e-SimFT: Pareto-Optimal Sampling of Generative Design Models Fine-tuned with Simulation Feedback

Hyunmin Cheong
Spectral Labs
San Francisco, CA 94111
hyunmin@spectrallabs.ai

Mohammadmehdi Ataei
Autodesk Research
Toronto, Ontario, Canada
mehdi.ataei@autodesk.com

Amir Hosein Khasahmadi
Autodesk Research
Toronto, Ontario, Canada

Pradeep Kumar Jayaraman
Autodesk Research
Toronto, Ontario, Canada

Abstract

Deep generative models have recently shown success in solving complex engineering design problems where models predict solutions that address the design requirements specified as input. However, there remains a challenge in aligning such models for effective design exploration. For many design problems, finding a solution that meets all requirements is infeasible. In such a case, engineers prefer to obtain a set of Pareto-optimal solutions with respect to those requirements, but uniform sampling of generative models may not yield a useful Pareto front. To address this gap, we first fine-tune generative design models with simulation feedback, and then apply epsilon-sampling, inspired by the epsilon-constraint method used for Pareto front generation with classical optimization algorithms, to construct a high-quality Pareto front with the fine-tuned models. Our framework, named e-SimFT, is shown to produce better-quality Pareto fronts than existing multi-objective alignment methods developed for large language models.

1 Introduction

Generative artificial intelligence (AI) has made remarkable impact in many domains, especially where creative automation is greatly desired. One notable area is engineering design, where various generative AI models have been proposed to help engineers develop solutions to their problems at a much faster pace than with the traditional design process [13]. Such progress could bring significant innovation to real-world problems and therefore amplify AI's positive impact on our society.

There remains a challenge in making use of generative AI models for engineering design in practice. For many design problems, finding a solution that meets all the requirements specified is often impossible. In such a scenario, engineers could focus on finding solutions for a relatively more important subset of the requirements. Or preferably, they would like to obtain a set of Pareto-optimal solutions so that they can understand the trade-offs and compare alternative solutions.

This challenge highlights new research opportunities in two aspects. First, we need to align a generative model with respect to specific design requirements preferred by the engineer. Next, we need a method to effectively sample a generative model to produce a high-quality Pareto front.

This work demonstrates that the alignment problem can be solved by adapting fine-tuning methods used for LLMs, but using *simulation feedback*. Because a simulator can be used to evaluate a given solution with respect to design requirements of interest, we can use it to either generate fine-tuning

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Frontiers in Probabilistic Inference: Sampling Meets Learning.

data or compute accurate rewards during reinforcement learning (RL). We propose *SimFT*, fine-tuning methods for generative design models with simulation feedback. In addition, we propose *epsilon-sampling*, which leverages the simulation fine-tuned models to produce a high-quality Pareto front. The technique is inspired by the epsilon-constraint method [8] used for constructing a Pareto front with gradient-based optimization.

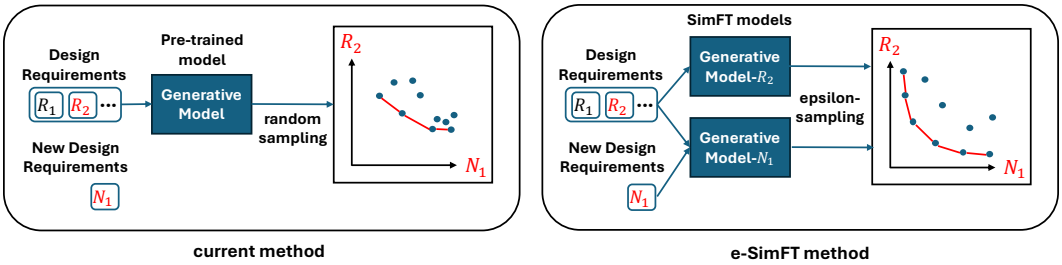


Figure 1: Randomly sampling a generative model (for engineering design) may not yield a good Pareto front with respect to the design requirements of interest. We address these issues with SimFT methods–using simulation feedback to fine-tune a generative model with respect to specific design requirements, and apply a new sampling strategy to create e-SimFT for Pareto front generation.

The overview of our method is illustrated in Figure 1. We showcase the performance improvements made with respect to specific design requirements using SimFT methods and various ablation studies to elucidate the nature of simulation fine-tuning. We also evaluate e-SimFT against several baselines including multi-objective alignment methods developed for LLMs and show its superior performance.

2 Related Work

Preference alignment aims to fine-tune LLMs to generate content that aligns with the user's values or objectives, mainly through Reinforcement Learning with Human Feedback (RLHF) [2, 19, 15, 10]. The most popular RL algorithm used is Proximal Policy Optimization (PPO) [14], while a non-RL approach such as Direct Preference Optimization (DPO) [11] uses a preference dataset to fine-tune LLMs [9, 1, 6]. Because user preferences for LLMs are likely multidimensional with trade-offs, e.g., helpfulness vs. harmlessness, multi-objective alignment methods have been proposed to produce Pareto-front aligned models [12, 18, 16, 7, 17].

Note that the purpose of multi-objective alignment methods has a strong parallel with the purpose of multi-objective optimization methods [4] where the goal is to find models or solutions that constitute a high-quality Pareto front [20]. Therefore, we were motivated to find inspirations from the techniques used in the latter domain such as the epsilon-constraint method [8] or non-dominated sorting [3].

