A complete framework for cosmological emulation and inference with `CosmoPower`

H. T. Jense,[1★] I. Harrison,[1] E. Calabrese,[1] A. Spurio Mancini,[2,3] B. Bolliet,[4,5] J. Dunkley,[6,7] J. C. Hill[8]
[1]*School of Physics and Astronomy, Cardiff University, The Parade, Cardiff, Wales CF24 3AA, UK*
[2]*Department of Physics, Royal Holloway, University of London, Egham Hill, Egham, UK*
[3]*Mullard Space Science Laboratory, University College London, Dorking, RH 5 6NT, UK*
[4]*Kavli Institute for Cosmology, University of Cambridge, Madingley Road, Cambridge CB3 0HA*
[5]*DAMTP, Centre for Mathematical Sciences, Wilberforce Road, Cambridge CB3 0WA, UK*
[6]*Joseph Henry Laboratories of Physics, Jadwin Hall, Princeton University, Princeton, NJ, USA 08544*
[7]*Department of Astrophysical Sciences, Peyton Hall, Princeton University, Princeton, NJ USA 08544*
[8]*Department of Physics, Columbia University, New York, NY 10027, USA*

Accepted XXX. Received YYY; in original form ZZZ

ABSTRACT

We present a coherent, re-usable `python` framework which further builds on the cosmological emulator code `CosmoPower`. In the current era of high-precision cosmology, we require high-accuracy calculations of cosmological observables with Einstein-Boltzmann codes. For detailed statistical analyses, such codes often incur high costs in terms of computing power, making parameter space exploration costly, especially for beyond-ΛCDM analyses. Machine learning-enabled emulators of Einstein-Boltzmann codes have emerged as a solution to this problem and have become a common way to perform fast cosmological analyses. With the goal of enabling generation, sharing and use of emulators for inference, we define standards for robustly describing, packaging and distributing them, and present software for easily performing these tasks in an automated and replicable manner. We provide examples and guidelines for generating your own sufficiently accurate emulators and wrappers for using them in popular cosmological inference codes. We demonstrate our framework by presenting a suite of high-accuracy emulators for the `CAMB` code's calculations of CMB C_ℓ, $P(k)$, background evolution, and derived parameter quantities. We show that these emulators are accurate enough for both ΛCDM analysis and a set of single- and two-parameter extension models (including $N_{\rm eff}$, $\sum m_\nu$ and $w_0 w_a$ cosmologies) with stage-IV observatories, recovering the original high-accuracy Einstein-Boltzmann spectra to tolerances well within the cosmic variance uncertainties across the full range of parameters considered. We also use our emulators to recover cosmological parameters in a simulated cosmic-variance limited experiment, finding results well within 0.1σ of the input cosmology, while requiring typically $\lesssim 1/50$ of the evaluation time than for the full Einstein-Boltzmann computation.

Key words: methods: statistical – cosmic background radiation – large-scale structure of the universe

1 INTRODUCTION

In the last two decades, cosmological observations have become a continuous source of ever-tightening constraints on models of the expansion and composition of the Universe. Bounds on cosmological parameters now come from a variety of measurements. Cosmic Microwave Background (CMB) temperature, polarisation and lensing data from satellite and ground-based experiments – from e.g., *Planck*[1] (Planck Collaboration VI 2020), the Atacama Cosmology Telescope[2] (ACT, Aiola et al. 2020; Choi et al. 2020; Madhavacheril et al. 2024; Qu et al. 2024b) and the South Pole Telescope[3] (SPT, Balkenhol et al. 2022; Pan et al. 2023) – yield percent-level limits on the parameters of both the standard ΛCDM cosmological model and some of its possible extensions. Tests of this model will become even more stringent in the next decade with the new, upcoming CMB observatories such as the Simons Observatory[4] (SO, Simons Observatory Collaboration 2019) and CMB-S4[5] (CMB-S4 Collaboration 2016). In addition, statistics of the late-time distribution of matter such as galaxy lensing and clustering add information on cosmological parameters which track the growth of structures caused by the matter and dark energy fields in the local Universe. These come from a number of large-scale-structure surveys – including the Dark

★ E-mail: jenseh@cardiff.ac.uk
[1] `https://www.cosmos.esa.int/web/planck/pla`
[2] `https://act.princeton.edu/`
[3] `https://pole.uchicago.edu/public/Home.html`
[4] `https://simonsobservatory.org/`
[5] `https://cmb-s4.org/`

Energy Survey[6] (DES, Abbott et al. 2022), the Kilo-Degree Survey[7] (KiDS, Heymans et al. 2021), the Hyper Suprime-Cam Survey[8] (HSC, More et al. 2023; Miyatake et al. 2023; Sugiyama et al. 2023) – which will soon be overtaken by major new experiments such as the Vera C. Rubin Observatory's Legacy Survey of Space and Time[9] (LSST, Mandelbaum et al. 2018), the *Euclid* satellite[10] (Scaramella et al. 2022), the Nancy Grace Roman Space Telescope[11] (Eifler et al. 2021) and the SPHEREx Observatory[12] (Doré et al. 2014). Finally, the imprint of cosmic perturbations on the baryonic matter is mapped by spectroscopic galaxy surveys – from the Baryon Oscillation Spectroscopic Survey (BOSS, Alam et al. 2021) to the new Dark Energy Spectroscopic Instrument (DESI, Aghamousa et al. 2016; Adame et al. 2024).

The high precision available from these experiments sets strong demands on the accuracy of theoretical modelling of their data vectors, in particular for the upcoming next-generation of surveys, usually labelled as Stage-IV experiments. Higher levels of physical and numerical accuracy in the codes which predict observables in a given cosmological model typically come at the expense of longer evaluation times. Full cosmological exploitation of the data relies on many such evaluations of this 'forward model' when calculating likelihood values (and subsequent posterior estimates) and the total amount of time required can easily become intractable.

Various works (see e.g., Spurio Mancini et al. 2022 and references therein) presented methods to speed up this process by means of emulating the Einstein-Boltzmann codes (typically `CAMB`[13], Lewis et al. 2000, or `class`[14], Lesgourgues 2011; Blas et al. 2011, which are commonly used to accurately compute linear-theory cosmological power spectra and background evolution quantities).

However, at present, each study typically generates its own emulator, tailored and limited to the specific need of the analysis performed. This means that there is limited general use for the wide variety of emulators currently existing, due to the lack of standardisation and cross-platform support for them. Aside from applicability and redundancy issues, the ad-hoc use emulators could also be a potential cause of inconsistencies between different analyses, and a limitation for model comparisons and for data combinations. There are many compelling reasons to expect that further advances in our understanding of cosmology will necessarily come from cross-correlation analyses between different experiments. Such analyses maximise both statistical constraining power and robustness to instrument and astrophysical systematics. They also require the possibility to analyse the different data using a single unified theoretical framework, in order to make consistent predictions for the different data types during inference. Though packages such as `Cobaya`[15](Torrado & Lewis 2019, 2021), `CosmoSIS`[16] (Zuntz et al. 2015) and MontePython (Brinckmann & Lesgourgues 2019; Audren et al. 2013) exist to enable this, there are still gaps which prevent data combinations which would otherwise be fruitful. By making emulators portable across platforms and frameworks, the work presented here will enable novel data combinations much more easily.

Lack of consistency across emulators also limits the ability to deploy these techniques for other applications. For example, a further use for emulators of Einstein-Boltzmann codes lies in the possibility of *autodifferentiation* (*autodiff.*), a computational method of quickly evaluating partial derivatives of the outputs with respect to their inputs. When these derivatives are known, more effective sampling methods such as Hamiltonian Monte Carlo (HMC), which requires accurately knowing the derivatives of often complex relations, become trivial to include. As an example, Campagne et al. (2023) presented a computational framework to autodifferentiate forward models for various cosmological observables. In their paper, they showed how using a specific implementation of HMC known as a No U-Turn Sampler (NUTS) can lead to statistical constraints similar to classical MCMC algorithms in $1/5$th of the time. While for classical Einstein-Boltzmann codes, finding these derivatives is a complicated if not impossible task, this becomes a trivial option when using emulators such as neural networks, in which the computational models used to map between inputs and outputs consist of multiple trivially differentiable units. This means commonly-used software libraries from the recent machine learning revolution, such as `tensorflow` or `jax`, are intrinsically able to take advantage of autodiff. Piras & Spurio Mancini (2023) also recently presented an example of the advantages of combining autodifferentiable emulators with HMC posterior sampling, achieving speed ups $O(10^3)$ relative to traditional Boltzmann codes combined with nested sampling methods.

In this paper we address the need of standardisation and maintenance of cosmological emulators by devising and releasing a framework which allows one to generate, re-use, and deploy emulators within the major infrastructure tools used by the cosmological community. Our work builds on, and expands, the initial `CosmoPower` (Spurio Mancini et al. 2022) software[17] and on the development of Stage-IV emulators started in Bolliet et al. (2023). We make use of `CosmoPower` because of its wide range of existing applications (Spurio Mancini & Pourtsidou 2022; Burger et al. 2023; Heydenreich et al. 2023; Linke et al. 2023; Balkenhol et al. 2022; Spurio Mancini & Bose 2023; Moretti et al. 2023; Reeves et al. 2024; Burger et al. 2024; Farren et al. 2023; Carrion et al. 2024; Giardiello et al. 2024; Qu et al. 2024a), but the type of packaging and interfaces applied here could also be used with other emulators of Einstein-Boltzmann codes (e.g. Aricò et al. 2021; Mootoovaloo et al. 2022; Nygaard et al. 2023; Bonici et al. 2023; Mauland et al. 2023) or other parts of astrophysical forward models.

In Bolliet et al. (2023), some of us presented high-accuracy emulators for `class`[18]. These emulators are capable of reproducing the CMB primary and lensing power spectrum to precision levels $< 10\%$ of the statistical error bars expected from Stage-IV CMB analyses. Bolliet et al. (2023) also released emulators for both the linear and non-linear matter power spectrum, as well as background-evolving quantities – validated using DES-Y1 and BAO analysis likelihoods. In this manuscript, as well as building the framework for community use of these emulators, we present an equivalent suite for `CAMB` that are accurate enough for Stage-IV analysis and beyond, demonstrating the emulators for cosmic-variance-limited datasets. We release a full software suite for `python` that allows easy creation, testing, and usage of `CosmoPower` emulators, alongside extensive documentation

[6] https://www.darkenergysurvey.org/
[7] https://kids.strw.leidenuniv.nl/
[8] https://hsc.mtk.nao.ac.jp/ssp/survey/
[9] https://www.lsstdesc.org/
[10] https://www.euclid-ec.org/
[11] https://roman.gsfc.nasa.gov/
[12] https://www.jpl.nasa.gov/missions/spherex
[13] https://github.com/cmbant/CAMB
[14] https://github.com/lesgourg/class_public
[15] https://github.com/CobayaSampler/cobaya
[16] https://github.com/joezuntz/cosmosis
[17] https://alessiospuriomancini.github.io/cosmopower/
[18] Similar emulators for SPT CMB analyses were presented in Balkenhol et al. (2022).

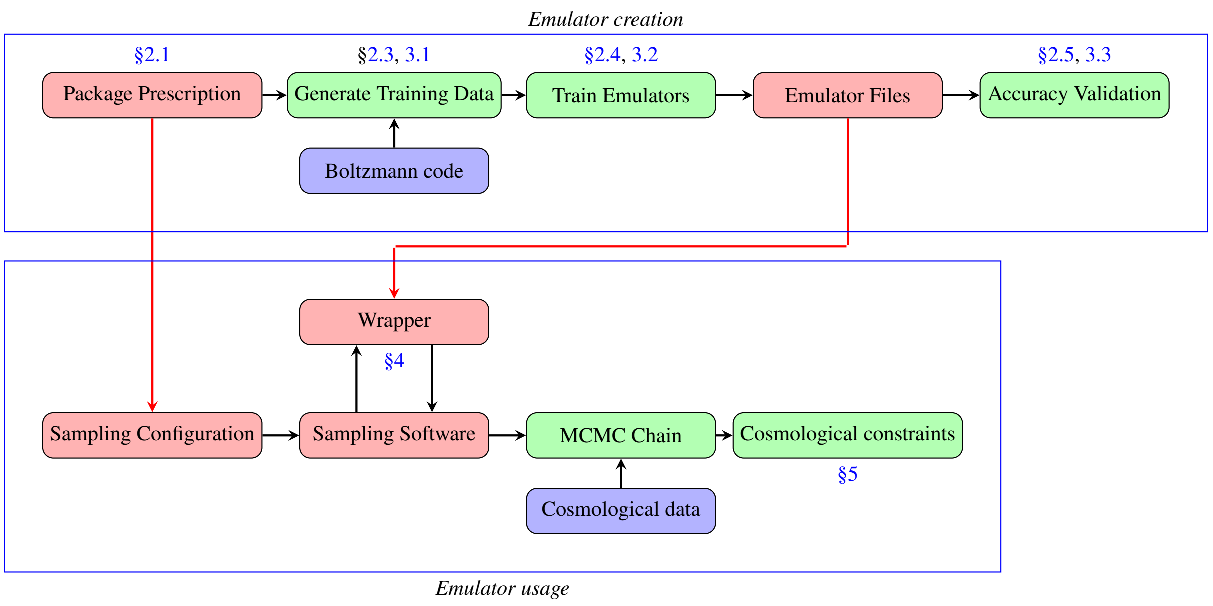


Figure 1. An overview of the workflow with `CosmoPower`: To create a new emulator (**top blue box**), we write a *packaging prescription*, use that to *generate training data*, and from that *train emulators* which outputs several *emulator files*, for which we can easily generate plots which *validate the accuracy* of the emulators. This packaging prescription and set of emulator files are then shared with the end-user (**red arrows**), who wants to use the emulators (**bottom blue box**): the prescription is put inside the *sampling configuration file*, which is given to our *software wrappers*, which provide the user with an *MCMC chain* that can be used to find *cosmological constraints*. The various labels refer to the sections where these individual steps are described in this paper.

and example notebooks. This allows the use of our `CAMB` and `class` emulators, as well as any future extensions or equivalents, within `Cobaya` and `CosmoSIS`, which are some of the most commonly used frameworks for Bayesian inference in cosmology.

