MNRAS **000**, 1–14 (2020) Preprint 8 December 2021 Compiled using MNRAS LATEX style file v3.0

Modelling stars with Gaussian Process Regression: Augmenting Stellar Model Grid

Tanda Li[1]★, Guy R. Davies[1]†, Alex Lyttle[1], Warrick Ball[1], Lindsey Carboneau[1], and A. N. Othe
[1] *School of Physics and Astronomy, University of Birmingham, Birmingham, B15 2TT, United Kingdom*

Accepted XXX. Received YYY; in original form ZZZ

ABSTRACT

Grid-based modelling is widely used for estimating stellar parameters. However, stellar model grid is sparse because of the computational cost. This paper demonstrates an application of the Gaussian Process (GP) Regression that turns a sparse model grid onto a continuous function. We train GP models to map five fundamental inputs (mass, equivalent evolutionary phase, initial metallicity, initial helium fraction, and the mixing-length parameter) to observable outputs (effective temperature, surface gravity, radius, surface metallicity, and stellar age). We test the GP predictions for the five outputs using off-grid stellar models and find no obvious systematic offsets, indicating good accuracy in predictions. As a further validation, we apply these GP models to characterise 1,000 fake stars. Inferred masses and ages determined with GP models well recover true values within one standard deviation. An important consequence of using GP-based interpolation is that stellar ages are more precise than those estimated with the original sparse grid because of the full sampling of fundamental inputs.

Key words: Star: Modelling – Machine Learning – Gaussian Process

1 INTRODUCTION

Theoretical stellar model has been developed for decades to simulate star structure and evolution. Star modelling is mostly grid-based (e.g. Choi et al. 2016) because computing many stellar models are time-consuming especially when a number of free input parameters are considered. Varying one of these adjusted parameters (mass, metallicity, helium fraction, mixing-length parameter, etc.) adds on an input demission and hence exponentially increases the computational cost.

A sparse grid is not ideal for the statistics analysis. Classical method like interpolation has been applied to overcome this disadvantage. For instance, Dotter (2016) developed a method to transform stellar evolution tracks onto a uniform basis and then interpolate to construct stellar isochrones. More recently, Rendle et al. (2019) uses Bayesian statistics and a Markov Chain Monte Carlo approach to find a representative set of interpolated models from a grid. The interpolation of both works achieve good accuracy for 3-demission girds (inputs are mass, age, and metallicity). However, this approach becomes less reliable for high-demission grid.

Machine learning is being applied to the field of stellar research in many ways to efficiently characterise stars. Verma et al. (2016) applied artificial neural network, which is a series of algorithms that endeavours to recognise underlying relationships in a set of data, to determine the evolutionary parameters of the sun and sun-like stars based on spectroscopic and seismic measurements. Using a similar artificial neural network interference, Hendriks & Aerts (2019) developed a method to provide the optimal starting point of model competitions for more detailed forward asteroseismic modelling. Moreover, Mombarg et al. (2021) trained neural networks to predict theoretical pulsation periods of high-order gravity modes, as well as the luminosity, effective temperature, and surface gravity for a given mass, age, overshooting parameter, diffusive envelope mixing, metallicity, and near-core rotation frequency. Using different machine learning tools, Bellinger et al. (2016) trained a random forest regressors(Ho 1995), which is an ensemble learning method for regression and operates by constructing a multitude of decision trees, to rapidly estimate fundamental parameters of solar-like stars based on classical and asteroseismic observations. Hon et al. (2018) developed a convolutional neural network classifier that analyses visual features in asteroseismic frequency spectra to distinguish between red giant branch stars and helium-core burning stars. Wu et al. (2019) determined masses and ages for massive RGB stars from their spectra with a machine-learning method based on kernel principal component analysis, which is a nonlinear form of principal component analysis using integral operator kernel functions and can efficiently compute principal components in high dimensional feature spaces related to input space by some nonlinear map (Schölkopf et al. 1997). Hon et al. (2020) applied the mixture density network (Bishop 1994), which learns a transformation from a set of input variables to a set of output variable, to determine stars' fundamental parameters like mass and age based on observed mode frequencies, spectroscopic and global seismic parameters.

★ E-mail: t.li.2@bham.ac.uk
† E-mail: G.R.Davies@bham.ac.uk

In above studies, the discriminative machine-learning model is mostly used. The discriminative model treats observables as given facts to directly infer star fundamental parameters. The method is efficient and easy for computation, while the downside is not allowing any priors for star properties like mass. In an opposite direction, the generative machine-learning model uses the star fundamental parameters as given facts to predict observables. This approach offers flexibility to prior fundamental parameters in the sampling. For instance, Lyttle et al. (2021) determined initial helium fraction and mixing-length parameters for a sample of Kepler dwarfs and subgiants with an artificial neural network to provide the generative model. This allowed them to prescribe prior distributions over the fundamental stellar parameters and, by extension, over population-level parameters such as a helium enrichment law. Priors encode our current knowledge and assumptions into inference from new data. This is especially important with noisy observations which span a large portion of parameter-space.

Constructing a comprehensive and fine model grid is computationally expensive. In this work, we aim to apply the machine learning tool to transform a sparse model grid onto a continuous function. We apply a machine learning algorithm that involves a Gaussian process (GP) that measures the similarity between data points (i.e., the kernel function) to predict values for unseen points from training data. We use the generative model and treat fundamental parameters as given facts to predict observables. This gives us flexibility to prior fundamental inputs when modelling stars. We organise the rest of the paper as follow. Section 2 contents descriptions about the computation of a representative stellar model grid. We then introduce the underline theory of GP and the setup of GP model in Section 3. Section 5 demonstrates GP predictions and their systematic uncertainties. Subsequently, we augment the grid to have a set of continuously-sampled stellar models and model 100 fake stars for testing the accuracy of our method in Section 6. Lastly, we discuss advantages and limitations of this approach, highlight areas where improvements can be found in the near future, and summary conclusions in Section 7.

2 REPRESENTATIVE MODEL GRID

2.1 Grid computation

We compute a stellar model grid as the training dataset. We aim to cover stars with approximate solar mass on the main-sequence and the subgiant phases. We consider four independent fundamental inputs which are stellar mass (M), initial helium fraction ($Y_{\rm init}$), initial metallicity ([Fe/H]$_{\rm init}$), and the mixing-length parameter ($\alpha_{\rm MLT}$). We calculated three model dataset for different purposes. The primary dataset is a standard model grid with uniform mass step. This model grid is used for all preliminary tests and also for the final training. We also calculate an additional dataset to increase the grid resolution for $M > 1.05{\rm M}_{\odot}$. Because we find that the blue hook feature (where global parameters sharply vary) is relatively hard to train. This dataset is only used in the final training. Details of parameter ranges and steps of the two grid are listed in Table 1. For validating and testing GP predictions, we computed off-grid models with randomly sampled fundamental inputs. The computation of evolutionary tracks starts at the Hayashi line **with pre-main-sequence temperature around 4600K** and terminates at the base of red-giant branch (RGB) where $\log g = 3.6$ dex. Note that we only use models after the zero-age-main-sequence (ZAMS), which is defined as the point where core-hydrogen burning contributes over 99.9% of the total luminosity.

Table 1. Computation of Stellar model grid.

Primary dataset		
Input Parameter	Range	Increment
M ($M_{\odot}$)	0.80 – 1.20	0.01
[Fe/H] (dex)	-0.5 – 0.2/0.2 – 0.5	0.1/0.05
$Y_{\rm init}$	0.24 – 0.32	0.02
$\alpha_{\rm MLT}$	1.7 – 2.5	0.2
Additional dataset		
Input Parameter	Range	Increment
M ($M_{\odot}$)	1.055 – 1.195	0.01
[Fe/H] (dex)	0.25 – 0.45	0.1
$Y_{\rm init}$	0.25 – 0.31	0.02
$\alpha_{\rm MLT}$	1.8– 2.4	0.2

2.2 Input physics

We use the stellar code Modules for Experiments in Stellar Astrophysics (MESA, version 12115) to construct stellar grids. MESA is an open-source stellar evolution package which is undergoing active development. Descriptions of input physics and numerical methods can be found in Paxton et al. (2011, 2013, 2015). We adopted the solar chemical mixture $[(Z/X)_{\odot} = 0.0181]$ provided by Asplund et al. (2009). The initial helium fraction ($Y_{\rm init}$) and initial metallicity ([Fe/H]$_{\rm init}$) are both independent inputs.

We use the MESA $\rho - T$ tables based on the 2005 update of OPAL EOS tables (Rogers & Nayfonov 2002) and OPAL opacity supplemented by low-temperature opacity (Ferguson et al. 2005). The grey Eddington $T - \tau$ relation is used to determine boundary conditions for modelling the atmosphere. The mixing-length theory is implemented and the convection is adjusted by the mixing-length parameter ($\alpha_{\rm MLT}$). We also apply the MESA **convective premixing scheme (Paxton et al. 2019), which an approach to handling mixing in convection zones that improves model structures at the convective boundary.** Atomic diffusion of helium and heavy elements was also taken into account. MESA calculates particle diffusion and gravitational settling by solving Burger's equations using the method and diffusion coefficients of Thoul et al. (1994) **as well as radiative turbulence formula given by Morel & Thévenin (2002)**. We consider eight elements (^{1}H, ^{3}He, ^{4}He, ^{12}C, ^{14}N, ^{16}O, ^{20}Ne, and ^{24}Mg) for diffusion calculations, and have the charge calculated by the MESA ionization module, which estimates the typical ionic charge as a function of T, ρ, and free electrons per nucleon from Paquette et al. (1986). **We only compute diffusion during the main-sequence stage before the central hydrogen abundance drops below 0.05, because its effects can be neglected in post main-sequence stages.** The MESA inlist used for the computation is available on `https://github.com/litanda/mesa_inlist/`.