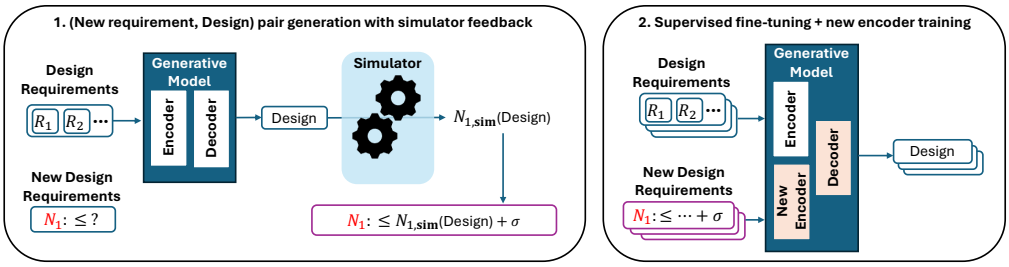
3 Methods

Our method involves first fine-tuning a generative design model with respect to each design requirement, and sampling from the collection of fine-tuned models to construct a Pareto front.

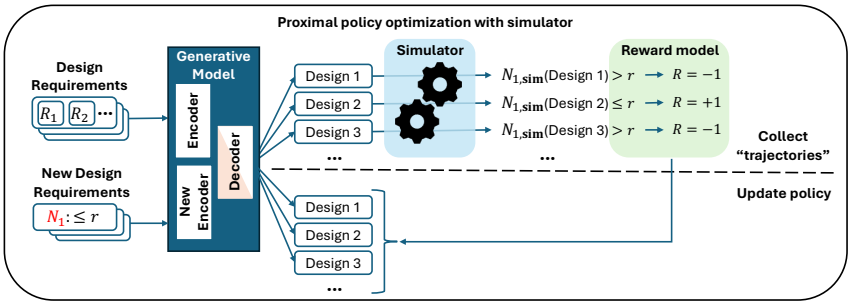
3.1 SimFT methods for requirements alignment

We use the typical two-step fine-tuning procedure developed for LLMs – supervised fine-tuning (SFT) followed by RL using PPO. Here, we focus on presenting methods developed to align a generative design model with respect to new design requirements that the model has never seen before (Figure 2). The alignment method with respect to the original requirements used to train the model is more straightforward and can be found in Appendix B.1.

SFT. In the first SFT stage, we generate the dataset in the following manner. We sample a design solution from the pre-trained model and compute the corresponding new requirement value using a simulator. We increase this value by some random variance and set it as the upper bound value of the requirement, synthetically generating a pair of the constraint bound value, n_i, and a solution that



(a) SimFT Step 1: SFT with simulation augmented data



(b) SimFT Step 2: PPO with a simulator to compute rewards.

Figure 2: SimFT data generation and training methods.

satisfies the constraint, x'. We fine-tune the pre-trained model using this dataset by minimizing the log probability loss (Eq. 1) but conditioned on the new requirement. We freeze the original encoder while training the decoder and also a new encoder that can encode the new requirement bound value.

$$\mathcal{L}(\theta) = -\mathbb{E}_{x' \sim \pi_\theta} \left[\log \pi_\theta(x' | r_i) \right] \tag{1}$$

PPO with a simulator. We employ PPO [14], using the simulator to compute accurate rewards for each solution during exploration. For the loss, we use both the clipped policy ratio (the full expression is omitted here for brevity) with the KL divergence penalty:

$$\mathcal{L}_{\text{RL}}(\theta) = -\mathbb{E}_{(x,n_i)\sim\mathcal{D}} \left[\text{clipped}\left(\frac{\pi_\theta(x \mid n_i)}{\pi_{\text{old}}(x \mid n_i)} \right) \mathcal{R}(x, n_i) \; - \; \beta \; \text{KL}(\pi_\theta \,\|\, \pi_{\text{old}}) \right]. \tag{2}$$

where $\mathcal{R}(x, n_i)$ is a reward function computed using the simulator output and normalized to $[-1, 1]$, i.e., $\mathcal{R} = 1$ if x is evaluated to meet the bound value n_i and approaches -1 as the violation increases.

In Appendix B.2, we also present using a simulator to generate a preference dataset with respective to different requirements and employing DPO to fine-tune the model.

3.2 Epsilon-sampling for Pareto front generation

Randomly sampling a generative design model multiple times likely would not result in a good Pareto front because generation is not conditioned on different requirement preferences. One could sample multiple models, each fine-tuned for different requirements, but this may get clusters of solutions at the extremes of only satisfying each requirement.

We propose epsilon-sampling (Figure 5, Appendix B.3) inspired by the epsilon-constraint method [8]. Given a pair of objectives, the method sets one objective as a constraint and solves multiple single-objective constrained optimization problems by incrementing the threshold value ϵ imposed on the constraint. Solutions to these problems form a Pareto front. With SimFT models, given a set of requirements $\vec{r}$, we assume that the model fine-tuned for r_i can best enforce that constraint; therefore, sampling from that model would be equivalent to posing r_i as a constraint and the rest of the requirements as objectives. We sample multiple solutions from this model by varying the target value by $r_i \pm \epsilon$, mimicking the epsilon-constraint method, which constitutes our *e-SimFT* framework.

4 Experiments and Discussion

We first evaluate the performance improvements made by SimFT methods. We then evaluate e-SimFT against several baselines in generating high-quality Pareto fronts. Details of experimental setup are presented in the Appendix C.

We used GearFormer, a generative model for gear train design [5] as the backbone, which was fine-tuned with respect to the original and new requirements using SimFT methods. Table 1 presents the improvements made by SimFT methods for each requirement. For original requirements (speed and position), the metrics are based on [5]. For new requirements (cost and bounding box volume), we calculate the percentage of the problems from the test dataset for which the first solution generated by the original GearFormer or SimFT models satisfy the respective requirement. We observed that DPO performed slightly better than PPO for both requirements.

See Appendix D.1 for additional details, including ablation studies performed to examine the importance of rejection sampling, performance trade-offs, and using different reward functions.