A schematic summary of the new aspects introduced in this paper and how they map into different sections is presented in Figure 1. More specifically:

- In Section 2 we include the details of the Einstein-Boltzmann emulators presented in this work: our models considered, parameter ranges, emulated observables, network structure, and training parameters. We also present the accuracy of these emulators in recovering power spectra.
- In Section 3 we give an overview of the packaging scheme and `python` interface we have developed for Machine Learning emulators. We give details of the specification of pre-trained emulators, and how these are exposed to the software. We also provide examples and guidance for others to create emulators using this framework.
- In Section 4 we present our wrappers for the `CosmoSIS` and `Cobaya` sampling software with a brief user guide.
- In Section 5 we use these wrappers to run Monte Carlo posterior estimation chains which shows our emulators recover cosmology at the observable level well within the forecast noise ranges of Stage-IV experiments.
- In Section 6 we summarise and conclude.

2 EMULATORS

In this section we describe the details of our emulators: what is emulated and with which inputs, how an emulation is performed and how the emulators are validated. This serves both as a full description of the emulators released with this manuscript and as guidelines on the creation of new emulators packaged and usable in the same way (e.g. for extended cosmological models). By *emulator* we mean a ‘black box’ code which is capable of ingesting a set of cosmological parameters $\vec{\theta}$ and outputting a set of predictions for the summary statistics of a set of observables $\{\vec{d}_1(\vec{\theta}), \vec{d}_2(\vec{\theta}), \ldots, \vec{d}_N(\vec{\theta})\}$ which are indistinguishable (within a given tolerance) from the set which would have been produced by a code which explicitly implements numerical models of the physics relating the $\vec{d}$ and $\vec{\theta}$. As the emulation works effectively as an interpolation of the quantities $\vec{d}$ between known points, we rely on the fact that the $\vec{d}$ vary smoothly with respect to the input parameters.

2.1 Emulated Quantities

In Table 1 we show the full list of quantities output by the Einstein-Boltzmann code (`CAMB` v.1.5.0) which we focus on emulating in this work. As output observables we generate the CMB temperature, polarisation and lensing potential angular power spectra; linear and non-linear matter power spectra (and their ratio); and a limited set of background expansion and derived perturbation quantities also output by the Einstein-Boltzmann code.

We compute the CMB angular power spectra in the multipole range $2 \leq \ell \leq 10000$ in each of TT, TE, EE, and BB combinations for different cosmological models. In the basic configurations, we use as inputs for our emulators the six cosmological parameters of the standard ΛCDM model: the baryon density $\Omega_b h^2$, the dark matter density $\Omega_c h^2$, the amplitude and spectral index of scalar perturbations $\ln(10^{10} A_s)$ and n_s, the optical depth to reionization $\tau_{\rm reio}$, and the Hubble constant H_0 in units of km/s/Mpc. We add additional model parameters to these for separate emulators for ΛCDM extension models as explained below. When not explicitly varied, neutrinos are described by fixing $N_{\rm eff} = 3.044$, corresponding to the contribution from the three Standard Model neutrino species, with one of them carrying a total 0.06 eV mass.

We also emulate the CMB lensing potential $\phi\phi$ power spectrum in

Quantity	Range	Emulator
C_ℓ^{TT}	$2 \le \ell \le 10000$	NN of log-spectra
C_ℓ^{TE}	$2 \le \ell \le 10000$	NN+PCA of spectra
C_ℓ^{EE}	$2 \le \ell \le 10000$	NN of log-spectra
C_ℓ^{BB}	$2 \le \ell \le 10000$	NN of log-spectra
$C_\ell^{\phi\phi}$	$2 \le \ell \le 10000$	NN+PCA of log-spectra
$P_{\rm lin}(k, z)$	$10^{-4} \le k \le 50$	NN of log-spectra
$P_{\rm NL}(k, z)$	$10^{-4} \le k \le 50$	NN of log-spectra
$P_{\rm NL}/P_{\rm lin}(k, z)$	$10^{-4} \le k \le 50$	NN of spectra ratio
$H(z)$	$0 \le z \le 20$	NN of evolution
$\sigma_8(z)$	$0 \le z \le 20$	NN of evolution
$D_A(z)$	$0 \le z \le 20$	NN of evolution
derived parameters	–	NN of value of derived parameters

Table 1. Emulated quantities, ranges of scales covered and type of emulator employed for each of them.

the same multipole range. For this we use the same parameter inputs except for the optical depth to reionization, given that the lensing potential power spectrum does not depend on it.

For the matter power spectrum $P(k, z)$, we compute the linear matter power spectrum $P_{\rm lin}(k)$ for five input parameters: $\Omega_b h^2$, $\Omega_c h^2$, $\ln(10^{10} A_s)$, n_s, and H_0, plus again the extra parameters for the extension models. For all matter power spectra we also treat the redshift z as an input parameter, resulting in an emulator function which acts as $P_{\rm lin}(k, \vec{\theta})$, where $\vec{\theta}$ includes the redshift. For the non-linear matter power spectrum, we emulate both the $P_{\rm NL}(k)$ spectrum itself and the non-linear boost $P_{\rm NL}/P_{\rm lin}(k) - 1$. For the emulators included in this paper, we emulate the 2020 version of HMCode described in Mead et al. (2021). We sample the wavenumber k at 500 points in the range $10^{-4} \le k \le 50\,{\rm Mpc}^{-1}$ with logarithmic spacing. Note that we compute $P(k)$ up to $k = 100\,{\rm Mpc}^{-1}$ for improved accuracy.

For background evolution quantities, we use redshift in the range $0 \le z \le 20$, sampled at 5000 equally-spaced points, as the modes along which we evaluate the redshift-evolution of the Hubble parameter $H(z)$, the angular diameter distance $D_A(z)$, and the clustering $\sigma_8(z)$ for the five input parameters $\Omega_b h^2$, $\Omega_c h^2$, $\ln(10^{10} A_s)$, n_s, and H_0, plus the extension model parameters where relevant. Adding these background quantities to our emulator packages allows for additional cosmological constraints from e.g., BAO measurements. Additional background quantities, such as $f\sigma_8(z) \equiv -(1+z){\rm d}\sigma_8/{\rm d}z$, can also be easily computed from these quantities with minimal overhead or loss of accuracy.

We also compute ten derived parameters, namely:

(i) The angular acoustic scale θ_* at the surface of last scattering;
(ii) The matter clustering parameter σ_8;
(iii) The primordial helium fraction $Y_{\rm He}$;
(iv) The redshift $z_{\rm reio}$ of reionization, defined as the midpoint of reionization described by a simple hyperbolic tangent;
(v) The optical depth $\tau_{r,{\rm end}}$ at the end of recombination;
(vi) The redshift z_* at the surface of last scattering;
(vii) The sound horizon scale r_* at the surface of last scattering;
(viii) The redshift z_d at the baryon drag epoch;
(ix) The sound horizon scale r_d at the baryon drag epoch;
(x) The effective number of relativistic species $N_{\rm eff}$.

It is common to use θ_*, the angular scale when optical depth is unity, or the approximate parameter $\theta_{\rm MC}$, as a sampled parameter in MCMC analyses of CMB data due to its lower level of covariance with other parameters than H_0[19]. As we also noted in Bolliet et al. (2023) however, CAMB and class use different points at which to evaluate the angular scale (with class defining θ_s as the angular scale at maximum visibility, which is close to but not the same as θ_*, which is used in CAMB). To maintain cross-compatibility between our emulators, and to remain consistent with our earlier work, we therefore use H_0 as an input, and not θ_*. Including these derived parameters as emulators allow us to recover the posterior distributions on these quantities, either directly storing their computed values while sampling the chain, or afterwards by post-processing a converged MCMC chain.

2.2 Cosmological Models

We provide emulators for the ΛCDM model with parameters $\{\Omega_b h^2, \Omega_c h^2, \ln(10^{10} A_s), n_s, H_0, \tau_{\rm reio}\}$ defined above as well as the following four extended models:

(i) ΛCDM+$N_{\rm eff}$: varying the effective number of relativistic species $N_{\rm eff}$;
(ii) ΛCDM+Σm_ν: varying the sum of neutrino masses Σm_ν;
(iii) ΛCDM+$N_{\rm eff}\Sigma m_\nu$: varying both the number and mass sum of neutrinos;
(iv) ΛCDM+$w_0 w_a$: varying the dark energy equation of state described with two parameters w_0 and w_a.

Each of these four extension models is emulated separately, with the extension parameters used as additional inputs. We chose to emulate $+N_{\rm eff}$ and $+\Sigma m_\nu$ separately, and the combination $+N_{\rm eff}+\Sigma m_\nu$ to explore the relation between model complexity and emulator accuracy. While, as we show later, our emulator for the full combination $+N_{\rm eff}+\Sigma m_\nu$ is accurate enough for cosmological analysis, we release the single parameter-extension model emulators as they offer greater accuracy over the higher-dimensional models.

For the non-linear $P_{NL}(k, z)$ and non-linear boost $P_{NL}/P_L(k, z) - 1$ emulators, we include the baryonic feedback parameters A_b, η_b, and $\log T_{\rm AGN}$ that appear in HMCode (Mead et al. 2021) and are otherwise fixed at their default CAMB values in the other emulators. For the remaining model choices, we set a primordial helium fraction set from BBN consistency using PRIMAT (Pitrou et al. 2018), recombination from the CosmoRec code (Chluba & Thomas 2010; Chluba et al. 2010), and reionization modeled with a simple hyperbolic tangent with a redshift width $\Delta z = 0.5$. Most of these options are the default settings in CAMB. We only changed the recombination code to CosmoRec, whereas the CAMB default is to use the older RECFAST code.

2.3 Training Data

Training of emulators involves creating a set of output data $\vec{d}$ at a finite sample of known parameter values $\vec{\theta}$ using the code to be emulated (i.e. the Einstein-Boltzmann code here). These data will then subsequently be used in Section 2.4 for the neural network to learn an approximate (but high accuracy) mapping between input and output. Training data must be generated at a high enough resolution in the input parameters that we can smoothly interpolate between outputs. The training data only need to be generated once, to train the emulator, and do not need to be generated using the computationally intensive numerical code again in any subsequent inference.

Following Spurio Mancini et al. (2022) and Bolliet et al. (2023),

[19] see note at https://cosmologist.info/cosmomc/readme.html

Parameter	Range	Default Value
$\Omega_b h^2$	$[0.015, 0.03]$	–
$\Omega_c h^2$	$[0.09, 0.15]$	–
$\ln(10^{10} A_s)$	$[2.5, 3.5]$	–
n_s	$[0.85, 1.05]$	–
$\tau_{\rm reio}$	$[0.02, 0.20]$	–
H_0 [km/s/Mpc]	$[40, 100]$	–
$z_{\rm pk}$	$[0, 5]$	–
A_b	$[2, 4]$	3.13
η_b	$[0.5, 1.0]$	0.603
$\log T_{\rm AGN}$	$[7.3, 8.3]$	7.8
$N_{\rm eff}$	$[1.5, 5.5]$	3.044
Σm_ν [eV]	$[0, 0.5]$	0.06
w_0	$[-2, 0]$	-1.0
w_a	$[-2, 2]$	0.0

Table 2. Table of parameter ranges over which we trained our emulators. Compare this with the textual specification in Figure 7. The **top** section of the table refers to the background cosmology parameters used in almost all emulators. The **middle** section of the table contains the redshift and baryonic feedback parameters used only in the $P(k)$ emulators, with their default values from CAMB used in the CMB and background evolution emulators. The **bottom** section of the table shows the ranges of the single-/two-parameter extension model emulators, and their default values taken in the base ΛCDM case. Each emulator takes in the first six parameters, and one or two extension parameters, with the exception for $C_\ell^{\phi\phi}$, and background quantities, which do not rely on $\tau_{\rm reio}$.

we generate $N_S = 10^5$ sets of output spectra as training data, of which 20% will be used for validating the network accuracy, and the rest for training. Our parameter space is shown in Table 2. We employ Latin Hypercube (LHC) sampling for ensuring our parameter space is evenly sampled. For extended models, we choose to generate slightly more spectra at $N_S = 1.2 \times 10^5$, to compensate for the expanded parameter space. To demonstrate the need for this and to provide some guidance on how to select N_S, we show a comparison of the mean prediction error versus the size of the training dataset in Figure 2, for a varying number of input parameters. The figure shows that there is not a simple linear scaling with the number of parameters. Although increasing the number of parameters always requires a larger training set to reach the desired target accuracy, the physical nature and range of variation of the specific additional parameter will impact the results. For example, if we extend ΛCDM varying $N_{\rm eff}$ or Σm_ν, we observe different behaviours, even if in both cases it is only one additional input parameter (7 input parameters compared to 6 for ΛCDM). We explain this by noting that cosmological observables have different responses to different parameters, according to the physics signature they are tracking. For example, the CMB C_ℓ^{TT} spectrum will exhibit a strong dependence on $N_{\rm eff}$ – changing both the peak position and amplitude at all scales, but less so on Σm_ν which will primarily appear at scales dominated by lensing. Hence in Figure 2 the ΛCDM+$N_{\rm eff}$ case requires more training than ΛCDM+Σm_ν. When we expand further the model to ΛCDM+$N_{\rm eff}\Sigma m_\nu$ (8 input parameters compared to 6 for ΛCDM), we observe a very similar behaviour to the 7-parameter case ΛCDM +$N_{\rm eff}$, because we have already covered most of the strongly-varying training region. We conclude that to achieve the desired convergence of the emulators, the user will need to monitor the behaviour of their specific model and perform some exploratory studies of how the emulators depend on the model parameters.

To meet the requirements for Stage-IV analyses, we use the CAMB accuracy settings suggested by McCarthy et al. (2022); Hill et al. (2022) as adequate for convergence of the likelihood value obtained from data with this level of precision, summarised in Figure 3.

