Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Machine Learning in Science

Master's Thesis Machine Learning

AutoML for Simulation-based Inference

Swagatam Haldar

31.10.2025

First Supervisor

Prof. Dr. Jakob H. Macke

Machine Learning in Science
Excellence Cluster Machine Learning
Tübingen AI Center
University of Tübingen

Second Supervisor

Prof. Dr. Katharina Eggensperger

Lamarr Institute
TU Dortmund University

Haldar, Swagatam:
AutoML for Simulation-based Inference
Master's Thesis Machine Learning
University of Tübingen
Processing period: 01.05.2025 - 31.10.2025

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst habe, dass ich keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe, dass ich alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe, dass die Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen ist, dass ich die Arbeit weder vollständig noch in wesentlichen Teilen bereits veröffentlicht habe, dass die Inhalte der für die Plagiatsprüfung eingereichten Datei mit den Inhalten des eingereichten gedruckten Berichts übereinstimmen, dass ich keine Inhalte aus Berichten anderer Studierender übernommen habe und ich mir bewusst bin, dass eine Plagiatsprüfung durchgeführt wird. Falls ich KI-basierte Textgeneratoren (z.B. ChatGPT oder GPT-3) für die Erstellung von Textteilen genutzt habe, habe ich sichergestellt, dass diese keine Plagiate enthalten. Mir ist bewusst, dass ich allein für die Qualität der eingereichten Arbeit verantwortlich bin.

Tübingen, 31st Oct, 2025 — Ort, Datum

Swagatam Haldar — Unterschrift

Abstract

Simulation-based inference (SBI) is a general tool for approximating Bayesian inference for complex simulators in science. Modern SBI methods—such as Neural Posterior Estimation (NPE)—rely on neural network based generative models. Building and training such SBI models can involve multiple design choices which can affect the quality of the inferred posteriors. Users typically manually tune them or fall back to default settings in available libraries that often leads to suboptimal performance. Automating the selection of SBI variants and their hyperparameters with AutoML approaches could enhance the effectiveness and accessibility of SBI methods. However, the potential of AutoML for SBI as well as the choice of suitable optimization objectives, has not been systematically investigated yet. We here address this challenge and explore how AutoML can automate a crucial part of the SBI pipeline: selecting and training inference networks, with a focus on NPE. We empirically identify validation loss as a tractable and reliable proxy objective for hyperparameter optimization (HPO), as lower validation losses correspond to posterior estimates that are closer to a reference distribution verified by metrics such as the Classifier Two-Sample Test (C2ST). This finding makes it possible to apply standard HPO methods such as random search and Bayesian optimization to SBI. On 10 widely used benchmark tasks, we show that HPO consistently outperforms current default settings, produces configurations that remain largely stable across simulation budgets ranging from 10^2 to 10^5, and often transfers well across tasks relative to the default. Our framework yields about $\approx 6\%$ C2ST improvement over defaults on average, highlighting AutoML's potential to improve the robustness and effectiveness of SBI and to enable future budget-aware and meta-learning extensions.

Acknowledgements

I am deeply indebted to my research advisors Dr. Jakob H. Macke and Dr. Katharina Eggensperger, for their support, mentorship and wisdom. I would like to express my gratitude to Jakob for giving me this opportunity to work alongside him, and also for encouraging me to bring this work to the community from the very beginning. I want to thank Katharina for serving as the second reader of this thesis, and introducing me to a beautiful community of researchers through the AutoML summer school.

I am also immensely thankful to my primary supervisor and go-to person in this project, Guy Moss. I thank him for agreeing to supervise this Master's thesis and regular check-ins. Many of the ideas and interpretations in this thesis often originated casually in the course of our meetings, and I am particularly impressed with his abilities to draw surprising connections and intriguing interpretations from a few (often a lot!) plots in a notebook. I am grateful to Jakob, Katharina and Guy for the numerous *tokens* they have spent on the discord channel with meticulous critiques and suggestions to refine the preliminary results and drafts. Their thoughts have significantly influenced my scientific thinking throughout the thesis period, and I hope to carry their counsel forward in my future research endeavors.

I thank all members of the *mackelab* for their sweet company at the office and occasionally during lunch at the MPI mensa. Thank you for being a welcoming crowd and for helping me out with many things and providing crucial feedback on the project from time to time. I also thank Jan-Matthis Lückmann for taking time out and sharing ideas from the benchmark paper. I also extend my gratitude to the "Wübben Wissenschaftsstiftung gGmbH" for the student grant for the 24-25 cycle.

Lastly, I want to thank my girlfriend Riya for her kind and joyous companionship, and my parents and sister for their unwavering love and support. They have always gently nudged me to pursue loftier goals. None of this would have been possible without them silently taking care of many (of my own) responsibilities so that I could focus on my academic life.

Table of Contents

Chapter 1
Introduction

Simulation-based inference (SBI) is a method to perform Bayesian inference, i.e., it fits a model to data and also provides uncertainty measures for the fits. SBI has been used to solve such parameter inference tasks with complex simulators in neuroscience, physics and other scientific disciplines (Gonçalves et al., 2020; Dax et al., 2023; Moss et al., 2025). Most modern SBI algorithms, such as Neural Posterior Estimation (NPE) (Papamakarios and Murray, 2016; Lueckmann, Goncalves, et al., 2017; Greenberg et al., 2019; Radev et al., 2020), use neural network based generative models, and have expanded the area of *neural* simulation-based inference.

Neural SBI is flexible in its applicability to arbitrary inference tasks (Cranmer et al., 2019; *SBI Applications Explorer* n.d.). But it has also inherited the complexity of training and tuning the underlying generative models from Deep Learning. There is a zoo of SBI algorithms (Boelts et al., 2024), and each of them, like any other machine learning method, comes with its own set of hyperparameters that can affect its performance. This scenario is deeply linked with the problems tackled by Automated Machine Learning (AutoML) (Hutter, Kotthoff, et al., 2019; Baratchi et al., 2024) that aims to circumvent this issue of hand-tuning hyperparameters.

The goal of this thesis is to bring AutoML to simulation-based inference pipelines. The application of SBI to a novel scientific inference problem involves a number of steps: from setting up the simulator and incorporating relevant domain knowledge in the prior, then choosing and training inference networks, to actually performing inference on one or more observations and finally scrutinizing the results by diagnostics. There are some guidelines and end-to-end example applications (Deistler, Boelts, et al., 2025) to gently onboard practitioners to SBI, but there is still a lack of automated, standardized approaches that can streamline these steps.

An automated way to choose amongst existing SBI variants and their hyperparameters is an attractive proposition for scientists and practitioners to efficiently identify an appropriate method for their inference task. Such a tool would not

only promote broader adoption of SBI methods but also let users focus on solving their scientific problem rather than tuning the inference process itself. It would further support SBI researchers by providing a basis for more robust and reproducible benchmarking of the growing range of available methods. However, the potential of AutoML for these tasks as well as the choice of suitable objectives, has not been systematically studied yet. This thesis addresses this challenge and attempts to automate a crucial part of the SBI pipeline: selecting and training inference networks, particularly for Neural Posterior Estimation (NPE).

Our contributions in this work can be outlined as follows. We empirically identify validation loss as a tractable and reliable proxy objective for hyperparameter optimization (HPO), as improvements in this loss translate into more faithful posteriors compared to a known reference distribution. This enables the application of standard HPO methods such as random search and Bayesian optimization (Bergstra et al., 2011; Akiba et al., 2019) to SBI settings. On 10 widely used benchmark tasks (Lueckmann, Boelts, et al., 2021), we show that (a) HPO can significantly improve over default settings, (b) the optimal hyperparameter setting remains largely stable across simulation budgets ranging from 10^2 to 10^5 samples, and (c) transferring optimized hyperparameter values from an already tuned task to other tasks typically yields better posteriors than the default setting on the SBI benchmark. Overall, we provide a framework for tuning SBI hyperparameters, resulting in $\approx 6\%$ C2ST improvement on average across all tasks and simulation budgets compared to available defaults. These results highlight AutoML's potential to improve the robustness and effectiveness of SBI methods, and they open up further possibilities for leveraging budget-aware extensions for HPO and meta-learning across tasks. Together, the results complement practical SBI guides by enabling practitioners to more effectively use SBI methods.

In the subsequent chapters, we first provide the necessary background on the overall SBI pipeline, Neural Posterior Estimation (NPE) and various metrics and diagnostics to assess the quality of the learned posterior distribution in Chapters 2 and 3. Then we describe our experimental setup and elaborate our main findings in Chapter 4. We also reflect on the results, discuss limitations and potential ideas for future work in Chapter 5.

Chapter 2
Background

In this chapter, we provide some necessary background information to facilitate better understanding of the components needed for AutoML for SBI, and subsequent experimental results. We briefly describe SBI and some fundamental ideas in AutoML methods in Sections 2.1 and 2.2 here.

2.1 Simulation-based Inference (SBI)

Simulation-based inference (SBI) is a collection of tools that perform probabilistic or Bayesian inference in a *likelihood-free* manner. We show a typical SBI pipeline with some of the design choices in Figure 2.1.1. A *simulator* here refers to a model of a physical phenomenon, and it is often implemented as a piece of computer code that can generate data ($\boldsymbol{x}$) given some parameters ($\boldsymbol{\theta}$). With such a simulator and one or more observations from the real-world phenomenon, the inference task is then to find the (hidden) parameters that could reproduce the observed data. It also quantifies uncertainty in multiple plausible parameter values. However, it is often impossible to numerically evaluate the likelihood $p(\boldsymbol{x}|\boldsymbol{\theta})$ (implicitly encoded by the simulator), thereby making traditional Bayesian inference methods inapplicable. In such scenarios, SBI methods can infer a posterior distribution over the parameter space $\boldsymbol{\theta}$, conditioned on a specific observation of interest $\boldsymbol{x}_0$. Rather than relying on tractable likelihoods, SBI typically utilizes samples from the simulator (or an appropriate joint distribution) and learns a conditional distribution (the posterior or the likelihood) by maximizing the log-probability of the (sampled) data under a generative model parameterized by a neural network. To this end, numerous algorithms have recently been proposed that use *neural networks* for density estimation, and have contributed to the general area of *neural* simulation-based Bayesian inference.

The prevalent algorithms are distinct from each other in the sense that each of them prioritize different aspects of the inference task. For instance, Neural Posterior Estimation (NPE) (Papamakarios and Murray, 2016; Lueckmann, Goncalves,

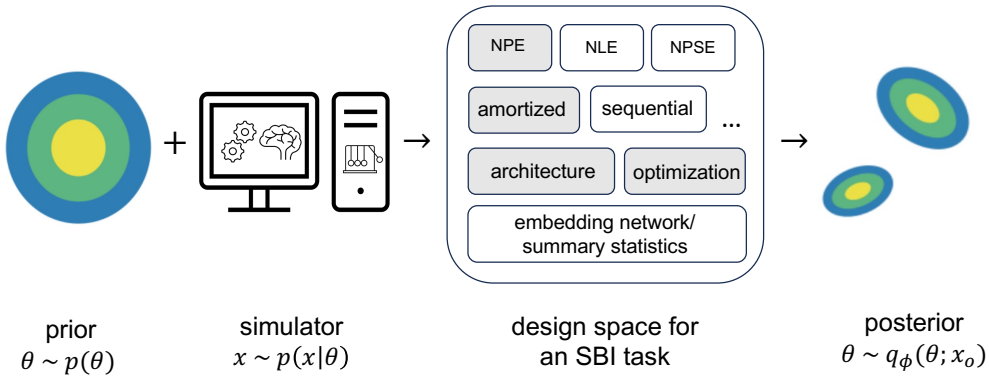

Figure 2.1.1.: SBI pipeline Application of SBI to a task involves a number steps from prior specification, inference method selection and training and finally to validating the obtained posterior. The focus of this work (automating inference network selection) is shaded in grey.

et al., 2017; Greenberg et al., 2019; Radev et al., 2020) approximates the posterior distribution on the parameter space directly, whereas Neural Likelihood Estimation (NLE) (Papamakarios, Sterratt, et al., 2018; Lueckmann, Bassetto, et al., 2019) first models the observation likelihood. In contrast, Neural Ratio Estimation (NRE) (Durkan, Murray, et al., 2020; Hermans et al., 2020) learns a likelihood-to-evidence ratio, allowing inference via binary classification between joint and independent samples. Other aspects include scalability: where amortized inference methods such as NPE are preferable when multiple observations are available, and simulation efficiency: which has motivated sequential extensions and effective modifications of a proposal distribution (Papamakarios and Murray, 2016; Deistler, Goncalves, et al., 2022). NLE and NRE however require an additional step to obtain the posterior by MCMC or variational inference. More recently, Neural Posterior Score Estimation (NPSE) (Geffner et al., 2023; Sharrock et al., 2022) and Flow Matching Posterior Estimation (Wildberger et al., 2023) has been proposed to learn the score (gradient of the log-posterior) with diffusion models or continuous normalizing flows.