Table 2 reports the hypervolumes of Pareto fronts obtained with e-SimFT and other baselines (C.4). It shows that on average, e-SimFT is the best method for both two- and three-requirements scenarios. One could also observe that on average, SimFT and epsilon-sampling on their own do not perform as well as e-SimFT. Only in one scenario, an alternative multi-objective alignment method, RiC, achieved the highest hypervolume. We also present the results obtained with a larger sample size and the Pareto fronts generated by different methods for sample design problems in the Appendix D.2.

Concluding Remarks The current work has demonstrated how the alignment techniques developed for LLM and a sampling technique inspired from multi-objective optimization can be combined to effectively construct a Pareto front with generative design models. We believe that many research opportunities exist at the intersection of generative AI, optimization, and engineering design.

Table 1: Performance improvements by SimFT methods

Requirement	**Baseline**	**SimFT method**	
		SFT	DPO / RL via PPO
speed [log(·)] ↓	0.0171	0.0139	N/A
position [m] ↓	0.0338	0.0317	N/A
cost ↑	52.8%	54.1%	66.9% / 65.4%
b.box ↑	49.4%	55.2%	62.3% / 59.1%

Table 2: Comparison of Pareto front hypervolumes. Number of test problems = 30 and samples = 30 per scenario. In the Design Scenario column, each label lists the requirements treated as simultaneous objectives (pairs or triplets). **speed** and **position** are original requirements (minimize deviation from their targets); **cost** and **b.box** are new requirements (design cost and bounding box volume; lower is better). Colons separate the jointly optimized objectives.

Design Scenario	**Method**					
	Baseline	R. Soup	R.-in-Context	SimFT	e-sample	e-SimFT
speed : position	$0.56^{\pm 0.19}$	$0.59^{\pm 0.20}$	$0.57^{\pm 0.21}$	$0.59^{\pm 0.20}$	$0.68^{\pm 0.15}$	$\mathbf{0.70}^{\pm 0.16}$
speed : cost	$0.61^{\pm 0.23}$	$0.61^{\pm 0.23}$	$0.60^{\pm 0.23}$	$0.61^{\pm 0.24}$	$0.66^{\pm 0.24}$	$\mathbf{0.67}^{\pm 0.23}$
speed : b.box	$0.71^{\pm 0.19}$	$0.72^{\pm 0.27}$	$0.72^{\pm 0.20}$	$0.75^{\pm 0.23}$	$0.74^{\pm 0.22}$	$\mathbf{0.81}^{\pm 0.20}$
position : cost	$0.46^{\pm 0.22}$	$0.48^{\pm 0.23}$	$0.48^{\pm 0.25}$	$0.47^{\pm 0.22}$	$0.47^{\pm 0.24}$	$\mathbf{0.49}^{\pm 0.25}$
position : b.box	$0.52^{\pm 0.22}$	$0.46^{\pm 0.22}$	$0.53^{\pm 0.22}$	$0.51^{\pm 0.20}$	$\mathbf{0.56}^{\pm 0.20}$	$0.54^{\pm 0.19}$
cost : b.box	$0.57^{\pm 0.26}$	$0.51^{\pm 0.28}$	$\mathbf{0.58}^{\pm 0.25}$	$0.56^{\pm 0.25}$	$0.52^{\pm 0.27}$	$0.52^{\pm 0.28}$
mean	$0.57^{\pm 0.22}$	$0.56^{\pm 0.24}$	$0.58^{\pm 0.23}$	$0.58^{\pm 0.22}$	$0.60^{\pm 0.22}$	$\mathbf{0.62}^{\pm 0.22}$
speed : position: cost	$0.38^{\pm 0.22}$	$0.40^{\pm 0.23}$	$0.40^{\pm 0.24}$	$0.43^{\pm 0.23}$	$0.48^{\pm 0.24}$	$\mathbf{0.48}^{\pm 0.22}$
speed : position : b.box	$0.44^{\pm 0.21}$	$0.44^{\pm 0.21}$	$0.43^{\pm 0.20}$	$0.51^{\pm 0.19}$	$0.50^{\pm 0.18}$	$\mathbf{0.53}^{\pm 0.20}$
speed : cost : b.box	$0.47^{\pm 0.24}$	$0.47^{\pm 0.26}$	$0.50^{\pm 0.23}$	$\mathbf{0.54}^{\pm 0.23}$	$0.48^{\pm 0.24}$	$0.52^{\pm 0.25}$
position : cost : b.box	$0.34^{\pm 0.20}$	$0.31^{\pm 0.21}$	$0.38^{\pm 0.23}$	$\mathbf{0.38}^{\pm 0.20}$	$0.35^{\pm 0.24}$	$0.37^{\pm 0.21}$
mean	$0.40^{\pm 0.22}$	$0.40^{\pm 0.23}$	$0.43^{\pm 0.22}$	$0.46^{\pm 0.21}$	$0.45^{\pm 0.22}$	$\mathbf{0.47}^{\pm 0.22}$

References

[1] Mohammad Gheshlaghi Azar, Zhaohan Daniel Guo, Bilal Piot, Remi Munos, Mark Rowland, Michal Valko, and Daniele Calandriello. A general theoretical paradigm to understand learning from human preferences. In *International Conference on Artificial Intelligence and Statistics*, pages 4447–4455. PMLR, 2024.

[2] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

[3] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[4] Kalyanmoy Deb, Karthik Sindhya, and Jussi Hakanen. Multi-objective optimization. In *Decision sciences*, pages 161–200. CRC Press, 2016.

[5] Yasaman Etesam, Hyunmin Cheong, Mohammadmehdi Ataei, and Pradeep Kumar Jayaraman. Deep generative model for mechanical system configuration design. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 16496–16504, 2025.