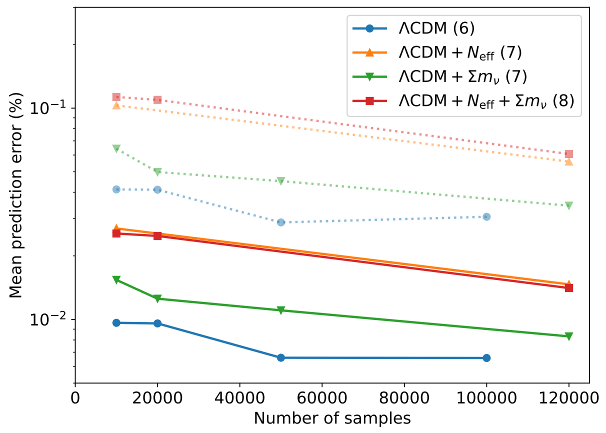


Figure 2. An overview of the accuracy reached by a trained C_ℓ^{TT} emulator given the number of training spectra used to train the emulator, for an increasing number of input parameters. The solid coloured lines and point represent the 68% error of a C_ℓ^{TT} emulator trained with N_S samples (the dotted shaded lines and points show the similar behaviour observed at 99%), averaged over the entire ℓ-range. We show the full-size emulators generated with $N_S = 100000$ for ΛCDM and $N_S = 120000$ for extended models, as well as emulators with a smaller training set to show how accuracy scales with N_S and input parameters. We train emulators for ΛCDM (6 parameters, **blue**), +Σm_ν (7 parameters, **orange**), and +$N_{\rm eff}$ + Σm_ν (8 parameters, **green**), each on a random smaller subset of the full training dataset, scaling the training batch size proportional to the size of the subset. We show how the mean emulation error decreases as the number of training spectra increases, and increases as we increase the complexity of the parameter space. We note, however, that scaling of emulators accuracy with number of input parameters is non linear, the nature and impact on the emulated quantity of the specific parameter will matter for this behaviour.

```
lmax: 10000
kmax: 10.0
k_per_logint: 130
nonlinear: True
lens_potential_accuracy: 8
lens_margin: 2050
lAccuracyBoost: 1.2
min_l_logl_sampling: 6000
DoLateRadTruncation: False
recombination_model: CosmoRec
```

Figure 3. Accuracy settings for CAMB, based on the settings earlier suggested in McCarthy et al. (2022); Hill et al. (2022). For an example of a full yaml file, see Appendix A.

We iterate over each of the N_S samples in our LHC, computing the CMB, lensing, and matter power spectra, as well as background quantities, and derived parameters with CAMB (see Table 1 for a summary of the outputs and their ranges), and store the results in a structured data file containing appropriate metadata (see Appendix B). Because of our choices of parameter limits as a hypercube, a small fraction ($\ll 1\%$) of our samples are in unphysical parts of parameter space and can cause issues in computations from CAMB. Because this number is small, these samples are simply discarded and ignored for future processing.

2.4 Network Design and Training

Following Spurio Mancini et al. (2022), we implement the emulators as dense neural networks, with four hidden layers of 512 neurons each. Each emulator takes the normalised parameters as input, and maps it to normalised spectra. We use the activation function from Spurio Mancini et al. (2022):

$$f(\vec{x}) = \left[\vec{\gamma} + \left(1 + e^{-\vec{\beta}\odot\vec{x}}\right)^{-1} \odot (1 - \vec{\gamma})\right] \odot \vec{x}, \tag{1}$$

where $\odot$ is the element-wise product. For the optimizer, we re-use the Adam optimizer. The input and output quantities are normalised with respect to mean and standard deviations of the respective ranges. For most quantities, as detailed in Table 1 we emulate the logarithm of the spectrum, as the high dynamic range of these values makes it easier for the emulator to reconstruct the log-values. We employ the same method for the background quantities $H(z)$, $\sigma_8(z)$, and $D_A(z)$, where we reconstruct the logarithm of the redshift evolution.

For the C_ℓ^{TE} emulator, the resulting raw spectra include zero-crossings which make emulating the log-spectra impossible. Because the unscaled spectra still contain a high dynamic range in values, we follow Spurio Mancini et al. (2022) in first decomposing the spectra with a Principal Component Analysis (PCA) and then subsequently emulating the sets of PCs. Similar to before, we decompose the C_ℓ^{TE} spectra into 512 PCs. Even though they remain completely positive, we also decompose the $C_\ell^{\phi\phi}$ spectra into 64 PCs. We find that this is more effective at emulating the $\phi\phi$ spectra, which we explain with the reduced dimensionality of the information contained in the $\phi\phi$ spectra. We introduce the procedure of constructing scree plots, showing the eigenvalues associated with each PC in the decomposition, to identify the "elbow" at which higher PC numbers no longer carry significant weight and can be discarded. For more details regarding this and for guidance on decisions regarding PCA see Appendix C. With this setup, our emulator design for the CMB spectra remains fully consistent with the original emulators from Spurio Mancini et al. (2022).

For the matter power spectra $P_{\rm lin}(k, z_{\rm pk})$ and $P_{\rm NL}(k, z_{\rm pk})$, we choose to emulate $\log P_{\rm lin}(k, z_{\rm pk})$ and the non-linear boost $P_{\rm NL}/P_{\rm lin}(k, z_{\rm pk}) - 1$ for best performance. These quantities are functions of two parameters, the wavenumber k and redshift $z_{\rm pk}$. Similar to previous emulators we have developed, we use k as the one-dimensional grid along which we sample our spectra, and use $z_{\rm pk}$ as an additional input for our $P(k)$ emulators.

The time it takes to train an emulator depends on many factors, including the size of the dataset, the number of inputs and outputs of the network, the hardware performance, as well as some inherently stochastic factors in the training process. At 10^5 training samples for a network, we find it takes $O(1h)$ to train a C_ℓ network on a GPU. If no GPU hardware or the required software is available, then the emulators can alternatively be trained on a CPU, which for the same case still only takes $O(10h)$ to perform.

2.5 Accuracy of Emulated Observables

To assess the accuracy of our emulators we perform a number of comparisons between the observables emulated and those calculated directly with `CAMB`. This allows us to understand if we have reached the theoretical calculation accuracy required for Stage-IV analyses. This functionality is now fully built into our released software as described later in Section 3.3.

In Figure 4 we report the difference between direct `CAMB` outputs and emulated observables, showing contours corresponding to the fraction of our training spectra (across the full parameter space) which lie within a given level of agreement with the emulated values. All the CMB spectra reach sub-percent accuracy (note that the TE higher values are numerical artefacts due to diving for a signal crossing zero, see Figure 5 for more details); the matter power spectrum is accurate at the few percent level relative to the `CAMB` prediction for very large range of wave numbers.

For the CMB observables, as done in previous works we can also compute the difference relative to (or 'in units of') a specific experiment's sensitivity which tracks the noise for each observable N_ℓ^{XY} with

$$\sigma_\ell^{XY} = \sqrt{\frac{1}{f_{\rm sky}(2\ell+1)}\left[(C_\ell^{XX} + N_\ell^{XX})(C_\ell^{YY} + N_\ell^{YY}) + (C_\ell^{XY} + N_\ell^{XY})^2\right]}, \tag{2}$$

where, for the cosmic variance limit, $f_{\rm sky} = 1$ and $N_\ell^{XY} = 0$ for all XY.

We show this accuracy of our emulators relative to the cosmic variance-limited experimental noise for ΛCDM in Figure 6 and extended models are shown in Appendix E in Figures E1 to E4 (for ΛCDM $+N_{\rm eff}$, $+\Sigma m_\nu$, $+N_{\rm eff}\Sigma m_\nu$, and $+w_0 w_a$ respectively).

All our emulators remain well within 10% of a cosmic variance-limited experimental uncertainty range. The only exception to this is our $w_0 w_a$ emulator (see Figure E4), for which some outliers at small scales in the CMB emulators can reach about 80% of this uncertainty. We attribute this effect to the parameter degeneracy of the model, as well as the complexity of this model and the relatively wide range of parameters we chose. However, in the absence of CMB sensitivity to the mechanics of dark energy, and in the interest of the recent results from DESI (DESI Collaboration 2024), we are still including this emulator.

3 PACKAGING DESCRIPTION

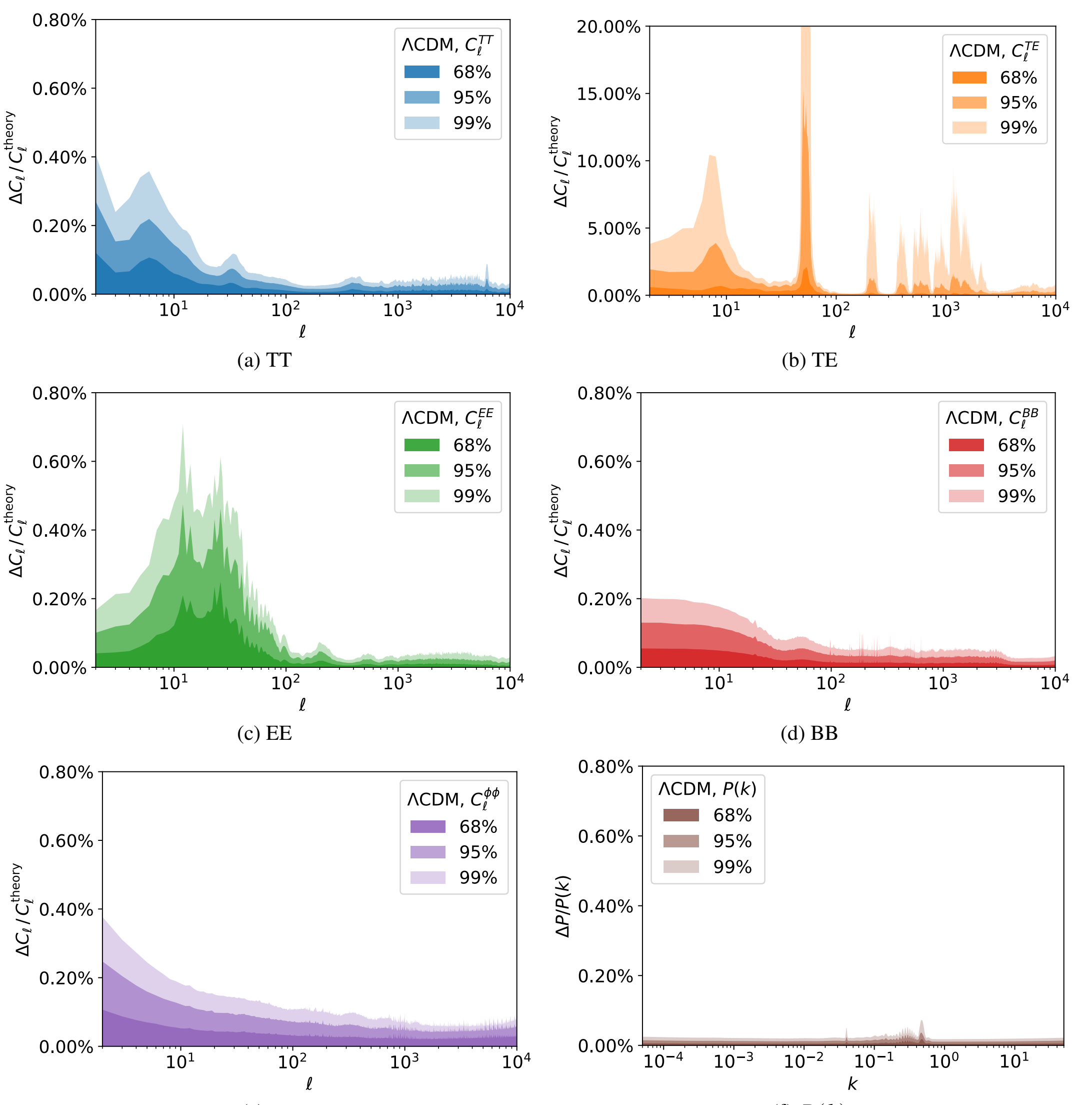
As part of this release, alongside new emulators we build a packaging prescription for `CosmoPower` emulators. This prescription is both human- and machine-readable and serves as a description of what the emulator is capable of and its full design specifications. The `CosmoPower` software package[20] has been updated to include a full parser for the packaging prescription.

To create and train a new emulator, the packaging prescription is designed to guide both the author and a later user through the process of considering what quantities are emulated, how, and to what accuracy.

In this section, we describe the main steps of creating an emulator, namely: (1) describing the input parameters and output data, and generating the training spectra with the Einstein-Boltzmann code, (2) detailing the specifications of the emulator and the training parameters, and performing the training process, and (3) testing the validation of emulators. We follow the creation of the emulators we specified in Section 2, and describe how the packaging prescription of these emulators is setup, as well as alternative options and choices available for the user.

We also create and release packaging for the emulators for the `class` Einstein-Boltzmann code presented in Bolliet et al. (2023)

[20] https://github.com/alessiospuriomancini/cosmopower



(a) TT (b) TE

(c) EE (d) BB

(e) $\phi\phi$ (f) $P(k)$

Figure 4. A validation graph generated from our trained networks for ΛCDM. We show the error in the reconstructed CMB power spectrum in C_ℓ^{TT} (blue, top-left), C_ℓ^{TE} (orange, top-right), C_ℓ^{EE} (green, centre-left), C_ℓ^{BB} (red, centre-left), $C_\ell^{\phi\phi}$ (purple, bottom-left), and linear $P_{\rm lin}(k)$ (brown, bottom-right) relative to the `CAMB` theory curve. The bands show the 68/95/99% contours (from darkest to lightest shades). Note the different scale for TE, for which errors get blown up due to the zero-crossings of the input power spectrum.

which also achieve Stage-IV-level accuracy, consistent with the `CAMB` emulators in this work. With our included packaging, these emulators can likewise be used in the inference frameworks with the same level of convenience and robustness.

3.1 Generating Training Data

In this subsection, we discuss the required prescription of the input parameters for emulators, and for the output of quantities that are desired to be emulated.

As mentioned above, `CosmoPower` uses LHC sampling, which allows for an evenly spaced grid of sampling points that are sufficiently distributed that the entire parameter space is covered with minimal variation in sampling density. In Figure 7 we show how to specify the LHC grid in the prescription file.

The `emulated_code` block of the packaging contains information about the Einstein-Boltzmann code being emulated, in particular the name and version number. If a customized version of a code is used, it is possible to manually specify the import path with the `boltzmann_path` keyword. The `inputs` keyword is the list of named parameters which will be varied as inputs to the Einstein-Boltzmann code. `extra_args` contains code parameters which embody any model choices or approximation and accuracy settings.

