cut on $\log(10^{10}A_s)$ than the Gaussian case.

G. Interpolation ability

In our baseline emulator, the models are trained on a Gaussian training set using a temperature $T_{\rm test} = T_{\rm train}/2$. However, when expanding a d-dimensional Gaussian distribution by tempering the covariance this way, the density of points generated near the fiducial cosmology decreases approximately as T^{-d} because the same number of points now covers a larger parameter space. This could mean that a model trained over a large temperature parameter distribution has not seen enough data near the fiducial cosmology to accurately model points sampled using a much lower temperature.

If the high-temperature testing set also has too few points near the fiducial model, then our metrics might miss the

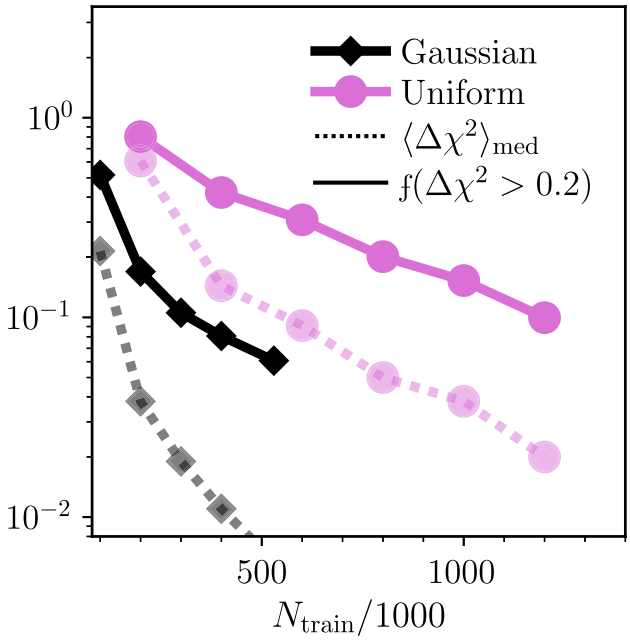

FIG. 12. Comparison between the uniform and the Gaussian sampling methods. The solid lines display the $f(\Delta\chi^2 > 0.2)$, and the dashed lines display the median $\Delta\chi^2$. The blue lines are for models trained on Gaussian samples on $T_{\rm train} = 256$ and tested on $T_{\rm test} = 128$, and the red lines are trained on the uniform samples drawn from the parameter ranges outlined in Table II. The Gaussian sampling provides around an order of magnitude improvement in both of the metrics displayed in the plot. However, with enough training data, about three to five times more, the uniform sampling can perform approximately as well as the Gaussian sampling.

poor performance of the emulator near the fiducial cosmology. This is particularly relevant as the emulator would likely be used for future CMB data whose allowed parameter space will be smaller than even the parameter space covered by our $T = 1$ tempered Gaussian. To verify that our models are not sensitive to this effect, the model trained on $T_{\rm train} = 256$ is tested on samples generated from Gaussian distributions with decreasing temperatures, down to $T = 1$.

Despite the decrease in the density of training points, we can see from Fig. 13 that the fraction of outliers tends to 0 as we decrease the temperature of the testing set. We additionally tested the performance of the emulator on a Planck ΛCDM sample chain using baseline PLIK half mission likelihoods over $30 \leq \ell \leq 2508$ in TT and $30 \leq \ell \leq 1996$ for TE and EE, HFI low-ℓ ($2 < \ell < 29$), and lensing data. Note that this parameter space is slightly smaller than our $T_{\rm test} = 1$ because our cosmic variance-limited Fisher matrix that we use to generate sample points only includes multipoles $30 \leq \ell \leq 400$ for the TT, TE, and EE power spectrum. When using this Planck sample chain, we find no outliers (i.e., testing points with $\Delta\chi^2$ values greater than 0.2).

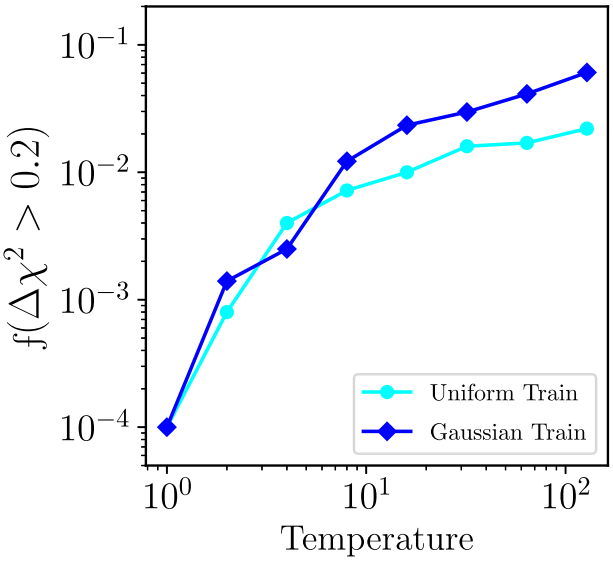

FIG. 13. Fraction of outliers, $f(\Delta\chi^2 > 0.2)$ as a function of the testing temperature $T_{\rm test}$. We train a transformer model on a Gaussian training set with $T_{\rm train} = 256$ and another model with uniform sampling, and test the model on all temperatures in powers of 2 between $T_{\rm test} = 128$ and $T_{\rm test} = 1$. To test the models trained on the uniform sets, we restrict $\log(10^{10}A_s) \leq 3.5$. We find, in both cases, that the $f(\Delta\chi^2 > 0.2)$ tends to zero as the temperature decreases to one, suggesting that, as the training temperature increases and the density of training points near the fiducial cosmology decreases, our emulators are still accurately able to replicate CAMB outputs near the fiducial cosmology. This is important because current and future CMB experiments will allow only a small volume parameter around the fiducial cosmology.

Furthermore, we show that a model trained with 1.2 million uniform sampled data vectors also has good interpolation ability in small regions around the fiducial point. Due to the different prior cuts we have for training in uniform sampling and Gaussian sampling, we test this model with a parameter cut of $\log(10^{10}A_s) \leq 3.5$. We see from Fig. 13 that the fraction of outliers from this model also decreases to 0 as we reduce the temperature of the Gaussian testing set. From this, we can see that either method of generating training samples can be used to run an MCMC. However, as seen in Sec. IV F, the Gaussian set simply achieves better accuracy with fewer training points. With enough training data, the model trained on uniform samples could achieve the same accuracy as the Gaussian trained samples at all temperatures but with a smaller prior in $\log(10^{10}A_s)$.

V. CONCLUSION

In this work, we emulate CMB TT, TE, and EE power spectra calculated by the CAMB Boltzmann code for the ΛCDM model. We extend previous analyses such as Ref. [16], which also emulated CMB anisotropy power spectra outputs of Boltzmann codes in two ways: (1) we broaden, relative to these previous analyses, the ΛCDM parameter space for which the emulator is validated, and the multipole range of the power spectra are emulated, and (2) we improve the emulator accuracy to be within cosmic variance uncertainties. This analysis also builds on Ref. [20] and Ref. [21], which applied self-attention mechanisms to optical weak lensing data. Overall, we conclude that we can emulate the CMB anisotropy power spectra outputs of CAMB to within cosmic variance limits out to $\ell \approx 5000$ while covering a very large parameter space.

We find that using a transformer architecture, which includes a self-attention mechanism, improves the emulator performance for all numbers of training points tested, relative to ResMLP-only architectures for both points near the fiducial cosmology as well as for points in the tails of the distribution. This latter point is important because it allows the emulator to cover a broader parameter space with fewer training points. Importantly, one may be able to achieve similar performance as the transformer architecture with a ResMLP-only architecture; however, this will require on the order of hundreds of thousands more training data vectors to achieve.

While the transformer architecture adds more trainable parameters relative to a ResMLP-only architecture, which in turn increases the time necessary to train the model, we conclude that it is still beneficial to use the transformer architecture. In general, we find the most time-consuming and computationally expensive part of emulating CMB power spectra is the initial calculation of all of the data vectors using CAMB to train the emulator. Therefore, reducing the number of training vectors necessary to

emulate the power spectra greatly expedites cosmological analysis.

We tested more self-attention layers as well as changed how the self-attention in the transformer blocks is calculated. However, we found at most marginal improvements relative to the improvement found by whether the model includes a self-attention layer or not. In Appendix E, we test using a 1D convolutional neural network, which performs about as well as the transformer. Additionally, we found that the dynamical activation function with learnable weights adopted by [16] considerably improves the emulation accuracy and efficiency compared to static activation functions like TANH.