[6] Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. Kto: Model alignment as prospect theoretic optimization. *arXiv preprint arXiv:2402.01306*, 2024.

[7] Yiju Guo, Ganqu Cui, Lifan Yuan, Ning Ding, Zexu Sun, Bowen Sun, Huimin Chen, Ruobing Xie, Jie Zhou, Yankai Lin, et al. Controllable preference optimization: Toward controllable multi-objective alignment. *arXiv preprint arXiv:2402.19085*, 2024.

[8] Yacov Haimes. On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE transactions on systems, man, and cybernetics*, (3):296–297, 1971.

[9] Joey Hejna, Rafael Rafailov, Harshit Sikchi, Chelsea Finn, Scott Niekum, W Bradley Knox, and Dorsa Sadigh. Contrastive prefence learning: Learning from human feedback without rl. *arXiv preprint arXiv:2310.13639*, 2023.

[10] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[11] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2024.

[12] Alexandre Rame, Guillaume Couairon, Corentin Dancette, Jean-Baptiste Gaya, Mustafa Shukor, Laure Soulier, and Matthieu Cord. Rewarded soups: towards pareto-optimal alignment by interpolating weights fine-tuned on diverse rewards. *Advances in Neural Information Processing Systems*, 36, 2024.

[13] Lyle Regenwetter, Amin Heyrani Nobari, and Faez Ahmed. Deep generative models in engineering design: A review. *Journal of Mechanical Design*, 144(7):071704, 2022.

[14] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[15] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021, 2020.

[16] Rui Yang, Xiaoman Pan, Feng Luo, Shuang Qiu, Han Zhong, Dong Yu, and Jianshu Chen. Rewards-in-context: Multi-objective alignment of foundation models with dynamic preference adjustment. *arXiv preprint arXiv:2402.10207*, 2024.

[17] Yifan Zhong, Chengdong Ma, Xiaoyuan Zhang, Ziran Yang, Haojun Chen, Qingfu Zhang, Siyuan Qi, and Yaodong Yang. Panacea: Pareto alignment via preference adaptation for llms. *arXiv preprint arXiv:2402.02030*, 2024.

[18] Zhanhui Zhou, Jie Liu, Jing Shao, Xiangyu Yue, Chao Yang, Wanli Ouyang, and Yu Qiao. Beyond one-preference-fits-all alignment: Multi-objective direct preference optimization. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 10586–10613, 2024.

[19] Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

[20] Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithms—a comparative case study. In *International conference on parallel problem solving from nature*, pages 292–301. Springer, 1998.

A Additional Background

The problem of our interest can be stated as follows. Suppose we have a generative model parameterized by θ that takes a set of design requirements $\vec{r} = \{r_1, r_2, ..., r_N\}$ as input and outputs a solution x that addresses those requirements, e.g., $\pi_\theta(x|\vec{r})$. First, an engineer might want to prioritize a specific requirement r_i. Therefore, we aim to find a fine-tuned model π_{θ,r_i} such that a solution sampled from the model is optimal with respect to r_i. In some scenarios, there may be a new design requirement n_j independent of the current generative model that an engineer would like the sampled solution to nevertheless satisfy. In such a case, the goal is to find a fine-tuned model π_{θ,n_j} from which a sampled solution would be optimal with respect to n_j.

Finally, given a set of prioritized requirements $\vec{p} \subset (\vec{r} \cup \vec{n})$, we aim to sample from the fine-tuned models a set of Pareto-optimal solutions with respect to $\vec{p}$ that maximizes a Pareto front quality, e.g., hypervolume [20].

A.1 Illustrative example: GearFormer

We use GearFormer, a recently developed generative model for gear train design [5], as an illustrative example for the current work. GearFormer is a Transformer-based model that takes multiple requirements as input via its encoder and outputs a gear train sequence via its decoder. The requirements it can handle are the speed ratio, output motion position, output motion direction, and input/output motion types. While it has been shown to outperform traditional search methods, an engineer does not have an option to express a preference of emphasizing one requirement over another, or explore multi-requirement trade-offs. Our goal is to fine-tune this model with respect to specific requirements and use the fine-tuned models to generate a high-quality Pareto front.

We consider the two types of requirements as expressed in the problem definition. *Original requirements* are those used to train GearFormer and therefore are used as an input to condition the output design. Note that these requirements are treated as equality constraints, i.e., they are target values such as speed ratio or output motion position that the design must meet. *New requirements* are those never seen by the model during training. We consider metrics such as the bounding box volume and design cost, which can be evaluated given a design. In contrast to the original requirements, we intentionally chose new requirements formulated as inequality constraints, i.e., an engineer will be willing to accept any value that is below the specified bound value.

A.2 Challenges of fine-tuning a generative design model

A generative model for engineering design such as GearFormer is trained with a synthetic dataset of ($\vec{r}$ = requirements, x = solution) pairs, where x can be first generated using some rules and $\vec{r}$ is evaluated using a simulator for the generated x. The model is then trained to predict x given $\vec{r}$, which means that the model has only seen designs that perfectly address the requirements. Therefore, during fine-tuning, any design that does not perfectly address a particular original requirement would likely deteriorate the performance of the pre-trained model. We show this effect in an ablation study presented in D.1.

Now, suppose a RL-based method is used to fine-tune a pre-trained model with a typical policy gradient loss

$$\mathcal{L}(\theta) = -\mathbb{E}_{x\sim\pi_\theta}\left[\log \pi_\theta(x|r_i)\mathcal{R}(x, r_i)\right] \tag{3}$$

where $\mathcal{R}$ is the reward for the solution sampled from the current policy. Since we aim to avoid degrading the policy with low-quality data, $\mathcal{R}$ can simply become a binary function that gives 1 for an x that satisfies r_i and 0 otherwise, i.e., equivalent to simply rejecting the sample. This means that with rejection sampled data, Eq 3 becomes a typical log probability loss used for supervised fine-tuning, e.g.,

$$\mathcal{L}(\theta) = -\mathbb{E}_{x'\sim\pi_\theta}\left[\log \pi_\theta(x'|r_i)\right] \tag{4}$$

where x' are solutions that satisfy the target value r_i. We therefore assume that SFT with rejection sampled data suffices as the fine-tuning step for original design requirements.