2.3 Equivalent Evolutionary Phase

Apart from the four independent model inputs, i.e., mass, metallicity, helium fraction, and the mixing-length parameter, stellar age is the fifth fundamental input. However, the dynamical range of age varies track by track. This makes GP models hard to map from age to global parameters. We need a uniform input to replace the age. The fractional age is an option but we find that global parameters (e.g. effective temperature) sharply change with the fractional age around the blue hook and the turn-off point (as shown in the left

panel in Figure 1). It requires a complex and spiky kernel function to fit the curvatures in this area and hence difficult for GP to learn. Dotter (2016) has introduced a quantity Equivalent Evolutionary Phase (*EEP*), which numbers evolutionary stages and transform stellar tracks onto a uniform basis. We follow this idea but define *EEP* in a different way to make global parameters change relatively smoothly. On each evolutionary track, we compute the displacement between consecutive models on the $\log T_{\rm eff} - \log g$ diagram. For instance, the displacement between model n and model $n-1$ can be calculated as

$$\delta d_n = ((\log T_{\rm eff,n} - \log T_{\rm eff,n-1})^2 + (\log g_n - \log g_{n-1})^2))^c, \quad (1)$$

where c is an adjusted parameter to scale the displacement. The total displacement of model n from the ZAMS (model 0) can be calculated with

$$d_n = \sum_{i=1}^{i=n} \delta d_i. \quad (2)$$

We then normalise d_n to the 0 –1 range and define it as *EEP*. On an evolutionary track, *EEP* equals to 0 at the ZAMS and 1 on the RGB where log = 3.6 dex. The factor c in Eq. 1 is introduced for modulating *EEP* to avoid obvious data gap in the parameter space, and we find that $c = 0.18$ gives the most uniform data distribution. In Figure 1, we demonstrate how the effective temperature changes with fractional age and *EEP*. It can be seen that *EEP* is a better choice than the fractional age because global parameters change smoother around the blue hook and the turn-off point.

2.4 Sampling method

There is a limitation of the data size in the GP framework, because the computational and memory complexity exponentially increase with the number of data points. In practice, the typical data size is on an order of 10^4. Given that the grid contents $\sim 10,000,000$ stellar models, only a small subset can be used for training. The sampling method is hence important. A flat sampling is not appropriate, because the evolving step is not uniform at different evolutionary stages due to the MESA step-control strategy. For instance, stellar models are dense at the main-sequence and lower RGB but quite sparse at the subgiant stage. We test a few methods and find that using the displacement (δd_n) defined in Eq. 1 as the weight to sample models on an evolutionary track gives a relatively uniform data distribution at different evolutionary stages.

3 GAUSSIAN PROCESS MODEL

A GP can be applied as a probabilistic model to a regression problem. Here we use the GP model to generalise a stellar model grid to a continuous and probabilistic function that maps inputs to observable quantities. This allow us to predict observable quantities for off-grid regions. We intend to train GP models that maps five fundamental inputs, i.e., mass (M), initial metallicity ([Fe/H]$_{\rm init}$), initial helium fraction($Y_{\rm init}$), the mixing-length parameter($\alpha_{\rm MLT}$), and equivalent evolutionary phase (*EEP*), to five model outputs including effective temperature($T_{\rm eff}$), surface gravity($\log g$), radius(R), surface metallicity(*[Fe/H]*), and stellar age(τ). We use the GP model as a non-parametric emulator, that is emulating the comparatively slow calls to models of stellar evolution. This emulator can be described as a function approximation problem. In fact, the way we have implemented the GP as function approximation means that we have used one GP for each of the outputs so that they can be described as

$$T_{\rm eff} = f_{T_{\rm eff}}(M, EEP, [Fe/H]_{\rm init}, Y_{\rm init}, \alpha_{\rm MLT}), \quad (3)$$

$$\log g = f_{\log g}(M, EEP, [Fe/H]_{\rm init}, Y_{\rm init}, \alpha_{\rm MLT}), \quad (4)$$

$$R = f_R(M, EEP, [Fe/H]_{\rm init}, Y_{\rm init}, \alpha_{\rm MLT}), \quad (5)$$

$$[Fe/H] = f_{[Fe/H]}(M, EEP, [Fe/H]_{\rm init}, Y_{\rm init}, \alpha_{\rm MLT}), \quad (6)$$

and

$$\tau = f_\tau(M, EEP, [Fe/H]_{\rm init}, Y_{\rm init}, \alpha_{\rm MLT}). \quad (7)$$

In the following, we introduce the underline theory of GP regression and the setup of training GP models.

3.1 Gaussian Process Application

In our application to a stellar model grid, a GP has a number of desirable properties. While a GP is a stochastic process, the distribution of a GP can be considered as a distribution of functions with a continuous domain. In fact, the marginal likelihood considered in function space is equal to the likelihood of the data given some function values, multiplied by the prior on those function values marginalised over all function values Williams & Rasmussen (1996). That is to say that, the GP allows for the analytical evaluation of a fit over many different functions (perhaps an infinite number) weighted by some concept of a prior and the agreement with the data. In addition, while the marginal likelihood will be assessed on discrete data, predictions can be made using linear algebra for new data in the continuous domain, but crucially again marginalised over these many different functional forms. It is possible to see how this might be useful for generalising (or emulating or augmenting) a discrete grid of stellar models in order to obtain predictions in the continuous domain.

In this section we will look at the required mathematics to be able to implement a GP for our application to grids of stellar models. We start with a series of definitions before dealing with the marginal likelihood and the posterior predictive distributions.

We start with a grid of stellar models containing N models with a label we want to learn, for example model effective temperature, which we will denote with the general symbol $\mathbf{y}$, and a set of on-grid inputs $\mathbf{X}$ (e.g., mass, *EEP*, metallicity, ...). We can use a GP to make predictions of the effective temperature (labelled y) for additional off-grid input values given by $\mathbf{X}_\star$. The vector $\mathbf{y}$ is arranged $\mathbf{y} = (y_i, ..., y_N)^T$ where the subscript label references the stellar model. The input labels are arranged into a $N \times D$ matrix where D is the number of input dimensions (e.g., $D = 3$ for mass, *EEP*, and metallicity) so that $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_N)^T$ where $\mathbf{x_i} = (x_{1,i}, ..., x_{D,i})^T$. The matrix of additional inputs $\mathbf{X}_\star$ has the same form as $\mathbf{X}$ but size $N_\star \times D$.

Williams & Rasmussen (1996), from which our description below is based, define a GP as a collection of random variables, where any finite number of which have a joint Gaussian distribution.

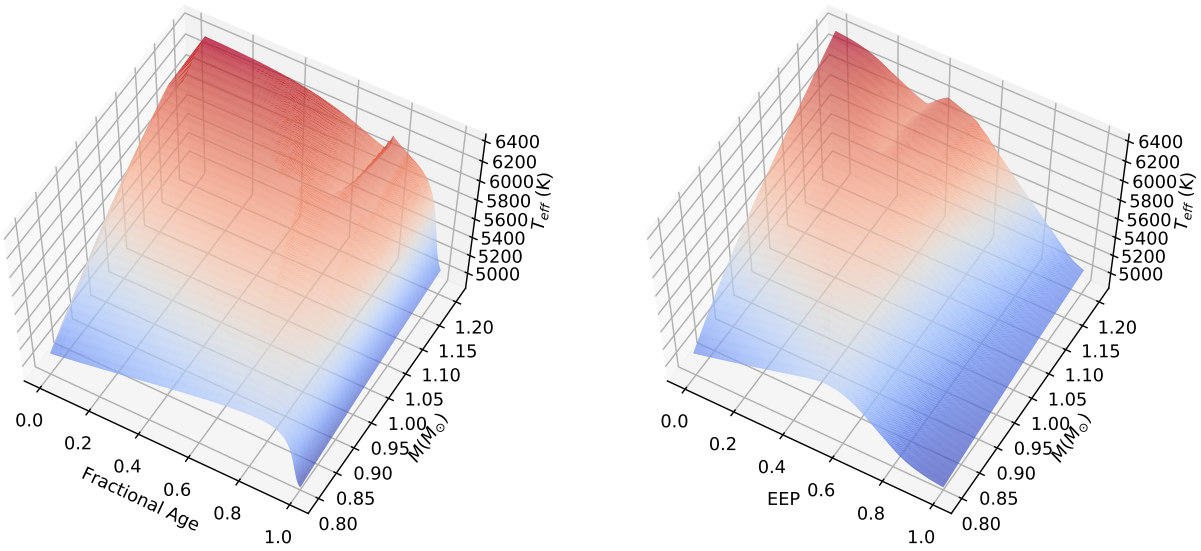

Figure 1. Surface plots of model effective temperature on the mass-fractional age (left) and mass-EEP (right) diagrams. Models in this figure are from the primary grid with fixed initial metallicity ($[\mathrm{Fe/H}]_{\mathrm{init}} = 0.0$), helium fraction ($Y_{\mathrm{init}} = 0.28$) and mixing-length parameter ($\alpha_{\mathrm{MLT}} = 2.1$). It can be seen that the effective temperature changes much smoother on the mass-EEP diagram at the blue hook and turn-off points.

In general terms, a GP may be written so that our on grid labels are random variables drawn from our GP distribution,

$$\mathbf{y}(\mathbf{X}) \sim \mathcal{GP}\left(m(\mathbf{X}), \mathbf{\Sigma}\right), \tag{8}$$

where $m(\mathbf{X})$ is some mean function, and $\mathbf{\Sigma}$ is some covariance matrix. The mean function controls the deterministic part of the regression and the covariance function controls the stochastic part. The mean function defined here could be any deterministic function and we will label the additional parameters, or hyperparameters, ϕ. Each element of the more familiar covariance matrix is defined by the covariance function or *kernel function* $\mathbf{K}$ which has hyperparameters θ and is given by,

$$\mathbf{\Sigma} = \mathbf{K}(\mathbf{X}, \mathbf{X}, \theta), \tag{9}$$

or

$$\mathbf{\Sigma}_{n,m} = k(\mathbf{X}_n, \mathbf{X}_m, \theta), \tag{10}$$

where the inputs $\mathbf{X}_n$ and $\mathbf{X}_n$ are D-dimensional vectors and the output is a scalar covariance. In addition to the covariance defined by the kernel function, we include additional white noise in the covariance matrix by adding an identity matrix $\mathcal{I}$ multiplied by a scalar value σ_w^2, so that,

$$\mathbf{\Sigma} = \mathbf{K}(\mathbf{X}, \mathbf{X}, \theta) + \sigma_w^2 \mathcal{I}, \tag{11}$$

where σ_w^2 is another hyperparameter to be learnt in training.