The `samples` block specifies the `Ntraining` training spectra to be generated. The packaging prescription recognises four different types of parameters in the `parameters` block:

(i) Sampled parameters, these are the parameters that the LHC is created over, and are defined with a minimum-maximum pair for the range over which the LHC is sampled, e.g. `ombh2: [0.015, 0.03]`;
(ii) Derived parameters, these are parameters that are trivially derived from other sampled parameters, and are defined with a text string prescribing a python lambda function equation

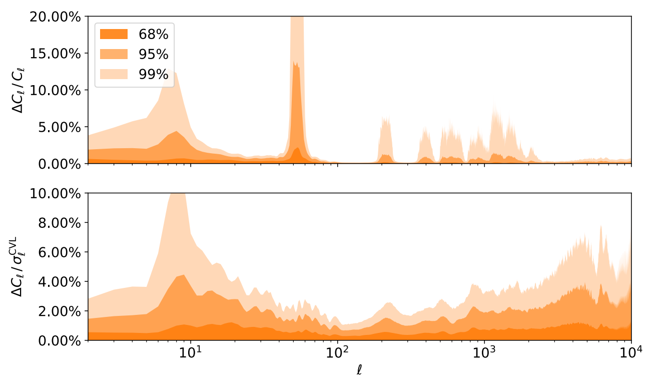


Figure 5. A direct comparison of the error in the C_ℓ^{TE} emulator as measured in fractional error with respect to the training spectrum (**top** and as in Figure 4), and as relative error with respect to a cosmic variance-limited noise curve (**bottom** and as in Figure 6 with more details in Section 2.5). The peaks in the top figure are due to the zero-crossings of the C_ℓ^{TE} power spectrum, which "blow up" any errors in the emulator. Using a cosmic variance limit noise curve provides a more realistic error measure, as shown in the bottom figure, where the inclusion of the C_ℓ^{TT} and C_ℓ^{EE} terms in the error wash out these zeroes and provide a more reasonable assessment for the error.

to derive them directly, e.g. `As: "lambda logA: 1.e-10 * np.exp(logA)"`;

(iii) Fixed parameters, these are simply defined by writing a single numerical value that the parameter is set to, e.g. `mnu: 0.06`;

(iv) Computed parameters, these are parameters that we cannot easily compute ourselves, but the Boltzmann code can, and these are defined by simply leaving an empty tag in the parameter list. These parameters are specified by variable names available to `CosmoPower` at the spectra generation stage via the python interfaces of the Einstein-Boltzmann codes being emulated, e.g. "`YHe:` " for $Y_{\rm He}$.

Any of these types of parameters can be used as an input to a network, and any of the first three types can be used as an input for the Einstein-Boltzmann code. It is for example possible to create an LHC over a range of Hubble parameter H_0, while using the angular scale θ_*, as computed by the Einstein-Boltzmann code, as an input for the emulators.

The `networks` block specifies the neural networks to be created using the training data. It is possible to specify multiple networks, each under a `quantity` heading, which each have their own set of network properties specified as further blocks and keywords. When creating `CosmoPower` networks, the current list of quantities to which can emulated is defined and described as follows:

- `Cl/xy`: referring to (lensed) CMB angular power spectra C_ℓ^{XY} with X, Y any combination of T/E/B (C_ℓ^{TT}, C_ℓ^{TE}, C_ℓ^{EE}, C_ℓ^{TB}, C_ℓ^{EB}, and C_ℓ^{BB});
- `Cl/pp`: CMB lensing potential spectrum for $C_\ell^{\phi\phi}$, there are also options available for cross-spectra with primary CMB via `Cl/pt`, `Cl/pe`, and `Cl/pb`;
- `Pk/lin` and `Pk/nonlin`: Matter power spectrum for linear $P_{\rm lin}(k, z)$ and non-linear $P_{\rm nl}(k, z)$;
- `Pk/nlboost`: The non-linear boost $(P_{\rm NL}/P_{\rm lin} - 1)(k, z)$ defined as the non-linear boost to the linear matter power spectrum;
- `Hubble`, `Omegab`, `Omegac`, `Omegam`, `sigma8` and `DA`: The redshift-evolving quantities $H(z)$, $\Omega_b(z)$, $\Omega_c(z)$, $\Omega_m(z)$, $\sigma_8(z)$, and $D_A(z)$.

It is also possible to specify `derived` quantities. This network will automatically use all parameters from the `parameter` block that are computed by the Einstein-Boltzmann code as outputs. So, when we specify a `derived` network in our emulators similar to our C_ℓ^{TT} emulator, we create an emulator that emulates the computation of the nine quantities mentioned in Section 2.1 (which are the nine parameters we listed in Figure 7).

In Figure 8 we show an example for the `network` block of an emulator trained on primary CMB C_ℓ^{TT} data for $2 \leq \ell \leq 10000$. We discuss the choices made in this block in more detail in Section 3.2.

Once the packaging file has been set up with the sections specified above, it becomes easy to generate training data for networks by calling:

```
python -m cosmopower generate <yamlfile>
```

In addition, the `--resume` flag can be used to increase more samples for an already existing set of data points, if it is found afterwards that the training set size is not large enough for training to result in good recovery of spectra from the emulator. When resuming the generation of samples, any pre-existing LHC will be used (if compatible with the given prescription) and any pre-existing samples are not re-generated. This can be used for continuing a run that was cancelled or stopped before, adding new quantities that were not computed earlier, or increasing the number of samples beyond the LHC that was generated beforehand.

We store the generated training data in hdf5 files, which are optimised for large, table-like datasets, and allow for both fast read-write access and good data compression. We also include the option to automatically split the data into multiple files, to prevent memory issues from opening a too large a single file at once. For our ΛCDM emulators, this means that we generate about 4 GB worth of training spectra per emulator, split across ten files.

3.2 Network Specification and Training

The `networks` block contains information on which emulator is to be trained, and how the network is designed; it contains:

(i) The type of emulator, either `NN` for a neural network emulating the spectra directly, or `PCAplusNN` for a NN emulating the PCA of the quantity;

(ii) The list of inputs used for the network, these can be different from the inputs to the Boltzmann code, and hence may need to be specified again;

(iii) Whether the network should be trained on log-spectra;

(iv) The range of modes (sampling points) over which the output spectrum is computed, and a text label for them (i.e. ℓs for C_ℓ spectra, k for $P(k)$ spectra, and redshifts z for background quantities);

(v) The specification for traits of the Neural Network emulator. For a dense neural network, the traits should contain the number of nodes per hidden layer. For a network that employs a PCA, the number of retained PCs must be given.

(vi) The specification for the steps taken when training the emulator (see below for details).

After the training data has been generated, training a network is done via a similar command:

```
python -m cosmopower train <yamlfile>
```

Training depends on a variety of parameters, which are set in the `training` block of the networks prescription. These parameters (explained below) are:

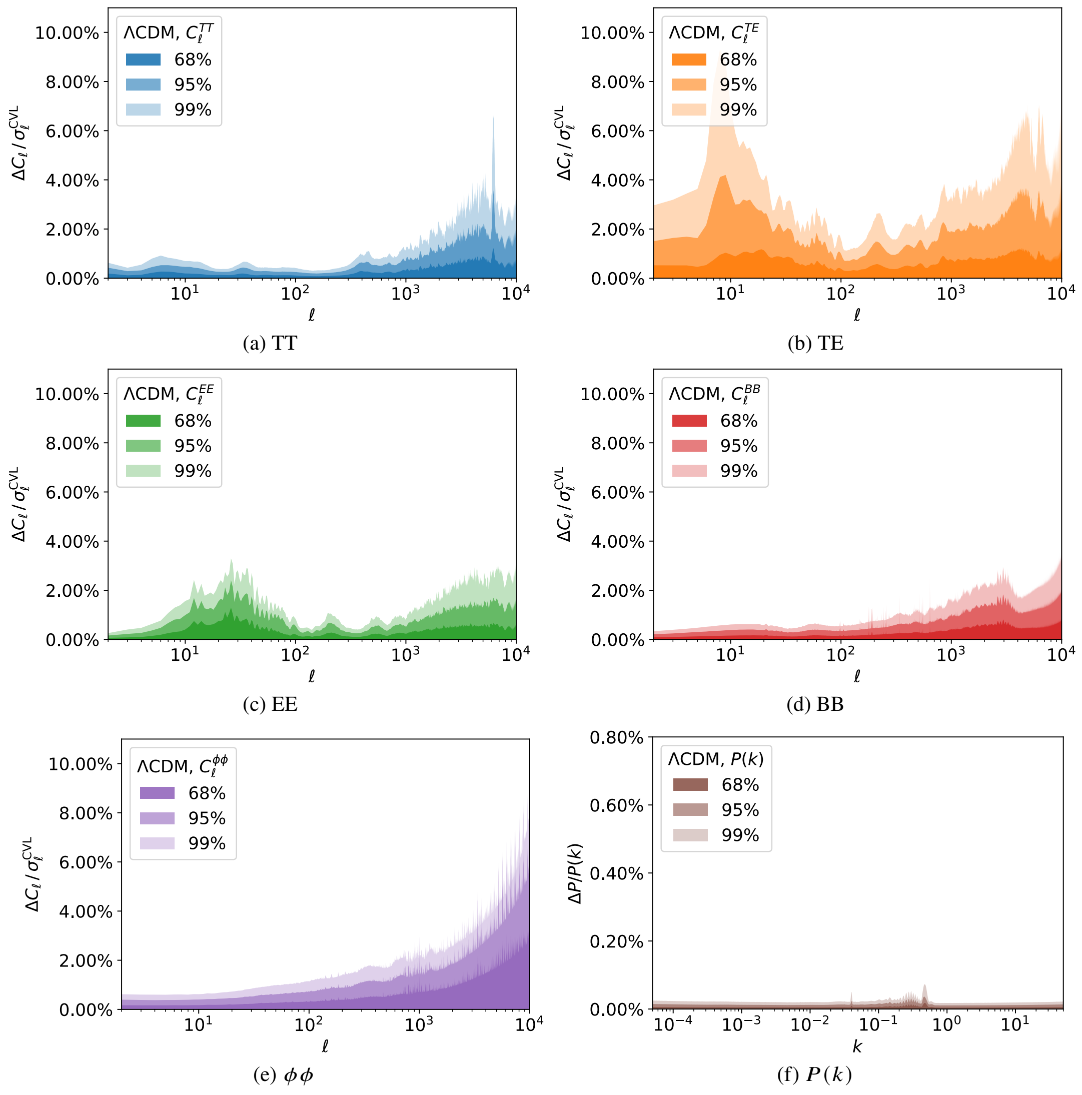


Figure 6. A validation graph generated from our trained networks for ΛCDM. We show the recovered CMB power spectrum C_ℓ^{TT} (blue, top-left), C_ℓ^{TE} (orange, top-right), C_ℓ^{EE} (green, center-left), C_ℓ^{BB} (red, center-right), $C_\ell^{\phi\phi}$ (purple, bottom-left), and linear $P_{\rm lin}(k)$ (brown, bottom-right), with respect to the cosmic variance limit for C_ℓ's, and as a fractional difference for $P(k)$. The bands show the 68/95/99% contours (from darkest to lightest shades).

(i) The learning rate, which controls the size of steps taken at each learning epoch;
(ii) A batch size, which controls the size of a batch over which a learning step is averaged;
(iii) The validation split, which controls how many spectra are kept aside of validation calculation;
(iv) The number of steps used for gradient accumulation;
(v) A patience value, which controls how long a network allows itself to be "stuck" at a loss value before continuing to the next learning iteration;
(vi) The maximum number of epochs in each learning iteration.

Each of these values can be set to either a single number or a list of length N_L, which indicates the number of *learning iterations* used. If a value is set to a single number, it is kept fixed over the course of each learning step, otherwise `CosmoPower` will iterate over the values in the list when training. If multiple values are to be iterated over, these lists need to be of the same length.

`CosmoPower` will train a network by iterating over these *learning iterations*, each of which consists of a number of *epochs* set by the `max_epoch` value. A fraction of samples equal to the `validation_split` is set aside each learning iteration, and the remainder is used as the training set. The training set is then grouped into *batches* determined by the `batch_size` value. Every epoch, each batch is passed through the emulator, and the trainable hyperparameters of the emulator are updated to reduce the loss function of the network. If a number of `gradient_accumulation_steps` $g > 1$ is given, then g consecutive steps are used to compute the total derivative of the loss function with respect to the hyperparameters as well, which can give a better learning rate, especially when using a GPU for increased computation of these derivatives. `CosmoPower` uses the *Adam* optimiser to determine how to tweak the hyperparameters, and the learning step size is multiplied by the `learning_rate` of this iteration. After going through a full epoch, the validation set is passed through the emulator and its loss is computed. If the validation loss has improved throughout this iteration, then the new hyperparameters are kept. If the `max_epoch` value is reached, or if

```
emulated_code:
  name: camb
  version: "1.5.0"
  inputs: [ombh2, omch2, As, ns, H0, tau]
  extra_args:
    <...>

samples:
  Ntraining: 100000
  parameters:
    ombh2: [0.015,0.03]
    omch2: [0.09,0.15]
    # We want to sample on log(10^10 As), but our
    # Boltzmann code takes As as an input.
    logA: [2.5,3.5]
    As: "lambda logA: 1.e-10 * np.exp(logA)"
    tau: [0.02, 0.20]
    ns: [0.85, 1.05]
    H0: [40.0, 100.0]
    # Parameters computed by the Boltzmann code
    thetastar:
    sigma8:
    YHe:
    zrei:
    taurend:
    zstar:
    rstar:
    zdrag:
    rdrag:
    Neff:
```

Figure 7. Code snippet for sampling and parameters block, compare this with Table 2. In the example here, we setup the aforementioned six parameters to sample over, add an intermediate parameter A_s, and add the nine parameters which are derived directly from the Boltzmann code, in this case `CAMB`. Note that `CAMB` expects the primordial amplitude A_s to be provided, but it is far more common to sample over $\ln(10^{10}A_s)$ instead. By defining the `logA` parameter and marking the `As` parameter as a derived parameter from that, we can perfectly accomplish this. At the bottom we show the nine parameters we derive from the Boltzmann code - in this case, they are computed by `CAMB`. It is possible to use any of the parameters defined in this block as an input to the networks, including the parameters derived from the Boltzmann code. The `extra_args` block would include any accuracy settings, as seen in Figure 3.

the validation loss has not improved over `patience_values` epochs in a row, then the emulator will go to the next learning iteration.

Because of the large amount of freedom in choosing these values, it can be hard to determine what settings are optimal for a good training pass. In addition, the impact of certain decisions can wildly vary from either minimal to substantial. As a result, we cannot provide clear guidance on what settings to use but there are a few rules of thumb that can be used when determining the training settings which we recommend:

- The validation split should be about 10-20%;
- Each iteration, the learning rate should go down and the batch size should go up;
- If a learning iteration reaches the maximum number of epochs instead of a patience value, that means it could have learned for longer, and it hasn't fully optimised yet - try to increase the batch size or learning rate for this iteration or an earlier one.