However for new design requirements, the fine-tuning scenario is very similar to the one with LLMs. The pre-trained model is not trained on the new requirement data and does not have any sense of which output is good or bad with respect to the requirement. Therefore, a solution that does not

perfectly satisfy the new requirement but is reasonably close can still provide useful signals for the model. We can therefore use a continuous reward value that reflects the degree of constraint violation. Based on these observations, a similar two-step technique used for fine-tuning LLMs such as using DPO or PPO with *simulation feedback* in addition to SFT can be considered.

B Additional Description of Methods

B.1 SimFT methods for original requirements

For an original requirement r_i, we perform a single SFT step as justified in the previous section. We generate an additional dataset for SFT by prompting the pre-trained model with a list of objectives (one of which is r_i), evaluate the designs generated using a simulator, and keep only the designs that meet r_i. This can be thought of as synthetic data generation via rejection sampling, but performed in an offline mode before training. Also, we have the advantage of using a simulator to accurately evaluate the data generated and keeping only the perfect design solutions.

The pre-trained model is fine-tuned with this dataset by minimizing the log probability loss (Eq. 4). Note that we freeze the encoder while updating the decoder only.

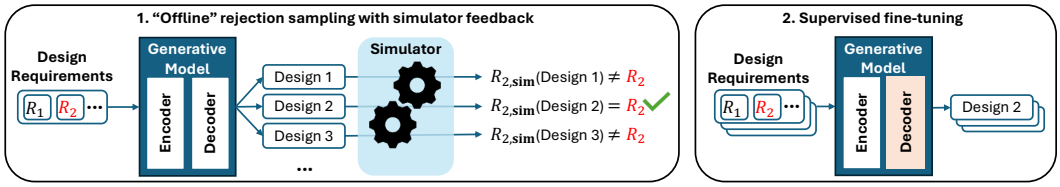


Figure 3: SimFT method for original requirements – SFT with simulation-augmented data

B.2 DPO with simulation-generated preference dataset

The preference dataset is generated by sampling a pair of solutions from the pre-trained model and using a simulator to evaluate the requirement values. The solution with the lower requirement value is labeled as preferred while the other as rejected. We then assume the mean of the two requirement values as the constraint bound value, i.e., the preferred solution would satisfy the bound value while the rejected solution would not. We freeze both the original encoder and the new encoder while updating the decoder by minimizing the DPO loss [11]:

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{(x_w,x_l,n_i)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(x_w \mid n_i)}{\pi_{\text{old}}(x_w \mid n_i)} - \beta\log\frac{\pi_\theta(x_l \mid n_i)}{\pi_{\text{old}}(x_l \mid n_i)}\right)\right]$$

where x_w and x_l are the preferred and rejected solutions for the requirement n_i. β is the KL divergence penalty parameter and π_{old} is the reference policy.

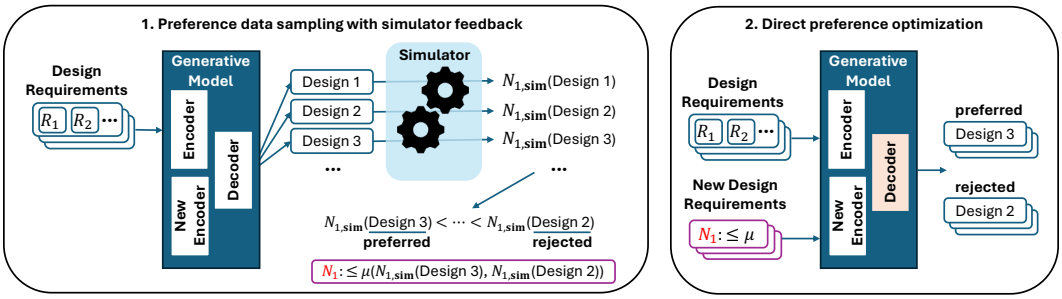


Figure 4: SimFT Step 2 alternative: DPO with simulation augmented preference dataset.

B.3 Epsilon-sampling

The depiction of the epsilon-sampling method is shown in Figure 5.

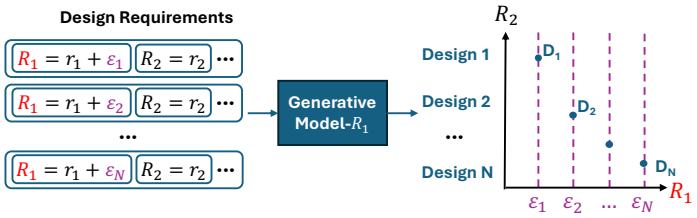


Figure 5: Epsilon-sampling with a SimFT model. The target requirement values for R_1 is incremented with ϵ_i and a SimFT model for R_1 is sampled to construct a Pareto front.

C Experimental Setups

C.1 Pre-trained model and simulator.

We use GearFormer [5] as the pre-trained model. We also use the simulator developed for GearFormer and extend it to compute the new requirements required for this work. While it would be ideal to test our method on multiple generative models, GearFormer was the only work that provided the model, simulator, and dataset required for our experiments.

C.2 Design requirements.

We consider four design requirements. Two original requirements of GearFormer, **speed** ratio and output motion **position**, are posed as equality constraints. The other two are new requirements that were not considered in training GearFormer, bounding box volume (**b.box**) and design **cost**, posed as inequality constraints.