In this thesis, we particularly focus on (amortized) NPE, specifically the single-round variant. It uses a normalizing flow (Dinh et al., 2017) based density estimator $q_\phi(\boldsymbol{\theta}; \boldsymbol{x})$ to model the true posterior distribution $p(\boldsymbol{x}|\boldsymbol{\theta})$. The amortized nature ensures that we can plug in any observation $\boldsymbol{x}_o$ to sample from the approximate posterior $q_\phi(\boldsymbol{\theta}; \boldsymbol{x}_o)$ without further training. Given enough data, and a sufficiently flexible model q_ϕ, it can approximate the posterior distribution $q_\phi(\boldsymbol{\theta}; \boldsymbol{x}) \approx p(\boldsymbol{\theta}|\boldsymbol{x})$ arbitrarily well (Papamakarios and Murray, 2016). Popular choices for the density estimator for NPE include mixture density networks (Bishop, 1994), autoregressive flows (Papamakarios, Pavlakou, et al., 2017), neural spline flows (Durkan, Bekasov,

et al., 2019) etc. These architectures allow for fast sampling as well as closed-form density evaluation $q_\phi(\boldsymbol{\theta}; \boldsymbol{x})$ for later use.

2.2 Automated Machine Learning (AutoML)

Although neural SBI methods are highly flexible and can be applied to a wide set of inference tasks (Cranmer et al., 2019), they also share the challenges of training and tuning neural networks. Each SBI algorithm involves numerous design choices that can lead to drastic variability in training stability and posterior quality. This dependence on hand tuning naturally connects SBI to the goals of Automated Machine Learning (AutoML) (Hutter, Kotthoff, et al., 2019; Baratchi et al., 2024), which aims to automate the search for well-performing configurations.

Automated Machine Learning (AutoML) frames the combined problem of algorithm and hyperparameter selection as an optimization task in itself. To perform this optimization, three components need to be defined: the search space $\mathcal{S}$, which specifies all feasible configurations; the search strategy, or how the optimization algorithm explores this space; and the performance measure f, which evaluates the quality of each configuration $s \in \mathcal{S}$. The search space typically includes both categorical and numerical hyperparameters, boolean design choices, and often exhibits conditional dependencies between them in the most general case. Common search strategies range from simple grid or random search to more sophisticated Bayesian optimization (Mockus, 1998; Shahriari et al., 2015) methods. In standard supervised learning, the performance measure or optimization objective is usually the prediction error on a held-out validation dataset.

For this work, we choose to use random search (RS) and a variant of Bayesian optimization (BO): Tree-structured Parzen Estimator (TPE) as baseline search strategies. BO based methods typically fit a probabilistic surrogate model on the evaluations of f until the current iteration. Then they propose the next candidate based on an acquisition function that balances exploration and exploitation. Common choices for the surrogate include Gaussian processes (GP) (Williams and Rasmussen, 2006), Tree-structured Parzen Estimator (Bergstra et al., 2011) where we can tractably compute the acquisition function. While GP models $p(f|s)$ directly, TPE models $p(s|f)$ and $p(f)$. Since TPE naturally supports conditional search spaces (i.e., can place a distribution for each hyperparameter $\in s$ separately to factorize $p(s|f)$ following the dependencies), we choose TPE as a baseline BO based method for AutoML for SBI.

In the next chapter, we describe specific choices for the search space $\mathcal{S}$ and a suitable objective function f for SBI.

Chapter 3
Components of AutoML for SBI

In previous chapters, we motivated the benefits of systematically automating the training of the inference network stage within the SBI pipeline. Now, we outline the key requirements needed to operationalize standard HPO methods for Neural Posterior Estimation (NPE). Specifically, we need to define the search space $\mathcal{S}$; specify a performance metric or objective function f for evaluating different configurations (often abbreviated as configs) s within that space; and select a search or navigation strategy to identify optimal configurations based on evaluations of the objective function.

3.1 Search space

We consider the hyperparameter space $\mathcal{S}_{\text{NPE}}$ for amortized NPE. We have 6 hyperparameters which we divide into two categories: ones defining the network architecture of the normalizing flow, and others for the training process. The hyperparameters along with their ranges are shown in Table 3.1.

Table 3.1.: The hyperparameter space $\mathcal{S}_{\text{NPE}}$ for NPE.

Name	Type	Domain
`density_estimator`	categorical (str)	{`maf`, `nsf`}
`num_transforms`	integer	$[2, 10]$
`hidden_features`	integer	$[10, 100]$
`training_batch_size`	categorical (int)	$\{50, 100, 200\}$
`learning_rate`	float	$[10^{-5}, 10^{-2}]$
`clip_max_norm`	float	$[1, 10]$

Informed by prior work (Lueckmann, Boelts, et al., 2021) and our own initial experiments, we limit the density estimators to masked autoregressive flows (`maf`) and

neural spline flows (`nsf`). For each kind of density estimator, the hyperparameters `num_transforms` and `hidden_features` define the network's *size* or *capacity*. As an example for `maf`, `num_transforms` defines how many masked affine autoregressive transform blocks are used to sample from a base distribution (e.g., gaussian) to the modelled posterior, and the `hidden_features` refers to the number of hidden neurons present in the (masked) linear layers inside each such block.

The `learning_rate` is used for the Adam (Kingma and Ba, 2014) optimizer used to train the networks. The hyperparameter `clip_max_norm` scales the gradients during training such that the total norm of the gradient vector does not exceed `clip_max_norm`. For typical neural network training it is usually set to 1 to avoid instability, but SBI, as observed empirically, can tolerate higher values.

The hyperparameter names in Table 3.1 particularly correspond to the NPE implementation in the `sbi` (Boelts et al., 2024) package. There are other design considerations, such as summarizing the observations ($\boldsymbol{x}$) using an embedding network or summary statistics (Figure 2.1.1) before fitting the density estimator that we are not considering here. Note that the hyperparameters for the embedding network, if present, can also be incorporated in the search space.

3.2 Performance metric or objective function

In supervised learning, the default AutoML objective is the error on a held-out validation dataset (Hutter, Kotthoff, et al., 2019). However for SBI, the final outcome is a probability distribution $p(\boldsymbol{\theta}|\boldsymbol{x})$, and for almost any real world inference task, the ground-truth posterior (or the parameters that generated the observations) is unattainable. We do not thus have an obvious performance metric that we can use as the optimization objective to identify optimal configurations in the search space.

To tackle this issue, we work with a suite of 10 benchmark tasks (Lueckmann, Boelts, et al., 2021) for which we can get samples from the ground-truth posterior, a.k.a. the reference posterior. This provides a way to judge the quality of a posterior (for a specific hyperparameter setting) by metrics such as C2ST (Lopez-Paz and Oquab, 2017). However, we emphasize that for arbitrary tasks, we will not have access to such samples and thus it is not tractable to directly incorporate the reference posterior in the optimization objective. Instead we can now leverage *proxy* measures (e.g., the log-likelihood loss for NPE) and investigate how well it correlates (or continues to correlate while we explicitly optimize the proxy measure in the search space) with any desired quality metric such as the C2ST.

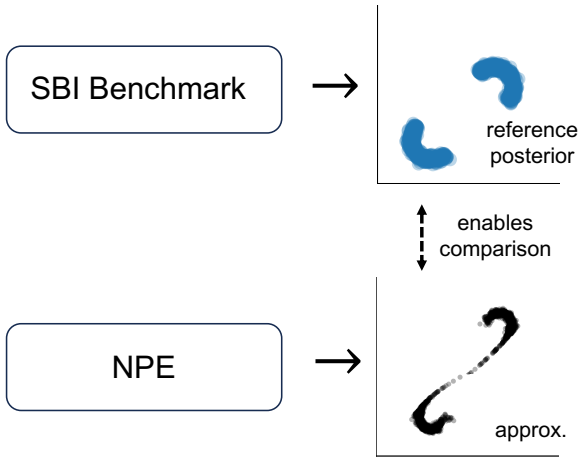

Figure 3.2.1.: No obvious objective for AutoML for SBI: For most real world SBI tasks, the ground-truth posterior (or samples from it) is not available. For the benchmark tasks (e.g., Two Moons (TM) here), we have samples from the reference posterior enabling a way to directly evaluate the quality of the NPE approximation. However, this is not a tractable AutoML objective as for most tasks reference samples are unavailable.

3.2.1 Validation loss as the optimization objective

For our experiments, we use NPE log-likelihood loss evaluated on a separate validation set of $N = 1000$ samples as the objective for the HPO methods. The validation set comes from the prior and simulator (i.e., sampled from the joint distribution), and therefore this objective does not depend on the reference posterior. For each hyperparameter setting or config $s \in \mathcal{S}$, we first train the inference network $q_{\phi(s)}(\boldsymbol{\theta}; \boldsymbol{x})$ for NPE until convergence (or a maximum number of epochs of 500), and then evaluate the loss on the validation set using

$$f(s) := \mathcal{L}_{\text{val}} = -\frac{1}{N} \sum_{i=1}^{N} \log q_{\phi(s)}(\boldsymbol{\theta}_i; \boldsymbol{x}_i). \tag{3.1}$$

We will often omit the explicit dependence for the trained weights ϕ on the config s, but eq. (3.1) is always computed after the training run has been completed. Note that this validation set is completely separate from the internal train-val split used to determine convergence of training. In subsequent chapters, we would often refer to this objective computed in equation (3.1) as simply the validation loss. We empirically establish in Chapter 4 that it is indeed a reliable objective for HPO.

3.2.2 C2ST as a verification metric

If we have samples from the reference posterior available, we can use a metric such as C2ST to evaluate the quality of the NPE posterior approximation. The SBI benchmark provides 10 independent observations for each task, and 10, 000 samples from the true posterior $p(\boldsymbol{\theta}|\boldsymbol{x}_o)$ for each observation $\boldsymbol{x}_o$. C2ST computation requires sampling from the posterior and training a classifier to discriminate between the two sets of samples. If the classifier accuracy is close to random chance (0.5), then the samples cannot be distinguished to be coming from two different distributions. A C2ST value close to 1 indicates poor approximation of the reference by the NPE posterior.

In the experiments, we use a set Y of 1000 samples from the posterior $q_\phi(\boldsymbol{\theta}; \boldsymbol{x}_o)$, and choose a subset X of size 1000, from the available $p(\boldsymbol{\theta}|\boldsymbol{x}_o)$ samples. We use a Random Forest (RF) classifier as the binary discrimination method. The metric is thus computed as

$$\text{C2ST}_k = \text{accuracy}(X_k, Y_k, \text{classifier=RF}) \tag{3.2}$$

for each observation $\{x_o\}_{k=1}^K$. We take an average of equation (3.2) for the available 10 independent observations, i.e.,

$$\text{C2ST} = \frac{1}{K} \sum_{k=1}^{K} \text{C2ST}_k, \tag{3.3}$$

where $K = 10$. This provides a way to keep track of how well improvements in the validation loss objective (f) translate into posterior quality.

3.3 Search strategies

Now that we have defined our search space ($\mathcal{S}_{\text{NPE}}$) and the objective function $f(s)$ (eq. **(3.1)**) for any $s \in \mathcal{S}_{\text{NPE}}$, we need a way to find optimal configurations s^* that minimize the objective. As baselines, we use random search (RS) and TPE, a variant of Bayesian optimization (BO) as the search strategies. For random search (RS), we sample 200 configurations from the search space and evaluate f for them independently in parallel. For TPE, we use the implementation available in the `optuna` (Akiba et al., 2019) library. We run it for 100 trials. The primary goal of these strategies is task-specific HPO: to find best hyperparameter settings for each of the benchmark tasks, and also to demonstrate benefits of doing HPO in terms of C2ST improvement.

We also implement a strategy aimed at identifying a single best configuration over all available benchmark tasks. If we could find a configuration that outperforms the current default for all tasks, it can be deployed as a new universal default for the SBI benchmark. We propose to use a straighforward objective by taking the average of equation **(3.1)** over all benchmark tasks to search for such a config. However, since this aggregated objective is computationally more expensive (each evaluation of objective requires training NPE separately for all 10 tasks) than the task-specific one, we apply it only post-hoc and for random search, once the objective evaluations are available for all configurations.

Chapter 4
Results

We first describe the experimental setup and few notations. We also briefly mention the research questions we will attempt to answer through these experiments. Then in the subsequent sections we elaborate more on our primary findings.

4.1 General setup

We run HPO using 3 strategies (RS, BO and agg., as described in Section 3.3) for NPE on 10 synthetic tasks from the SBI benchmark. The task names, along with their parameter ($\boldsymbol{\theta}$) and observation ($\boldsymbol{x}$) dimensions are listed in Table 4.1.

Table 4.1.: SBI benchmark tasks.