Pre-processing the CMB data vectors can significantly improve emulator accuracy as it can reduce the amount of parameter correlations that the emulator needs to learn. In particular, we found improvement when rescaling the amplitude of the CMB data vectors by $A_s e^{-2\tau}$ for both the transformer and ResMLP-only architectures. In Appendix F, we show that rescaling the amplitude of the damping tail may also improve emulator performance.

Optimally sampling the parameter space can improve the performance of the emulator while also allowing for good coverage of a broad parameter space. Specifically, we found, as it was previously found in Ref. [21], that using a tempered Gaussian sampling method reduces the number of training data vectors necessary to achieve a set emulation accuracy threshold relative to uniform sampling. We attribute this improvement to taking into account the correlation between parameters when generating the larger parameter space so that the emulator does not need to learn to emulate cosmologies that are far from the fiducial parameters in a direction orthogonal to the CMB parameter degeneracies. In effect, we are creating a more targeted parameter volume to emulate. Increasing the volume of parameter space where the emulator is valid is beneficial as a proxy for extended cosmological models, as well as a tool for collaborations employing blinding methods to shift the data vector by an arbitrary amount and still achieve reasonable results from the emulator.

Improving emulator accuracy and precision to be in agreement with CAMB to within cosmic variance is valuable because this enables the emulator to be applicable to real data for the next couple of decades of CMB analyses. In Appendix B, we show this explicitly using the actual Planck data covariance matrix as well as forecasts for Simons Observatory-like, CMB-S4-like, and CMB-HD-like experiments. We additionally take the Atacama Cosmology Telescope (ACT) DR6 data MCMC [44] and calculate the shift in χ^2 when replacing CAMB with our baseline emulator and find the $f(\Delta\chi^2_{\rm ACT} > 0.2) < 9\%$ with only 200 thousand training points on uniform sampling testing set. With the same emulator trained and tested on Gaussian sampling, $f(\Delta\chi^2_{\rm ACT} > 0.2)$ is further suppressed to under 0.1%. We conclude that our emulator has broad applicability to both current and future CMB experiments.

ACKNOWLEDGMENTS

Special thanks to Kunhao Zhong for helpful discussions about the convolutional neural networks. V. M. and Y. Z. were partially supported by the Roman Project Infrastructure Team "Maximizing Cosmological Science with the Roman High Latitude Imaging Survey" (NASA Contracts No. 80NM0018D0004 and No. 80NM0024F0012). V. M. was also partially supported by the Roman Project Infrastructure Team "A Roman Project Infrastructure Team to Support Cosmological Measurements with Type Ia Supernovae" (NASA Contract No. 80NSSC24M0023). E. S. acknowledges support from the Yang Institute for Theoretical Physics and Stony Brook University as a graduate assistant, partly made possible by generous contributions from the Simons Foundation. The work of A. S. G. is supported in part by the National Science Foundation Grant No. PHY-2210533. Simulations in this paper use High Performance Computing (HPC) resources supported by the University of Arizona TRIF, UITS, and RDI and maintained by the UA Research Technologies department. The authors would also like to thank Stony Brook Research Computing and Cyberinfrastructure and the Institute for Advanced Computational Science at Stony Brook University for access to the high-performance SeaWulf computing system.

DATA AVAILABILITY

The data that support the findings of this article are openly available at [45], embargo periods may apply.

APPENDIX A: CAMB SETTINGS

We adopt the CAMB settings in Appendix A of [46]; however, we increase the ACCURACY-BOOST parameter to 1.5 instead of 1.1. We assume results from this setting as

TABLE VII. The CAMB settings we use to generate our power spectra. Our choices are based on [46]; however, we specifically changed ACCURACY-BOOST to 1.5 for more robust accuracy. Future works may consider even higher settings for ACCURACY-BOOST.

CAMB setting	Value
ACCURACY-BOOST	1.5
ℓ-SAMPLE-ACCURACY	10
ℓ-ACCURACY-BOOST	3
DO-LATE-RAD-TRUNCATION	FALSE
$\ell_{\rm max}$	15000
LENS-POTENTIAL-ACCURACY	30
LENS-MARGIN	2050
CAMB-PARAMS-NONLINEAR	CAMB-MODEL-NONLINEAR-BOTH
CAMB-PARAMS-NONLINEARMODEL	MEAD2016

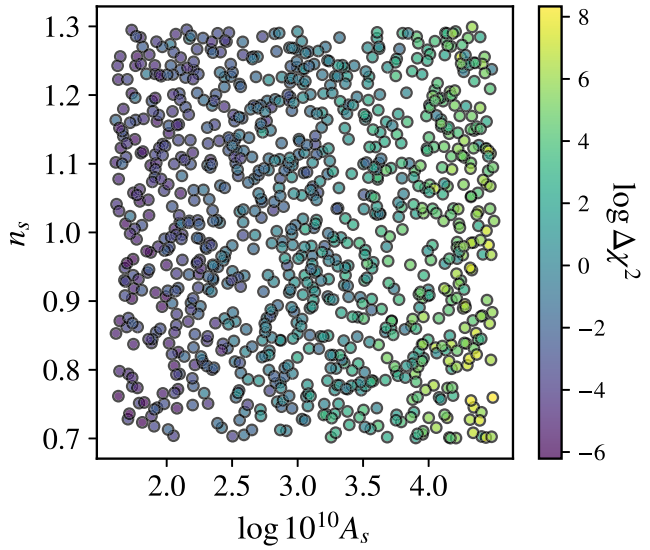

FIG. 14. Distribution of $\log(\Delta\chi^2)$ for a cosmic variance limited experiment between CAMB outputs with ACCURACY-BOOST $= 1.5$ and 1.8, with all other settings fixed. There are significant deviations between outputs with those two settings in the high A_s region.

our "truth" value for CMB power spectra and use the same setting for training, testing, and validation sets as different settings, including $\ell_{\max}$, can numerically alter the power spectra. We also do not consider the comparison between numerical outputs between CAMB and CLASS [47]. A summary is provided in Table VII. We noticed that there is some numerical inaccuracy that is visible under cosmic variance covariance in the power spectrum generated by this setting when we compared those data vectors to some data set generated by more accurate settings.

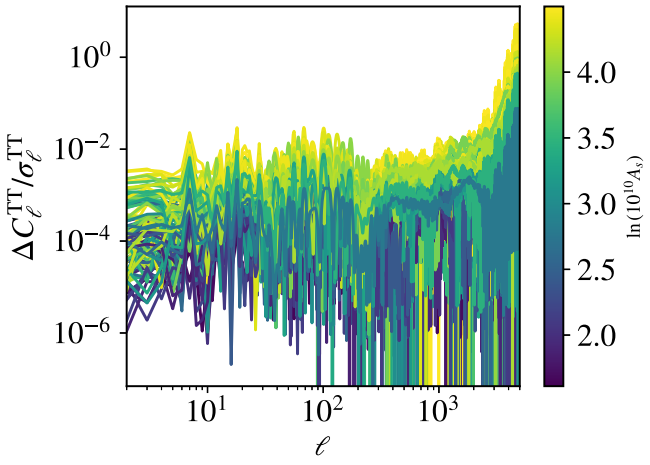

FIG. 15. Distribution of $\Delta C_\ell^{TT}/\sigma_\ell^{TT}$ between CAMB outputs with ACCURACY-BOOST $= 1.5$ and 1.8, with all other settings fixed, over 100 different cosmologies. The colorbar shows the scaling of the $\ln(10^{10}A_s)$, and we can see that the discrepancy between outputs from those two settings, especially on high ℓ, grows as $\ln(10^{10}A_s)$ grows.

In Figs. 14 and 15, we compare $\log(\Delta\chi^2)$ under cosmic variance covariance between CAMB outputs with ACCURACY-BOOST $= 1.5$ and 1.8, and noticeably, the magnitude of this inaccuracy, especially in small scale, increases as A_s increases, which we attribute to higher A_s values increasing the effect of lensing. Thus, for the application of our models, users need to be aware of the necessity of picking a satisfactory accuracy setting. For future works, we need to set the ACCURACY-BOOST to even higher values, possibly around 2.5. We note for our purposes that potential CAMB numerical instabilities, relative to cosmic variance uncertainties and resulting from lower ACCURACY-BOOST settings, would make training an emulator more difficult. This is because the emulator will attempt to learn these numerical instabilities, which are inherently random. Using a higher ACCURACY-BOOST setting could potentially lower the number of training data vectors necessary to train an emulator to within a desired accuracy.