3.1.1 *The likelihood*

Conceptually we value the GP because of it's ability to marginalise over many functions $\mathbf{f}$ and return a marginal likelihood,

$$p(\mathbf{y}|\mathbf{X}) = \int p(\mathbf{y}|\mathbf{f}, \mathbf{X}) p(\mathbf{f}|\mathbf{X})\, \mathrm{d}\mathbf{f}, \tag{12}$$

noting that this function space marginal likelihood is weighted by the probability of the data given the function and the probability of the function. This integral could be evaluated. However, by noting that a GP is a collection of random variables, where any finite number of which have a joint Gaussian distribution, the marginal probability of our data $\mathbf{y}$ is also the joint likelihood of a multivariate normal distribution,

$$p(\mathbf{y}|\mathbf{X}) = \mathcal{N}(m(\mathbf{X}), \mathbf{\Sigma}), \tag{13}$$

which can be straightforward to evaluate. Thus the marginal likelihood is,

$$p(\mathbf{y}|\mathbf{X}) = (2\pi)^{k/2} \det(\mathbf{\Sigma})^{-0.5} \exp\left(\frac{-1}{2}(\mathbf{X} - m(\mathbf{X}))^T \mathbf{\Sigma}^{-1} (\mathbf{X} - m(\mathbf{X}))\right), \tag{14}$$

which can be evaluated without integrating over all possible function space. While this marginal likelihood expression is clearly more computationally feasible that the integral over functional space it not without it's limitations. Because it is necessary to calculate the determinant and the inverse of the covariance matrix, typically applied algorithms, make this a $O(N^3)$ or $O(N^2 \log N)$ operation. This naturally limits the size of the data set for which the likelihood, and optimisations of the likelihood, can be applied.

3.1.2 *Making predictions*

If we want to obtain predictive distributions for the output $\mathbf{y}_\star$ given the inputs $\mathbf{X}_\star$, the joint probability distribution of $\mathbf{y}$ and $\mathbf{y}_\star$ is Gaussian and given by

$$p\left(\begin{bmatrix}\mathbf{y}\\ \mathbf{y}_\star\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}m(\mathbf{X})\\ m(\mathbf{X}_\star)\end{bmatrix}, \begin{bmatrix}\mathbf{\Sigma} & \mathbf{K}_\star\\ \mathbf{K}_\star{}^T & \mathbf{K}_{\star\star}\end{bmatrix}\right), \tag{15}$$

where the covariance matrices $\mathbf{\Sigma}$ and $\mathbf{K}$ are computed using the kernel function so that,

$$\mathbf{\Sigma}_{n,m} = k(\mathbf{X}_n, \mathbf{X}_m), \tag{16}$$

which is an $N \times N$ matrix.

$$\mathbf{K}_{\star\, n,m} = k(\mathbf{X}_n, \mathbf{X}_{\star\, m}), \tag{17}$$

which is an $N \times N_\star$ matrix, and finally

$$\mathbf{K}_{\star\star\, n,m} = k(\mathbf{X}_{\star\, n}, \mathbf{X}_{\star\, m}), \tag{18}$$

which is an $N_\star \times N_\star$ matrix. The predictions of $\mathbf{y}_\star$ are again a Gaussian distribution so that,

$$\mathbf{y}_\star \sim \mathcal{N}(\hat{\mathbf{y}}_\star, \mathbf{C}), \tag{19}$$

where

$$\hat{\mathbf{y}}_\star = m(\mathbf{X}_\star) + \mathbf{K}_\star^T\, \mathbf{\Sigma}^{-1}\, (\mathbf{y} - m(\mathbf{X})), \tag{20}$$

and

$$\mathbf{C} = \mathbf{K}_{\star\star} - \mathbf{K}_\star^T\, \mathbf{\Sigma}^{-1}\, \mathbf{K}_\star. \tag{21}$$

At point we can make predictions on model properties given a grid of stellar models using equation 19. But these predictions will likely be poor unless we select sensible values for the form and hyperparameters of the mean function and covariance function. In the following section we detail a number of kernel functions that will be tested against the data. We will then discuss the method for determining the values of the hyperparameters to be used.

3.2 Setup of GP Models

3.2.1 Tool package

We adopt a tool package named GPyTorch, which is a GP framework developed by Gardner et al. (2018). It is a Gaussian process library based on an open source machine-learning framework PyTorch (`https://pytorch.org`). The package provides significant GPU acceleration, state-of-the-art implementations of the latest algorithmic advances for scalability and flexibility, and easy integration with deep learning frameworks. Source codes and detailed introductions could be found on `https://gpytorch.ai`. We train GP models on a NVidia Tesla V100 graphics processing unit (GPU) with 32GB GPU Memory. The GPU captivity allows a training dataset with up to ~20,000 data points.

3.2.2 Training procudure

The training procedure of a GP model includes training, validating, and testing. In the training process, we literately optimise hyperparameters of a GP model to learn the underline function which maps inputs to outputs from on-grid evolutionary tracks (training dataset). In each iteration, the GP model is validated by comparing true and GP predicted values of some off-grid tracks (validating dataset). Although the validating dataset is not directly involved in training hyperparameters, it still constructs the GP model to some extend because the optimal solution is the one that best fits the validating dataset. For this reason, the validating dataset does not give a completely independent validation for a GP model. We hence have a testing process after the training. The testing dataset contents some other off-grid tracks which are reserved from the training and validating process. The testing dataset are also used to estimate the systematic uncertainties of GP model.

Here we briefly summary the setup of GP model training. We apply an Architecture Neural Network (ANN) including 6 hidden layers and 128 nodes per layer as the mean function. **Note that this is not training an ANN to learn the data in detail. The ANN is quickly trained at the beginning to interpret the complex mean function in multiple-demission space to accelerate the whole training process. In the GP model training, the mean function is normal uninteresting because all the inference effort is spent on estimating the correct covariance function. In our tests, GP models with the linear or the constant mean function could achieve similar results, but it takes more time for models to converge.** The GPyTorch standard likelihood for regression, which assumes a standard homoskedastic noise model, is applied as the likelihood function. We use the negative logarithm of the likelihood as the loss function. The optimiser for training is called 'Adam', which is a combination of the advantages of two other extensions of stochastic gradient descent, specifically, Adaptive Gradient Algorithm and Root Mean Square Propagation (Kingma & Ba 2014). More detailed discussions about these choices can be seen in the Appendix A.

We set up the so-called 'Early Stopping' procedure to decide when to terminate the training. The procedure evaluates GP models on a holdout validation dataset after each iteration. If the performance of the GP model on the validation dataset starts to degrade or stops upgrading after many iterations, then the training process is terminated (see discussions in Anzai 2012; Goodfellow et al. 2016). The 'Early Stopping' procedure can reduce overfitting and improve the generalisation of GP models. We use the aforementioned validating EI to monitor the training and terminate it when there is no improvement for 300 iterations.

To save the best learned GP models, we check the validating EI after every iteration. The current model will be saved to replace the last saving if it has the so-far lowest validation EI. This is to say, the final saved model is the one with the best performance in the training process.

3.2.3 Kernel Function

To select the proper kernel function for training GP models, we test four basic kernels and a number of combined kernels. The four basic kernels are listed as follow.

- **RBF**: Radial Basis Function kernel (also known as squared exponential kernel)
- **RQ**: Rational Quadratic Kernel (equivalent to adding together many RBF kernels with different lengthscales)
- **Mat12**: Matern 1/2 kernel (equivalent to the Exponential Kernel)
- **Mat32**: Matern 3/2 kernel

These four kernels are all universal, and we can integrate each of them against most functions. Every function in its prior has infinitely many derivatives. **The differences between these kernels, in a simply way, can be understood as their smoothness/flexibility levels. The RBF kernel is very smooth function and can be expressed as a product of a polynomial.** It hence suits for the case when the data follow a slowly varying function. The RQ kernel, as a combination of many RBF kernels, is more complex and is able to fit to data with a number of smooth underline functions (e.g. when the output depends on multiple inputs). On the opposite, the Mat12

gives the absolute exponential kernel, which is hence very spiky. It can fit to any sharp variations in the data. The Mat32 kernel has a smoothness somewhere between the RBF and Mat12 kernels, because it is a combination of an exponential and a polynomial. It is a smooth function but has significantly more extrema than the RBF kernel. **For each kernel, there are hyper parameters (e.g. the lengthscale) to modulate the smoothness as well. The hyper parameters can be either prioritised or purely determined in training process. A less smooth kernel like Mat32 with a large lengthscale could behave similarly to the RBF kernel. However, RBF kernel with small lengthscale can not reproduce those extrema in Mat32 kernel.** These differences are not chance coincidence, and the origin of these differences are crucial for interpreting the results. Choosing a good kernel for a particular application is necessary for good predictions. If the generates functions that are too spiky, then the analysis will have variance that is too high. On the contrary, if the kernel function is too smooth, it may not fully capture the variability of the underlying function and lead to an increase in bias. Combining basic kernels can increase the flexibility of kernel. For instance, the combination of a RBF kernel and a Mat12 kernel is able to fit to both smooth and spiky features in the data. **However, overfitting is a risk when using combined kernel because the kernel could be over-flexible.**

According to the stellar theory, changing fundamental input parameters smoothly changes the dependent outputs. Hence the kernel function for our application needs to be smooth. Moreover, we can notice in Figure 1 that the observable quantities fast vary at some particular regions (e.g., around the blue hook). This means that a slowly varying function like the RBF kernel may under fit in these areas. We do a number of preliminary studies of training GP models with different kernel functions to choose the proper kernel function. Details of training will be mentioned in Section 4.2. Here we only summary the results. We apply four basic kernels and a number of their combinations (RBF + Mat21, RQ + Mat21, Mat32 + Mat21, RBF + Mat32, RQ + Mat32) to train GP models. The combined kernel RBF+Mat21 gives the best fit to the training data, however, its testing errors are large. This indicates that the kernel is too flexible and hence overfits to the data. The kernel having the best performance is the Mat32. The GP model with Mat32 fits training data reasonably well and gives the best predictions for the testing models. The results match our expectations. What we need is a smooth function but not too smooth to fit to the quick variations at some particular evolutionary phases. The Mat32 kernel is apparently suitable for our application.

4 PRELIMINARY STUDIES

Before training the whole model grid (with five input demissions), we start with a number of preliminary studies on low-demission dataset. These preliminary studies are for several purposes. In the 1D problem (training data on a single evolutionary track), we compare GP predictions with the classical interpolator. In the 2D problem, we train GP models on a mass – *EEP* platform to test the performances of using different kernel functions. We also discuss about introducing a new error index for validating and testing GP models instead of using a global error quantity such like Root Mean Square Error. In the 3D problem, where GP maps three fundamental inputs (mass, EEP, and metallicity) onto observables, we solve the training strategy for the large dataset whose data size excesses the practical limitation.