	Task (t)	dim($\boldsymbol{\theta}$)	dim($\boldsymbol{x}$)
1	Two Moons (TM)	2	2
2	Gaussian Linear (GL)	10	10
3	Gaussian Linear Uniform (GLU)	10	10
4	Gaussian Mixture (GM)	2	2
5	Bernoulli GLM (BGLM)	10	10
6	Bernoulli GLM Raw (BGLMR)	10	100
7	SIR	2	10
8	Lotka Volterra (LV)	4	20
9	SLCP	5	8
10	SLCP Distractors (SLCP-D)	5	100

In SBI, depending on how costly it is to sample from the simulator $p(\boldsymbol{x}|\boldsymbol{\theta})$, we can have a varied number of training samples a.k.a. simulation budget. We also investigate the effect of simulation budget as a separate dimension for HPO. In our experiments, we attempt to train the inference network (for every configuration)

for four simulation budgets (b): $10^2, 10^3, 10^4$ and 10^5. However, we observe that for a very low simulation budget (e.g., 100), often we fail to learn a reasonable posterior (indicated by C2ST values close to 1), on the other hand, for very high simulation budgets ($100,000$), often the training of a single configuration takes very long, prohibiting the application of sequential HPO methods such as BO for 100 trials.

We use the current default hyperparameters ($C_{\text{def}} \in \mathcal{S}_{\text{NPE}}$) for NPE in `sbi` as a baseline configuration . For the three search strategies described in Section 3.3, we denote the best configurations (s^*) found by each of them as $C_{\text{RS}}(t, b)$, $C_{\text{BO}}(t, b)$ and $C_{\text{agg}}(b)$ respectively. While RS and BO (TPE) strategies are applied separately for each task t, the aggregate or "all-tasks-combined" objective identifies a single top configuration for all tasks. To make the dependence on the simulation budget (b) explicit, we also indicate it within the brackets here, but will be omitted later.

For each $\langle t, b \rangle$ tuple, we run RS and BO once (with a single random seed) to identify the top-performing configuration for each task and budget. Unless otherwise specified, we then re-evaluate each such selected top configuration (across tasks or simulation budgets when required, e.g., for investigating transferability) using 5 different seeds to estimate the mean and standard deviations in the C2ST values.

4.2 Research questions

We investigate the following main research questions through our experiments.

RQ1. Can we find a configuration that leads to an accurate posterior (i.e., with good C2ST) by just optimizing the validation loss objective in the search space?

RQ2. Do we improve (and by how much) on the C2ST relative to the current default (C_{def}) in `sbi` if we do task-specific HPO?

RQ3. Can we find a single best configuration that outperforms C_{def} for all tasks?

RQ4. Are optimal configurations found at each budget b stable across the range of simulation budgets?

RQ5. Do optimal configurations transfer well across tasks, i.e., if the optimal hyperparameters from task t_1 (source) are used to train the NPE for task t_2 (target), does it stil achieve better-than-default performance for t_2?

4.3 Validation loss is a reliable AutoML objective

In Section 3.2, we proposed to use the validation loss **(3.1)** as a tractable objective to identify top-performing configurations for AutoML for NPE. First, we investigate

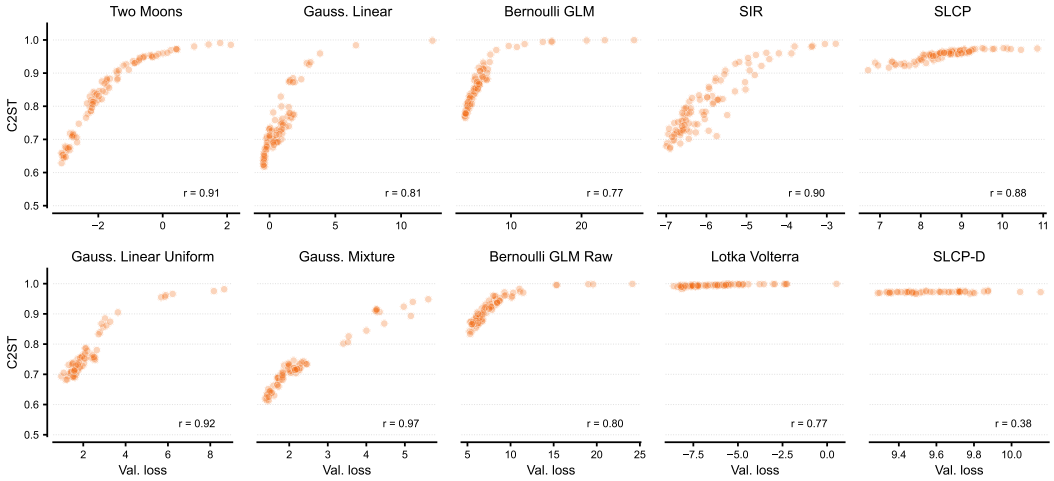

Figure 4.3.1.: Val. loss and C2ST are highly correlated in the search space for most tasks. The figure shows the validation loss and C2ST on 100 configurations from random search for all 10 tasks for a simulation budget of 1000.

whether validation loss is a reliable proxy for improvements in posterior quality verified by C2ST **(3.3)**.

4.3.1 Correlation between val. loss and C2ST

In Figure 4.3.1, we present correlation plots for all 10 tasks (for 1000 simulation budget). In each of them, we plot the val. loss on the x-axis, and the corresponding C2ST achieved on the y-axis for the 100 configurations for random search. Typically all tasks follow the trend that lower val. losses correspond to lower C2ST values (e.g., high correlation coefficients as shown in bottom right for each plot). For tasks which are known to be hard to learn for NPE (such as LV and both SLCP tasks), we see almost horizontal lines, i.e., C2ST does not improve much while val. loss goes down. Since C2ST is bounded above by 1.0, we notice this saturation for high loss values for other tasks (e.g., Ber. GLM) as well. However, on increasing budgets (10^4 or 10^5), this correlation improves (Figures A.2.2 and A.2.3 in Appendix). Note that, in this figure, the configurations are sampled randomly in the search space, but we observe similar high correlation for configs proposed during sequential optimization with BO. Also, we observe that each task operates on a different scale and range on the val. loss axis, so the objective values across tasks are not directly comparable.

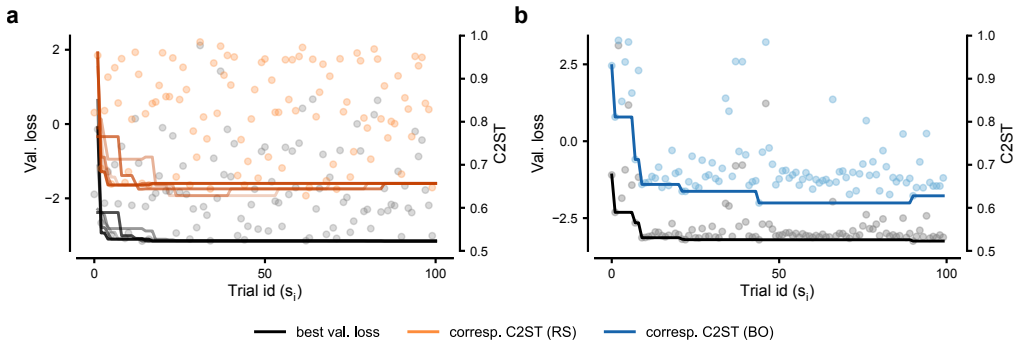

Figure 4.3.2.: Validation loss is a reliable optimization objective. The figure shows how validation loss (explicitly optimized) and C2ST evolve with the number of trials for **a**) Random search (RS), and **b**) Bayesian optimization (TPE). For both, we see choosing the config with best val. loss also leads to good C2ST values. Note that for RS, since there is no fixed order for the trials, we show 5 random orderings for the solid lines. Both plots are for Two Moons task with 1000 simulation budget.

4.3.2 Optimization trajectories

In Figure 4.3.1, we saw that the proposed objective and posterior quality are highly correlated in both good (low loss) and bad regions (high loss) of the search space. Now, we want to investigate whether the correlation persists while we explicitly optimize for val. loss in the search space. In Figure 4.3.2, we inspect if we do reach configurations with reasonably well C2ST by just optimizing for val. loss with RS (left) and BO (right). Here, we track the best val. loss observed until trial i (black solid line), and the corresponding C2ST (orange or blue solid line). We also show the actual trial's loss and C2ST by grey and orange (or blue) dots. Since RS, does not impose any explicit ordering among the trials, we show the lines for 5 shuffled orderings. For BO, the trials are naturally ordered as the sequential optimization progresses. We note a few interesting observations from this Figure. First, we see for both RS and BO, if we choose the config with best val. loss, we also improve on the C2ST (not explicitly optimized) as the coloured lines go down. Interestingly, the C2ST values can show slight degradation (e.g., the blue line shifts a bit up towards the end) when BO found a config with lower loss. This is also observed in few other cases, however, the increase in C2ST is not substantive. In other words, the optimization first reaches a region of the search space with good C2ST values. Then, the correlated nature of the space ensures we do not end up too far as many configs with low loss and similar C2ST exist, and choosing any one of them is sufficient. Second, we see the latter trials from BO tend to highly concentrate while,

as expected, the trials from RS are scattered over the entire range. Finally, for BO we observe a sharp decrease in both loss and C2ST in the first 10-20 trials for most tasks, while for RS depending on orderings it may or may not be the case. However, we do not see a high difference in quality (by C2ST) for the optimal configs found by RS and BO (discussed in Section 4.4).

4.4 C2ST improvements by HPO

Now, we study the performance of the best (or optimal) hyperparameter configs found by the three strategies (RS, BO and aggregate) relative to the current default setting in `sbi` (C_{def}). We compare the strategies by measuring the quality of the NPE posteriors learned with the best configs ($C_{\text{RS}}, C_{\text{BO}}$ and C_{agg}) by C2ST **(3.3)**. Note that for the 10^5 budget case, we are not able to retrieve optimal configs using BO using our setup since evaluating each trial takes longer (runtimes in Appendix A.1) and we frequently run into rejection sampling issues with a poor posterior.

Table 4.2.: Change (%) in C2ST (↓ is better) for the three strategies with respect to current default setting in `sbi` across all benchmark tasks by budget. The best strategy in each row is highlighted in **bold**.

	Strategies		
Budget	RS	BO	Agg.
10^2	**−1.35**	−1.16	0.32
10^3	−6.69	**−7.87**	−3.91
10^4	−8.42	**−9.04**	−6.10
10^5	**−7.04**	—	−5.31

Table 4.2 shows the percentage improvements in C2ST relative to the default combined across all 10 tasks. We stratify it by simulation budgets (rows), as we observe non-uniform gains of doing HPO with the budgets. For a very low simulation budget like 100, we do not see much improvement in C2ST after doing HPO. Typically with 100 simulations, the posteriors are already poor, i.e., close to $0.9 - 1$ C2STs for almost all tasks (leftmost plot in Figure 4.4.1 and in Appendix A.3.2). The C2ST improvements become significant at a budget of 1000 simulations, i.e., we see $\approx 7 - 8\%$ drop for task-specific HPO by RS and BO, and $\approx 4\%$ drop using the C_{agg} config. With higher budgets, the improvements still go up but also saturate quickly as the default C2ST also improves substantially (C_{def} dotted line shifts down in the plots in Fig. 4.4.1).

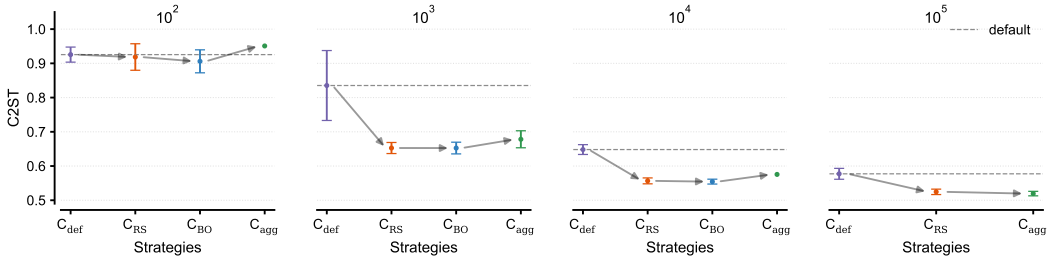

Figure 4.4.1.: HPO strategy comparison: For sufficiently large budgets, all HPO strategies (random search, bayesian optimization per task, as well as random search aggregating all the benchmark tasks) outperform the current `sbi` defaults. However, there is not a big benefit C2ST-wise to perform BO over random search over the tasks (Two Moons in this figure) and hyperparameters considered here.

As a representative example, we show the C2STs for the four configs for the Two Moons (TM) task in Figure 4.4.1. Each plot (column) in the figure corresponds to a different simulation budget. The error bars around each point is the standard deviation for the C2ST after training NPE on the config with 5 different seeds.