`CosmoPower` keeps track of the validation loss for every epoch, and saves this to a plain text file for post-training analysis and diagnosis of training issues.

```
networks:
  - quantity: "Cl/tt"
    inputs: [ombh2, omch2, logA, ns, H0, tau]
    type: NN
    log: True
    modes:
      label: l
      range: [2,10000]
    n_traits:
      n_hidden: [512, 512, 512, 512]
    training:
      validation_split: 0.1
      learning_rates: [1.e-2, 1.e-3, 1.e-4, 1.e-5,
          1.e-6, 1.e-7]
      batch_sizes: [1000, 2000, 5000, 10000,
         20000, 50000]
      gradient_accumulation_steps: 1
      patience_values: 100
      max_epochs: 1000
```

Figure 8. Code snippet for network block. We setup a network that emulates $\log_{10}(C_\ell^{TT})(\vec{\theta})$ with our six input parameters $\vec{\theta} = \{\Omega_b h^2, \Omega_c h^2, \log(10^{10}A_s), n_s, h, \tau\}$ and ℓ between 2 and 10000. The network is a fully connected dense neural network with 4 hidden layers of 512 neurons each. Our training block defines the fraction of example spectra used for validation estimation, the learning rates of each learning step, the batch size over which we average, any gradient accumulation steps, patience values, and maximum number of training epochs.

3.3 Assessing Accuracy

The validation loss for the emulators is only one quantity to evaluate the accuracy, but it is important to explicitly evaluate the accuracy of the output emulator quantities. We include functionality to generate accuracy plots, that show the average difference between the emulated quantity and the original quantity as computed by the Einstein-Boltzmann code, relative to ('in units of') an observable error.

For a trained emulator, one can evaluate the accuracy of the emulator by invoking the command:

```
python -m cosmopower show-validation <yamlfile>
```

This command will pass a fraction of all original samples through the each trained emulator and plot the emulator error. The accuracy of the emulated observables can be defined as either the fractional difference to the true value, or relative to some observational error, as defined in e.g. Section 2.5. There are options to use either the public Simons Observatory noise curves noise curves, presented in The Simons Observatory collaboration (2019), or a cosmic variance-limited uncertainty.

4 WRAPPER DESCRIPTION

As an additional component for our `CosmoPower` extension, we provide wrapper functionality that interfaces the basic `CosmoPower` functionality with the inference software packages `CosmoSIS` and `Cobaya`. Because most of the emulator specification will be present in the packaging prescription file, interfacing these emulators with the sampling software is as simple as pointing the wrapper to a packaging file. The remaining interfacing is then provided for with these wrappers. We will show here how to interface the emulators with `CosmoSIS` and `Cobaya`, and show that these wrappers, with the emulators we have described in the previous section, can recover parameter constraints equivalent to those recovered with the original Einstein-Boltzmann code.

4.1 CosmoSIS

The wrapper for using CosmoPower in CosmoSIS inference pipelines involves specifying the CosmoPower module in the usual way in the ini file:

```
[cosmopower]
file = path/to/interface/cosmopower_interface.py
package_file = /path/to/packaging/
    package_prescription.yaml
extra_renames = {'cosmosis_parameter_name' :
                        'network_parameter_name'}
```

The options available and their default values for the module are specified in its associated module.yaml. In particular we note that care should be taken with parameter naming conventions, with any necessary translations specified using the extra_renames keyword. The CosmoSIS wrapper allows for the use of CosmoPower to compute CMB and matter power spectra, and the background evolution and derived quantities also described in Section 2.1. If desired, it is also possible to use CosmoPower only to perform the computation of spectra from the perturbations, and the native Einstein-Boltzmann code for the (relatively) faster background calculations (e.g. by only requesting the CMB from CosmoPower and including a CAMB module with mode = background). Here we note that caution should be taken to not generate inconsistent results through inconsistent choices of CAMB parameters when running in this mode.

4.2 Cobaya

When CosmoPower is installed, the wrapper for using it in Cobaya can be used by simply adding the cosmopower block to the Cobaya configuration file. This is similar to how one normally adds CAMB or class as their Einstein-Boltzmann code. Due to the new interface using the packaging prescription, the CosmoPower wrapper requires minimal settings, and a full block can look as simple as:

```
cosmopower:
  root_dir: /path/to/packaging
  package_file: package_prescription.yaml
```

Here, the (optional) root_dir keyword points the wrapper to the root directory where the packaging file is saved, and the package_file option points to the packaging prescription file that you want to load in. From this point, the wrapper parses the packaging prescription, interfaces with Cobaya, loads in the emulators that are required to compute all desired quantities, and provides the likelihoods with the computed quantities during the chain sampling.

4.3 Fall through to native Einstein-Boltzmann code

In order to increase the robustness of the use of CosmoPower emulators, we also include a feature which allows a given evaluation to 'fall through' to the native Einstein-Boltzmann code, in a limited and configurable set of circumstances. By specifying the fall_through = True option in the wrapper being used, CosmoPower will check that a python module corresponding to the emulated_code and version can be imported. If so, then if a set of parameters is requested by the sampler which is outside of the trained range of the emulator specified in the parameters block (e.g. if the prior being used is wider than the training range) then CosmoPower will give a warning, but also calculate the requested quantities using the native Einstein-Boltzmann code. Whilst this may be desirable in a limited set of circumstances, care should be taken that the expected computational cost does not overwhelm that of augmenting the training set with a broader range of parameters and re-training the emulator.

Parameter	Fiducial Value
$\Omega_b h^2$	2.2383×10^{-2}
$\Omega_c h^2$	12.011×10^{-2}
H_0	67.32 km/s/Mpc
n_s	0.966
$\log(10^{10} A_s)$	3.0448
τ	5.43×10^{-2}
A_b	3.13
η_b	0.603
$\log T_{\rm AGN}$	7.8
Σm_ν	0.12eV

Table 3. The fiducial parameters used for generating the smooth data vector. The first six parameters refer to the cosmology, while the middle three are the baryonic feedback parameters used in the non-linear model of CAMB. The last parameter is specific for the extension model we tested, with a neutrino mass for the inverted hierarchy to ensure that we could recover a closed posterior for our $+\Sigma m_\nu$ emulators. The remaining accuracy settings are the same as in Figure 3.

5 COMPARISON OF RECOVERED COSMOLOGY

We now demonstrate that we can use our emulators in parameter inference analysis, generating posterior samples using Monte Carlo chains with each of the Cobaya and CosmoSIS wrappers above using the same packaged network. In order to utilise all of the output quantities we do this for a set of observables: primary CMB, CMB lensing, galaxy weak lensing, and galaxy clustering. Note that this allows for quick and easy cross-validation of the results from using different Einstein-Boltzmann codes between different inference packages (e.g., class in CosmoSIS and CAMB in Cobaya). This is particularly important because leading cosmology collaborations adopt different combinations of these codes while releasing results which we want to compare and combine.

5.1 Simulated data vectors

For full validation, it is important to check that not only the emulators recover the cosmological observables to high accuracy, but also that there is no inherent bias when using our emulators for estimation of the final cosmological parameters. To do this, we can generate simulated data for the observables we emulate with a theoretical covariance matrix and perform a parameter inference analysis on them using the wrappers described above.

5.1.1 Cosmic-variance-limited CMB data

For our testing purposes, we generate a smooth data vector with cosmic-variance-limited noise (such that our conclusions apply to all current and future experiments). This data vector contains data from a fiducial cosmology (see Table 3) for the CMB power spectra C_ℓ^{TT}, C_ℓ^{TE}, and C_ℓ^{EE}, as well as the lensing potential spectrum $C_\ell^{\phi\phi}$. For the CMB data vector, the cosmic-variance-limited noise model is similar to Section 2.5, with $N_\ell^{XX} = N_\ell^{XY} = 0$ for all combinations of XX and XY. We constrain our analysis to the multipole range $2 \leq \ell \leq 6000$. To explore the parameter space we add a log-likelihood function as a simple Gaussian chi-square distribution:

$$\log \mathcal{L} = -\frac{1}{2} \sum_{\ell} \left(\frac{C_\ell^{\text{pred}} - C_\ell^{\text{data}}}{\sigma_\ell} \right)^2 . \tag{3}$$

Since the data vector is smooth, we expect to recover the exact input parameters with a final $\chi^2 = 0$.

5.1.2 *Stage-IV-like 3x2pt LSS data*

We also simulate a Large Scale Structure dataset for demonstrating and validating the $P(k)$ emulators. This consists of 3x2pt data for cosmic shear, galaxy clustering and galaxy-galaxy lensing, as is typically constrained by experiments such as DES, HSC and KiDS+BOSS+2dFLens. Here we approximate the constraining power of a Stage-IV LSS survey (such as LSST or *Euclid*), with a number of caveats. In order to be able to make use of existing theoretical modelling and likelihoods which are implemented in *both* `Cobaya` and `CosmoSIS` we use real space data rather than power spectra and set up the redshift and angular binning of the data to be the same as the DES-Y1 configuration, as described in Abbott et al. (2018). Likewise, we both simulate and model the data using the DES-Y1 model for Intrinsic Alignments, linear galaxy bias, shear and redshift calibration biases etc. For a covariance matrix we create a Gaussian covariance using the `save_2pt` module of `CosmoSIS`. We do not contend such a model will be accurate for describing real Stage-IV data; here we are seeking to understand if differences between the calculation of $P(k)$ with either `CosmoPower` or `CAMB` can be detected when 3x2pt statistics are measured with Stage-IV precision. To that end we assume a sky fraction, redshift distribution, total galaxy number density and shape noise as appropriate for an LSST-Y10 3x2pt survey (as specified in the LSST-SRDC by Mandelbaum et al. 2018) when simulating and analysing the data. Full details of the configuration are given in Appendix D.

5.2 Results

Figure 9 shows the recovered contours of `Cobaya+CosmoPower` and `CosmoSIS+CosmoPower` versus the `Cobaya+CAMB` and `CosmoSIS+CAMB` posteriors from a CMB cosmic-variance-limited dataset. We show that we can reproduce the `CAMB` best-fit cosmology and posterior distribution to $< 0.1\sigma$ of the cosmic variance limit error bars in both inference codes. Figure 10 shows the same result within `Cobaya` for the $+\sum m_\nu$ emulator as an example for an extended model.

The main advantage from running `CosmoPower` is the speed increase over `CAMB`. For a simple ΛCDM model and the cosmic-variance-limited CMB data, we found that a `CAMB` chain took $\sim$ 10 hours, while for `CosmoPower` it takes only $\sim$ 20 minutes to run to convergence. Most of this speed-up comes from the fact that at this level of accuracy, an evaluation of a `CAMB` power spectrum takes $\sim 20s$ to compute, while the same computation takes `CosmoPower` $\sim 0.1s$, at which point computing any non-trivial likelihood function becomes the limiting factor. When going to beyond-ΛCDM models, the time it takes to run a `CAMB` chain will go up due to the increased complexity or accuracy requirements from the computations. For `CosmoPower` however, the pre-trained emulators do not require more complicated computation when running these chains, and as such the time it takes a `CosmoPower` chain to converge will only increase slightly due to the larger parameter space that needs to be explored.

Similarly for the Stage-IV-like LSS data we find times for each individual likelihood evaluation with the `CosmoPower CosmoSIS` module to be $\sim$ 0.5 seconds, compared to $\sim$ 42 seconds for the `CAMB` `CosmoSIS` module. In this case the need for Limber integration dominates the likelihood evaluation time for `CosmoSIS` ($\sim$ 2 seconds) when the Boltzmann emulator is used. Rather than expending significant computational expense on a fully converged `CAMB` chain, in Figure 11 we show the log Posterior values calculated in a short chain using both `CAMB` and `CosmoPower` within `CosmoSIS` for the LSS data set described in Section 5.1.2. As can be seen the relative differences in log Posterior between the numerical code and the emulator are less than 0.005%, representing an indistinguishable difference in estimates of posterior credible intervals and summary statistics. See Figure D1 for full estimated posteriors showing the parameter constraining power of this data set.

6 CONCLUSIONS

We have presented a coherent framework for specifying, creating, packaging and utilising emulators of cosmological Einstein-Boltzmann codes, building on the `CosmoPower` package. These emulators can speed up by orders of magnitude the estimation of posteriors on cosmological and nuisance parameters from experimental data and hence enable investigation of models which extend the fiducial ΛCDM cosmology and the checking of the robustness of any conclusions made to a plethora of modelling choices. By creating a specification for packaging and distributing such emulators and providing wrappers for their use in popular inference packages we hope to improve efficiency and reproducibility in cosmological studies, by allowing appropriate emulators to be widely used by many different studies once they have been trained. This kind of reproducibility across platforms will also assist in combining different data sets to improve statistical constraining power and investigate more models in more detail.

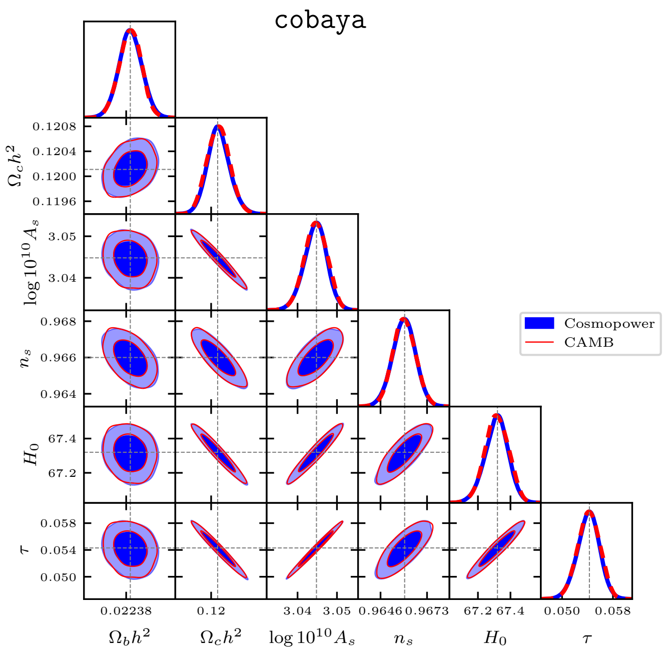
We have used the framework to produce a suite of emulators of quantities calculated by `CAMB` v1.5.0: CMB primary angular power spectra $C_\ell^{TT}, C_\ell^{TE}, C_\ell^{EE}, C_\ell^{BB}$; CMB lensing power spectra $C_\ell^{\phi\phi}$; linear and non-linear matter power $P(k)_{\text{lin}}$, $P(k)_{\text{NL}}$ and a variety of background and derived quantities. We have demonstrated the accuracy of the emulators at both the spectrum level and the parameter-recovery level to accuracy appropriate for Stage-IV data (and beyond to the cosmic variance limit for the CMB spectra).