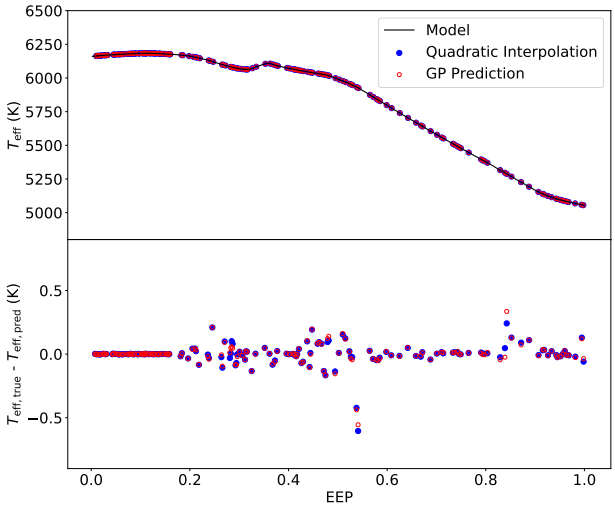

Figure 2. GP application on 1D problem. Models on this track are split into training and testing data by 70-to-30, Top: the evolution of effective temperature for a $1.1M_{\odot}$ track. The grey line is the evolutionary track computed with MESA; blue and red circles indicate predictions for the testing data from the quadratic interpolator and the GP model. Bottom: residuals of predictions in the top graph.

4.1 1D Problem

We first demonstrate an example of GP application on an 1D problem. We train a GP model using the Mat32 kernel to learn the evolution of effective temperature for a $1.1M_{\odot}$ track. We split the model data points on this track into training and testing data by 70-to-30. We train a GP model which maps *EEP* to effective temperatures and then test GP-predicted effective temperatures with truths. As it can be seen in Figure 2, the GP model gives very good predictions with residuals less than ±0.5 K. As a comparison, we fit the training data with the quadratic function and use the fitted function to do the same prediction. We find very similar results from the two methods. It suggests that GP can be an alternative of classical interpolators on the 1D problem.

4.2 2D Problem

As a further step, we train GP models on a 2D problem where GP models map mass and *EEP* to the five observable outputs (Outputs = $f(M, EEP)$). Training data are selected from the primary grid with fixed $[Fe/H]_{\rm init}$ (0.0), $Y_{\rm init}$ (0.28), and $\alpha_{\rm MLT}$ (2.1). There are 41 evolutionary tracks which content 24,257 models, and we sample 20,000 of them as training data. To validate and test GP models, we compute 44 evolutionary tracks with the same $[Fe/H]_{\rm init}$, $Y_{\rm init}$, and $\alpha_{\rm MLT}$ but randomly sampled M. We split off-grid tracks half-to-half as validating and testing datasets. The script developed based on the SIMPLE GP REGRESSION example (https://docs.gpytorch.ai/en/stable/examples/01_Exact_GPs/Simple_GP_Regression.html). We change the mean function and optimiser in the example and add an early stopping and a model saving modules. We follow the aforementioned training procedure to train, validate, and test GP models. We illustrate the learned GP model for effective temperature on the mass–*EEP* diagram in Figure 3. As it shown that, GP transforms the sparse

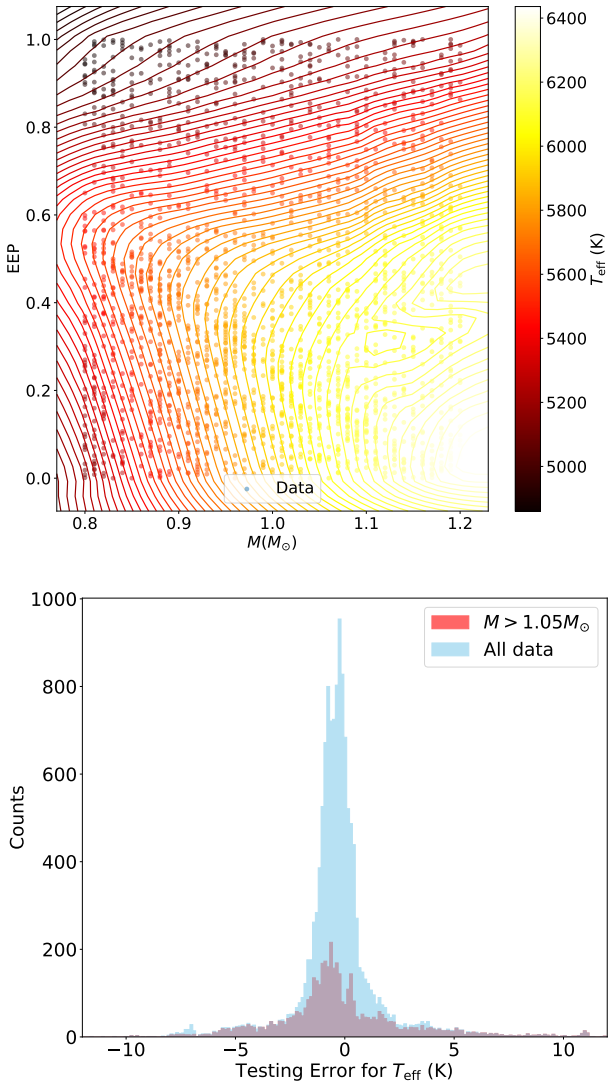

Figure 3. Top: The 2D GP model for $T_{\rm eff}$. Bottom: probability distributions of validating errors of the GP model.

data onto a continuous function and hence is able to predict values for unseen points in the grid.

It can be seen in Figure 3 that kernels in the area of $M \geq 1.05 M_\odot$ and $EEP \leq 0.7$ is more complex than those for other regions. There are two regions where global parameters vary relatively fast. The first is around the blue hook and main-sequence-turn-off point ($EEP \sim 0.4$) where high-mass tracks sharply turns on the HR diagram. The second is at early subgiant pahse ($EEP \sim 0.6$) where stars fast restructure. Features in these particular areas are relatively difficult to learn and hence poorly predicted by the GP model. When there is a subregion in which the GP model performs worse than other areas, the error distribution would not follow a Normal function. As shown at the bottom of Figure 3, the density distribution of testing errors form long tails which contents about 10% data. **The cases for other two global parameters surface gravity and radius are similar to the effective temperature.**

We also find substructures when inspecting testing errors for metallicity and age. The region where the surface metallicity quickly changes is at the early subgiant phase for relatively high-mass tracks. This is because high-mass models maintain shallow convective envelope and hence have strong diffusion effect during the main-sequence stage. At the early subgiant phase, the quick expansion of the surface convective envelope mixes up the settled heavy elements, leading to a fast raise of the surface metallicity. The accuracy of age prediction drops down for very old low-mass stellar models. This is because age values vary in a relatively big dynamic range (15 - 50 Gyr) in a small fraction of data points. Poor GP predictions are caused by the low age resolution.

The error distribution causes an issue in validating and testing GP models. What we normally use are some global errors, such like Root Mean Square Error (RMSE), to represent the validating or testing results. For our case, a global error is not able to point out how GP performs in regions where an observable quantity quickly varies. We want to have an error index that can reflect the GP model performance in general as well as in those sub-areas. By inspecting the error distributions of all five outputs, we find the data points in the tails (outside the 3 times of full width at half maximum) are around 10% (8 - 12% for different outputs). For the majority (90%) of data points, which from a Gaussian-like profile, the 68% confidential interval (1-σ uncertainty) can be used to reflect the global accuracy. For the worst 10% of the data, we could use the 95% and 99.7% confidential intervals (2- and 3-σ uncertainties) to describe the median and the length of the tail. Thus, we define an Error Index (EI), which is the sum of 68%, 95%, and 99.7% cumulative values of the absolute errors. For the case in Figure 3, cumulative values at 68%, 95%, and 99.7% are 1.1, 4.9, and 11.1K, which give a testing EI equals to 17.1K. We apply this EI in all following training processes to validate and test GP models.

We train GP models using different kernel functions to investigate which is the best for our application. We do this with the 2D data because the training is fast to be able to test many different options. As mentioned in Section 3.2.3, we find Mat32 is the most suitable kernel for mapping the stellar model grid.

4.3 3D Problem: Strategy for Large Data Sample

We apply GP to a 3D problem where GP maps three fundamental inputs, i.e., M, EEP, and $[Fe/H]_{\rm init}$ to observables. The main purpose of this preliminary study is investigate the strategy for training large data sample that exceeds the data size limitation of 20,000. We select training data from the primary grid with $Y_{\rm init} = 0.28$ and $\alpha_{\rm MLT} = 2.1$. The training dataset contents ~300,000 data points which is 15 times the limit of training data size (20,000). For validating and testing purposes, we compute another 174 evolutionary tracks with the same input $Y_{\rm init}$ and $\alpha_{\rm MLT}$ but random input M and $[Fe/H]_{\rm init}$.

We start with sampling 20,000 training data and train a set of GP models. We then obtain testing EI for each output. For instance, the testing EI for $T_{\rm eff}$ is 23.5K (2.0, 5.8, and 15.7 K at 68th, 95th, and 99.7th). We then apply two state-of-the-art approaches designed for large dataset to train the model. These two approaches are named as Stochastic Variational GP (SVGP) and Structured Kernel Interpolation (SKI GP). We only find some minor improvements in the GP predictions. Comparing the testing EI for $T_{\rm eff}$, SVGP gives a results of EI = 24.1K (2.2, 6.8, and 15.1K at at 68th, 95th, and 99.7th) and SKI GP ends up with EI = 22.9K (2.0, 6.1, and 14.8K at 68th, 95th, and 99.7th). Details about these two implements and some discussions about the results can be seen in appendix B.