When doing HPO with 10^3 simulations for instance, the C2ST improves from 0.83 (C_{def}) to ≈ 0.65 for RS and BO, and to 0.68 for C_{agg}. We see a slight degradation in C2ST for C_{agg} relative to C_{RS} or C_{BO}. This trend is also observed for other tasks and budgets. We here emphasize that although C_{agg} is not universally better than C_{def} for arbitrary task and budget pairs, for most tasks it still outperforms the default or is only slightly worse. Also, for many tasks (TM, SIR, GM) we observe high variance in C_{def} while configs from HPO are more resistant to random seed variability.

4.5 Scaling behaviour of optimal configs

We have seen that task-specific HPO improves inference quality as reflected in C2ST values. However, if one wants to do HPO in the high budget regime ($\geq 10k$ simulations), it often becomes infeasible due to high training cost (Appendix A.1), i.e., evaluation of objective for each configuration. From the previous experiments, for each task t we already obtained four optimal configurations $C_{\text{RS}}(t, b)$ for $b \in \{10^2, 10^3, 10^4, 10^5\}$. Although the configurations differ in their specific hyperparameter values (e.g., in Table 4.3), we now examine their performance across budgets. In particular, we ask whether it is possible to avoid expensive HPO at high budgets by reusing the best configuration found at a lower simulation budget.

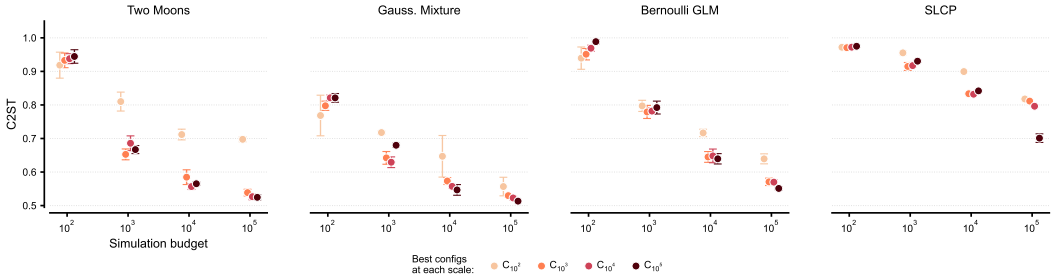

Figure 4.5.1.: Most optimal configs remain consistent performers across budgets. Optimal configurations (as found by Random Search) and performance on their respective budgets, as well as other budgets. Except for very low simulation budgets (100), the C2STs of optimal hyperparameters for learning a task are robust to the simulation budget. Here we show for 4 tasks (others in Appendix).

Figure 4.5.1 illustrates the robustness of hyperparameters to simulation budgets for four representative tasks. We report results for configurations obtained via random search, since Bayesian optimization results were not available for the 10^5 budget. Optimal configurations at each budget are shown in shades of orange, with lighter to darker tones representing budgets from 10^2 to 10^5. We note a few consistent trends. Except for very low simulation budgets (100, lightest points in the Figure), the C2STs of the optimal configurations cluster closely together. For instance, for the Two Moons task, we see a clear deviation of the optimal config at 100 simulations from others, while HPO on remaining budgets yields similar results. We also notice similar clustering of C2STs for other tasks (Figure A.4.1 in Appendix). In conclusion, these results suggest that conducting HPO at a moderate simulation budget (e.g., $1,000$ or $10,000$ simulations, if feasible) is sufficient in these benchmark tasks. The resulting configuration can then be reliably reused to learn the same task in the high-budget regime without noticeable degradation in posterior quality.

Table 4.3.: We show the optimal configs (from RS) for the Two Moons task at different budgets. Even though there are differences in hyperparameter values, they are quite close in terms of C2STs achieved across budgets (except for $C_{\text{RS}}(\text{TM}, 10^2)$). For corresp. C2STs, please refer to Tab. A.3 in Appendix.

Config	`est.`	`#tfs.`	`#feats.`	`lr`	`#batch`	`clip norm`
$C_{\text{RS}}(\text{TM}, 10^2)$	`maf`	10	100	0.003	100	7.0
$C_{\text{RS}}(\text{TM}, 10^3)$	`nsf`	10	10	0.003	50	1.0
$C_{\text{RS}}(\text{TM}, 10^4)$	`nsf`	5	100	0.0007	128	3.0
$C_{\text{RS}}(\text{TM}, 10^5)$	`nsf`	10	100	0.001	200	3.0

4.6 Cross-task transferability of optimal configs

So far we focused on doing HPO for the benchmark tasks individually and saw that it boosts C2ST relative to the default, and the performance of optimal configurations remain relatively stable with simulation budgets. Now we address the question whether we can completely avoid doing HPO when we encounter a new SBI task given we already have a knowledge *database* of HPO results available for many known tasks. Specifically, we want to investigate how well optimal configurations from one task perform on another. We evaluate it by inspecting if the "ported" optimal configuration still yields better C2STs when compared to the default, i.e., does not suffer from overfitting to a single task.

Figure 4.6.1 shows two such transfer examples where the source tasks are Two Moons (top) and Gaussian Linear (bottom). We choose budget $\geq 10^3$ as lower budgets resulted in poor posteriors. Here, we first get the optimal (random search) config for the source task, and use it to train NPE on the target task, and compare it with the default. On the x-axis we lay out all 10 target tasks in an arbitrary order, and the y-axis shows the difference in C2STs. The light green shaded region ($y \leq 0$) indicates the ported config could achieve a lower C2ST than C_{def}. Note that we have also added the source task on the x-axis, e.g., TM to TM, (which is just task-specific HPO) for completeness.

With Two Moons (TM) as source task (top plot in Fig. 4.6.1), we notice the C2ST degrades mainly for three other tasks: GL, BGLM and BGLMR. For other tasks, it is comparable or better than C_{def}. This can be explained by inspecting the optimal density estimator choices of these tasks (Table 4.4). For 10^4 budget, the optimal configs of GL, BGLM and BGLMR use `maf` while the ported config from TM uses `nsf`. This also explains why the C2ST degrades for TM when we use the ported config from GL. Same holds for the GM task which also prefers `nsf` in its optimal config. Interestingly, we notice the C2ST for BGLM also degrades (bottom middle) while it prefers `maf`. Upon further inspection, we noticed that the ported config uses a smaller network (2 transforms with 20 hidden features), while the optimal for BGLM is a slightly larger network (5 transforms and 50 hidden features, which is the same network used in C_{def}). So density estimator choice alone is not always sufficient to explain transfer performance. We highlight here that for both GL and BGLM we do not significantly improve over the default C2ST upon doing HPO (Figure A.3.2 in Appendix). The consistent large error bars in Figure 4.6.1, e.g., for TM, GM and SIR tasks with 10^3 simulations in both plots, are often an artefact of the high variance observed for the default C2ST (also observed in Figures A.3.2 and A.3.1).

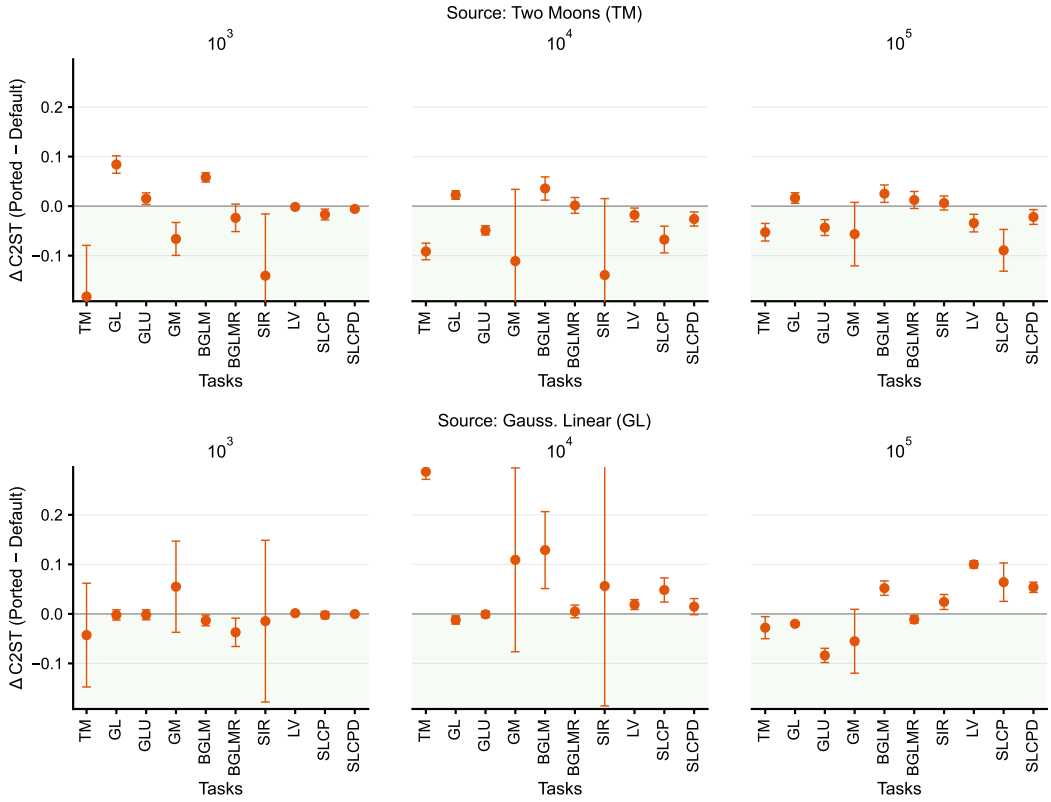

Figure 4.6.1.: Portability of optimal hyperparameters across tasks. Performance of the optimal parameters as learned on a task (source) when applied for training the inference network on a new task (targets on x-axis), as compared to the `sbi` default. On average, optimal parameters can be ported with improved performance, but there is some task variability as explained in the text. Here we show for two source tasks: Two Moons and Gaussian Linear. The consistent large error bars, e.g., for TM, GM and SIR tasks are often an artefact of the high variance for default C2ST (also observed in Figures A.3.2 and A.3.1).

As another example, we turn to the task pair GL (source) and GLU (target). The tasks are only distinct with respect to the choice of the prior distribution ($\mathcal{N}(\mathbf{0}, 0.1 \odot \mathbf{I}_{10})$ for GL and $\mathcal{U}[-1, 1]$ for GLU), but prefer different density estimators for learning (Figure 4.6.2). For GL, the best configs (for budget $\leq 10^4$) mostly use `maf`, while for GLU, `nsf` configs are consistently better for all budgets. Thus, when we port the optimal config (using `maf`) from GL to GLU, we do not see much or any improvement. However for the 10^5 budget, the C2ST for GLU drops significantly since now the ported config from GL uses `nsf`.

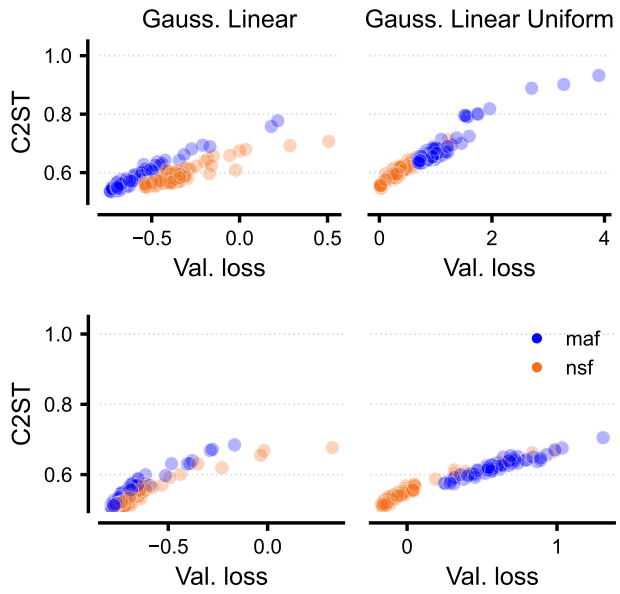

Figure 4.6.2.: Similar tasks might prefer totally different density estimators. The tasks GL and GLU are different only w.r.t the choice of the prior. We show the correlation between C2ST and val. loss for different random configs in the search space coloured separately for maf and nsf. The plots are for a simulation budget of 10^4 (top) and 10^5 (bottom). For GL, the best configs mostly use maf, while for GLU, nsf configs are significantly better. This can be explained by the fact that the gaussian prior is a conjugate prior for GL task, and it likely aids autoregressive flows. Note that for 10^5 budget (bottom left), the gap closes completely for GL, and in fact the optimal RS config uses nsf (Tab. 4.4).

Table 4.4.: Optimal configs (from random search) for the source and target tasks. We also include C_{def} for reference.