In principle, this standardisation of emulator packaging extends in scope beyond Einstein-Boltzmann codes to other numerically-intensive codes amenable to emulation, such as Interstellar Medium models (e.g. Palud et al. 2023), supernova radiative transfer (e.g. Kerzendorf et al. 2021), early-Universe re-ionisation models (e.g. Schmit & Pritchard 2018) and others.

The framework described here will form a new release of the `CosmoPower` code, with the website `https://alessiospuriomancini.github.io/cosmopower/` providing full API documentation and extensive demo scripts and tutorial notebooks.

ACKNOWLEDGEMENTS

HTJ, IH and EC acknowledge support from the European Research Council (ERC) under the European Union's Horizon 2020 research



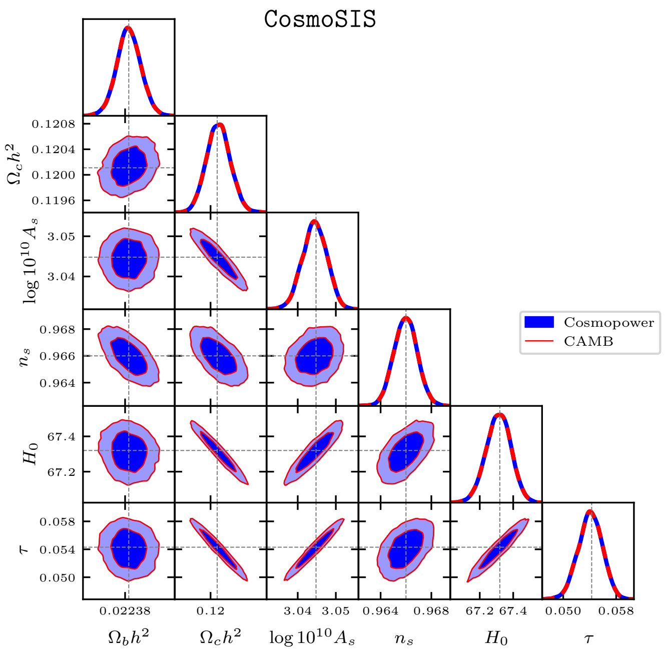


Figure 9. To illustrate that we can estimate posteriors in both `Cobaya` and `CosmoSIS` we show the same 68% and 95% confidence levels for ΛCDM parameters from CMB cosmic-variance-limited power spectra, obtained from a full MCMC run done either with the `Cobaya` wrapper for `CosmoPower` (**blue**) or with the `CAMB` (**red**) on the *left* for `Cobaya` and *right* for `CosmoSIS`. This Figure also demonstrates the correct recovery of the cosmological likelihood in each case (note that for `Cobaya` two separate sets of posterior samples are taken, whilst for `CosmoSIS` we re-evaluate the likelihood at the same posterior samples, resulting in visually identical contours).

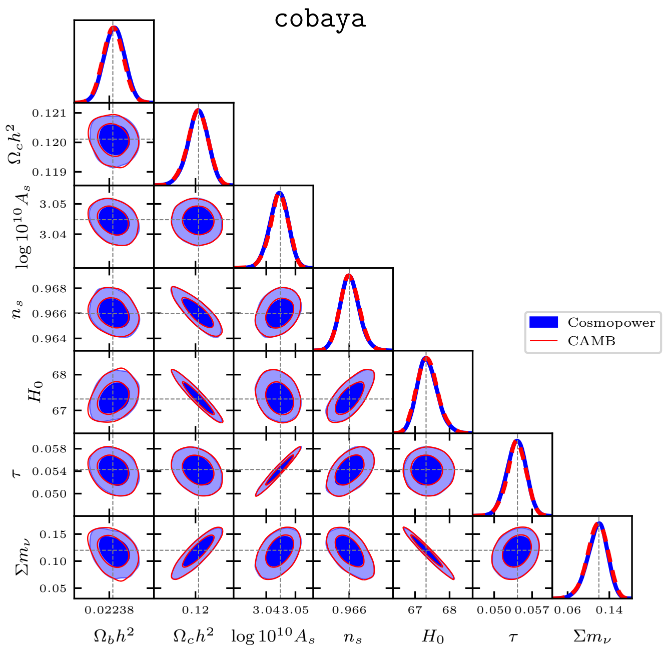


Figure 10. Similar to Figure 9 but for ΛCDM $+\Sigma m_\nu$: *Left:* 68% and 95% confidence levels for ΛCDM parameters from CMB cosmic-variance-limited power spectra, obtained from a full MCMC run done either with the `Cobaya` wrapper for `CosmoPower` (**blue**) or with the existing `CAMB` wrapper for `Cobaya` (**red**). The dotted lines show the fiducial value of the input data vector, and both posterior distributions recovered this fiducial value within $< 0.1\sigma$. The `CosmoPower` sampler converged within ~ 100 minutes, while the `CAMB` sampler converged after ~ 28 hours.

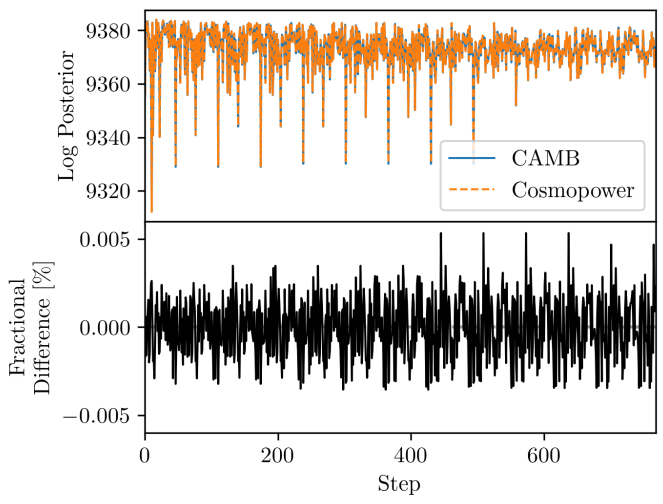


Figure 11. Log Posterior differences for the Stage-IV-like 3x2pt LSS data set described in Section 5.1.2 between estimations made using the original `CAMB` Boltzmann code and the `CosmoPower` emulator.

and innovation programme (Grant agreement No. 849169). ASM acknowledges support from the MSSL STFC Consolidated Grant ST/W001136/1. We acknowledge the support of the Supercomputing Wales project, which is part-funded by the European Regional Development Fund (ERDF) via Welsh Government. We thank Antony Lewis for input on precision parameters and support with `Cobaya`; Jens Chluba and Yacine Ali-Haïmoud for discussions on recombination codes; and Joe Zuntz for discussions on `CosmoSIS`. In addition to the references in the main text we thank the authors and maintainers of public software codes including `NumPy` (Harris et al.

2020), `SciPy` (Virtanen et al. 2020), `matplotlib` (Hunter 2007), `TensorFlow` (Abadi et al. 2015), and `GetDist` (Lewis 2019).

Author contributions

We list here the roles and contributions of the authors according to the Contributor Roles Taxonomy (CRediT)[21].
Hidde T. Jense: Conceptualization (equal), Investigation (equal), Methodology (equal), Software (lead), Validation (equal), Visualization (lead), Writing - original draft (equal). **Ian Harrison**: Conceptualization (equal), Investigation (equal), Methodology (equal), Software (supporting), Supervision (supporting), Validation (equal), Visualization (supporting), Writing - original draft (equal). **Erminia Calabrese**: Conceptualization (equal), Methodology (supporting), Supervision (lead), Visualization (supporting), Writing - original draft (equal). **Alessio Spurio Mancini**: Conceptualization (equal), Methodology (equal), Writing - original draft (supporting). **Boris Bolliet**: Conceptualization (equal), Writing - original draft (supporting). **Jo Dunkley**: Writing - original draft (supporting). **J. Colin Hill**: Conceptualization (supporting), Writing - original draft (supporting).

REFERENCES

Abadi M., et al. 2015, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, https://www.tensorflow.org/
Abbott T. M. C., et al., 2018, Phys. Rev. D, 98, 043526
Abbott T. M. C., et al., 2022, Phys. Rev. D, 105, 023520
Adame A. G., et al., 2024, preprint (arXiv:2404.03002)
Aghamousa A., et al., 2016, preprint (arXiv:1611.00036)
Aiola S., et al. 2020, Journal of Cosmology and Astroparticle Physics, 2020, 047–047
Alam S., et al., 2021, Phys. Rev. D, 103, 083533
Aricò G., Angulo R. E., Zennaro M., 2021, preprint (arXiv:2104.14568)
Audren B., et al. 2013, JCAP, 1302, 001
Balkenhol L., et al., 2022, preprint (arXiv:2212.05642)
Blas D., Lesgourgues J., Tram T., 2011, Journal of Cosmology and Astroparticle Physics, 2011, 034
Bolliet B., et al. 2023, preprint (arXiv:2303.01591)
Bonici M., Bianchini F., Ruiz-Zapatero J., 2023, preprint (arXiv:2307.14339)
Brinckmann T., Lesgourgues J., 2019, Phys. Dark Univ., 24, 100260
Burger P. A., et al., 2023, Astron. Astrophys., 669, A69
Burger P. A., et al. 2024, Astronomy &amp; Astrophysics, 683, A103
CMB-S4 Collaboration 2016, preprint (arXiv:1610.02743)
Campagne J.-E., et al. 2023, The Open Journal of Astrophysics, 6
Carrion K., et al. 2024, preprint (arXiv:2402.18562)
Chluba J., Thomas R. M., 2010, Monthly Notices of the Royal Astronomical Society, pp no–no
Chluba J., Vasil G. M., Dursi L. J., 2010, Monthly Notices of the Royal Astronomical Society, 407, 599–612
Choi S. K., et al., 2020, JCAP, 12, 045
DESI Collaboration 2024, preprint (arXiv:2404.03002)
Doré O., et al., 2014, preprint (arXiv:1412.4872)
Eifler T., et al., 2021, Mon. Not. Roy. Astron. Soc., 507, 1746
Farren G. S., et al. 2023, preprint (arXiv:2311.04213)
Giardiello S., et al., 2024, preprint (arXiv:2403.05242)
Harris C. R., et al., 2020, Array Programming with NumPy, doi:10.1038/s41586-020-2649-2, https://arxiv.org/abs/2006.10256v1
Heydenreich S., et al. 2023, Astron. Astrophys., 672, A44
Heymans C., et al., 2021, Astron. Astrophys., 646, A140
Hill J. C., et al. 2022, Phys. Rev. D, 105, 123536
Hunter J. D., 2007, Computing in Science and Engineering, 9, 90
Kerzendorf W. E., et al. 2021, ApJ, 910, L23
Lesgourgues J., 2011, preprint (arXiv:1104.2932)
Lewis A., 2019, preprint (arXiv:1910.13970)
Lewis A., Challinor A., Lasenby A., 2000, ApJ, 538, 473
Linke L., et al. 2023, Astron. Astrophys., 672, A185
Madhavacheril M. S., et al., 2024, Astrophys. J., 962, 113
Mandelbaum R., et al., 2018, preprint (arXiv:1809.01669)
Mauland R., Winther H. A., Ruan C.-Z., 2023, preprint (arXiv:2309.13295)
McCarthy F., Hill J. C., Madhavacheril M. S., 2022, Phys. Rev. D, 105, 023517
Mead A. J., et al. 2021, Monthly Notices of the Royal Astronomical Society, 502, 1401–1422
Miyatake H., et al., 2023, Phys. Rev. D, 108, 123517
Mootoovaloo A., et al. 2022, Astron. Comput., 38, 100508
More S., et al., 2023, Phys. Rev. D, 108, 123520
Moretti C., et al. 2023, JCAP, 12, 025
Nygaard A., et al. 2023, JCAP, 05, 025
Palud P., et al. 2023, A&A, 678, A198
Pan Z., et al., 2023, Phys. Rev. D, 108, 122005
Piras D., Spurio Mancini A., 2023, arXiv e-Prints
Pitrou C., et al. 2018, Submitted to Phys. Rept.
Planck Collaboration VI 2020, A&A, 641, A6
Qu F. J., et al. 2024a, preprint (arXiv:2404.16805)
Qu F. J., et al., 2024b, Astrophys. J., 962, 112
Reeves A., et al. 2024, JCAP, 01, 042
Scaramella R., et al., 2022, Astron. Astrophys., 662, A112
Schmit C. J., Pritchard J. R., 2018, Mon. Not. Roy. Astron. Soc., 475, 1213
Simons Observatory Collaboration 2019, Journal of Cosmology and Astroparticle Physics, 2019, 056–056
Spurio Mancini A., Bose B., 2023, preprint (arXiv:2305.06350)
Spurio Mancini A., Pourtsidou A., 2022, Mon. Not. Roy. Astron. Soc., 512, L44
Spurio Mancini A., et al. 2022, Monthly Notices of the Royal Astronomical Society, 511, 1771
Sugiyama S., et al., 2023, Phys. Rev. D, 108, 123521
The Simons Observatory collaboration 2019, Journal of Cosmology and Astroparticle Physics, 2019, 056
Torrado J., Lewis A., 2019, Cobaya: Bayesian analysis in cosmology ([ascl:1910.019]), https://ascl.net/1910.019
Torrado J., Lewis A., 2021, Journal of Cosmology and Astroparticle Physics, 2021, 057
Virtanen P., et al., 2020, Nature Methods, 17, 261
Zuntz J., et al. 2015, Astron. Comput., 12, 45

APPENDIX A: FULL ΛCDM EMULATOR PRESCRIPTION

Here we present the full yaml prescription for our ΛCDM emulators.

```yaml
network_name: jense_2023_camb_lcdm
path: jense_2023_camb_lcdm

# Details on the boltzmann code we emulate
emulated_code:
  name: camb
  version: "1.5.0"
  inputs: [ ombh2, omch2, As, ns, H0, tau ]
  extra_args:
    lens_potential_accuracy: 8
    kmax: 10.0
    k_per_logint: 130
    lens_margin: 2050
    AccuracyBoost: 1.0
    lAccuracyBoost: 1.2
    lSampleBoost: 1.0
    DoLateRadTruncation: false
    min_l_logl_sampling: 6000
```