We seek for better strategy for training large data sample. The GPU memory captivity limits the actual number of data that induce the kernel. This limitation becomes crucial for the high-demission problems. Because we need much more data give the fact that the pa-

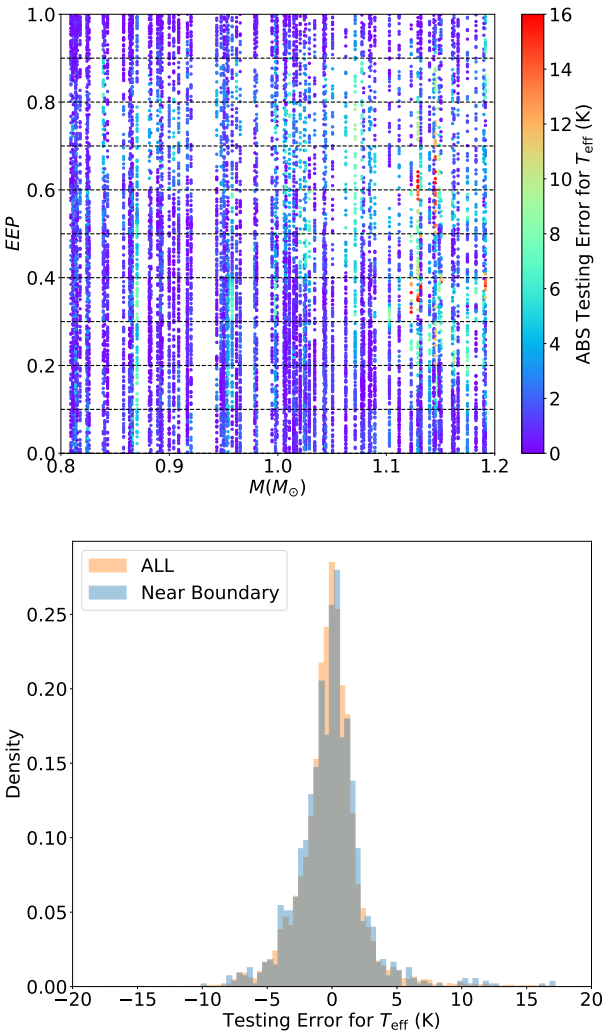

Figure 4. Top: Testing errors of 3D GP model for $T_{\rm eff}$ on the $M - EEP$ diagram. Dashes indicates section boundaries. Bottom: examination of the edge effects of the section scenario. Probability distributions of testing errors of all testing data and those near the boundary (±0.01 EEP) in the upper graph are compared. As it can be seen, testing errors do not raise around the boundary.

rameter space exponentially increase with the demission. A simple way to overcome this issue is breaking the grid into many sections and train GP models for each section separately. We divide the training dataset into 10 equal sections by *EEP* and sample 20,000 training data in each. A set of GP models are then trained for each *EEP* section. Using this section scenario, we improve the testing EIs for the five output parameters by around 10%. For instance, the testing EI for $T_{\rm eff}$ decreases from 23.5 to 21.6K. (1.7, 5.0, 14.9K at at at 68th, 95th, and 99.7th). EI values for the five outputs before and after sectioning can be seen in Table 3. The section scenario outperforms the SVGP and SKI GP methods. We hence apply it as our training strategy.

The section scenario improves the performance of GP model, but there is a major concern about the edge effect at the boundary between segments. If a GP model works significantly poorly at these boundaries, it will be difficult to map the systematic errors across the whole parameter space. We hence examine potential edge effects as illustrated in Figure 4. We inspect absolute testing errors for each output on the $M - EEP$ diagram. No obvious edge effect is found. We also do a statistical comparison between all errors and those around section boundaries ($\pm 0.01 EEP$). As shown in the bottom graph, the density distributions of the two samples are very similar to each other.

5 AUGMENTING THE STELLAR GRID

Based on what we find from preliminary studies, we now apply GP to mapping the whole 5D model grid. The setup of GP model is summarised in Table 2. The training data are sampled from both primary and additional girds (as described in Table 1). The additional grid increases the grid resolution for relatively high-mass models, and this gives more information about the blue hook for GP to learn. We also computed 4,880 off-grid tracks. These off-grid tracks are split by 50-to-50 for validating (in the training progress) and testing (after the training progress) GP models.

The section scenario is applied. For each section, we train a set of GP models for each output parameter with 20,000 training and 20,000 validating data. The number of sections need to be tested to obtain the best efficiency. To do this, we gradually increase the number of sections from 1 to 100 and track down the changes in testing EI. We find significant improvement from 1 to 10 sections but no further improvements for more than 10 sections. We list the testing EI with different numbers of sections in Table 3. It turns out that dividing the grid into 10 sections (corresponding to a 2% sampling rate) is the most efficient.

We use GP models for the 10-sections case as our final result. All following analysis and discussion are based on it. When testing GP models, we do not section the dataset because the data size limitation for testing is not strict. We sample 100,000 off-grid stellar models as the testing dataset. Note that we do not use models with $\tau \geq 20.0$ Gyr, $[Fe/H]_{\rm surf} \leq$ -0.6 dex, or $T_{\rm eff} \geq 7000K$ for testing because we find strong edge effects in those ranges.

5.1 Overview of Results

A overview of testing errors (Truths - GP predictions) can be seen in Figure 5, where we plot rolling medians and rolling standard deviations for all outputs' errors against fundamental inputs. Median values are approximate along zero in most plots, indicating good agreement between GP predictions and true values. The 68% confidential intervals are generally small and their dynamical ranges do not significantly vary across input ranges. However, the 95% confidential intervals have more significant changes and are not well scaled to the 68% confidential intervals. This corresponds to the tail feature as seen in Figure 3. Because GP predictions are relatively poor in some particular regions. For instance, predictions for the effective temperature are more scattered in high-mass because of the appearances of the blue hook. From these results, it can be see that the model systematic uncertainty is not uniform across the parameter space. Proper estimates of model uncertainty are hence necessary.

5.2 Mapping Systematic Uncertainties

A learned GP model predicts output quantities with uncertainties based on its noise model. In our preliminary studies, uncertainties are properly determined for the 1D and 2D problems, but we find obviously underestimated uncertainties in the 3D problem. In the 5D

Table 2. Setup of GP Models

GP model inputs		
Parameter	Notation	Range
Mass	M	0.8–1.2 $M_\odot$
Equivalent evolutionary phase	EEP	0 – 1
Initial metallicity	$[Fe/H]_{init}$	-0.5 – 0.5 dex
Initial helium fraction	Y_{init}	0.24 – 0.32
Mixing-length parameter	α_{MLT}	1.7 – 2.5
GP model outputs		
Parameter	Notation	Trust-worth range[a]
Effective temperature	T_{eff}	$\leq$ 7000 K
Surface gravity	$\log g$	-
Radius	R	-
Surface metallicity	$[Fe/H]$	$\geq$ -0.6 dex
Stellar age	τ	$\leq$ 20 Gyr
Setup of training		
Item	Adopted	
Kernel	Mat32	
Mean Function	6 layers x 128 notes Neural Network	
Likelihood Function	Gaussian Likelihood Function	
Loss Function	Exact marginal likelihood	
Optimiser	Adam including AMSGRAD variant	
Termination	Early Stoping (monitoring the validating EI)	

[a] The ranges without strong edge effects.

Table 3. Training and validating errors for GPR Models

Model Type	Inputs	$N_{Training}$	Sampling rate	Testing Errors (at 68/95/99.7%)				
				T_{eff} (K)	$\log g$ (10^{-3}dex)	R ($10^{-3}R_\odot$)	$[Fe/H]_{surf}$ (10^{-3}dex)	τ (10^{-2}Gyr)
GP	2D	20,000 x 1	96%	1/5/11	1/3/8	2/6/14	0.5/2/12	1/3/9
GP	3D	20,000 x 1	5%	2/6/16	1/4/10	3/7/17	2/6/22	2/7/22
GP with 10 sections	3D	20,000 x 10	50%	2/5/15	1/4/11	2/7/17	1/3/20	2/6/19
GP	5D	20,000 x 1	0.2%	3/9/34	2/5/18	4/11/36	2/7/30	3/9/27
GP with 3 sections	5D	20,000 x 3	0.6%	3/8/27	2/5/18	3/7/26	1/4/24	3/7/22
GP with 5 sections	5D	20,000 x 5	1%	2/7/25	1/4/15	3/7/24	1/4/21	2/6/22
GP with 10 sections	5D	20,000 x 10	2%	2/7/27	1/4/14	2/7/26	1/4/20	2/6/21
GP with 20 sections	5D	20,000 x 20	4%	2/7/26	1/4/14	2/7/27	1/3/18	2/6/22
GP with100 sections	5D	20,000 x 100	20%	2/7/25	1/4/14	2/7/26	1/3/17	2/6/18

problem, GP models also predict significantly small uncertainties: they are mostly one order of magnitude smaller than testing errors. This is to say, the learned GP models are over-confident for high-demission cases. The reason could be the equally spaced training data, from which GP model learn few variations at the scale smaller than the grid step and hence turns to fit with large lengthscale values. Because GP models do not give reliable uncertainties, we intend to use testing errors to estimate the systematic uncertainty in GP prediction. As shown in Figure 5, systematical uncertainties relate to M, EEP, and $[Fe/H]_{\rm init}$ but not to $Y_{\rm init}$ or $\alpha_{\rm MLT}$. We can treat this as a 3D problem and train another GP model, in which GP model systematic uncertainty is a function of M, EEP, and $[Fe/H]_{\rm init}$.