Config	`est.`	`#tfs.`	`#feats.`	`lr`	`#batch`	`clip norm`
C_{def}	`maf`	5	50	0.0005	200	5.0
$C_{\text{RS}}(\text{GL}, 10^4)$	`maf`	2	20	0.0001	50	7.0
$C_{\text{RS}}(\text{GLU}, 10^4)$	`nsf`	2	10	0.0003	200	5.0
$C_{\text{RS}}(\text{BGLM}, 10^4)$	`maf`	5	50	0.0005	100	3.0
$C_{\text{RS}}(\text{GL}, 10^5)$	`nsf`	5	10	0.00001	50	5.0
$C_{\text{RS}}(\text{GLU}, 10^5)$	`nsf`	2	10	0.0003	200	5.0

4.7 Hyperparameter characteristics

Lastly, we look at the effect of different hyperparameters on the validation loss objective in the search space. For this we use the fANOVA (Hutter, Hoos, et al., 2014) which finds the marginal importance of each hyperparameter by a random forest approximation on the (100) trial outcomes. We show one such example for the Two Moons task with 10^4 budget in Figure 4.7.1.

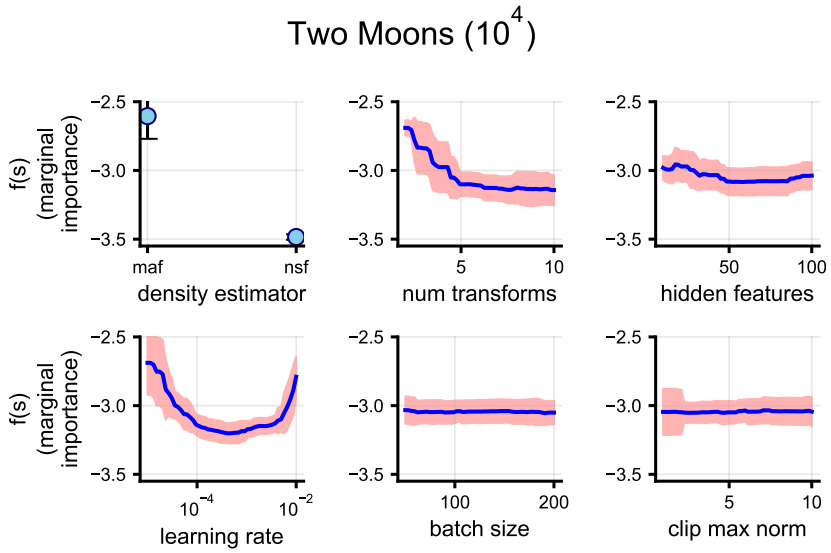

Figure 4.7.1.: Marginal objective values (lower the better) for each hyperparameter in the search space. We observe that network type and capacity hyperparameters show high variability, while few others are relatively ineffectual.

As we have seen before, the density estimator choice has a large effect, and in particular, using `nsf` brings down the objective considerably, as also observed in Lueckmann, Boelts, et al., 2021. We also note that higher number of transforms (i.e., capacity) are often needed to fit larger training simulations. The learning rate hyperparameter shows plenty of variability with the objective becoming worse at very low or very high ends. We also observe that batch size and clip max norm have relatively smaller effect.

We supply the fANOVA plots for GL task in Figure 4.7.2 which shows some task variability as also noted in Section 4.6. Note that here the $y-$axis is scaled differently for each hyperparameter. We notice that GL prefers smaller `maf` networks, while batch size and clip max norm still remain relatively flat in the search space.

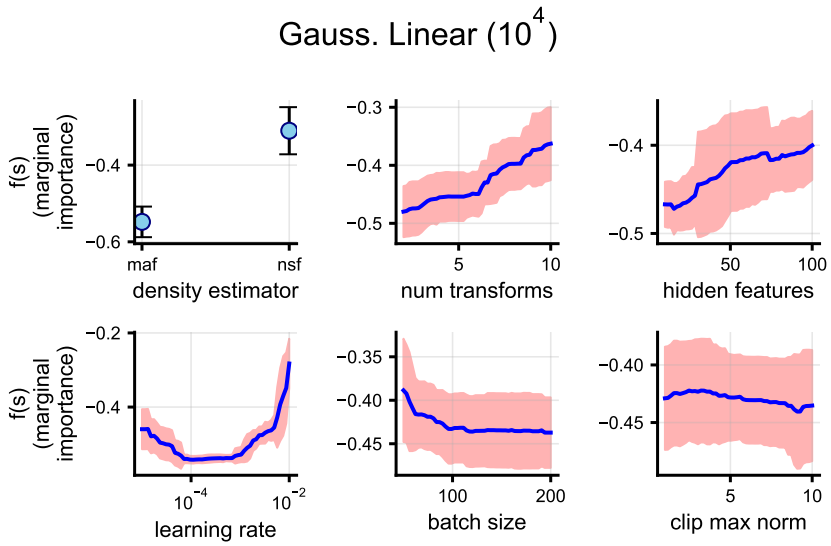

Figure 4.7.2.: For the Gauss. Linear task, we observe a few differing trends compared to Two Moons. For example, it prefers `maf` and smaller networks.

Chapter 5
Discussion

Here, we summarize and discuss limitations of our results and provide an outlook on future research directions. In this work, we brought AutoML to simulation-based inference pipelines. We asked whether we can leverage standard principles from AutoML to automate some part of the SBI pipeline, and focused particularly on choosing hyperparameters for training the inference network for Neural Posterior Estimation (NPE). Although we explicitly focus on NPE, the ideas such as search space design, validation loss as the objective, using C2ST for inspection etc. can be extended to other SBI methods (e.g., NLE, NRE, NPSE) in a straighforward manner. These methods come with many design choices that affect the inference quality in unintuitive ways, and automating the selection of them would ultimately improve the accessibility and robustness of SBI.

We began by specifying and exploring the hyperparameter space for NPE. The core challenge that makes AutoML for SBI distinct from traditional supervised learning is the absence of the ground-truth posterior distribution for the observed data. We proposed to use the validation loss objective for NPE to navigate the search to find optimal configurations. This objective does not depend on the ground-truth so can be operationalized for any SBI inference task. We found that this objective is highly correlated with posterior quality (via C2ST) on the SBI benchmark, and also at different simulation budgets. We emphasize here that this is specifically observed for NPE, where inference network tries to directly model the posterior, and under some assumptions (Papamakarios and Murray, 2016) $q_\phi(\boldsymbol{\theta}; \boldsymbol{x}) \approx p(\boldsymbol{\theta}|\boldsymbol{x})$ for any $\boldsymbol{x}$. The C2ST (Lopez-Paz and Oquab, 2017) also measures an aspect of the posterior (sample diversity or coverage of all modes). The high correlation signifies we often get a good approximate of the posterior by NPE for the SBI benchmark tasks. For other algorithms, such as NLE where we model the likelihood $p(\boldsymbol{x}|\boldsymbol{\theta})$ instead of the posterior, or NPSE where we learn the score of the posterior $\nabla_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\boldsymbol{x})$ involve another layer of approximation to get to the posterior. For these methods,

the network's loss can still likely be used as an AutoML objective, but the degree of correlation with posterior quality needs to be inspected first.

For NPE, we performed HPO in a 6-dimensional hyperparameter space. Through fANOVA (Hutter, Hoos, et al., 2014) plots, we assessed the sensitivity of the loss to each hyperparameter, and found that the network type and capacity hyperparameters as well as `learning_rate` often had higher variation on the range, while `batch_size` and `clip_max_norm` were relatively ineffectual.

From our empirical results, we did not observe a substantive difference in C2ST between random search and Bayesian optimization, with the optimal configs from BO performing slightly better (Table 4.2). We note a caveat here: for random search, we sampled configurations on a grid for some numerical hyperparameters (e.g., `hidden_features` was sampled from $\{10, 20, 50, 100\}$), while BO could sample from the full range of integers $[10, 100]$. This might have hurt the reachability (and thus expressivity) of random search and could explain the slight degradation in C2ST relative to BO. While random search enjoys the benefits of being embarrassingly parallel, we consistently noticed that BO often concentrates to a region of the space with low loss considerably quickly (10-25 trials, with the first 10 trials being random to initialize the TPE surrogate). Although we tried to run the optimization for 100 trials for most tasks, it might not be necessary for BO, especially with higher budgets as it significantly improves all configurations. As another caveat, we note the small dimensionality of our hyperparameter space (6) that could have benefitted the search for all strategies to find good configs quickly. For subsequent results, we explicitly used the optimal configs obtained from random search but many of the conclusions (budget stability, transferability etc.) also hold for best configs produced by Bayesian optimization.

In our experiments, we chose to work with 10 SBI benchmark tasks, and saw that we could significantly improve the C2STs relative to the current best known setting or the default configuration. We note here that this C2ST improvement was not homogeneous both across tasks and simulation budgets. In particular, highest gains from HPO were achieved with moderately high budgets (10^3 and 10^4). For a very low budget (10^2) the quality was too poor, and at a very high budget (10^5), most strategies performed equally good and often the default was very close in C2ST. The C2STs optimal configurations found at each budget were typically found to concentrate together as well (Section 4.5).

We now note some characteristics of the tasks that do not show much C2ST improvement after HPO. They include SLCP (with distractors) and Lotka Volterra which are known to be hard inference tasks for NPE. We see Gaussian Linear and

Bernoulli GLM also do not exhibit much benefits of doing HPO relative to the default. The difference however is that with higher simulation budgets, the C2ST improves considerably here, unlike SLCP-D and LV where the C2ST stagnates at around 0.8-0.85. This implies some SBI tasks might be bottlenecked more by the budget rather than choice of hyperparameters. A comprehensive list and figures can be found in the Appendix (Table A.2 and Fig. A.3.1). These results suggest that few tasks in the SBI benchmark continue to be hard for NPE (even when supplemented by HPO), and encourage to look for other SBI methods. Future work can explore the linking of sequential methods to the budget extrapolation characteristics observed in our results.

We highlight here that we could not find a best configuration (or a new default) that is universally better for all tasks using the aggregate strategy. However, we did observe that optimal config of one task is often portable, but there is some task-specific nuances like density estimator preference or inductive biases for small networks as seen in Section 4.6 which make it hard to draw general conclusions.

In conclusion, this thesis demonstrated that established AutoML methods can be effectively adapted to automate the selection and training of inference networks for NPE, an amortized SBI method. By empirically validating validation loss as a reliable optimization objective and showing consistent improvements across the SBI benchmark, this work highlights a encouraging path toward making neural SBI more robust, effective and accessible and ultimately empowering scientists to focus on discovery rather than on tuning.

References

Akiba, Takuya, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama (2019). "Optuna: A Next-Generation Hyperparameter Optimization Framework". In: *The 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623–2631.

Baratchi, Mitra, Can Wang, Steffen Limmer, Jan N Van Rijn, Holger Hoos, Thomas Bäck, and Markus Olhofer (2024). "Automated machine learning: past, present and future". In: *Artificial intelligence review* 57.5, p. 122.

Bergstra, James, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl (2011). "Algorithms for hyper-parameter optimization". In: *Advances in neural information processing systems* 24.

Bishop, Christopher M (1994). "Mixture density networks". In.

Boelts, Jan, Michael Deistler, Manuel Gloeckler, Álvaro Tejero-Cantero, Jan-Matthis Lueckmann, Guy Moss, Peter Steinbach, Thomas Moreau, Fabio Muratore, Julia Linhart, et al. (2024). "sbi reloaded: a toolkit for simulation-based inference workflows". In: *arXiv preprint arXiv:2411.17337*.

Cranmer, Kyle, Johann Brehmer, and Gilles Louppe (2019). "The frontier of simulation-based inference". In: *Proceedings of the National Academy of Sciences*.

Dax, Maximilian, Stephen R Green, Jonathan Gair, Michael Pürrer, Jonas Wildberger, Jakob H Macke, Alessandra Buonanno, and Bernhard Schölkopf (2023). "Neural importance sampling for rapid and reliable gravitational-wave inference". In: *Physical review letters* 130.17, p. 171403.

Deistler, Michael, Jan Boelts, Peter Steinbach, Guy Moss, Thomas Moreau, Manuel Gloeckler, Pedro LC Rodrigues, Julia Linhart, Janne K Lappalainen, Benjamin Kurt Miller, et al. (2025). "Simulation-Based Inference: A Practical Guide". In: *arXiv preprint arXiv:2508.12939*.

Deistler, Michael, Pedro J Goncalves, and Jakob H Macke (2022). "Truncated proposals for scalable and hassle-free simulation-based inference". In: *Advances in neural information processing systems* 35, pp. 23135–23149.

Dinh, Laurent, Jascha Sohl-Dickstein, and Samy Bengio (2017). "Density estimation using Real NVP". In: *International Conference on Learning Representations*.

Durkan, Conor, Artur Bekasov, Iain Murray, and George Papamakarios (2019). "Neural spline flows". In: *Advances in neural information processing systems* 32.

Durkan, Conor, Iain Murray, and George Papamakarios (2020). "On contrastive learning for likelihood-free inference". In: *International conference on machine learning*. PMLR, pp. 2771–2781.

Geffner, Tomas, George Papamakarios, and Andriy Mnih (2023). "Compositional score modeling for simulation-based inference". In: *International Conference on Machine Learning*. PMLR, pp. 11098–11116.