[21] https://credit.niso.org/

```yaml
    recombination_model: CosmoRec

# Details on the parameters we sample and derive.
samples:
  Ntraining: 100000

  parameters:
    # Our latin hypercube
    ombh2: [0.015,0.030]
    omch2: [0.09,0.15]
    logA: [2.5,3.5]
    tau: [0.02, 0.20]
    ns: [0.85, 1.05]
    h: [0.4,1.0]
    # Parameters derived directly from our LHC
    H0: "lambda h: h * 100.0"
    As: "lambda logA: 1.e-10 * np.exp(logA)"
    # Parameters computed by our boltzmann code
    thetastar:
    sigma8:
    YHe:
    zrei:
    taurend:
    zstar:
    rstar:
    zdrag:
    rdrag:
    N_eff:

# Details on each of the emulators we want to create.
networks:
  - quantity: "derived"
    type: NN
    n_traits:
      n_hidden: [ 512, 512, 512, 512 ]
    training:
      validation_split: 0.1
      learning_rates: [ 1.e-2, 1.e-3, 1.e-4, 1.e-5, 1.e-6, 1.e-7 ]
      batch_sizes: [ 1000, 2000, 5000, 10000, 20000, 50000 ]
      gradient_accumulation_steps: [ 1, 1, 1, 1, 1, 1 ]
      patience_values: [ 100, 100, 100, 100, 100, 100 ]
      max_epochs: [ 1000, 1000, 1000, 1000, 1000, 1000 ]

  - quantity: "Cl/tt"
    type: NN
    log: True
    modes:
      label: l
      range: [2,10000]
    n_traits:
      n_hidden: [ 512, 512, 512, 512 ]
    training:
      validation_split: 0.1
      learning_rates: [ 1.e-2, 1.e-3, 1.e-4, 1.e-5, 1.e-6, 1.e-7 ]
      batch_sizes: [ 1000, 2000, 5000, 10000, 20000, 50000 ]
      gradient_accumulation_steps: [ 1, 1, 1, 1, 1, 1 ]
      patience_values: [ 100, 100, 100, 100, 100, 100 ]
      max_epochs: [ 1000, 1000, 1000, 1000, 1000, 1000 ]

  - quantity: "Cl/te"
    type: PCAplusNN
    modes:
      label: l
      range: [2,10000]
    p_traits:
      n_pcas: 512
      n_batches: 10
    n_traits:
      n_hidden: [ 512, 512, 512, 512 ]
    training:
      validation_split: 0.1
      learning_rates: [ 1.e-2, 1.e-3, 1.e-4, 1.e-5, 1.e-6, 1.e-7 ]
      batch_sizes: [ 1000, 2000, 5000, 10000, 20000, 50000 ]
      gradient_accumulation_steps: [ 1, 1, 1, 1, 1, 1 ]
      patience_values: [ 100, 100, 100, 100, 100, 100 ]
      max_epochs: [ 1000, 1000, 1000, 1000, 1000, 1000 ]

  - quantity: "Cl/ee"
    type: NN
    log: True
    modes:
      label: l
      range: [2,10000]
    n_traits:
      n_hidden: [ 512, 512, 512, 512 ]
    training:
      validation_split: 0.1
      learning_rates: [ 1.e-2, 1.e-3, 1.e-4, 1.e-5, 1.e-6, 1.e-7 ]
      batch_sizes: [ 1000, 2000, 5000, 10000, 20000, 50000 ]
      gradient_accumulation_steps: [ 1, 1, 1, 1, 1, 1 ]
      patience_values: [ 100, 100, 100, 100, 100, 100 ]
      max_epochs: [ 1000, 1000, 1000, 1000, 1000, 1000 ]

  - quantity: "Cl/bb"
    type: NN
    log: True
    modes:
      label: l
      range: [2,10000]
    n_traits:
      n_hidden: [ 512, 512, 512, 512 ]
    training:
      validation_split: 0.1
      learning_rates: [ 1.e-2, 1.e-3, 1.e-4, 1.e-5, 1.e-6, 1.e-7 ]
      batch_sizes: [ 1000, 2000, 5000, 10000, 20000, 50000 ]
      gradient_accumulation_steps: [ 1, 1, 1, 1, 1, 1 ]
      patience_values: [ 100, 100, 100, 100, 100, 100 ]
      max_epochs: [ 1000, 1000, 1000, 1000, 1000, 1000 ]

  - quantity: "Cl/pp"
    inputs: [ ombh2, omch2, logA, ns, h ]
    type: PCAplusNN
    log: True
    modes:
      label: l
      range: [2,10000]
    p_traits:
      n_pcas: 64
      n_batches: 10
    n_traits:
      n_hidden: [ 512, 512, 512, 512 ]
    training:
      validation_split: 0.1
      learning_rates: [ 1.e-2, 1.e-3, 1.e-4, 1.e
        -5, 1.e-6, 1.e-7 ]
      batch_sizes: [ 1000, 2000, 5000, 10000,
        20000, 50000 ]
      gradient_accumulation_steps: [ 1, 1, 1, 1,
        1, 1 ]
      patience_values: [ 100, 100, 100, 100, 100,
        100 ]
      max_epochs: [ 1000, 1000, 1000, 1000, 1000,
        1000 ]
```

APPENDIX B: DATASET FILE STRUCTURE

We opted to standardise the dataset file structure for `CosmoPower`, as a way to streamline the emulator building process for the end-user. At the `python`-interface side, we included a `cosmpower.Dataset` class that wraps around the file structure easily and handles the file parsing in a safe manner.

The main file format we settled on is Hierarchical Data Format revision 5 (HDF5), which is a file format designed to handle large datasets of tabular nature, something that lends itself specially well for this issue. Via the `h5py` library in `python`, HDF5 is also a relatively fast and memory-efficient read/write access, offering both good compression for hard drive storage and decompression rates for RAM access during runtime.

The training data needs to accurately match the $\vec{d}(\vec{\theta})$ mapping of our emulators well, while also being robust against potentially missing datapoints and multi-threaded reading access. We opted to split this mapping into two different files, a `parameters` file which contains the main LHC of the dataset and is only used for spectra generation, and a (set of) files for the computed observable quantities, which are named as `Cl_tt.0.hdf5`, `Cl_tt.1.hdf5`, etc. for e.g. C_ℓ^{TT}. The quantity files are split into several files, to allow multi-threaded write access without having to worry about data races, and to prevent issues when opening data files which are larger than a device's available RAM.

The `parameters` file contains a header and a main body. The header contains an ordered list of strings for the p parameters that are to be passed on to the Boltzmann code. The main body contains a $p \times N$ table of N samples from the LHC. Because the LHC is relatively small in size and quick to generate, this file never needs to be written to in different threads and can be kept as one file. It is stored separately from the main dataset in case a spectra generation run is interrupted and needs to be resumed at a later stage, in which case it can be ensured that new spectra are sampled from the same LHC as before.

Each quantity file also contains a header and a main body. The header contains a list of M modes for the quantity, the names of the parameters that are to be used for the emulator. In the main body, there is a $p \times N$ array of input parameters for each spectra, and a $M \times N$ array where each M-length spectrum is stored. In addition, there is a N-array of indices stored, the entries of which refer to the indices of the `parameters` file that each sample was computed from. Because quantity files are pre-allocated before they are filled, an index of -1 indicates that a spectrum has not been computed yet.

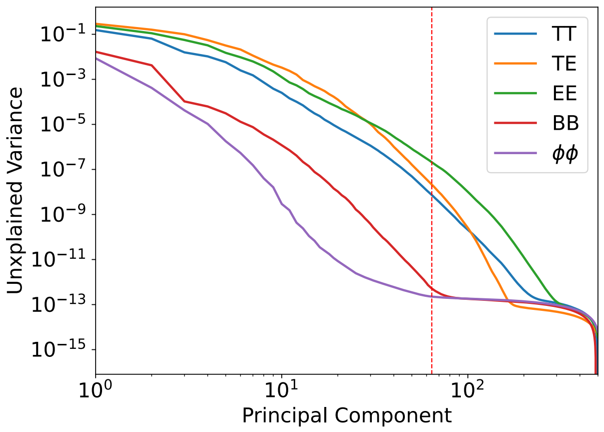


Figure C1. A scree plot, showing the *unexplained variance* of a PCA compression for the various CMB quantities, as a function of the number of retained principal components. The "scree" of each line is the flat plateau of each line. We observed that for $C_\ell^{\phi\phi}$, this scree lies around 64 principal components (vertical red line), and hence a PCA compression of 64 retained components is effective for a $C_\ell^{\phi\phi}$ emulator. Conversely, however, observing the scree for C_ℓ^{BB} at around 100 principal components, we expected the same to see for this quantity. We attribute the lack of an improvement in emulation for this quantity to the presence of important features which shift in ℓ-space for that quantity, which would not be retained by our implementation of PCA.

APPENDIX C: PRINCIPAL COMPONENT ANALYSIS

The use of Principal Component Analysis (PCA) can be worthwhile in improving the accuracy of the emulator by compressing the full data into a smaller number of free components. While the reduction in freedom in the output is reduced and has therefore less capacity to accurately recover the original spectra, the reduced dimensionality of the output vector means that the emulator can more efficiently train on this reproduction.

The choice of whether or not to use PCA is not trivial, and there is no simple test that can conclusively show that the use of PCA compression is guaranteed to be beneficial before training an emulator. While for some cases, like C_ℓ^{TE}, the use of a PCA is needed due to the zero-crossing of the observed quantity, it may not be obvious *a priori* that the use of a PCA can improve it for other quantities as well.

It was observed in Spurio Mancini et al. (2022) that the $C_\ell^{\phi\phi}$ emulator improved in accuracy when employing PCA compression. We observed that this can be explained by making a *scree* plot, which is a line plot of the eigenvalues of all retained PCA components. We show a scree plot of the training data for our ΛCDM emulators in Figure C1. By observing where this line flattens out (the "scree" of the line), one can estimate the amount of components that need to be retained in the PCA. For the $C_\ell^{\phi\phi}$ spectra, we found that this scree appears around 60 components, which means around 64 components should be sufficient to accurately decompose the 10000 ℓ modes of the spectra without loss of information. Similarly, a scree plot showed that a few hundred components should be sufficient for C_ℓ^{TE}.

However, a scree plot is not necessarily conclusive. We observed that the C_ℓ^{BB} are also dense enough that about 200 PCA components should be capable of accurately recovering them. Upon training such an emulator however, we found that direct emulation of C_ℓ^{BB} was more accurate than one that employed PCA compression. We think this is due to the fact that the BB spectra contain features which vary in ℓ under certain parameter variations, and hence cannot be properly accounted for in PCA compression. Since our regular emulators were shown to be more than accurate for physical analysis, we did not do an in-depth analysis of this discrepancy. Further investigation, or a different type of information compaction that does allow for horizontal shifts in ℓ-space, can perhaps allow for more accurate emulators in the future.

APPENDIX D: SPECIFICATION OF STAGE-IV-LIKE 3X2PT DATA

For assessing the accuracy of our emulation of $P(k)$ at Stage-IV levels of precision on LSS data, we create a data set containing angular correlation functions for galaxy clustering $w(\theta)$, galaxy-galaxy lensing $\gamma_t(\theta)$, and cosmic shear $\xi_\pm(\theta)$. In addition to the fiducial cosmological model and parameters for ΛCDM shown in Table 3, we include linear galaxy bias parameters for the lens galaxies, a two-parameter NLA model for galaxy intrinsic alignments, one-parameter per tomographic bin central shift parameters for redshift distributions of the sources and lenses, and one parameter per tomographic bin for multiplicative shear bias calibration of the sources. Following the LSST-SRDC (Mandelbaum et al. 2018) specification for LSST-Y10 we assume a redshift distribution for both sources and lenses given by $n(z) \propto z^2 \exp\left[-(z/z_0)^\alpha\right]$ with $\alpha = 0.783$, $z_0 = 0.176$ and convolve this with a Gaussian of width $\sigma_z = 0.05(1+z)$. Sources are placed into four tomographic bins and lenses placed into five tomographic bins, all equally populated with the total number density of galaxies $n_{\rm gal} = 27$ [arcmin^{-2}] (note that this tomographic binning is not the one expected for the LSST analysis, but matches the DES-Y1 model). When modelling the covariance we assume a $\sigma_e = 0.26$ and a sky area of 14, 300 deg^2. In Figure 11 we show the ΛCDM model constraints from this data set (using `CosmoPower`), alongside the offical Dark Energy Survey Y1 results from Abbott et al. (2018) (which use the same model and likelihood pipeline) to give a sense of the relative power.

APPENDIX E: ACCURACY PLOTS FOR EXTENSION MODEL EMULATORS

Here we reproduce Figure 6 for the extended models we consider beyond ΛCDM, with all models showing acceptable levels of accuracy as discussed in Section 2.5.

This paper has been typeset from a TEX/LATEX file prepared by the author.

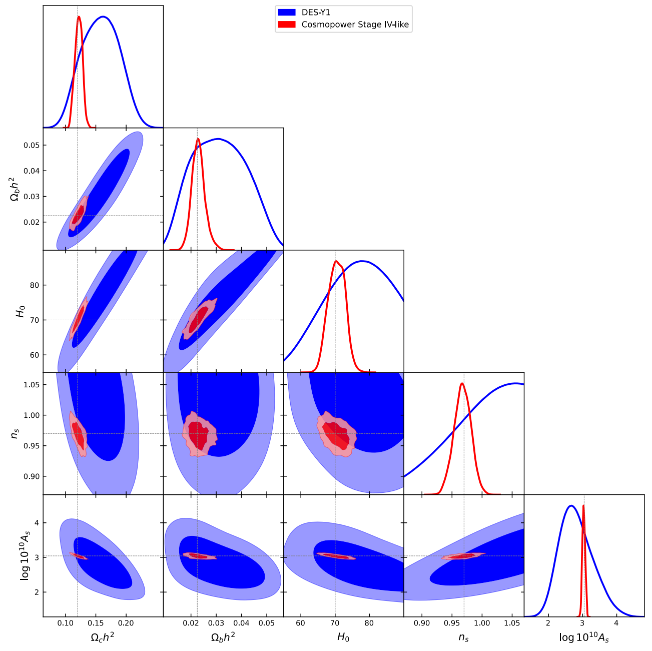


Figure D1. The ΛCDM model constraining power of the Stage-IV-like 3x2pt Large Scale Structure data set used to benchmark the trained $P(k)$ emulator. For scale we show the official DES-Y1 (Abbott et al. 2018) chain, which use the exact same likelihood pipeline but with their real data.