We inspect the testing errors in the M-EEP-$[Fe/H]_{\rm init}$ space and find that their local medians vary smoothly. We hence apply the constant mean function and the RBF kernel. The testing dataset contents 100,000 which exceeds the data size limitation. We use the SVGP approach but not the section scenario for this training, because the SVGP can well handle large data following smooth function (see Appendix B for detailed discussions about SVGP). We split the testing error data by 75-to-25 for training and validating. The variational evidence lower bound (ELBO) is adopted as the loss function because it is designed for when there is too much data for the exact inference. We set up Early Stopping by tracking the RMSE value and terminate training when the RMSE value stops decreasing for 100 iterations. The outputs of GP models are the local medians of testing errors. We use them to infer the systematic uncertainties for five observable quantities (referred as $\sigma_{T_{\rm eff}}$, $\sigma_{\log g}$, σ_R, $\sigma_{[Fe/H]_{\rm surf}}$, and σ_τ). To differentiate these GP models, we refer

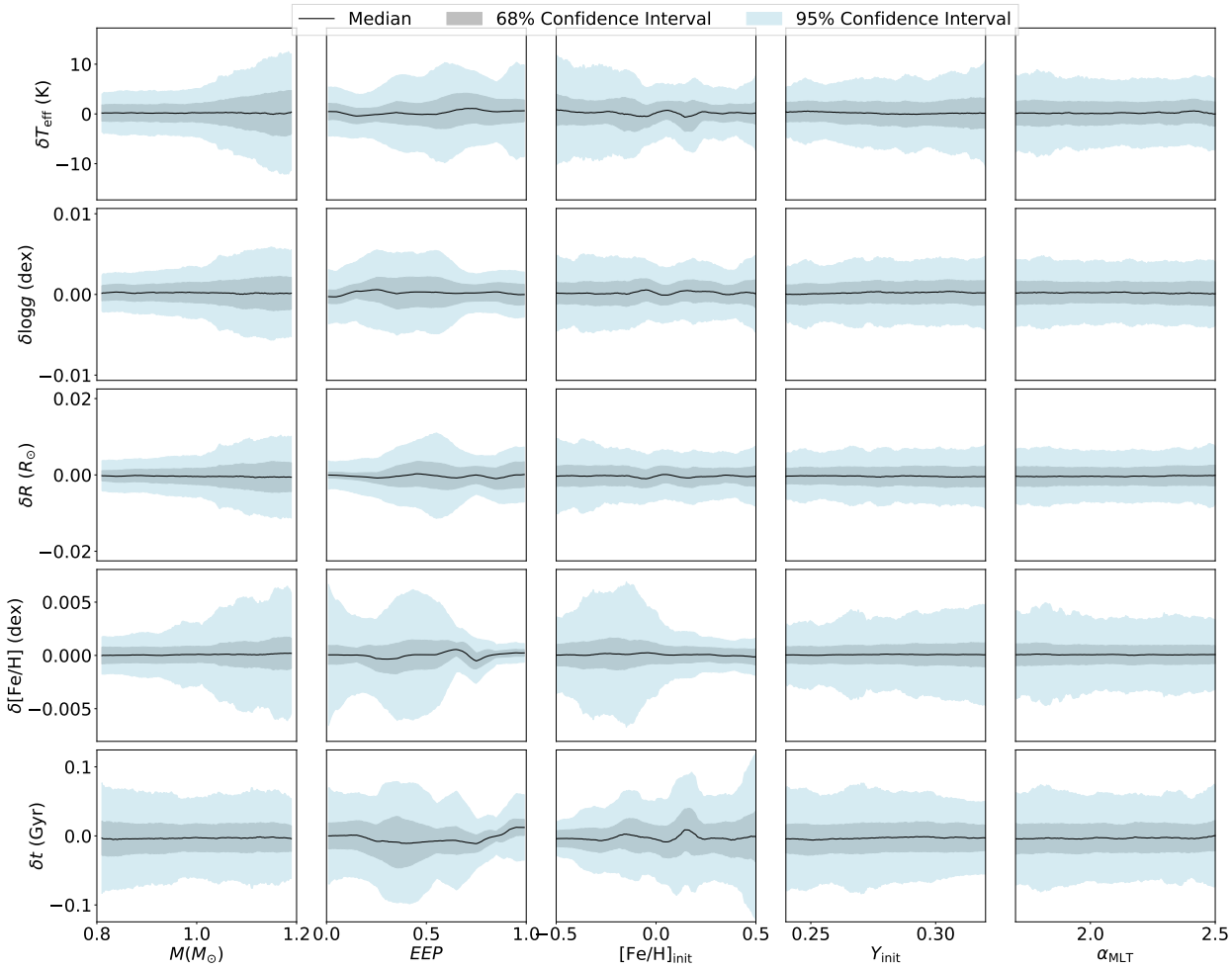

Figure 5. Roll medians and 68/95% confidential intervals of testing errors against GP model inputs. Black solid lines indicate the median value; grey and blue shadowes represent the 68% and 95% confidential interval. Testing errors of $T_{\rm eff}$, log g, and R mainly depend on M andEEP. Metallicity error strongly depends on M ,EEP, and $[Fe/H]_{\rm init}$, and age error has a significant correlation to EEP and $[Fe/H]_{\rm init}$. However, testing errors do not obviously relate to $Y_{\rm init}$ or $\alpha_{\rm MLT}$.

to them as GP-SYS models. In Figure 6, we compare the actual local systematic uncertainties for $T_{\rm eff}$ at $[Fe/H]_{\rm init} \simeq 0.0$ with those given by the GP-SYS models. It shows that the GP-SYS model well reproduces the $\sigma_{T_{\rm eff}}$ distributions.

6 MODELLING STARS WITH GP PREDICTIONS

6.1 Augmenting the model grid

Now we are able to use learned GP models to augment the original stellar grid. We randomly sample 5,000,000 data points with uniform distributions for five fundamental inputs (M, EEP, $[Fe/H]_{\rm init}$, $Y_{\rm init}$, and $\alpha_{\rm MLT}$). We then predict output quantities using GP models and their systematic uncertainties using GP-SYS models. This GP-based model dataset can be downloaded following the instruction at https://github.com/litanda/GPGird. In Figure 7, we demonstrate the original grid, GP predictions, and GP systematic uncertainties on the Kiel diagram.

6.2 GP-based Modelling for 1,000 Fake Stars

As a final test of our method, we use GP-predicted stellar models to characterise 1,000 fake stars to examine whether the method recovers true stellar properties. We compute 1,000 fake model stars with the same input physics as the grid but randomly sampled input fundamental parameters. To avoid the edge effect, fake stars are computed in the range of $T_{\rm eff}$ = [4700K, 6800K], log g = [3.7, 4.6], $[Fe/H]_{\rm surf}$ = [-0.35,0.35], M = [0.85,1.15], EEP = [0.05,0.95], $Y_{\rm init}$ = [0.25,0.31], and $\alpha_{\rm MLT}$ = [1.8,2.4]. We use four observables, i.e., $T_{\rm eff}$, log g, R, and $[Fe/H]_{\rm surf}$, as observed constraints. We apply typical observed uncertainty that is ±50K for $T_{\rm eff}$ (high-resolution spectroscopy), ±0.005dex for log g (seismology), 3% for R (seismology), and ±0.05dex for $[Fe/H]_{\rm surf}$ (high-resolution spectroscopy). Observed value for each constraint is calculated with true value plus a random noise which follows a Gaussian distribution.

We fit fake stars using the Maximum Likelihood Estimate (MLE) method. Note that the variance term in the MLE function contents observed uncertinaty and also the model systematic uncertainty determined with GP-SYS models ($\sigma^2 = \sigma^2_{obs} + \sigma^2_{sys}$). We

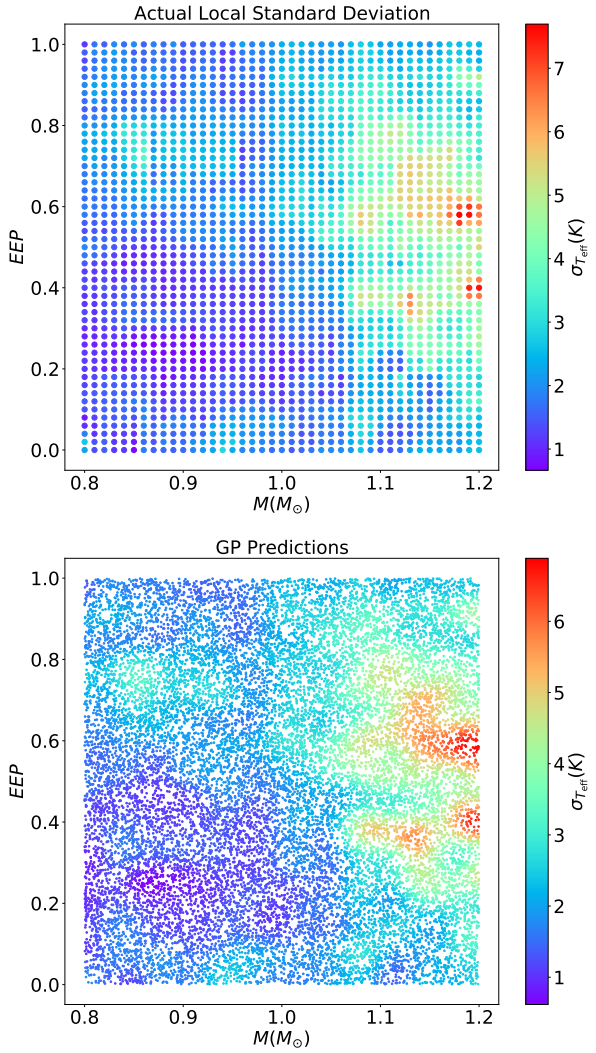

Figure 6. Comparison between actual and GP predicted local systematic uncertainties (1-σ) for $T_{\rm eff}$ on the $M-EEP$ diagram at $[Fe/H]_{\rm init} \simeq 0.0$. To calculate the actual local values, we separate the mass range into 40 equally-spaced segments and the *EEP* range into 50, and then measure the median testing error for each segment.

measure the 16th, 50th, and 84th percentiles of likelihood distribution to estimate a parameter and its uncertainty.

We present inferred stellar parameters for a representative fake star in Figure 8. Observed constraints for this fake star are $T_{\rm eff}$ = 5652±50K, $\log g$ = 4.424±0.005, $[Fe/H]_{\rm surf}$ = 0.31±0.05, and R = 1.047±0.031$R_\odot$. True fundamental parameters are M = 1.062M$_\odot$, τ = 3.79Gyr, $[Fe/H]_{\rm init}$ = 0.364, $Y_{\rm init}$ = 0.292, and $\alpha_{\rm MLT}$ = 1.984 (blue dashes). We fit with both the original model grid and the GP predictions for comparing. As it shown that the GP-based modelling has a completed statistical sampling and hence gives more sensible posterior distributions than the grid-based modelling. The improvement for the age is obvious. GP-based modelling infers an age of $4.4^{+2.0}_{-1.8}$Gyr, which is more accurate and precise than that determined with the original grid ($3.8^{+2.6}_{-2.2}$Gyr). It can be seen that the age estimate with the original grid does not actually converge because of under sampling. For initial metallicity, initial helium fraction, and the mixing-length parameter, at it shown that, GP-based models make it possible to properly estimate them without computing a very fine model grid. It clearly shows that GP is an efficient tool to augment a typical stellar model grid and overcome the under sampling issue, as a result, improves the precision of estimate.