C.3 Design scenarios.

We generated 30 new random test problems based on the distributions of requirement metrics obtained from the original GearFormer dataset [5]. For each test problem, we consider 10 different trade-off scenarios – the all possible two-way and three-way combinations of the four requirements under consideration. For each design scenario, a sample budget of $\mathcal{N}$ is assumed. How this budget is used varies depending on the methods employed, as explained in the following section.

C.4 Baselines.

The first baseline involves sampling the pre-trained GearFormer $\mathcal{N}$ times for each design scenario.

Two distinct and recent multi-objective alignment methods are chosen as additional baselines.

Rewarded Soup: First, *Rewarded Soup* (RS) [12] linearly interpolates the weights of models fine-tuned for each specific objective, given the preference weights assigned for each objective. A Pareto front can be constructed by sampling from multiple of these linearly interpolated models with varying preference weights. We define $\mathcal{N}_s$ combinations of preference weights and allocate $\mathcal{N}/\mathcal{N}_s$ sampling budget for each combination.

For a given weight preference combination, we linearly interpolate the parameters (weights) of SimFT models to create a new model specific for that combination. For Pareto-front generation, we use the following weight combinations. For the two-requirement scenarios, $w_1 = [0, 0.2, 0.4, 0.6, 0.8, 1]$ and $w_2 = 1 - w_1$. For the three-requirement scenarios, $w_1 = [0, 0, 0, 0.33, 0.5, 0.5, 1]$, $w_2 = [0, 0.5, 1, 0.33, 0, 0.5, 0]$, and $w_3 = 1 - w_1 - w_2$.

Rewards-in-Context: We also chose *Rewards-in-Context* (RiC) [16]. RiC performs SFT on the pre-trained model with outputs associated with their reward/preference values, which are encoded as additional input to the model. For this work, we train a new encoder that can take in preference values for each requirement, indicating which requirements to prioritize. We define $\mathcal{N}_r$ combinations of requirement preferences and allocate $\mathcal{N}/\mathcal{N}_r$ sampling budget for each combination.

Using the problems defined in D_{ft}, the training data is generated by sampling a solution for each problem using GearFormer and determining whether the solution meets each of the four requirements of interest or not. Based on this, we can create a preference weight vector that indicates whether a particular requirement is met or not. We then perform supervised fine-tuning with this dataset using log probability loss, while training two new encoders – one for encoding the new requirements and another for encoding the preference weight vector. We train until the validation loss increases, which was at epoch #7 . We used the learning rate of 1e-6 for all models and the batch size of 64. For Pareto-front generation, we use the following weight combinations. For the two-requirement scenarios, $w_1 = [0, 1, 1]$ and $w_2 = [1, 0, 1]$. For the three-requirement scenarios, $w_1 = [0, 0, 1, 0, 1, 1, 1]$, $w_2 = [0, 1, 0, 1, 0, 1, 1]$, and $w_3 = [1, 0, 0, 1, 1, 0, 1]$.

C.5 e-SimFT.

Relevant SimFT models are chosen based on the design scenario and we allocate $\mathcal{N}/2$ or $\mathcal{N}/3$ (for two- or three-requirements) sampling budget to each model. We then create an evenly spaced values of ϵ, sized either $\mathcal{N}/2$ or $\mathcal{N}/3$, within [-5, 5] for original requirements and [0, 10] for new requirements, and add these values to the requirement values before sampling SimFT models.

We also test two conditions as ablation: SimFT only or epsilon-sampling only. For the former, we use the same sampling budget allocation as e-SimFT with relevant SimFT models but do not employ epsilon-sampling. For the latter, we follow the same epsilon-varying schedule as the e-SimFT method but use the pre-trained GearFormer model.

C.6 Evaluation metric.

For each requirement, the degree of constraint violation for the design solutions generated by different models is normalized as $[0, 1]$. These values are used to determine Pareto-optimal solutions for each problem and the hypervolume of the Pareto front [20] is used to compare e-SimFT versus other baselines.

C.7 Dataset

We use the validation and test portion of the original GearFormer dataset [5] for all our fine-tuning and testing, $|D| = 7360$. 5% of the dataset was withheld for testing SimFT methods including their ablation studies. The rest was used for fine-tuning, i.e., $D_{\text{ft}} \cup D_{\text{test}} = D, |D_{\text{ft}}| = 6992, |D_{\text{test}}| = 368$.

For the original requirement SimFT, use the whole D_{ft} for SFT; where 90% is used for training and 10% is held for validation.

For the new requirement SimFT, use the first half $D_{\text{ft},1}$ for SFT and the other half $D_{\text{ft},2}$ for DPO/RL, i.e., $D_{\text{ft},1} \cup D_{\text{ft},2} = D_{\text{ft}}, |D_{\text{ft},i}| = 3496$. Again for each subset, we use 90% for training and 10% for validation.

C.8 Training SFT models.

SFT for original requirements: Training was performed until the validation loss increased. For the speed SimFT, it was stopped at epoch #16, and for the position SimFT, it was stopped at epoch #19. We used the learning rate of 1e-6 and the batch size of 64 for both.

SFT for new requirements: Training was performed until the validation loss increased. For both the speed and position SimFT models, it was stopped at epoch #6. We used the learning rate of 1e-5 (including the new encoder) and the batch size of 64 for both.

DPO for new requirements: Training was carried out for 20 epochs and the model checkpoint with the best requirement improvement was picked *post hoc*, subject to the criterion that 95% of the generated solutions are valid. For the cost SimFT, this was at epoch #15 while for the boundinb box volume SimFT, it was at epoch #10. We used the learning rate of 1e-6, $\beta = 0.1$, and the batch size of 64 for both.