Gonçalves, Pedro J, Jan-Matthis Lueckmann, Michael Deistler, Marcel Nonnenmacher, Kaan Öcal, Giacomo Bassetto, Chaitanya Chintaluri, William F Podlaski, Sara A Haddad, Tim P Vogels, et al. (2020). "Training deep neural density estimators to identify mechanistic models of neural dynamics". In: *Elife*.

Greenberg, David S., Marcel Nonnenmacher, and Jakob H. Macke (2019). "Automatic Posterior Transformation for Likelihood-Free Inference". In: *International Conference on Machine Learning*.

Hermans, Joeri, Volodimir Begy, and Gilles Louppe (2020). "Likelihood-free mcmc with amortized approximate ratio estimators". In: *International conference on machine learning*. PMLR, pp. 4239–4248.

Hutter, Frank, Holger Hoos, and Kevin Leyton-Brown (2014). "An efficient approach for assessing hyperparameter importance". In: *International conference on machine learning*. PMLR, pp. 754–762.

Hutter, Frank, Lars Kotthoff, and Joaquin Vanschoren, eds. (2019). *Automated Machine Learning - Methods, Systems, Challenges*. Springer.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Lopez-Paz, David and Maxime Oquab (2017). "Revisiting Classifier Two-Sample Tests". In: *International Conference on Learning Representations*.

Lueckmann, Jan-Matthis, Giacomo Bassetto, Theofanis Karaletsos, and Jakob H. Macke (2019). "Likelihood-free inference with emulator networks". In: *Proceedings of The 1st Symposium on Advances in Approximate Bayesian Inference*.

Lueckmann, Jan-Matthis, Jan Boelts, David Greenberg, Pedro Goncalves, and Jakob Macke (2021). "Benchmarking simulation-based inference". In: *International Conference on Artificial Intelligence and Statistics*.

Lueckmann, Jan-Matthis, Pedro J Goncalves, Giacomo Bassetto, Kaan Öcal, Marcel Nonnenmacher, and Jakob H Macke (2017). "Flexible statistical inference for mechanistic models of neural dynamics". In: *Advances in Neural Information Processing Systems*.

Mockus, Jonas (1998). "The application of Bayesian methods for seeking the extremum". In: *Towards global optimization* 2, p. 117.

Moss, Guy, Vjeran Višnjević, Olaf Eisen, Falk M Oraschewski, Cornelius Schröder, Jakob H Macke, and Reinhard Drews (2025). "Simulation-based inference of surface accumulation and basal melt rates of an Antarctic ice shelf from isochronal layers". In: *Journal of Glaciology* 71, e44.

Papamakarios, George and Iain Murray (2016). "Fast ϵ-free Inference of Simulation Models with Bayesian Conditional Density Estimation". In: *Advances in Neural Information Processing Systems*.

Papamakarios, George, Theo Pavlakou, and Iain Murray (2017). "Masked autoregressive flow for density estimation". In: *Advances in neural information processing systems* 30.

Papamakarios, George, David C. Sterratt, and Iain Murray (2018). "Sequential Neural Likelihood: Fast Likelihood-free Inference with Autoregressive Flows". In: *International Conference on Artificial Intelligence and Statistics*.

Radev, Stefan T, Ulf K Mertens, Andreas Voss, Lynton Ardizzone, and Ullrich Köthe (2020). "BayesFlow: Learning complex stochastic models with invertible neural networks". In: *IEEE transactions on neural networks and learning systems* 33.4, pp. 1452–1466.

SBI Applications Explorer (n.d.). Streamlit App: https://sbi-applications-explorer.streamlit.app/. Accessed: 2025-09-15.

Shahriari, Bobak, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas (2015). "Taking the human out of the loop: A review of Bayesian optimization". In: *Proceedings of the IEEE* 104.1, pp. 148–175.

Sharrock, Louis, Jack Simons, Song Liu, and Mark Beaumont (2022). "Sequential neural score estimation: Likelihood-free inference with conditional score based diffusion models". In: *arXiv preprint arXiv:2210.04872*.

Wildberger, Jonas, Maximilian Dax, Simon Buchholz, Stephen Green, Jakob H Macke, and Bernhard Schölkopf (2023). "Flow matching for scalable simulation-based inference". In: *Advances in Neural Information Processing Systems* 36, pp. 16837–16864.

Williams, Christopher KI and Carl Edward Rasmussen (2006). *Gaussian processes for machine learning*. Vol. 2. 3. MIT press Cambridge, MA.

List of Abbreviations

SBI	Simulation-based Bayesian Inference
NPE	Neural Posterior Estimation
NRE	Neural Ratio Estimation
NLE	Neural Likelihood Estimation
NPSE	Neural Posterior Score Estimation
MAF	Masked Autoregressive Flow
NSF	Neural Spline Flow
C2ST	Classifier 2-Sample Test
AutoML	Automated Machine Learning
HPO	Hyper-parameter Optimization
BO	Bayesian Optimization
RS	Random Search

Appendices

Appendix A
Additional results

A.1 Training times

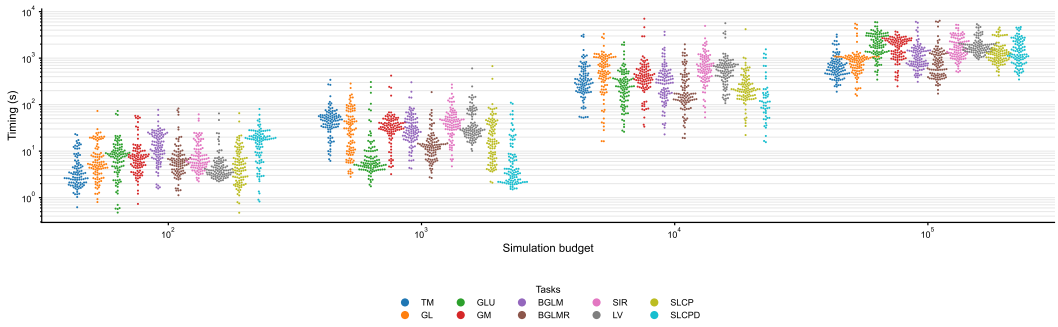

Figure A.1.1.: Runtimes for each trial by budget for all tasks. Tasks are represented in 10 different colours.

All the NPE training runs are conducted on CPU cores (36 cores: 2 x Intel Xeon Gold 6240, 18 cores/die, 2.6GHz).

In Figure A.1.1, we present the runtimes for the 100 trials (each single dot is a trial colored by tasks) for the 10 tasks on the range of simulation budgets (on the x-axis). In "runtime", we include both the simulation time (to generate the $\{(\boldsymbol{\theta}, \boldsymbol{x})\}$ pairs for training) and training the density estimator for NPE. We exclude the sampling time later to compute the C2ST (limited to $150s$ for each of the 10 observations). Note that for these benchmark tasks, the simulation is fast, so the runtime is mainly dominated by the training cost.

As expected, we see higher runtimes with high budgets. Interestingly, we also observe higher variance in runtimes in the high budget regime (the y-axis is on log scale). In Table A.1 we also provide the mean runtimes with the standard deviation across the 100 trials. We can observe that for 10^5 simulations, each trial often takes ≈ 1 hour, making sequential methods like BO compute intensive if one wants to run

them for 100 trials. Use of GPU compute cores to train NPE with 10^5 simulations with even higher batch sizes (≥ 1000) could result in some amount of speed-up.

Table A.1.: Mean runtimes (in seconds) for each trial (with 1 standard deviation) for the benchmark tasks at the 4 simulation budgets.

Tasks	10^2	10^3	10^4	10^5
Two Moons	4.9 ± 4.6	54.5 ± 49.4	461.9 ± 532.4	782.8 ± 554.8
Gaussian Linear	8.6 ± 9.5	44.8 ± 49.1	676.3 ± 566.7	1029.9 ± 918.9
Gaussian Linear Uniform	9.7 ± 11.2	17.2 ± 43.5	350.6 ± 377.2	2203.7 ± 1196.4
Gaussian Mixture	10.3 ± 12.0	42.2 ± 51.5	566.2 ± 883.4	1804.9 ± 835.3
Bernoulli GLM	15.6 ± 11.7	39.2 ± 39.7	445.7 ± 542.2	1278.8 ± 990.0
Bernoulli GLM Raw	9.5 ± 14.0	17.7 ± 21.8	312.0 ± 361.1	1052.5 ± 1120.7
SIR	10.0 ± 9.6	51.3 ± 43.1	778.5 ± 654.0	1887.7 ± 1036.8
Lotka Volterra	6.2 ± 8.3	52.1 ± 68.5	716.9 ± 786.0	1989.9 ± 916.1
SLCP	7.1 ± 8.6	34.9 ± 76.1	305.4 ± 435.5	1482.5 ± 878.3
SLCP Distractors	17.7 ± 12.8	11.1 ± 19.2	254.7 ± 356.7	1622.5 ± 1027.2

A.2 Correlation plots

In Figures A.2.1, A.2.2 and A.2.3, we show the correlation plots between validation loss and C2ST for all benchmark tasks. With higher budgets, the correlation seems to improve for all tasks.

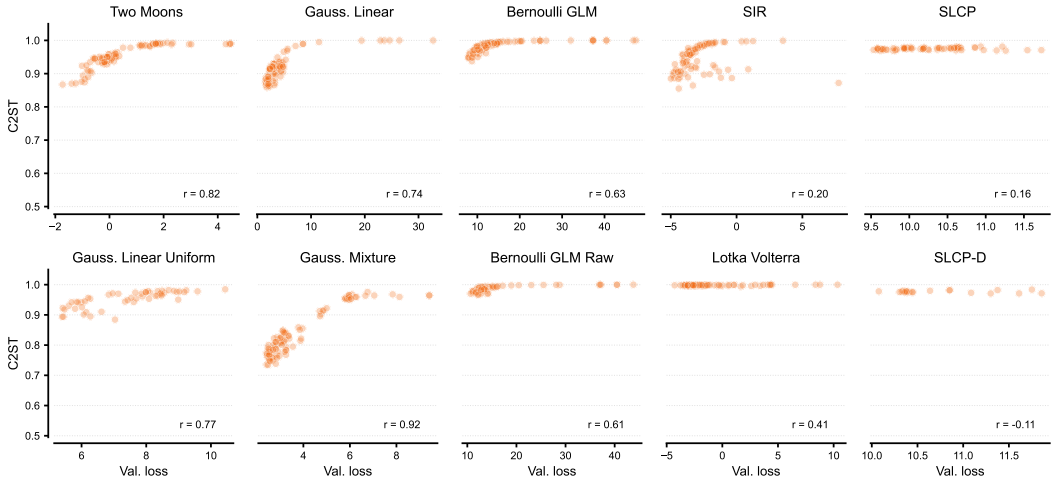

Figure A.2.1.: Val. loss and C2ST correlation for a simulation budget of 100.

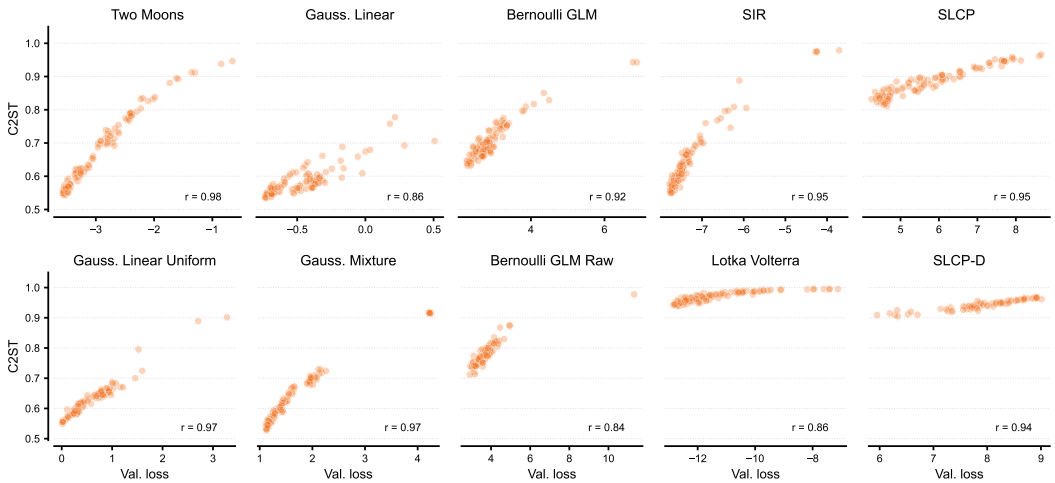

Figure A.2.2.: Val. loss and C2ST correlation for a simulation budget of $10,000$.