We compared estimates for the 1,000 fake stars and find remarkable improvements in GP-based modelling. The average precisions are 4.7% for mass and 26% for age with the original grid and 4.5%and 19% with GP predictions. The accuracy of GP-based modelling is examined as illustrated in Figure 9, in which we compare modelling solutions with fake stars' true masses and ages. Good consistence is found, and we also see that differences nicely follow a normal distribution, saying that the fitting is only affected by random noises in observations. Comparing between grid-based and GP-based modelling, their results for the mass both consist with truths, but the median age difference shifts to around -0.2 for the grid-based modelling. This corresponds to what is seen in Figure 8, that the grid-based modelling can underestimate the age because of the in-completed sampling. We hence conclude that GP-based modelling can better recover stellar properties.

7 CONCLUSIONS

In this work, we apply a machine-learning algorithm that involves a Gaussian process as an interpolator to augment a stellar grid. We train GP models to convert a sparse model grid into continuous functions, which map five fundamental inputs to observable quantities. We find good precision and accuracy in GP-based interpolations when testing them with off-grid models. A GP-predicted model dataset is then generated and we use it to model 1,000 fake stars. Comparing with the original sparse grid, GP-predicted models have complete sampling across the parameter space and hence improve the accuracy and the precision of estimates, particularly for the stellar age.

The application of GP's make it efficient to generate a statistically-sound model dataset based on a sparse model grid. It works well for high-demission case (up to 5D in this work) and this can be a big advantage compared with traditional interpolators. The choice of kernel is crucial for the success of a particular case. We hence do a number of preliminary studies for the best option. A limitation of the training step for the GP is the size of the training data set and the requirement to perform matrix inversion and calculate the matrix determinant. For training a normal GP model, the practical training data size upper limit is around 20,000 which is apparently not big enough for the high-demission grid including ~10,000,000 models. To overcome this issue, we section the training data according to their evolutionary stage (*EEP*) and train GP models for each section. This section scenario significantly improves overall accuracy of GP predictions and we find no edge effects at the boundary of sections. When inspecting the systematic uncertainty of GP models, we notice that GP predictions are very over-confident in the high-demission problem: the uncertainties given by GP models are mostly smaller than testing errors by an order of magnitude. Because of this, we use the testing errors to estimate systematic uncertainties across the parameter space. Eventually, we provide a GP-based model dataset including 5,000,000 models with randomly sampled fundamental inputs.

We use GP-based models to characterise 1,000 fake stars to examine whether truths of stellar properties can be recovered. We find that GP-based masses and ages are consistent with the injected truth values. The uncertainties are dominated by observational noise, saying that, the systematic uncertainty due to the GP approximation

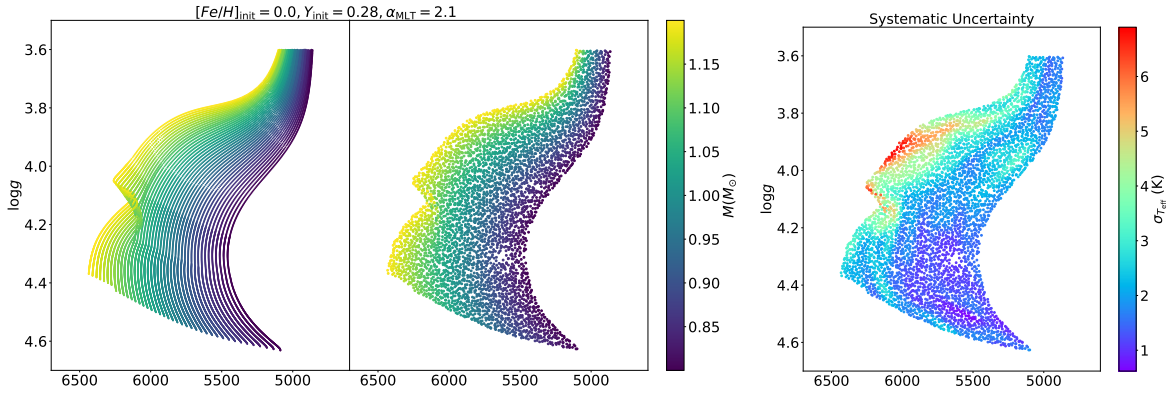

Figure 7. Left: Comparing the original grid and GP predictions on Kiel diagram. Right: The systematical uncertainties for $T_{\rm eff}$ given by GP-SYS models.

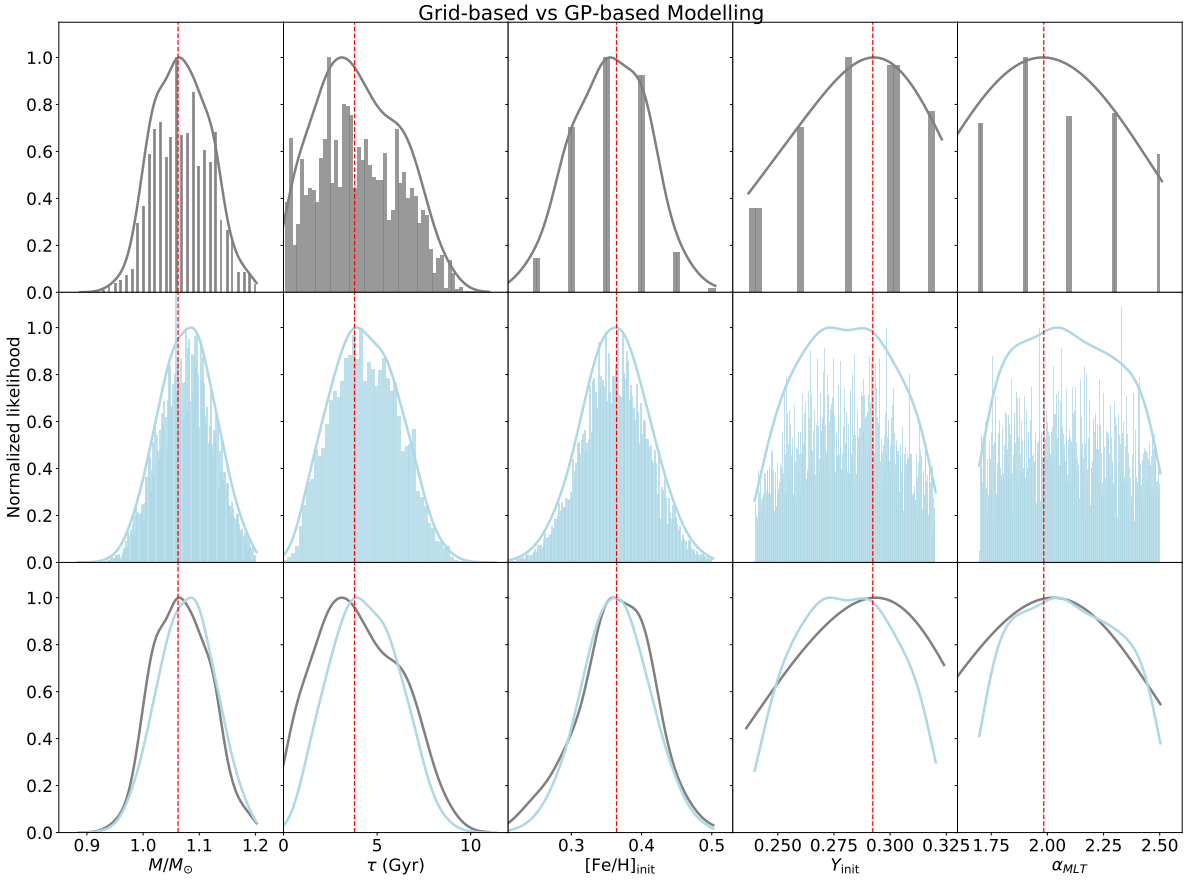

Figure 8. replot grey lines for feh, Y, and alpha Probability distributions of estimated fundamental parameters from grid-based (Top) and GP-based modelling (Bottom) for a fake star. Observed constraints for this fake star are $T_{\rm eff}$ = 5652±50K, $\log g$ = 4.424±0.005, $[Fe/H]_{\rm surf}$ = 0.31±0.05, and R = 1.047±0.031$R_\odot$. True fundamental parameters are M = 1.062M$_\odot$, τ = 3.79Gyr, $[Fe/H]_{\rm init}$ = 0.364, $Y_{\rm init}$ = 0.292, and $\alpha_{\rm MLT}$ = 1.984 (blue dashes).

does not obviously affect the modelling interferences. Comparing with the probability distributions of original sparse grid, GP models are fully sampled in the input range and hence improve the accuracy and precision of inferred parameters. The improvement is remarkable for the stellar age (by 7%). Moreover, the continuous sampling makes it possible to properly estimate some fundamental parameters which are sparse in the grid, e.g., the helium fraction. These results indicate that the method demonstrated in this work is reliable and efficient for interpolating an established model grid and it can improve the modelling solutions because of the statistically-sound sampling.

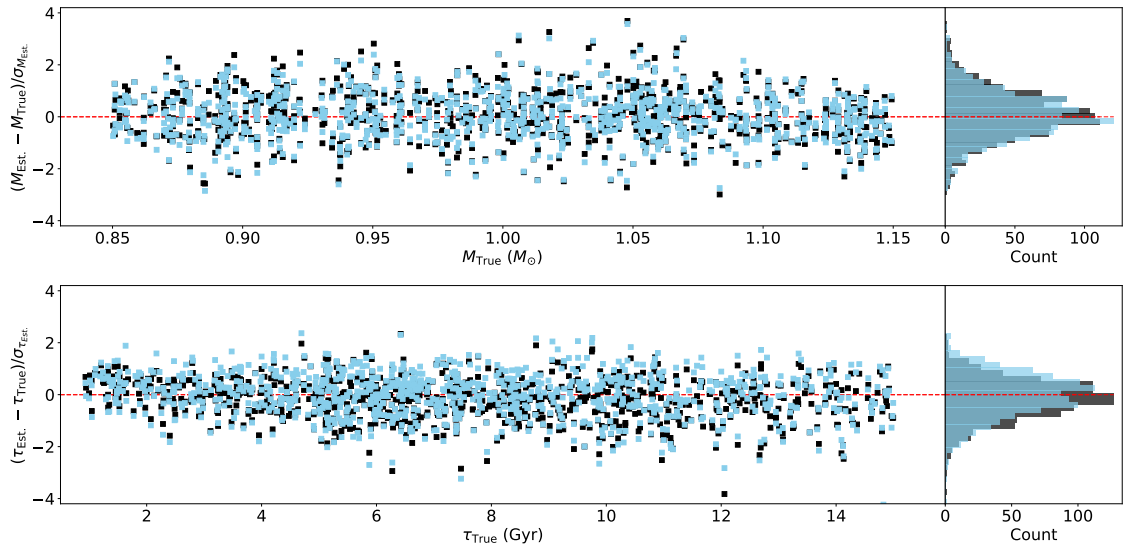

Figure 9. Differences between true and estimated stellar masses and ages over their estimated uncertainty of 1,000 fake stars. Black and blue symbols represent inferences with the original grid and with GP predictions. Count distributions of offsets are demonstrated on the right side.