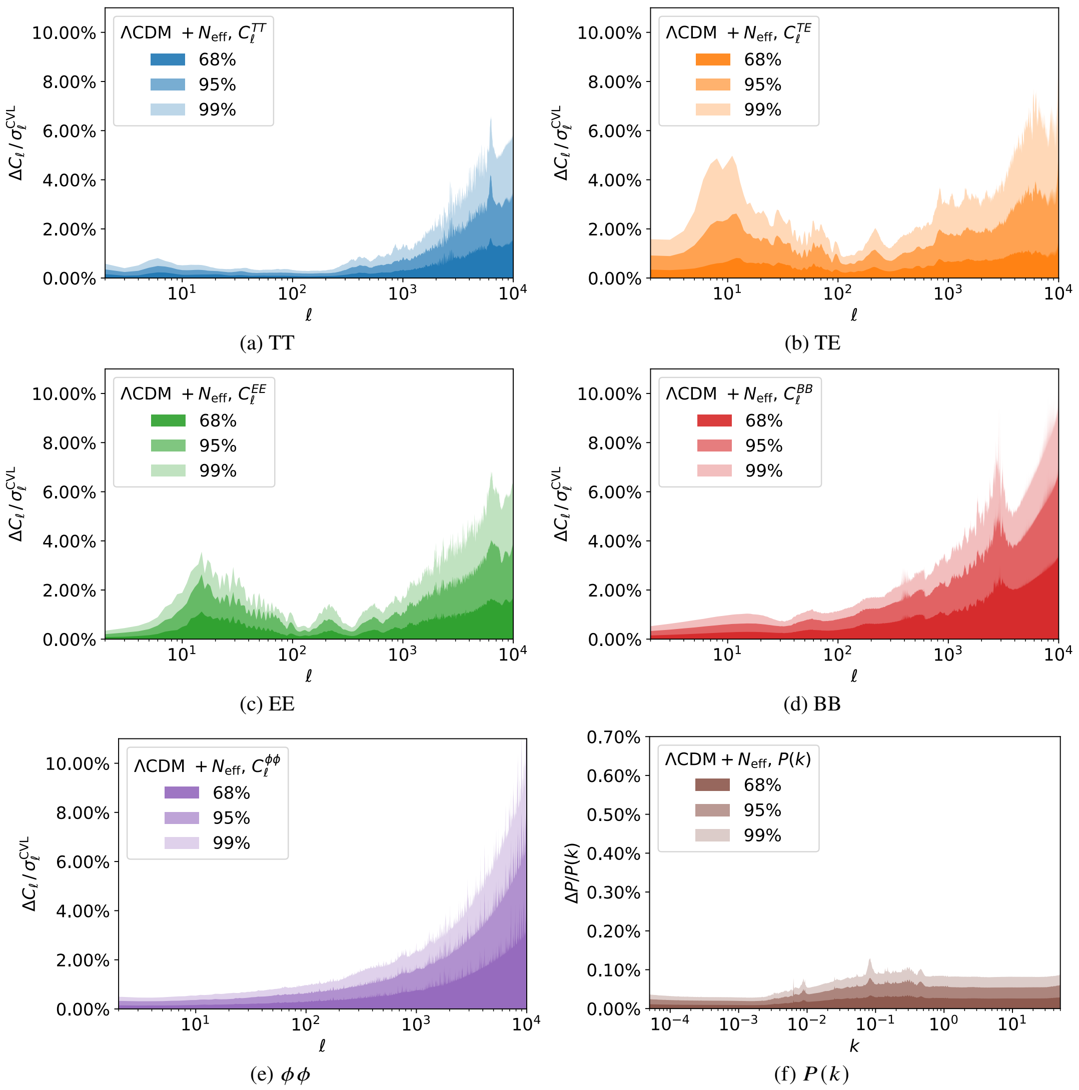
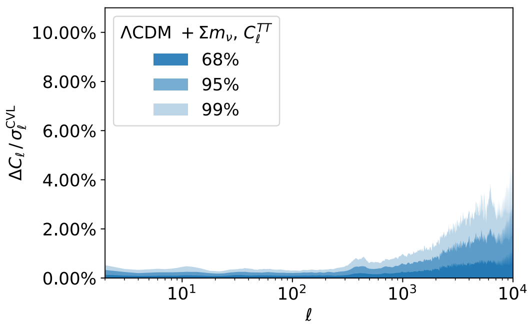
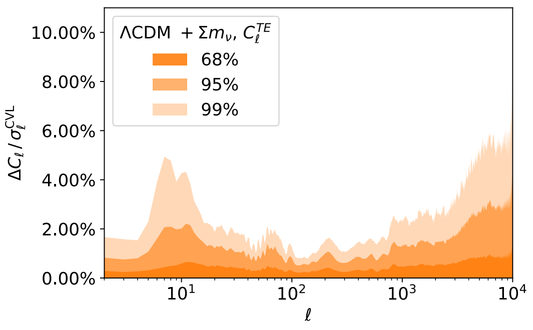


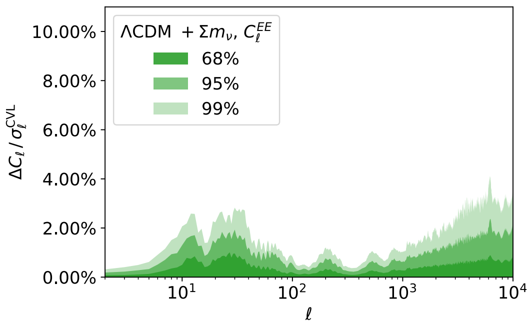
Figure E1. Same as Figure 6 but for ΛCDM + $N_{\rm eff}$.



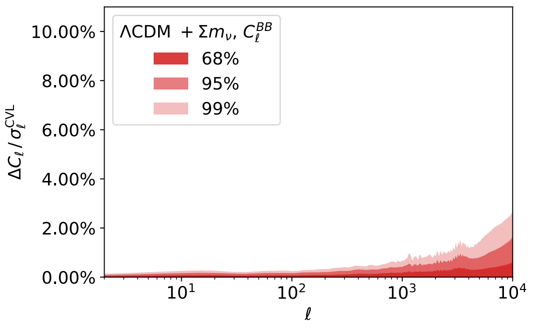
(a) TT



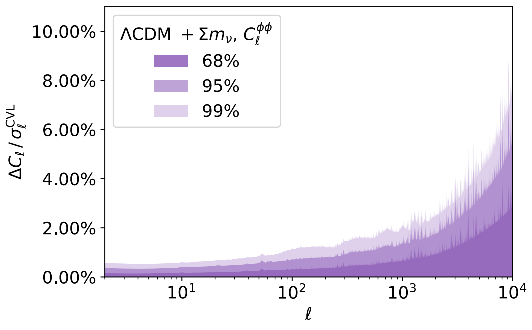
(b) TE



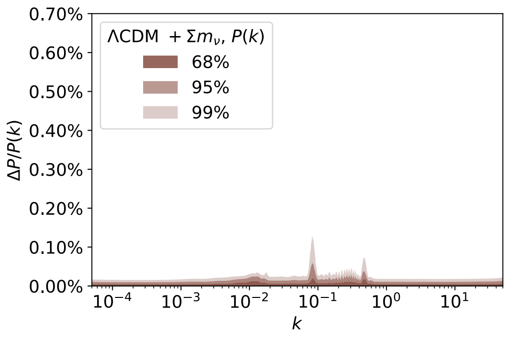
(c) EE



(d) BB

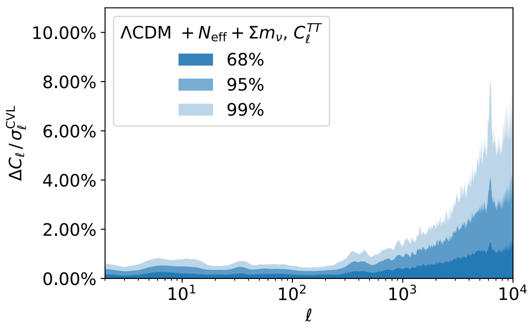


(e) $\phi\phi$

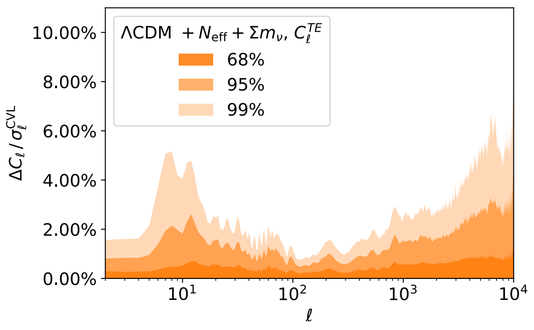


(f) $P(k)$

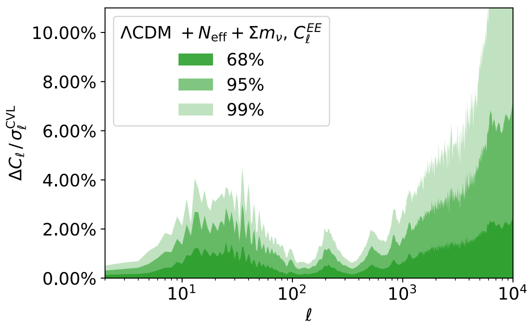
Figure E2. Same as Figure 6 but for $\Lambda\mathrm{CDM} + \Sigma m_\nu$.



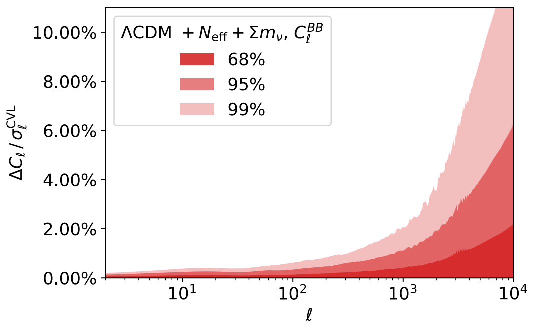
(a) TT



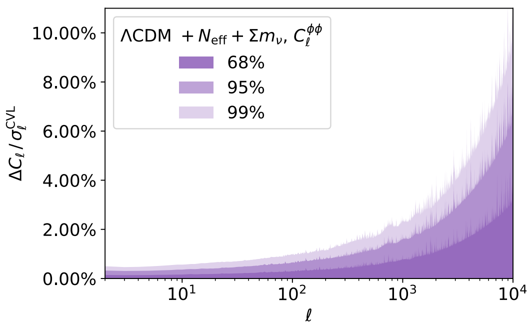
(b) TE



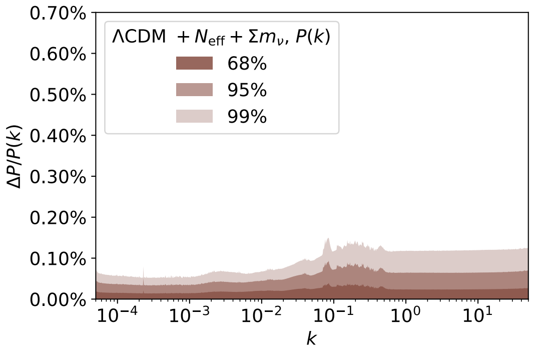
(c) EE



(d) BB



(e) $\phi\phi$



(f) $P(k)$

Figure E3. Same as Figure 6 but for ΛCDM $+ N_{\rm eff} + \Sigma m_\nu$.

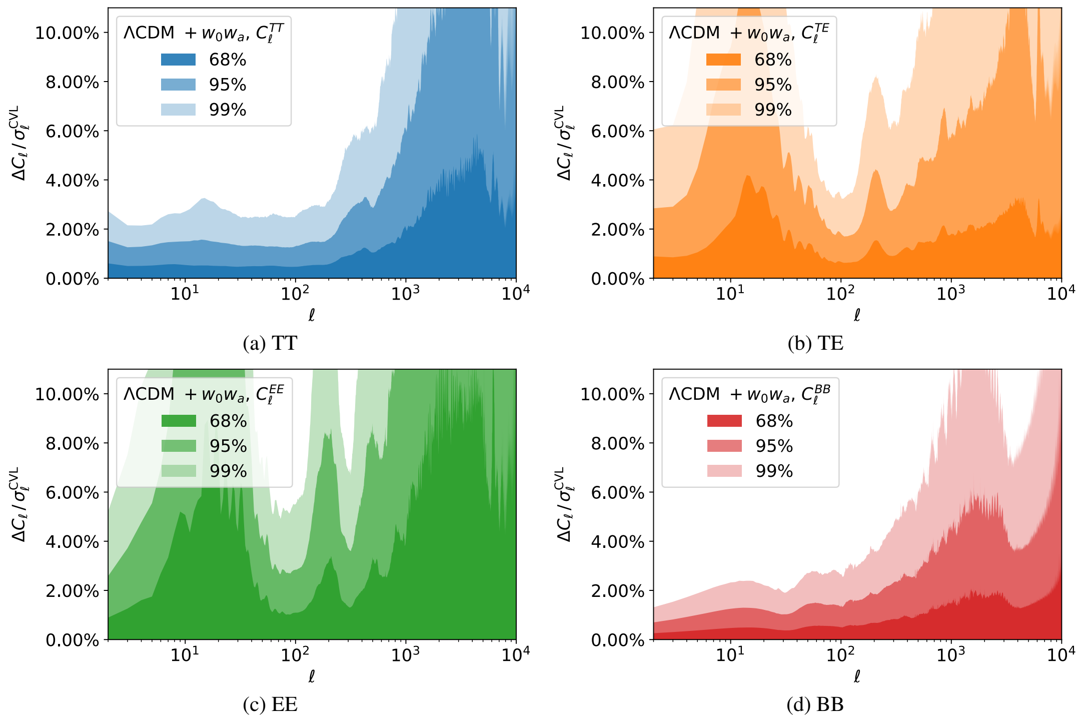


Figure E4. Same as Figure 6 but for $\Lambda\mathrm{CDM} + w_0 w_a$.