APPENDIX B: EMULATOR PERFORMANCE ESTIMATION FOR CURRENT AND FUTURE EXPERIMENTS

We test our emulator performance given Planck-like uncertainties as well as forecasted data covariances for Simons Observatory-like, CMB-S4-like, and CMB-HD-like experiments in Fig. 16, with binning strategies covariance matrices from [37,40,46,48] and from the GitHub repository for Planck likelihood.[5] For all experiments, $\ell_{\min} = 30$. For the Planck-like experiment, $\ell_{\max} = 2508$ for the TT power spectrum, and $\ell_{\max} = 1996$ for TE and EE. For the TT power spectrum from the Simons Observatory-like and CMB-S4-like experiments, $\ell_{\max} = 3000$. For the Simons Observatory-like [1,2] and CMB-S4-like experiments TE and EE power spectra, $\ell_{\max} = 5000$ [46]. For CMB-HD [46], the collaboration plans to measure $1000 \leq \ell \leq 20000$. However, our emulator was only trained to give outputs up to $\ell = 5000$, so we restrict the maximum multipole range from the CMB-HD-like forecast to $\ell = 5000$. We assume for the CMB-HD-like forecast that they can append data taken from Simons Observatory and CMB-S4 for the lower ℓ range and test up to the upper limit of our emulator, so we perform the forecast for the CMB-HD-like experiment in the range of $30 \leq \ell \leq 5000$. The covariance matrix and binning files for SO come from the NERSC repository released in 2023.[6] All covariance matrices for CMB-S4 and CMB-HD come from the GitHub Repository linked in this footnote,[7] and are binned and delensed.

[5]Planck Likelihood: https://github.com/benabed/clik.
[6]https://portal.nersc.gov/cfs/sobs/users/MFLike_data/v0.8.tar.gz
[7]https://github.com/CMB-HD/hdfisher/tree/main/hdfisher/data/covmats

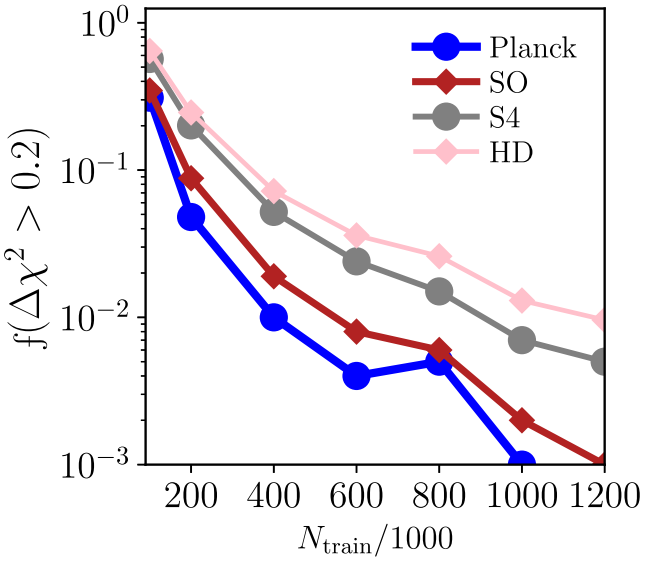

FIG. 16. Comparison of the fraction of outlier points, $f(\Delta\chi^2 > 0.2)$, given configurations for Planck-like (blue), Simons Observatory-like (red), CMB-S4-like (gray), and CMB-HD-like (pink) experiments. With the Planck-like and Simons Observatory-like analyses, an emulator trained on uniform sampling set with 200 thousand training data vectors would be sufficient in precision, while for more advanced configurations in the future, like a CMB-S4-like or CMB-HD-like experiment, we will need around 400 thousands training points to acquire the necessary precision.

We find that for the Planck-like and Simons Observatory-like cases, we can train emulators on uniform sampling sets with 200 thousand training data vectors to achieve $f(\Delta\chi^2 > 0.2) \leq 10\%$. For the CMB-S4-like and CMB-HD-like forecasts, which have higher precision, we require around 400 thousand training data vectors to train a reliable emulator. As expected, when we target realistic experiments, we do not need as many training data vectors as when we train with a cosmic variance-limited covariance matrix on every multipole. Note that we compute with ranges set by experiment limits and our current emulator training setting, described in Fig. 16. More detailed verification for some exact experimental configurations will be necessary in the actual application.

In the light of the new ACT data release [44], we apply the new ACT CMB power spectra only likelihood from their Data Release 6 (DR6) to compare the $\Delta\chi^2_{\rm ACT}$ between CAMB outputs and emulator outputs under their binning and covariance matrix, up to $\ell = 5000$. For the emulator, we use the same baseline transformer architecture trained with a uniform sampling set with various numbers of training points. We ignore higher multipole contributions due to the emulator training range and BB power spectrum since we have not yet explored this signal. We show in Fig. 17 that $\Delta\chi^2_{\rm ACT}$ of the emulator is well bounded below 10^{-1}, with around 600 thousand training data vectors on uniform sampling and with around 200 thousand training data vectors on Gaussian sampling, supporting that these emulators are reliable to replace CAMB in actual MCMC chains.

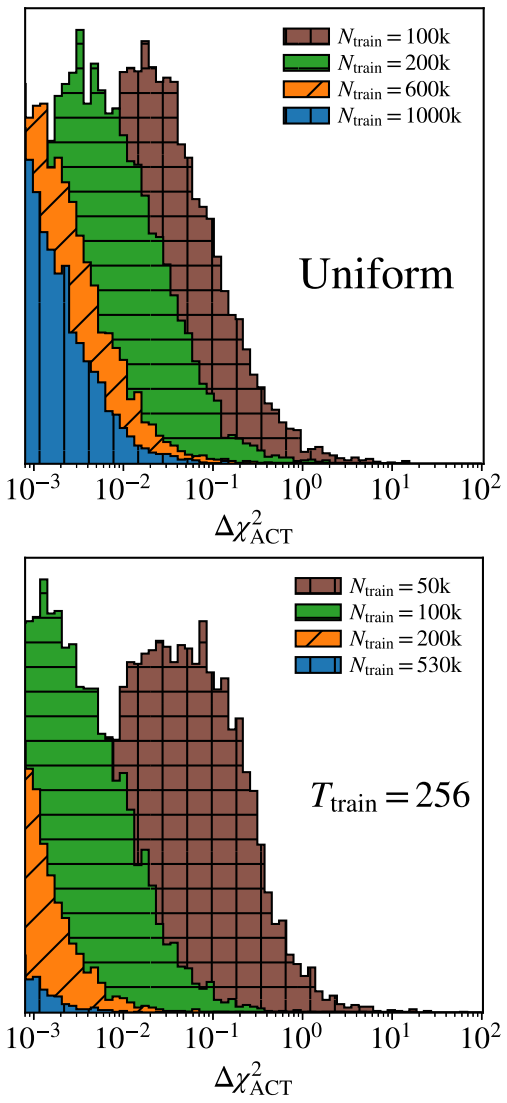

FIG. 17. $\Delta\chi^2_{\rm ACT}$ of emulators trained on 100, 200, 400, and 1000 thousand data vectors on uniform sampling and the ones trained on 50, 100, 200, 300, and 530 thousand data vectors on Gaussian sampling. All computations are evaluated with the binning and covariance matrix in ACT-DR6-CMB-only log-likelihood. For the uniform sampling models, the deviations are well controlled under the level of 10^{-1} with 600 thousand training data vectors, and for the Gaussian sampling models, the deviations are well controlled under the level of 10^{-1} with 200 thousand training data vectors. The $f(\Delta\chi^2_{\rm ACT} > 0.2)$ is each 9%, 1%, 0.2%, and 0% for the tests we perform on uniform sampling and is each 16%, 0.4% for the Gaussian sampling models with 50 and 100 thousand training data vectors, and there are no outliers for models trained with 200 thousand and above.

APPENDIX C: WEAK LENSING AND GALAXY CLUSTERING EMULATOR

The emulators presented in Parts I and II [20,21] each present outstanding results for emulating weak lensing observables. We test how the techniques explored in this work can improve the weak lensing emulators. To begin, we observed that simply tempering the parameter covariance matrix resulting from a forecasted LSST Y1 Fisher analysis does not completely populate the parameter space (see Figure 7 in [21]). This results from a combination of hard priors and strong correlations in the sampling parameters, particularly with Ω_b. As such, we reduce the correlations of all parameters with Ω_b by a factor of 2, but we did not reduce the autocorrelation of Ω_b.