ACKNOWLEDGEMENTS

This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (CartographY GA. 804752). Development of GPyTorch is supported by funding from the Bill and Melinda Gates Foundation, the National Science Foundation, and SAP.

APPENDIX A: SETUP OF GP MODEL TRAINING

This section includes detailed discussions about the selections of mean function, likelihood and loss function, optimiser, and early stopping.

A1 Mean Function

We first investigate the mean function. As discussed above, the data distribution is generally smooth but complex in some regions of parameter space (e.g., the sub-giant hook). **Although the choice of mean function is not crucial for training GP models, we find that using a constant or a linear mean function leads to a significantly long training time. Hence, we apply a neural network mean function which is flexible enough to manage both simple and complex features to accelerate the training. We adopt an architecture based on that of Lyttle et al. (2021) comprising 6 hidden layers and 128 nodes per layer. All layers are fully-connected and the output of each layer, except for the last, is transformed by the Exponential Linear Unit (ELU) activation function Clevert et al. (2015). The ELU activation function provides a smooth function from inputs to outputs, which is preferred over its more common, faster counterpart, the Rectified Linear Unit (RELU).**

A2 Likelihood and Loss Function

Our training dataset is a theoretical model grid, hence there is no observed uncertainty for each data point, but a tiny random uncertainty exists due to the approximations in the MESA numerical method. We model this using σ_w. This noise model is then assumed to be a Gaussian function with a very small variance. A likelihood specifies the mapping from input values $f(X)$ to observed labels y. We adopt the the standard likelihood for regression which assumes a standard homoskedastic noise model whose conditional distribution is

$$p(y|f(x)) = f + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma^2), \tag{A1}$$

where σ is a noise parameter. We used a small and fixed noise parameter and run a few tests. However, the strict noise parameter makes GP models hard to converge. When this noise parameter is set as free, it reduces to a small value anyway in the training progress because it is data-driven. For this reason, we did not put strict constraint for or prioritise this noise parameter. In practice, we only set up a loose upper limit ($\sigma < 0.1$) to speed up the training. One thing should be noted that a GP model with a large noise parameter is not a proper description for the stellar grid. Because of this, we only adopt GP models with $\sigma \lesssim 10^{-4}$. We train GP models model using the negative logarithm of the likelihood function as the loss function.

A3 Optimiser

We compared two optimisers named SGD and Adam. Here SGD refers to Stochastic Gradient Descent, and Adam is a combination of the advantages of two other extensions of stochastic gradient descent, specifically, Adaptive Gradient Algorithm and Root Mean Square Propagation. The SGD optimiser in the GPyTroch package involves the formula given by Sutskever et al. (2013). The formula makes it possible to train using stochastic gradient descent with momentum thanks to a well-designed random initialisation and a particular type of slowly increasing schedule for the momentum parameter. The application of momentum in SGD could improve its efficiency and make it less likely to stuck in local minimums. On the other hand, the Adam optimiser includes the 'AMSGrad' variant developed by Reddi et al. (2018) to improve its weakness

in the convergence to an optimal solution. With these new developments, the two optimisers give very similar results. We finally choose Adam because it works relatively efficiently and stable. We adaptive learning rate in the training process. Our training starts with a learning rate of 0.01 and decreases by a factor of 2 when the loss value does not reduce in previous 100 iterations.

APPENDIX B: STATE-OF-THE-ART IMPLEMENT FOR LARGE DATASET

In section 4.3, we investigate the strategy for large dataset. We test two State-of-the-art approaches that designed for training big data whose data size is more than the limit of a GP model. Here we mention some details about the two methods and the results.

We first consider the Stochastic Variational GP (SVGP) approach based on the GPyTorch ApproximateGP module. We train our data based on the SVGP example on `https://docs.gpytorch.ai/en/v1.1.1/examples/04_Variational_and_Approximate_GPs/SVGP_Regression_CUDA.html`. SVGP is an approximate scheme rely on the use of a series of inducing points which can be selected in the parameter space. It trains using mini batches and hence is able to deal with large data size. The other key point of SVGP is the number of inducing points. Because the kernel is only built on these points, the number determines the complexity of kernel. When the underline function is simple, for instance, a power law, a small number of inducing points is enough. For our application, a large number of inducing points are required. Underline principles and detailed descriptions of this approach can be found in Hensman et al. (2014). In our tests on 3D problem, we find a practical issue with the SVGP approach. When we load in a large training sample which takes a lot GPU memory, the rest memory can capacitate only 10,000 inducing points. This is to say, we use more training data but sacrifice the kernel complexity. The result shows that using SVGP model does not improve the GP predictions compared with normal GP model. For instance, the 68th, 95th, and 99.7th testing errors for $T_{\rm eff}$ are 2.2, 6.8, and 15.1K (EI = 24.1 K) for the SVGP and 2.0, 5.8, and 15.7 K (EI = 23.5K) for the normal GP model. This is because the evolutionary feature are complex across multiple demissions. Reducing the kernel complexity is not ideal. We conclude that the SVGP is suitable for training large data which have relatively simple variations but not a good choice for training the model grid.

We also investigate another approach designed for large dataset named Structured Kernel Interpolation (SKI GP). SKI GP was introduced by Wilson & Nickisch (2015). It produces kernel approximations for fast computations through kernel interpolation and is a great way to scale a GP up to very large datasets (100,000+ data points). We follow the example on `https://docs.gpytorch.ai/en/stable/examples/02_Scalable_Exact_GPs/KISSGP_Regression.html` to develop our script. We run a few tests to train a 3D SKI GP model with 100, 000 training data. Compare with the Normal GP and SVGP, its testing errors for $T_{\rm eff}$ are slightly improved to 2.0, 6.1, and 14.8K (EI = 22.9K). However, the further test on the 5-demission data is not ideal: a SKI GP model using 100, 000 training data performs much worse than a normal model with only 20,000 training data. The poor behaviour consists with what has been discussed by Wilson & Nickisch (2015): the SKI GP poorly scale to data with high dimensions, since the cost of creating the grid grows exponentially in the amount of data. We attempt to make some additional approximations with the GPyTorch AdditiveStructureKernel module. It makes the base kernel to act as one-dimension kernels on each data dimension and the final kernel matrix will be a sum of these 1D kernel matrices. However, the testing errors are not significantly improved.

REFERENCES

Anzai Y., 2012, Pattern recognition and machine learning. Elsevier
Asplund M., Grevesse N., Sauval A. J., Scott P., 2009, Annual Review of Astronomy and Astrophysics, 47, 481
Bellinger E. P., Angelou G. C., Hekker S., Basu S., Ball W. H., Guggenberger E., 2016, ApJ, 830, 31
Bishop C. M., 1994
Choi J., Dotter A., Conroy C., Cantiello M., Paxton B., Johnson B. D., 2016, ApJ, 823, 102
Clevert D.-A., Unterthiner T., Hochreiter S., 2015, arXiv e-prints, p. arXiv:1511.07289
Dotter A., 2016, ApJS, 222, 8
Ferguson J. W., Alexander D. R., Allard F., Barman T., Bodnarik J. G., Hauschildt P. H., Heffner-Wong A., Tamanai A., 2005, ApJ, 623, 585
Gardner J. R., Pleiss G., Bindel D., Weinberger K. Q., Wilson A. G., 2018, in Advances in Neural Information Processing Systems.
Goodfellow I., Bengio Y., Courville A., 2016, Deep learning. MIT press
Hendriks L., Aerts C., 2019, PASP, 131, 108001
Hensman J., Matthews A., Ghahramani Z., 2014, arXiv preprint arXiv:1411.2005
Ho T. K., 1995, in Proceedings of 3rd international conference on document analysis and recognition. pp 278–282
Hon M., Stello D., Yu J., 2018, MNRAS, 476, 3233
Hon M., Bellinger E. P., Hekker S., Stello D., Kuszlewicz J. S., 2020, MNRAS, 499, 2445
Kingma D. P., Ba J., 2014, arXiv e-prints, p. arXiv:1412.6980
Lyttle A. J., et al., 2021, MNRAS,
Mombarg J. S. G., Van Reeth T., Aerts C., 2021, arXiv e-prints, p. arXiv:2103.13394
Morel P., Thévenin F., 2002, A&A, 390, 611
Paquette C., Pelletier C., Fontaine G., Michaud G., 1986, ApJS, 61, 177
Paxton B., Bildsten L., Dotter A., Herwig F., Lesaffre P., Timmes F., 2011, The Astrophysical Journal Supplement Series, 192, 3
Paxton B., et al., 2013, The Astrophysical Journal Supplement Series, 208, 4
Paxton B., et al., 2015, The Astrophysical Journal Supplement Series, 220, 15
Paxton B., et al., 2018, ApJS, 234, 34
Paxton B., et al., 2019, ApJS, 243, 10
Reddi S., Kale S., Kumar S., 2018, in International Conference on Learning Representations.
Rendle B. M., et al., 2019, MNRAS, 484, 771
Rogers F. J., Nayfonov A., 2002, ApJ, 576, 1064
Schölkopf B., Smola A., Müller K.-R., 1997, in International conference on artificial neural networks. pp 583–588
Sutskever I., Martens J., Dahl G., Hinton G., 2013, in International conference on machine learning. pp 1139–1147
Thoul A. A., Bahcall J. N., Loeb A., 1994, ApJ, 421, 828
Verma K., Hanasoge S., Bhattacharya J., Antia H. M., Krishnamurthi G., 2016, MNRAS, 461, 4206
Williams C. K., Rasmussen C. E., 1996
Wilson A., Nickisch H., 2015, in International Conference on Machine Learning. pp 1775–1784
Wu Y., et al., 2019, MNRAS, 484, 5315

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