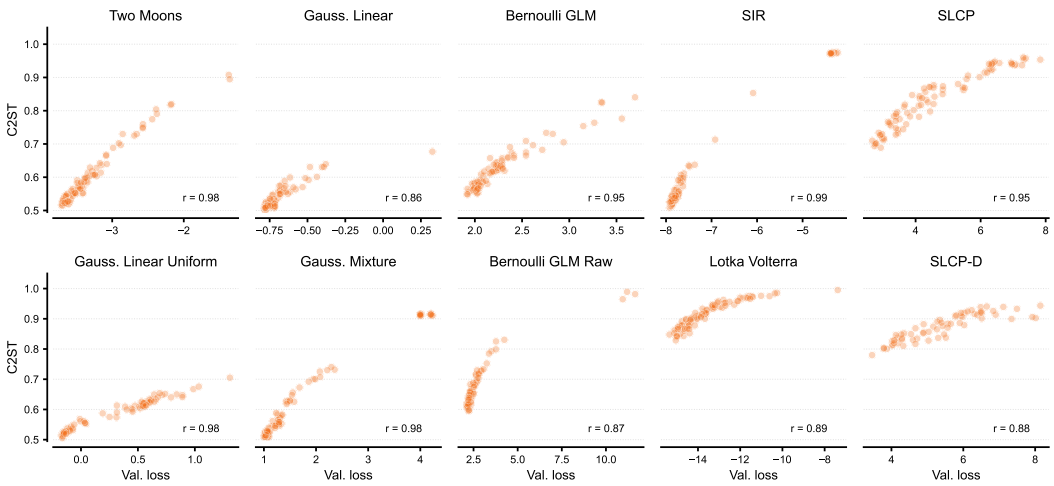

Figure A.2.3.: Val. loss and C2ST correlation for a simulation budget of $100,000$.

A.3 C2ST improvements by HPO

Here we supply the plots to see the effect on C2ST by using different HPO strategies. As a summary, we provide the improvement figures per task (aggregated over all budgets) in Table A.2. For the top 6 benchmark tasks in the table (SIR, TM, GM, GLU, SLCP, BGLMR), we observe larger drops in C2STs compared to the default.

Table A.2.: Mean change (%) in C2ST w.r.t default (↓ is better) for the three strategies across all budgets. The tasks are ordered by the drop in C2ST in the RS column.

	Strategies		
Task	RS	BO	Agg.
SIR	−11.69	−15.17	−9.68
TM	−11.47	−12.82	−9.32
GM	−9.07	−11.69	−9.14
GLU	−8.38	−5.33	−4.86
SLCP	−6.02	−4.71	−5.52
BGLMR	−3.64	−3.28	−1.39
SLCP-D	−2.79	−2.00	−0.99
GL	−2.12	−1.50	2.60
BGLM	−1.92	−2.91	1.88
LV	−1.66	−0.81	−1.06

We present the 4 tasks that do not improve much in Fig. A.3.1 and rest in Fig. A.3.2. Thus, C2ST improvements by HPO for SBI tasks is to be taken with some caution, as we do not see improvement for all tasks or budgets, e.g., for GM, only with 10^3 or higher simulations, we see the benefit of doing HPO.

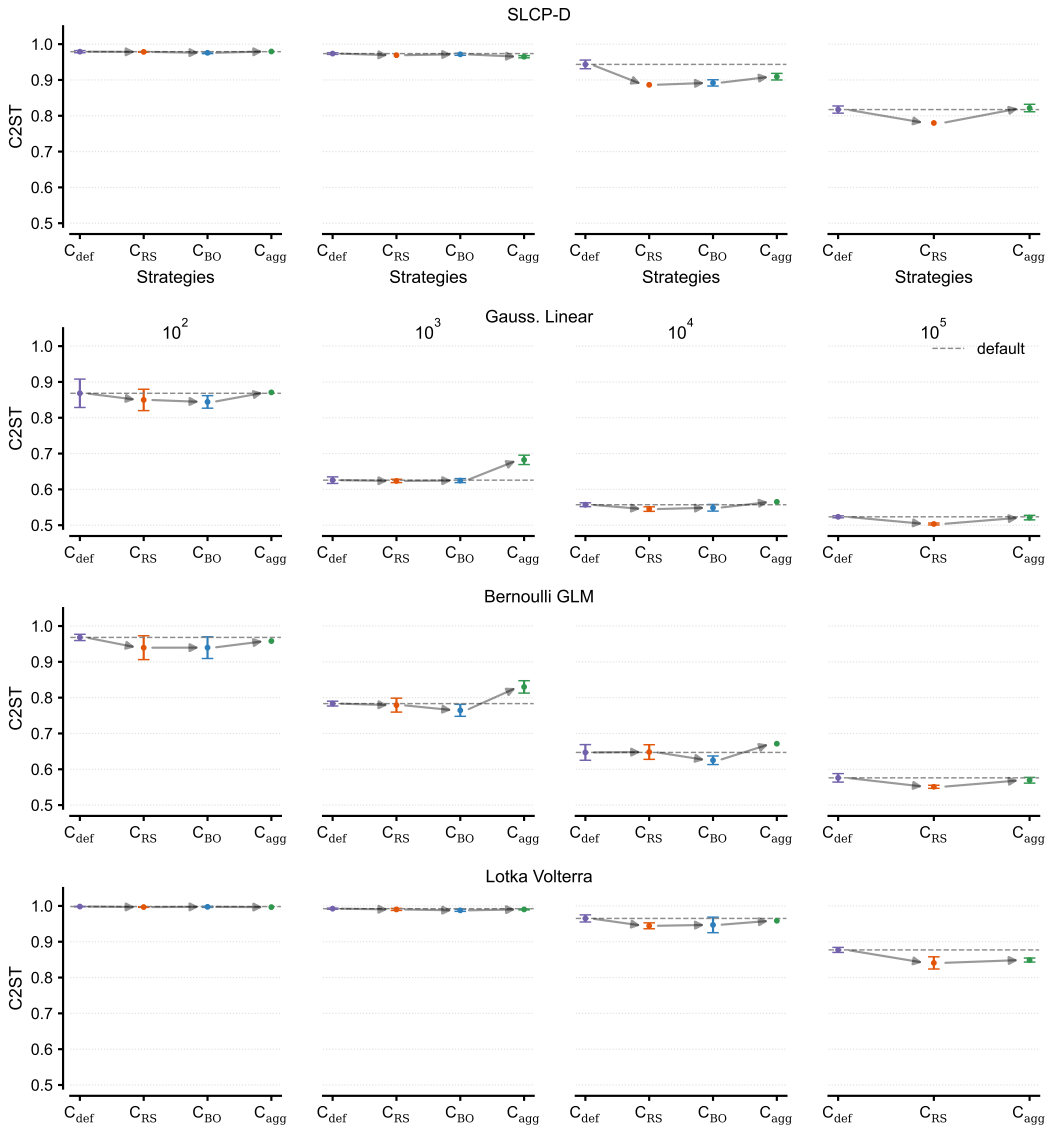

Figure A.3.1.: C2STs for these tasks do not improve much after HPO.

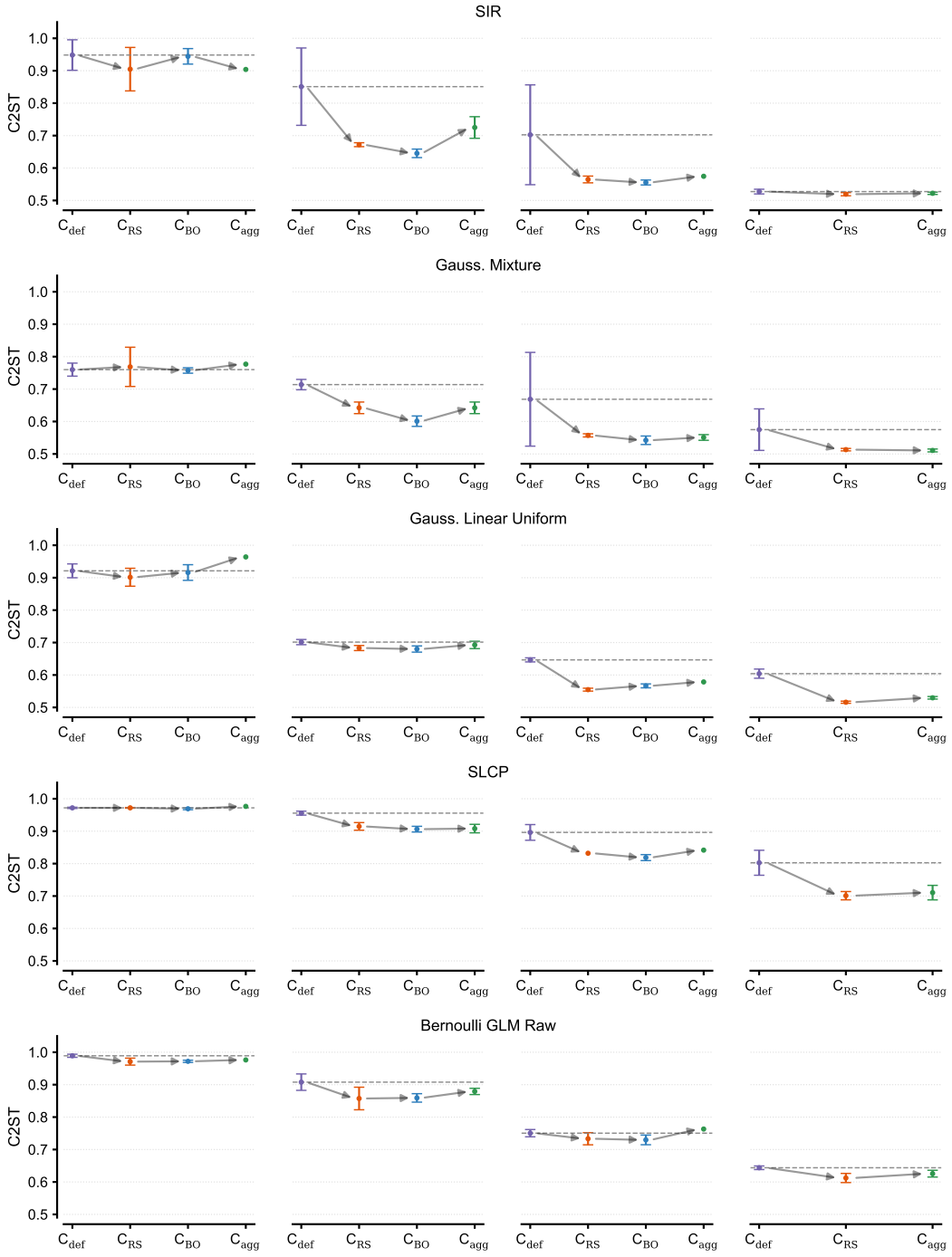

Figure A.3.2.: Tasks exhibiting considerable C2ST improvement after HPO for some budget (Two Moons plot already shown in Fig 4.4.1).

A.4 Scaling with budgets

In Figure A.4.1 we show the robustness of the C2STs obtained from optimal hyperparameters to simulation budgets for the remaining 6 tasks (not shown in the main text). One interesting case here is the Gaussian Linear (GL) task, where we observe the optimal setting obtained for 10^5 simulations (darkest shade) is considerably worse at lower simulations. This can be explained by the density estimator choice: at lower simulations, GL prefers `maf` but the optimal setting at 10^5 budget uses `nsf`. This is also discussed in more detail in Figure 4.6.2.

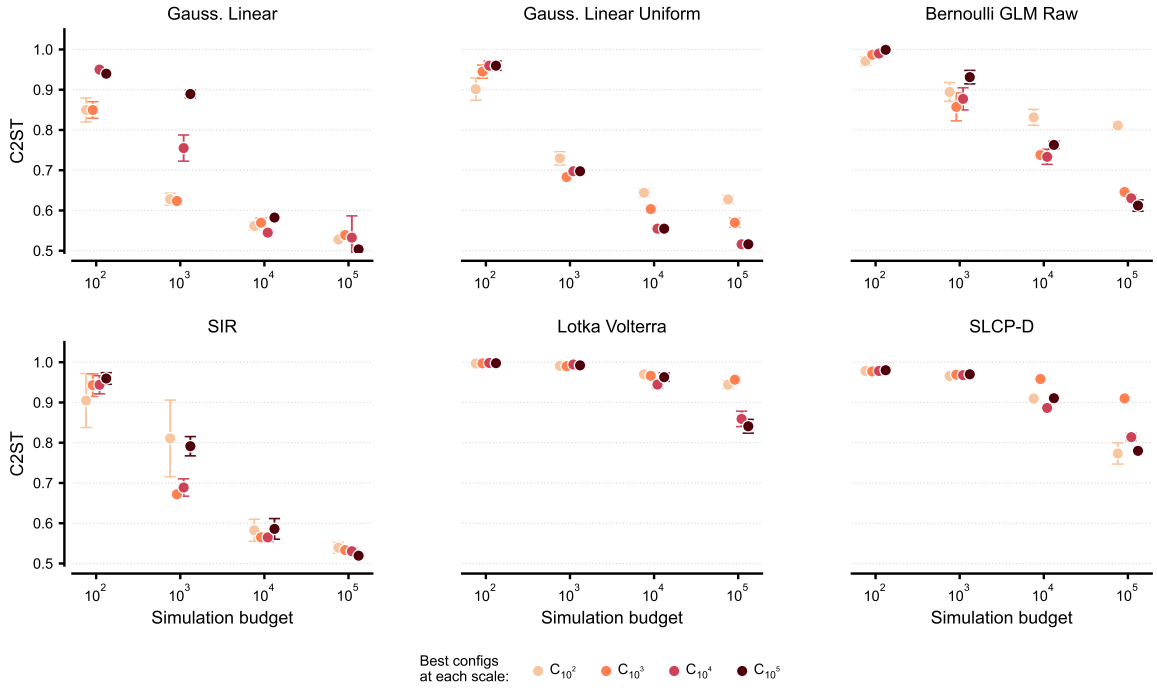

Figure A.4.1.: C2STs from optimal hyperparameters for most tasks are robust to the budgets.