Next, the 3×2 point data vectors require more parameters to enter the emulation and result in a larger data vector. Correspondingly, we expand the internal dimensions of the neural network as well. The input dimension of 22 (cosmological parameters, five source photo-z, five lens photo-z, two intrinsic alignment, and 5 galaxy bias) is mapped to an internal dimension of 512 in the RESMLP portion of the network. The transformer portion has an internal dimension of 3840 with 60 channels, reflecting the number of sample bins in the resulting data vector. The output data vector has a dimension of 1560.

Finally, and perhaps most importantly, in an effort to minimize the effect of outliers, we adopt an outlier-friendly loss function that will prevent large errors from degrading the overall performance of the emulator. In particular, we use a hyperbolic loss function given by $\mathcal{L}(\Delta\chi^2_{XY})=\langle\sqrt{1+\Delta\chi^2_{XY}}\rangle$, which is similar to the $\mathcal{L}_3$ loss function defined in Eq. (20). This provides an enormous improvement over the results of Part II, where outliers dominated the overall performance of the emulator. This change allows us to use just 10^5 training points. With fewer training points, we correspondingly reduce the batch size to 32.

There are a few things to note about these changes. First, the RESTRF tends to greatly benefit from smaller batch sizes. With 10^5 training points and a batch size of 256, we find the RESMLP slightly outperforms the RESTRF, demonstrating the importance of choosing hyperparameters. Both architectures, however, perform better with a smaller batch size. The larger internal and output dimensions provide a computational cost, as the size of the output layer is much larger than in the cosmic shear case. Thus, while the 3×2 pt emulator is slower than the cosmic shear, it is still orders of magnitude faster than numerical methods. Lastly, due to redshift space distortions being used to compute the galaxy-galaxy autocorrelation, the galaxy bias is not a fast parameter and must be included in the emulation.

With these changes, the results are significantly improved over those in Part II. We train the emulator with temperatures in powers of 2 up to $T_{\rm train}=1024$, which was the largest value we set before encountering errors. A summary of the results can be found in Table VIII.

TABLE VIII. Summary of the results of the 3×2 point emulator, showing the median $\Delta\chi^2$ in the second row and the fraction of testing points with $\Delta\chi^2>0.2$ in the last row. All temperatures we consider are able to pass our threshold of having $f(\Delta\chi^2>0.2)<0.1$. Most prominently, even with a temperature as high as 128, we are able to have zero testing points with $\Delta\chi^2>0.2$. In all cases, the median $\Delta\chi^2$ is well under control, being smaller than all values quoted in Part II.

$T_{\rm train}$	128	256	512	1024
$\langle\Delta\chi^2\rangle_{\rm med}$	0.006	0.010	0.012	0.033
$f(\Delta\chi^2>0.2)$	<0.001	0.001	0.012	0.074

We find that all temperatures we can test are able to pass our imposed threshold, with the fraction of testing points with $\Delta\chi^2>0.2$ being less than 0.1. Distributions of the $\Delta\chi^2$ are shown in a histogram in Fig. 18 and a heatmap in Fig. 19. With the adoption of the new loss function Eq. (20) and activation function Eq. (6), we are able to outperform our results from Part II with far fewer training points, now only needing 10^4 training points. Attempting to populate a larger region of parameter space by decorrelating the

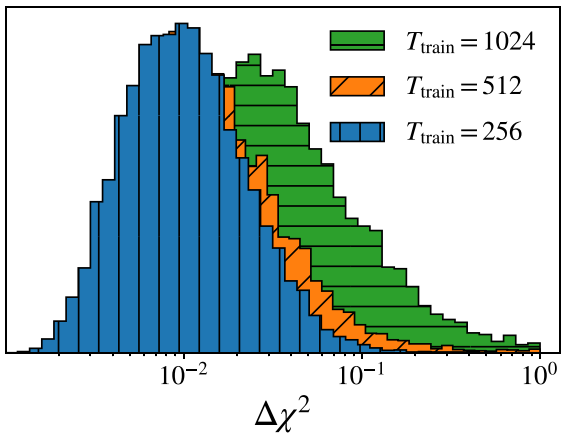

FIG. 18. Histograms displaying the $\Delta\chi^2$ distribution resulting from optical weak lensing 3×2 point forecasts for LSST Y1 for training temperatures of $T_{\rm train}=1024$, 512, and 256. The values in the histogram represent evaluation of 10^4 cosmologies on a testing set with $T_{\rm test}=T_{\rm train}/2$. The $T_{\rm train}=512$ and 256 distributions are both centered around $\Delta\chi^2=10^{-2}$ and have small tails that extend beyond $\Delta\chi^2=0.1$, reflecting the results in Table VIII. The $T_{\rm train}=1024$ distribution is centered closer to $\Delta\chi^2=0.1$ with a tail that extends beyond $\Delta\chi^2=1$. Even with a loss function that does not penalize outliers, we see that the tails are not extended far into higher values of $\Delta\chi^2$. Furthermore, we see that lower temperatures do not necessarily improve in overall performance. This is likely caused by small gradients in the loss function near the minimum, meaning the model does not have the ability to fit the data more accurately. It is seen, however, that the tails of the distribution are under better control with lower training temperatures.

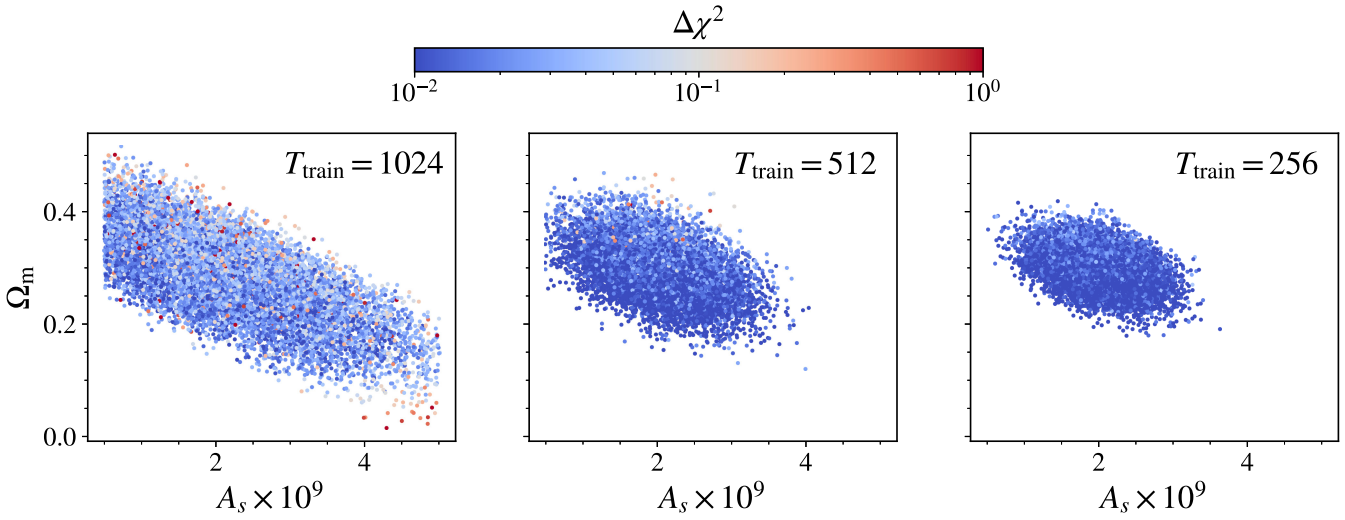

FIG. 19. Projected spatial distribution of $\Delta\chi^2$ emulator errors into the $A_s \times 10^9 - \Omega_{\rm m}$ plane. Each model was tested with 10^4 testing points from a distribution with $T_{\rm test} = T_{\rm train}/2$. Interestingly, we find a slight bias, where larger values of $\Omega_{\rm m}$ perform worse than the smaller values, a feature not present when doing the same training procedure on a correlated posterior. This bias results from the additional parameter space introduced by the decorrelation procedure and appears at simultaneously large values of $\Omega_{\rm m}$ and H_0. Thus, the bias is absent at the smaller temperatures. Aside from the bias, we can see that the training data is better centered on the chosen fiducial cosmology, meaning our training data is a more faithful representation of our input cosmology.

posterior introduced a mild bias in the emulator performance, despite the better average performance. This can possibly be resolved by selecting a more complicated parameter normalization during the emulator training, although we leave this for future work. The bias is not present when training on correlated posteriors. We also find that the transformer greatly benefits from having a large number of batches and seems to only outperform the ResMLP models when the batch size is small.