PPO for new requirements: Same as DPO, training was done for 20 epochs and the model checkpoint with the best requirement improvement was picked *post hoc*, subject to the criterion that 95% of the generated solutions are valid. For the cost SimFT, this was at epoch #9 while for the bounding box volume SimFT, it was at epoch #4. We used the learning rate of 1e-5, $\beta = 0.1$, the rollout size of 64, and the mini-batch size of 8 for both.

For the reward function, we used the following normalization function based on the evaluation of each solution using a simulator:

$$\mathcal{R}(x) = \begin{cases} 1, & \text{if } \tilde{n}(x) \leq n \\ 1 - \frac{2(\tilde{n}(x)-n)}{1+\tilde{n}(x)-n}, & \text{otherwise} \end{cases}$$

where x is the solution, n is the target requirement value, and $\tilde{n}(x)$ is the evaluated requirement value. Note that because we compute the exact reward for each solution as a whole, we employed an actor-only method (i.e., no critic model).

D Full Experimental Results

D.1 Evaluation of SimFT methods

Table 1 presents the improvements made by SimFT methods for each requirement. As an example, Figure 6 shows solutions obtained for the same design problem with the original GearFormer model vs. a SimFT model.

Importance of offline rejection sampling for SFT. To show the importance of using only the samples that satisfy the equality constraint requirements, we ran an ablation study where any new sample drawn was accepted for the SFT training data. The performances for the speed and position requirements dropped to 0.0179 and 0.0339, compared to the baseline performances of 0.0171 and 0.0338.

Performance trade-offs during DPO. Because DPO is performed using a preference loss that differs from the original cross-entropy loss used for the pre-trained model, over-training the model can lead to significant deterioration of its original performance. Figure 7 shows that after 16 and 12 epochs, respectively, the percentage of valid designs produced by GearFormer drops below 95% (the performance reported in [5]) and at a significant rate in the subsequent epochs. Note that the DPO training loss and the requirement-met performance continues to improve over these epochs. Therefore, the best model checkpoint should be picked after confirming that other performances of the model have not deteriorated below the required thresholds.

Using binary reward function for PPO. Considering the new requirements are inequality constraints, we could implement a binary reward function for PPO that simply assigns a score of 1 or -1 if the requirement is met or not. We found that fine-tuning the model with this reward function for 20 epochs, the best accuracies were 62.3% for the cost requirement and 60.4% for the bounding box volume requirement, versus 65.4% and 59.1% reported in Table 1.

D.2 Evaluation of e-SimFT for Pareto front generation

We considered two settings for the hypervolume comparison. In the first setting, we used the sampling budget of $\mathcal{N} = 30$, which for each design scenario takes about 90 seconds of inference and simulation time on a machine with a Tesla V100-SXM2-16GB GPU and a AMD EPYC 16-core processor. This is assumed to be a relatively short time that an engineer needs to wait to obtain a Pareto front with multiple optimal solutions. The results for the first setting was shown in Table 2, showing that on average, e-SimFT is the best method for both two- and three-requirements scenarios. One could also observe that on average, SimFT and epsilon-sampling on their own do not perform as well as e-SimFT.

In the second setting, we set $\mathcal{N} = 300$ to examine how each method scales with increased sampling. Table 3 shows a similar pattern when the sampling size is increased by 10 times. On average, e-SimFT is still found to be the best for both two- and three-requirement scenarios.

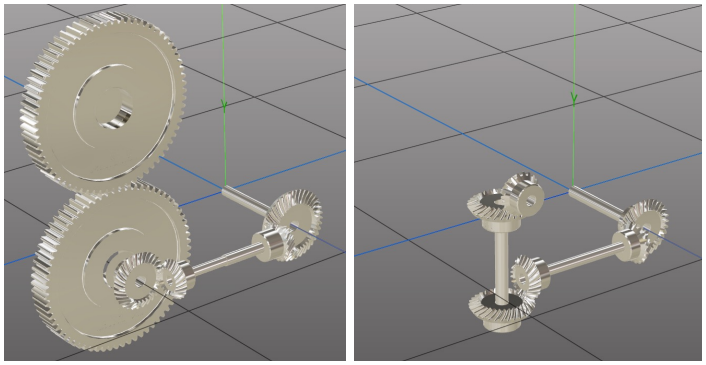


Figure 6: Example of gear designs produced for a sample problem with the original GearFormer versus a SimFT model fine-tuned for bounding box volume. The first design has a volume of $0.018m^3$ while the second design has a much lower volume of $0.008m^3$.

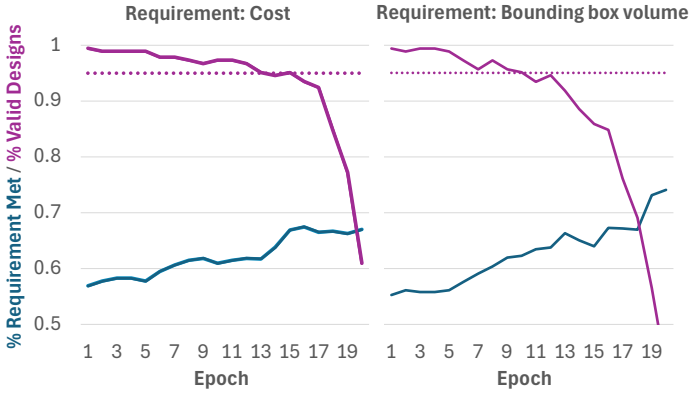


Figure 7: DPO improves the percentage of requirements met at the expense of the percentage of valid designs.

Table 3: Comparison of Pareto front hypervolumes. Number of test problems = 30 and samples = 300 per scenario.