Table A.3.: C2STs of optimal configs across budgets for the Two Moons task.

Config	10^2	10^3	10^4	10^5
$C_{\text{RS}}(\text{TM}, 10^2)$	0.92 ± 0.04	0.81 ± 0.03	0.71 ± 0.02	0.70 ± 0.01
$C_{\text{RS}}(\text{TM}, 10^3)$	0.93 ± 0.02	0.65 ± 0.02	0.58 ± 0.02	0.54 ± 0.01
$C_{\text{RS}}(\text{TM}, 10^4)$	0.94 ± 0.01	0.69 ± 0.02	0.56 ± 0.01	0.53 ± 0.01
$C_{\text{RS}}(\text{TM}, 10^5)$	0.94 ± 0.02	0.67 ± 0.01	0.57 ± 0.01	0.52 ± 0.01
C_{def}	0.93 ± 0.02	0.84 ± 0.10	0.65 ± 0.01	0.58 ± 0.02

A.5 Cross-task transferability

We provide plots (Figure A.5.1) for portability of optimal configurations for 3 more source tasks here: GLU, GM and SIR. We choose these tasks for transfer because they also show significant C2ST improvements w.r.t default after HPO.

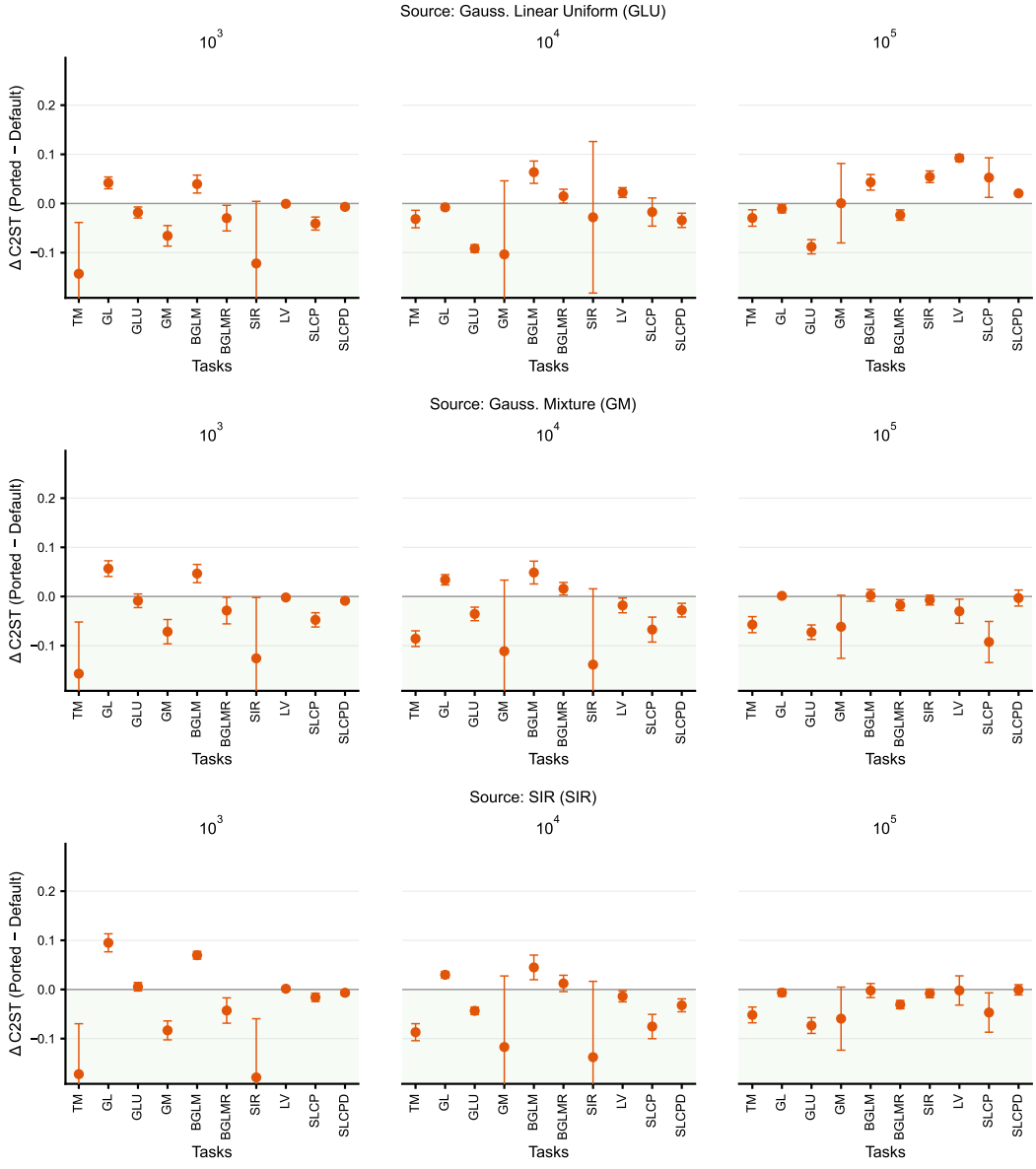

Figure A.5.1.: We show transferability plots for 3 more candidate source tasks here.

Appendix B

Usage of GenAI

The author of this thesis uses coding IDEs (e.g., VSCode and Cursor) with built-in LLM (GPT-5, Claude-4.5-Sonnet) support, so LLMs have been used to support the development of software with potential bug-fixes and menial coding tasks by tab completion, but they are not used to generate larger program sections. All ideas around the problem setup, experiments, visualizations and contributions of the thesis emerged through numerous discussions between the author and the supervisors with no input from any LLM. LLM tools have also been used only very sparingly in writing, not more than one would consult a thesaurus or dictionary. The AI usage declaration has been attached at the end as required.

Erklärung

Laut Beschlüssen der Prüfungsausschüsse Bioinformatik, Informatik, Informatik Lehramt, Kognitionswissenschaft, Machine Learning, Medieninformatik und Medizininformatik der Universität Tübingen vom 05.02.2025. Gültig für Abschlussarbeiten (B.Sc./M.Sc./B.Ed./M.Ed.) in den zugehörigen Fächern. Bei Studienarbeiten und Hausarbeiten bitte nach Maßgabe des/der jeweiligen Prüfers/Prüferin.

1. Allgemeine Erklärungen

Hiermit erkläre ich:

- Ich habe die vorgelegte Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt.
- Ich habe alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet.
- Die Arbeit war weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens.
- Falls ich ein elektronisches Exemplar und eines oder mehrere gedruckte und gebundene Exemplare eingereicht habe (z.B., weil der/die Prüfer/in(nen) dies wünschen): Das elektronisch eingereichte Exemplar stimmt exakt mit dem bzw. den von mir eingereichten gedruckten und gebundenen Exemplar(en) überein.

2. Erklärung bezüglich Veröffentlichungen

Eine Veröffentlichung ist häufig ein Qualitätsmerkmal (z.B. bei Veröffentlichung in Fachzeitschrift, Konferenz, Preprint, etc.). Sie muss aber korrekt angegeben werden. Bitte kreuzen Sie die für Ihre Arbeit zutreffende Variante an:

☑ Die Arbeit wurde bisher weder vollständig noch in Teilen veröffentlicht.

☐ Die Arbeit wurde in Teilen oder vollständig schon veröffentlicht. Hierfür findet sich im Anhang eine vollständige Tabelle mit bibliographischen Angaben.

3. Nutzung von Methoden der künstlichen Intelligenz (KI, z.B. chatGPT, DeepL, etc.)

Die Nutzung von KI kann sinnvoll sein. Sie muss aber korrekt angegeben werden und kann die Schwerpunkte bei der Bewertung der Arbeit beeinflussen. Bitte kreuzen Sie alle für Ihre Arbeit zutreffenden Varianten an und beachten Sie, dass die Varianten 3.4 - 3.6 eine vorherige Absprache mit dem/der Betreuer/in voraussetzen:

☐ 3.1. Keine Nutzung: Ich habe zur Erstellung meiner Arbeit keine KI benutzt.

☑ 3.2. Korrektur Rechtschreibung & Grammatik: Ich habe KI für Korrekturen der Rechtschreibung und Grammatik genutzt, ohne dass es dabei zu inhaltlich relevanter Textgeneration oder Übersetzungen kam. Das heißt, ich habe von mir verfasste Texte in derselben Sprache korrigieren lassen. Es handelt sich um rein sprachliche Korrekturen, sodass die von mir ursprünglich intendierte Bedeutung nicht wesentlich verändert oder erweitert wurde. Im Zweifelsfall habe ich mich mit meinem/r Betreuer/in besprochen. Alle genutzten Programme mit Versionsnummer sind im Anhang meiner Arbeit in einer Tabelle aufgelistet.

☑ 3.3. Unterstützung bei der Softwareentwicklung: Ich habe KI als Unterstützung beim Schreiben von Code in der Softwareentwicklung genutzt. Es handelt sich hierbei lediglich um Unterstützung und nicht um die automatische Generierung von größeren Programm-Teilen. Im Zweifelsfall habe ich mich mit meinem/r Betreuer/in besprochen. Alle genutzten Programme mit Versionsnummer sind im Anhang meiner Arbeit in einer Tabelle aufgelistet.

☐ 3.4. Übersetzung: Ich habe *nach vorheriger Absprache und mit Erlaubnis meines/r Betreuer/in* KI zur Übersetzung von mir in einer anderen Sprache geschriebenen Texte genutzt. Jede derartige Übersetzung ist im laufenden Text gekennzeichnet und der Anhang meiner Arbeit enthält eine Tabelle mit einem vollständigen Nachweis aller übersetzten Textstellen und der verwendeten Programme mit Versionsnummer.

☐ 3.5. Code-Generierung: Ich habe *nach vorheriger Absprache und mit Erlaubnis meines/r Betreuer/in* KI zur Erzeugung von Code in der Softwareentwicklung genutzt. Der Anhang meiner Arbeit enthält eine Tabelle mit einem vollständigen Nachweis aller derartigen Nutzungen, der verwendeten Programme mit Versionsnummer und der verwendeten Prompts.

☐ 3.6. Text-Generierung: Ich habe *nach vorheriger Absprache und mit Erlaubnis meines/r Betreuer/in* KI zur Erzeugung von Text in meiner Arbeit genutzt. Jede derartige Verwendung von KI ist im laufenden Text gekennzeichnet und der Anhang meiner Arbeit enthält eine Tabelle mit einem vollständigen Nachweis aller derartigen Nutzungen, der verwendeten Programme mit Versionsnummer und der verwendeten Prompts.

Falls ich in irgendeiner Form KI genutzt haben (siehe oben), dann erkläre ich:

Mir ist bewusst, dass ich die Verantwortung trage, falls es durch die Verwendung von KI zu fehlerhaften Inhalten, zu Verstößen gegen das Datenschutzrecht, Urheberrecht oder zu wissenschaftlichem Fehlverhalten (z.B. Plagiaten) kommt.

4. Abschluss und Unterschrift(en)

Mir ist bekannt, dass ein Verstoß gegen diese Erklärung prüfungsrechtliche Konsequenzen haben und insbesondere dazu führen kann, dass die Prüfungsleistung mit „nicht ausreichend" bzw. die Studienleistung mit „nicht bestanden" bewertet wird und bei mehrfachem oder schwerwiegendem Täuschungsversuch eine Exmatrikulation erfolgen bzw. ein Verfahren zur Entziehung eines eventuell verliehenen akademischen Titels eingeleitet werden kann.

Swagatam Haldar	Tübingen, 31.10.2025	Swagatam Halda.
Vorname, Nachname Student/in	Ort, Datum	Unterschrift

Die Punkte 3.4 - 3.6 erfordern eine Zustimmung des/r Betreuer/in. Sollten Sie einen dieser Punkte angekreuzt haben, dann sollte der/die Betreuer/in bitte hier unterschreiben:

Ich habe der oben genannten Nutzung von KI zur Erstellung der Arbeit zugestimmt.

Vorname, Nachname	Ort, Datum	Unterschrift