APPENDIX D: SUPERNOVA DISTANCE EMULATOR

In this Appendix, we describe the emulator used for the supernovae (SNe) luminosity distances. Parametrizing the dark energy equation of state using a basis of principal components (PCs), we aim to create a training set of cosmological parameters and distances. Following the analysis in [49], we write

$$w(z_j) = w_{\rm fid} + \sum_{i=1}^{N_{z,\rm PC}} \alpha_i \mathbf{e}_i(z_j). \tag{D1}$$

For numerical stability, we will normalize our basis vectors such that

$$\sum_{i=1}^{N_{z,\rm PC}} [\mathbf{e}_i(z_j)]^2 = \sum_{j=1}^{N_{z,\rm PC}} [\mathbf{e}_i(z_j)]^2 = N_{z,\rm PC}. \tag{D2}$$

We study smooth dark energy models that produce deviations from ΛCDM, which we assume is our fiducial model, and where $w_{\rm fid} = -1$. We construct the Fisher matrix for the average magnitude of the SNe, and then we rotate to the basis where it is diagonal,

$$F_{ij}^{\rm SN} = \sum_\alpha \sigma_\alpha^{-2} \frac{dm(z_\alpha)}{d\theta_i} \frac{dm(z_\alpha)}{d\theta_j}, \tag{D3}$$

where $m(z_\alpha) = 5\log[\mathrm{H_0 d_L(z_\alpha)}] + \mathcal{M}$ is the average apparent magnitude of the SNe in the redshift bin denoted by z_α, σ_α is the error in the average magnitude given in Appendix A of [49], and $\mathcal{M} = M - 5\log(H_0/\mathrm{Mpc}^{-1}) + 25$ is a constant related to the unknown absolute magnitude of the SNe. Within the parametrization that we have chosen, the parameters that are relevant are

$$\boldsymbol{\theta} = \{\alpha_1, \ldots, \alpha_{N_{z,\rm PC}}, \Omega_m, \Omega_m h^2\}, \tag{D4}$$

where $h = H_0/(100\ \mathrm{km\,s^{-1}\,Mpc^{-1}})$.

Our goal is to use this parametrization of $w(z)$ to generate a training set of cosmological inputs $\boldsymbol{\theta}$ with their corresponding luminosity distances $d_L(z)$, which will be used to train the emulator. Before proceeding, we first study the completeness of the PC basis that we use. Our goal is to determine the number of PCs we need to retain in order to construct $d_L(z)$ within the acceptable uncertainty. For the decomposition in (D1), we use $N_{z,\rm PC} = 1000$ PCs. In order to determine how many of these PCs we need to use to train the emulator, we compare the reconstructed $m(z)$ with the corresponding analytic expression. We find that retaining additional PCs beyond $N_{z,\rm PC} \sim 20$ leads to minimal changes in the computed $\Delta\chi^2$. This result is illustrated

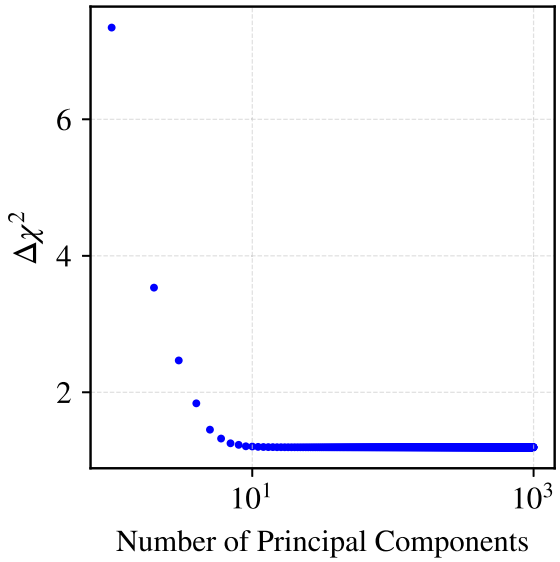

FIG. 20. $\Delta\chi^2$ between the average magnitude $m(z)$ reconstructed using the principal component basis and the average magnitude $m(z)$ calculated analytically as a function of the number of principal components used in the reconstruction for given cosmological parameters $\boldsymbol{\theta}$.

in Fig. 20, where we plot $\Delta\chi^2$ values as a function of the number of PCs used to reconstruct $m(z)$. Note that for this analysis, we have ranked the PCs in order of increasing uncertainty. To remain conservative, and since our approach does not involve computationally restrictive analytic calculations, we opt to retain 50 PCs. Using this setup, we generate a training set of $\boldsymbol{\theta}$ and their corresponding $d_L(z)$. For the range of coefficients α_i sampled in the training set, we follow the constraints outlined in [49].

For the emulator, we find that the ResMLP architecture performs sufficiently well. The neural network is designed with an input layer consisting of 52 cosmological parameters (50 PC amplitudes, Ω_m, H_0) and includes 8

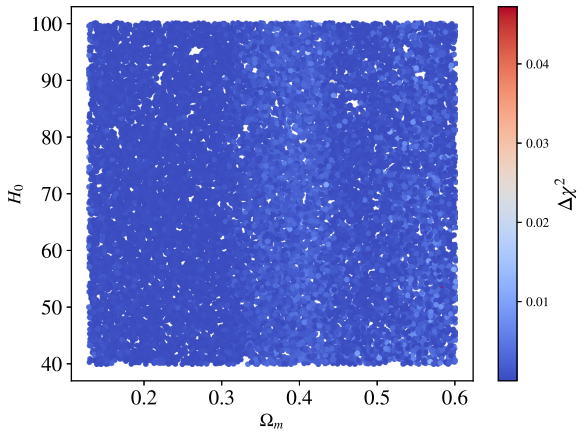

FIG. 21. Heatmap of $\Delta\chi^2$ values comparing the emulator's $\log \mathrm{H}_0 d_L(z)$ predictions to the true $\log \mathrm{H}_0 d_L(z)$ from the test dataset. The plot shows $\Delta\chi^2$ as a function of Ω_m and H_0, with PC amplitudes implicit in order to maintain clarity.

ResBlocks. We use the Adam optimizer and the average $\Delta\chi^2$ over all z bins as the loss function during training, with the error definitions for each redshift bin provided in Appendix A of [49]. The training, validation, and testing datasets are generated by uniformly sampling the allowed ranges of the cosmological parameters Ω_m, H_0, and PC amplitudes. Figure 21 shows the $\Delta\chi^2$ values comparing the emulator's predictions with the known luminosity distance $d_L(z)$ for the cosmologies in the test set.

APPENDIX E: 1D CONVOLUTIONAL NEURAL NETWORK

There are additional, nonattention-based architectures that are worth exploring. A popular architecture used in

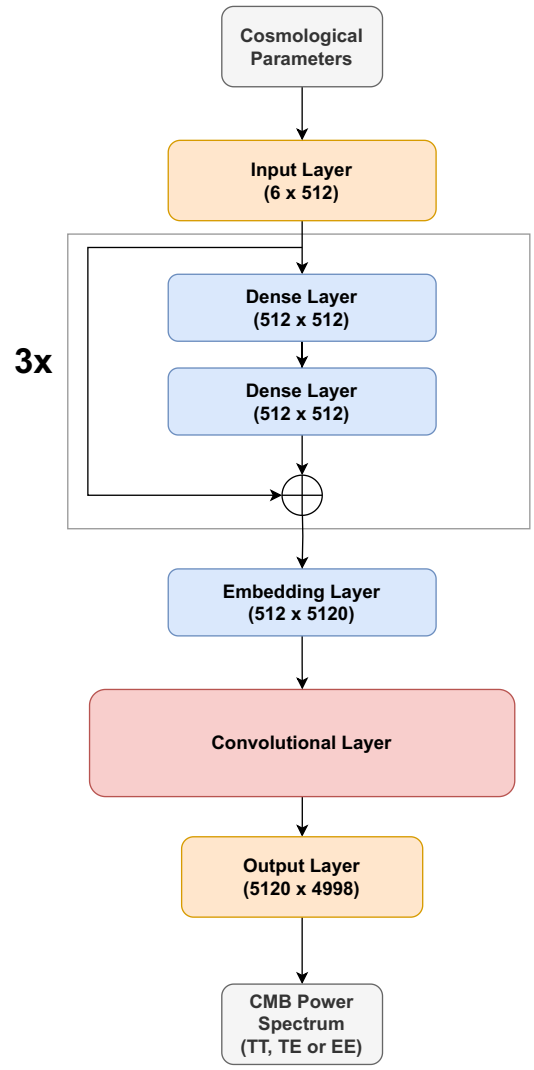

FIG. 22. Architecture of the 1D convolutional neural network we use. Similar to the case with the transformer, we first pass the data through 3 ResBlocks. We then replace the attention and transformer with a convolutional layer. The convolutional layer contains KERNEL-SIZE = 5, STRIDE = 16, PADDING = 2, and OUT-CHANNEL = 16.