Design Scenario	**Method**					
	Baseline	R. Soup	R.-in-Context	SimFT	e-sample	e-SimFT
speed : position	$0.68^{\pm 0.17}$	$0.70^{\pm 0.16}$	$0.68^{\pm 0.19}$	$0.70^{\pm 0.16}$	$0.77^{\pm 0.13}$	$\mathbf{0.78}^{\pm 0.14}$
speed : cost	$0.71^{\pm 0.22}$	$0.72^{\pm 0.22}$	$0.72^{\pm 0.21}$	$0.72^{\pm 0.22}$	$0.76^{\pm 0.19}$	$\mathbf{0.78}^{\pm 0.20}$
speed : b.box	$0.83^{\pm 0.15}$	$0.84^{\pm 0.20}$	$0.87^{\pm 0.14}$	$0.87^{\pm 0.15}$	$0.87^{\pm 0.15}$	$\mathbf{0.91}^{\pm 0.13}$
position : cost	$0.58^{\pm 0.21}$	$0.58^{\pm 0.22}$	$0.57^{\pm 0.23}$	$0.59^{\pm 0.23}$	$0.59^{\pm 0.24}$	$\mathbf{0.59}^{\pm 0.24}$
position : b.box	$0.63^{\pm 0.20}$	$0.56^{\pm 0.21}$	$0.63^{\pm 0.20}$	$0.62^{\pm 0.19}$	$\mathbf{0.68}^{\pm 0.19}$	$0.65^{\pm 0.19}$
cost : b.box	$0.72^{\pm 0.23}$	$0.62^{\pm 0.26}$	$\mathbf{0.70}^{\pm 0.24}$	$0.69^{\pm 0.25}$	$0.68^{\pm 0.26}$	$0.67^{\pm 0.27}$
mean	$0.69^{\pm 0.20}$	$0.67^{\pm 0.22}$	$0.70^{\pm 0.20}$	$0.70^{\pm 0.21}$	$0.72^{\pm 0.20}$	$\mathbf{0.73}^{\pm 0.20}$
speed : position : cost	$0.51^{\pm 0.23}$	$0.50^{\pm 0.24}$	$0.52^{\pm 0.24}$	$0.53^{\pm 0.23}$	$0.57^{\pm 0.22}$	$\mathbf{0.59}^{\pm 0.23}$
speed : position: b.box	$0.57^{\pm 0.19}$	$0.52^{\pm 0.21}$	$0.58^{\pm 0.21}$	$0.59^{\pm 0.19}$	$0.65^{\pm 0.17}$	$\mathbf{0.66}^{\pm 0.18}$
speed : cost : b.box	$0.60^{\pm 0.23}$	$0.60^{\pm 0.25}$	$0.62^{\pm 0.23}$	$0.65^{\pm 0.23}$	$0.66^{\pm 0.23}$	$\mathbf{0.69}^{\pm 0.23}$
position : cost : b.box	$0.47^{\pm 0.22}$	$0.41^{\pm 0.23}$	$0.49^{\pm 0.23}$	$0.50^{\pm 0.23}$	$\mathbf{0.52}^{\pm 0.23}$	$0.50^{\pm 0.25}$
mean	$0.54^{\pm 0.22}$	$0.51^{\pm 0.23}$	$0.55^{\pm 0.23}$	$0.57^{\pm 0.22}$	$0.60^{\pm 0.22}$	$\mathbf{0.61}^{\pm 0.22}$

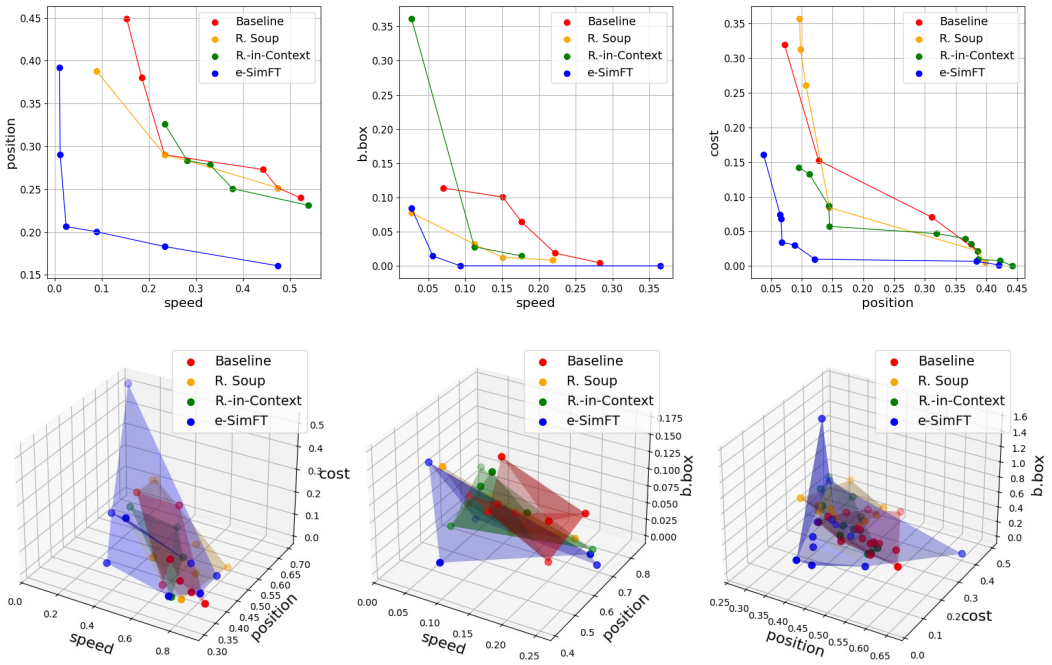


Figure 8: Pareto fronts generated by e-SimFT versus other methods for sample design problems.