image analysis is a convolutional neural network (CNN) [50]. These models work by convolving an image with a kernel matrix. Since our data is vector-like, we instead use a convolution vector, creating a one-dimensional CNN (1D-CNN) [51]. We believe that this architecture can allow us to capture the smooth, short-range correlations in the CMB power spectra. We test one architecture in which we append a 1D-CNN layer without pooling after 3 ResBlocks, as shown in Fig. 22. The convolutional layer is set to have KERNEL-SIZE $= 5$, STRIDE $= 16$, PADDING $= 2$, OUT-CHANNEL $= 16$. The choice of OUT-CHANNEL is analogous to the $N_{\rm channel}$ of transformers in the sense that the data vectors are rearranged into channels while the model processes the data vectors in pieces. Similarly, we want to move the kernel with the same amount of steps during convolution, so we have the same number setting for STRIDE. We choose one common KERNEL-SIZE number for this primary test, and the setting for PADDING is then chosen to give us the correct output dimension once the KERNEL-SIZE is fixed.

For the CMB power spectrum, we train it on $N_{\rm train} = 530$ thousand data vectors with $T_{\rm train} = 256$ training set, and test it on $T_{\rm test} = 128$ testing set, which returns a

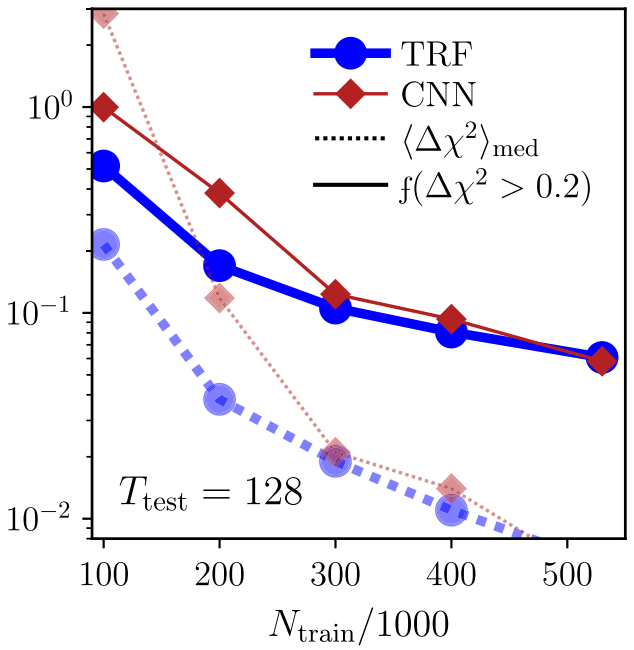

FIG. 23. Comparison of the accuracy of a transformer and a 1D-CNN. The solid lines are for fraction of points with $\Delta\chi^2 > 0.2$, and the dashed lines are for median $\Delta\chi^2$. The thick blue lines are for transformers, and the thin red lines are for 1D-CNN. We did not optimize the hyperparameters for the 1D-CNN. With about 1×10^5 training points, the transformer outperforms the 1D-CNN by about a factor of two in $f(\Delta\chi^2 > 0.2)$ *and by about an order of magnitude in* $\langle\Delta\chi^2\rangle_{\rm med}$. However, as the number of training points increases to about 3×10^5 and beyond, the performance of the transformer and 1D-CNN are approximately equal.

$f(\Delta\chi^2 > 0.2) = 0.059$. This is roughly the same as the transformer architecture result, which has $f(\Delta\chi^2 > 0.2) = 0.061$. This can be seen in Fig. 23, which compares the performance of a CNN architecture to the performance of a transformer architecture for various numbers of training vectors. CNN has the advantage of being faster to train since it does not have many trainable parameters.

APPENDIX F: IMPROVED MODELING OF THE DAMPING TAIL AND LENSING

The presence of a damping tail in the CMB power spectra leads to additional difficulties in training machine learning models. By pre-processing the TT power spectrum using the techniques in Sec. III B, we are able to reduce the span of the data vectors by orders of magnitude, which assists in training the emulator. To improve this, we deploy symbolic regression to develop a fit of the TT power spectrum damping/lensing tail.

Genetic algorithms (GAs) are a class of machine learning techniques that learn to fit data by mimicking evolutionary biology, such as random gene mutation and crossbreeding [52]. GAs have been widely used in the field of cosmology, for example, to model the linear matter power spectrum [53], nonlinear matter power spectrum [54], and matter transfer function [55]. Symbolic regression, in particular, is a kind of GA that attempts to learn a symbolic expression to fit a given dataset.

We design a GA model that takes a fixed functional form for the lensed damping tail. The functional form for our fit has two components, the damping tail and the lensing effect, and it has a total of 15 free parameters, α_i. The lensing contribution is described in Eq. (F12). For the damping tail, we follow the work done in [41]. There, the rescaling factor, to go from the undamped power spectrum without lensing tail to the full damped unlensed power spectrum, is given by

$$T_D(\ell) = \frac{A_s}{e^{2\tau}} \mathcal{D}_\ell^2 \mathcal{R}_\ell^2 \mathcal{P}_\ell, \tag{F1}$$

where $\mathcal{D}_\ell$ is the diffusion damping contribution, $\mathcal{R}_\ell$ is the reionization damping contribution, and $\mathcal{P}_\ell$ is the potential envelope damping contribution. The diffusion-damping contribution is modeled as

$$\mathcal{D}_\ell = e^{-(\ell/\ell_D)^m}, \tag{F2}$$

where ℓ_D and m are given by

$$\ell_D / r_\theta^\star = a_1 (\Omega_b h^2)^{0.291} [1 + a_2 (\Omega_b h^2)^{1.80}]^{-1/5}, \tag{F3}$$

$$m = a_3 (\Omega_b h^2)^{a_4} [1 + (\Omega_b h^2)^{1.80}]^{1/5}, \tag{F4}$$

where $r_\theta^\star$ is the comoving angular-size distance to last scattering. The a_i are power law functions of $\Omega_m h^2$, given by

$$a_1 = \alpha_1(\Omega_m h^2)^{\alpha_2}[1 + \alpha_3(\Omega_m h^2)^{\alpha_4}], \tag{F5}$$

$$a_2 = \alpha_5(\Omega_m h^2)^{\alpha_6}[1 + \alpha_7(\Omega_m h^2)^{\alpha_8}]^{-1}, \tag{F6}$$

$$a_3 = \alpha_9(\Omega_m h^2)^{\alpha_{10}}, \tag{F7}$$

$$a_4 = \alpha_{11}(\Omega_m h^2)^{\alpha_{12}}. \tag{F8}$$

The reionization damping term is given by

$$\mathcal{R}_\ell^2 = \frac{1 - e^{-2\tau}}{1 + c_1 x + c_2 x^2 + c_3 x^3 + c_4 x^4} + e^{-2\tau}, \tag{F9}$$

with $x = \ell/(\ell_r + 1)$ and constants $c_1 = -0.276$, $c_2 = 0.581$, $c_3 = -0.172$, and $c_4 = 0.0312$. The parameter $\ell_r = (\eta_0 - \eta_r)/\eta_r$, where η_r is the comoving distance to reionization. In this study, we set the redshift when reionization occurs at $z = 10$.

The potential envelope damping contribution is approximately given by

$$\mathcal{P}_\ell = 1 + A \exp(-1.4\ell_{\rm eq}/\ell), \tag{F10}$$

where $\ell_{\rm eq} = k_{\rm eq} r_\theta^\star$, and $k_{\rm eq}$ is the scale that crosses the horizon at matter-radiation equality. The amplitude A is fixed by the expression

$$A = 25\left(1 + \frac{4}{15} f_\nu\right)^{-2} \frac{(1 + R_\star)^{-1/2} + (1 + R_\star)^{-3/2}}{2} - 1, \tag{F11}$$

where $f_\nu = \rho_\nu/(\rho_\nu + \rho_\gamma)$ is the fraction of the radiation energy density comprised of neutrinos, and $R_* = \frac{3}{4}\rho_b(\eta_*)/\rho_\gamma(\eta_*)$ is the ratio of baryon and photon densities at matter-radiation equality.

Thus constructed, Eqs. (F1) through (F11) comprise the functional form for the damping tail fit of the TT power spectrum we consider. The learned parameters are listed in Table IX.

For the lensing effect, we employ symbolic regression (SR). This technique works by randomly generating a population of expression trees, known as candidate models, and evolving them over a number of evolutionary steps. At each step, either one fit expression is chosen from the population and randomly mutated to produce a child expression, or two fit expressions are selected as parents and mixed, or crossed over, to produce two child expressions. Whichever the selected operation, the resulting expression(s) replace the oldest expression(s) in the population. As this process repeats, the fittest expressions tend to survive as the overall population of candidate expressions evolves to fit the data. By the end of learning, the fittest expression in the final population is returned.

TABLE IX. The GA learned parameters of the damping tail fit as described by Eqs. (F1)–(F11).

Parameter	Value	Parameter	Value
α_1	0.082	α_7	1.3
α_2	0.035	α_8	−0.11
α_3	5.0	α_9	1.3
α_4	0.47	α_{10}	0.022
α_5	1300	α_{11}	0.059
α_6	0.46	α_{12}	0.17

To perform SR, we use the Python package PySR, an open-source Python library for SR built on a highly optimized Julia backend. The core PySR algorithms are described in [56]. PySR performs SR on independent populations of expressions asynchronously, and allows for expressions to migrate across populations after a set number of evolutionary steps, leading to efficient and parallelizable learning.

PySR's inner loop is a simple evolutionary algorithm performed on each population and constitutes one evolutionary step. First, an evolution is chosen from the following: a selected expression is randomly mutated, two fit expressions are selected to be crossed over, a selected expression is algebraically simplified, or a selected expression's constants are optimized. To select an expression for evolution, or two if crossing over, a random subset of the population is chosen, and the best fit expression(s) within, according to a specified fitness function, is selected. If child expressions are produced via random mutation or crossing-over, these then replace the oldest expressions in the population. If simplification or optimization of the selected expression is chosen, the algorithm directly modifies the original expression in memory.

The outer loop in PySR allows the most fit expressions in each population to migrate across populations in an effort to mitigate the risk that the isolated populations get stuck in a local minimum of the search space. As populations are evolving, PySR stores the best expressions it has seen by population. After all populations have gone through a set number of evolutions, the best seen expressions have a chance at being randomly inserted back into other populations. Following migration, the populations are evolved independently again. The outer loop constitutes one iteration of learning and runs for however many iterations the user specifies.

Besides featuring a robust and optimized implementation of SR, PySR is also highly configurable. PySR allows the user to define a template expression, within which SR learning takes place in a structured way. The user fixes the outer form of the expression and specifies which inner expression(s) and parameters(s) PySR may learn. We take a power-law function as the template expression for our fitting formula of the lensing tail,

$$L(\ell) = 1 + w(\ell)\left[\alpha_{13}\left(\frac{\ell}{\alpha_{14}}\right)^{\alpha_{\rm SR}(\Omega_b h^2, \Omega_m h^2)} - 1\right], \quad \text{(F12)}$$

where

$$w(\ell) = \frac{1}{1 + e^{-(\ell-\alpha_{15})/100}} \quad \text{(F13)}$$

is a sigmoid function, included to smoothly transition into the lensing tail. The width of the sigmoid is fixed to 100. The parameters α_i are to be tuned by PySR in the process of learning. α_1 and α_2 are added to allow for some flexibility in the scaling of the power-law, and α_3 allows the midpoint of the sigmoid to be learned. $\alpha_{\rm SR}$ is the power-law exponent, which we leave for PySR to learn to form an analytic expression for in terms of $\Omega_b h^2$ and $\Omega_m h^2$.

We set PySR to learn for 200 iterations with 20 populations of 1000 candidate expressions. We leave the fitness function as the PySR default, which is the mean square error. We restrict PySR's set of possible operators in constructing expressions to be addition, subtraction, multiplication, and exponentiation. The maximum complexity of the expressions, which limits the number of nodes possible in an expression tree, is set to 30. All other settings and hyperparameters are left at their PySR default values.

After learning, PySR converged to the following values for α_i:

$$\alpha_{13} = 0.83,$$
$$\alpha_{14} = 3218,$$
$$\alpha_{15} = 3240.$$

The learned exponent is

$$\alpha_{\rm SR} = \Omega_m h^2 (\Omega_b h^2)^{-0.877} - 3.342\Omega_m h^2 - 1.118. \quad \text{(F14)}$$

Thus constructed, Eqs. (F12) through (F14) comprise the functional form for the lensing tail fit of the TT power spectrum we consider. We show how well we recover the rough shape and order of magnitude of the lensing tail in Fig. 25.

As a preliminary test, we emulate the CMB TT power spectrum assuming ΛCDM with three free parameters: $\Omega_b h^2, \Omega_c h^2$, and H_0. We employ a ResMLP architecture with PCA decomposition of the data vectors. We use 1.8×10^4 data vectors for training. The parameter value ranges in which we train and test the models are listed in Table X. The results of removing and not removing the damping tail are in Table XI, and a comparison between

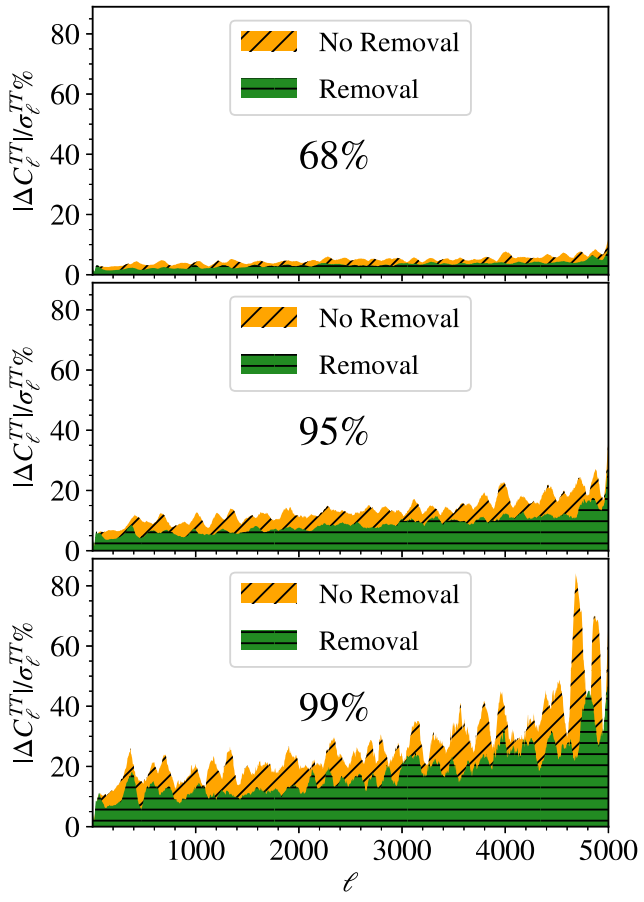

FIG. 24. Plot showing the distributions of the ratio between the absolute error of the emulator and the cosmic variance standard deviation in percentage, with and without removal of the damping tail, both trained on the same architecture and training settings. We can see that we get better emulation on high ℓ region with the damping tail removal, which helps us to get a factor of 2 improvement in $\langle \Delta\chi^2 \rangle_{\rm med}$.

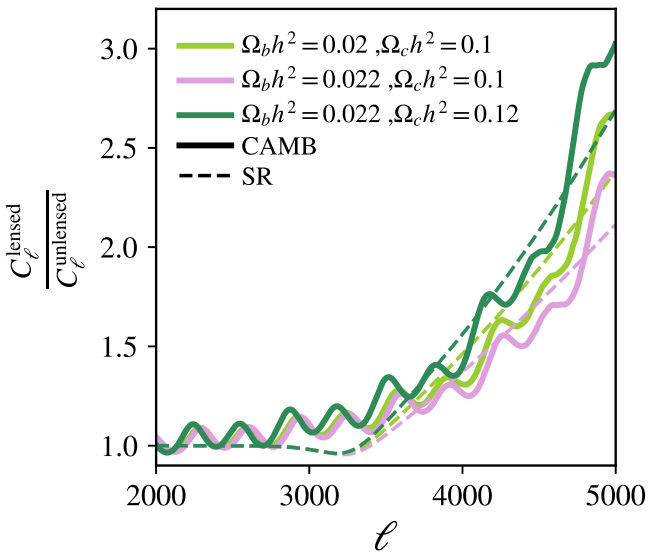

FIG. 25. Plot showing the comparison of the lensing tail ratio between lensed and unlensed CMB TT power spectrum computed from CAMB and from the analytical approximation trained with PySR, on 3 cosmologies. We can see the SR recovers the lensing tail up to the same order of magnitude.

TABLE X. The parameter limits we use in testing the removal of lensed damping tail. We only vary three parameters within a relatively smaller range than in Table II.

Parameter	Remove-damp train	Remove-damp test
$100\Omega_b h^2$	[1.5, 3.3]	[1.6, 3.2]
$10\Omega_c h^2$	[0.3, 1.8]	[0.4, 1.7]
H_0	[45, 95]	[50, 90]

TABLE XI. Comparison between ResMLP models trained with and without removal of the lensed damping tail. The number outside of the parenthesis is the $\langle\Delta\chi^2\rangle_{\rm med}$, and the number inside the parenthesis is the fraction of outlier $f(\Delta\chi^2 > 10)$. We compare $f(\Delta\chi^2 > 10)$ because those models have not reached the level of precision as high as before.

ResMLP	$N_{\rm train} = 18$ k Train
No removal	10.1 (0.50)
Removal	4.6 (0.23)

the distribution of ratios between the absolute error of the emulator and the cosmic variance standard deviation in percentage is shown in Fig. 24.

The removal of lensed damping tail provides an improvement of about a factor of 2 in both $\langle\Delta\chi^2\rangle_{\rm med}$ and $f(\Delta\chi^2 > 10)$ with our fitting results. We leave a more comprehensive analysis, including the extension to a larger parameter volume box, the inclusion of additional parameters, and the adoption of alternative analytical expressions, to future work. Here, we highlight that incorporating a more accurate pre-processing step to account for the effects of damping could potentially significantly improve emulator performance.

[1] P. Ade *et al.* (Simons Observatory Collaboration), J. Cosmol. Astropart. Phys. 02 (**2019**) 056.
[2] M. Abitbol *et al.* (Simons Observatory Collaboration), J. Cosmol. Astropart. Phys. 08 (**2025**) 034.
[3] K. Abazajian *et al.*, arXiv:2203.08024.
[4] N. Sehgal *et al.*, Bull. Am. Astron. Soc. **51**, 1 (2019).
[5] A. Lewis, A. Challinor, and A. Lasenby, Astrophys. J. **538**, 473 (2000).
[6] J. Lesgourgues, arXiv:1104.2932.
[7] D. Blas, J. Lesgourgues, and T. Tram, J. Cosmol. Astropart. Phys. 07 (2011) 034.
[8] J. a. Rebouças, D. H. F. de Souza, K. Zhong, V. Miranda, and R. Rosenfeld, J. Cosmol. Astropart. Phys. 02 (2025) 024.
[9] A. Hajian, Phys. Rev. D **75**, 083525 (2007).
[10] M. Bonici, F. Bianchini, and J. Ruiz-Zapatero, Open J. Astrophys. **7**, 10 (2023).
[11] F. Ge *et al.* (SPT-3G Collaboration), Phys. Rev. D **111**, 083534 (2025).
[12] D. Piras and A. Spurio Mancini, Open J. Astrophys. **6**, 20 (2023).
[13] J. U. Lange, Mon. Not. R. Astron. Soc. **525**, 3181 (2023).
[14] F. Feroz, M. P. Hobson, and M. Bridges, Mon. Not. R. Astron. Soc. **398**, 1601 (2009).
[15] M. Kamionkowski, Phys. Rev. D **104**, 063512 (2021).
[16] A. Spurio Mancini, D. Piras, J. Alsing, B. Joachimi, and M. P. Hobson, Mon. Not. R. Astron. Soc. **511**, 1771 (2022).
[17] H. T. Jense, I. Harrison, E. Calabrese, A. Spurio Mancini, B. Bolliet, J. Dunkley, and J. C. Hill, RAS Tech. Instrum. **4**, rzaf002 (2025).
[18] G.-J. Wang, C. Cheng, Y.-Z. Ma, J.-Q. Xia, A. Abebe, and A. Beesham, Astrophys. J. Suppl. Ser. **268**, 7 (2023).
[19] B. Bolliet, A. Spurio Mancini, J. C. Hill, M. Madhavacheril, H. T. Jense, E. Calabrese, and J. Dunkley, Mon. Not. R. Astron. Soc. **531**, 1351 (2024).
[20] K. Zhong, E. Saraivanov, J. Caputi, V. Miranda, S. S. Boruah, T. Eifler, and E. Krause, Phys. Rev. D **111**, 123519 (2025).
[21] E. Saraivanov, K. Zhong, V. Miranda, S. S. Boruah, T. Eifler, and E. Krause, Phys. Rev. D **111**, 123520 (2025).
[22] E. Krause *et al.*, arXiv:2105.13548.
[23] Y. Bengio, P. Simard, and P. Frasconi, IEEE Trans. Neural Networks **5**, 157 (1994).
[24] R. Pascanu, T. Mikolov, and Y. Bengio, arXiv:1211.5063.
[25] K. He, X. Zhang, S. Ren, and J. Sun, arXiv:1512.03385.
[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, arXiv:1706.03762.
[27] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, in *Proceedings of the International Conference on Machine Learning (ICML)* (PMLR, 2020).
[28] A. Vyas, A. Katharopoulos, and F. Fleuret, *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)* (Curran Associates, Inc., 2020).
[29] R. Li, J. Su, C. Duan, and S. Zheng, arXiv:2007.14902.
[30] R. Dolga, L. Maystre, M. Cobzarenco, and D. Barber, arXiv:2402.17512.
[31] N. Kitaev, Ł. Kaiser, and A. Levskaya, arXiv:2001.04451.
[32] S. Zhai, W. Talbott, N. Srivastava, C. Huang, H. Goh, R. Zhang, and J. Susskind, arXiv:2105.14103.
[33] J. Alsing, H. Peiris, J. Leja, C. Hahn, R. Tojeiro, D. Mortlock, B. Leistedt, B. D. Johnson, and C. Conroy, Astrophys. J. Suppl. Ser. **249**, 5 (2020).
[34] A. Nygaard, E. B. Holm, S. Hannestad, and T. Tram, J. Cosmol. Astropart. Phys. 05 (**2023**) 025.
[35] J. R. Bond, A. H. Jaffe, and L. E. Knox, Astrophys. J. **533**, 19 (2000).
[36] H. K. Eriksen, J. B. Jewell, C. Dickinson, A. J. Banday, K. M. Górski, and C. R. Lawrence, Astrophys. J. **676**, 10 (2008).

[37] N. Aghanim *et al.* (Planck Collaboration), Astron. Astrophys. **641**, A5 (2020).
[38] H. Prince and J. Dunkley, Phys. Rev. D **105**, 023518 (2022).
[39] C. Devon Lin and B. Tang, arXiv:2203.06334.
[40] N. Aghanim *et al.* (Planck Collaboration), Astron. Astrophys. **641**, A6 (2020).
[41] W. Hu and M. J. White, Astrophys. J. **479**, 568 (1997).
[42] J. Adamo, H.-J. Huang, and T. Eifler, Phys. Rev. D **110**, 123517 (2024).
[43] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu, arXiv:2104.09864.
[44] E. Calabrese *et al.* (ACT Collaboration), J. Cosmol. Astropart. Phys. 11 (2025) 063.
[45] https://github.com/CosmoLike.
[46] A. MacInnis, N. Sehgal, and M. Rothermel, Phys. Rev. D **109**, 063527 (2024).
[47] J. Lesgourgues, arXiv:1104.2934.
[48] N. Aghanim *et al.* (Planck Collaboration), Astron. Astrophys. **641**, A1 (2020).
[49] M. J. Mortonson, W. Hu, and D. Huterer, Phys. Rev. D **79**, 023004 (2009).
[50] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, arXiv:1905.03554.
[51] S. Pandya, Y. Yang, N. Van Alfen, J. Blazek, and R. Walters, arXiv:2504.05235.
[52] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (The MIT Press, Cambridge, MA, 1992).
[53] J. B. Orjuela-Quintana, D. Sapone, and S. Nesseris, Phys. Rev. D **113**, 063526 (2026).
[54] D. J. Bartlett, B. D. Wandelt, M. Zennaro, P. G. Ferreira, and H. Desmond, Astron. Astrophys. **686**, A150 (2024).
[55] J. B. Orjuela-Quintana, S. Nesseris, and W. Cardona, Phys. Rev. D **107**, 083520 (2023).
[56] M. Cranmer, arXiv:2305.01582.