PerturBench: Benchmarking Machine Learning Models for Cellular Perturbation Analysis

Yan Wu[*]
Altos Labs
San Diego, US

Esther Wershof[*]
Altos Labs
Cambridge, UK

Sebastian M Schmon[*]
Altos Labs
Cambridge, UK

Marcel Nassar[*]
Altos Labs
San Diego, US

Błażej Osiński[*]
Altos Labs
Cambridge, UK

Ridvan Eksi[*]
Altos Labs
San Diego, US

Kun Zhang
Altos Labs
San Diego, US

Thore Graepel
Altos Labs
Cambridge, UK

Abstract

We present a comprehensive framework for predicting the effects of perturbations on cellular states, designed to standardize benchmarking in this rapidly evolving field. Our framework, PerturBench, includes a user-friendly platform, diverse datasets, metrics for fair model comparison, and detailed performance analysis. Extensive evaluations of published and baseline models reveal limitations like mode or posterior collapse, and underscore the importance of rank metrics that assess the ordering of perturbations alongside traditional measures like RMSE. Our findings show that simple models can outperform more complex approaches. This benchmarking exercise sets new standards for model evaluation, supports robust model development, and advances the potential of these models to use high-throughput and high-content genetic and chemical screens for disease target discovery.

1 Introduction

Experiments in which biological systems, such as cell lines, are perturbed through chemical treatments or genetic modifications, can help unravel the causal drivers of diseases and identify promising therapeutics. Advances in CRISPR technology and automation have enabled these experiments, which we refer to as perturbation screens, to be conducted at scale with up to hundreds of thousands of perturbations applied in parallel in a single experiment (Shalem et al., 2014). These perturbation screens have been coupled with modern RNA-sequencing technology (see e.g. Lowe et al., 2017, for an overview) that can measure gene expression profiles at single cell resolution, creating a rich understanding of perturbation effects (Tang et al., 2009; Macosko et al., 2015; Adamson et al., 2016; Dixit et al., 2016; Bock et al., 2022; Srivatsan et al., 2020). However, measuring the effect of perturbing the tens of thousands of currently known genes or the 10^{60} known drug-like chemicals across different tissues and cell types is prohibitively expensive with current technologies, especially for combinations of perturbations (Replogle et al., 2022; Reymond, 2015). Thus, there is an increased interest in computational approaches that can predict the effects of perturbations on gene expression.

[*] Equal contribution. Author order determined by random sampling.

AI for New Drug Modalities at NeurIPS 2024.

Specifically, researchers have developed models that can generate counterfactual, out of sample (oos) predictions of perturbation effects (Gavriilidis et al., 2024). One use case, which we call covariate transfer, involves training a model on perturbation effects measured in a set of covariates (i.e. cell lines) and predicting those effects in another covariate where they were never observed. Another use case, which we call combo prediction, involves training a model on individual perturbation effects and predicting the effects of multiple perturbations in combination. The ultimate goal of these models is to enable in-silico screening across the vast space of unobserved perturbations and pave the way for targeted disease treatment strategies.

It is difficult to judge the performance of published perturbation effect prediction models against each other due to differences in benchmarking datasets and metrics. The NeurIPS 2023 perturbation prediction challenge helped address this issue by providing a chemical perturbation dataset with a model evaluation framework (Burkhardt et al., 2023). However, the challenge was limited to a single dataset on the covariate transfer task and models were evaluated using a single metric that may not fully capture performance. Additionally, the challenge did not benchmark published models. The *sc-perturb* database provides datasets with unified metadata, but does not attempt to benchmark models (Peidli et al., 2024). Ahlmann-Eltze et al. (2024) and Wenteler et al. (2024) assessed the performance of single cell foundation models and GEARS on only the combo prediction and unseen perturbation prediction tasks, and both studies used MSE as their evaluation metric, which again may not capture key aspects of model performance.

In this work, we provide a rigorous quantitative assessment of the field of perturbation response modelling by: **(1)** introducing a highly modular and user-friendly framework in the form of a code base for model development and evaluation, **(2)** curating diverse perturbational datasets and defining biologically relevant tasks, **(3)** defining a set of metrics that enable comparison of different models on an equal basis, **(4)** performing extensive evaluation of existing perturbation models and their individual components across different datasets. Figure 1 illustrates our approach.

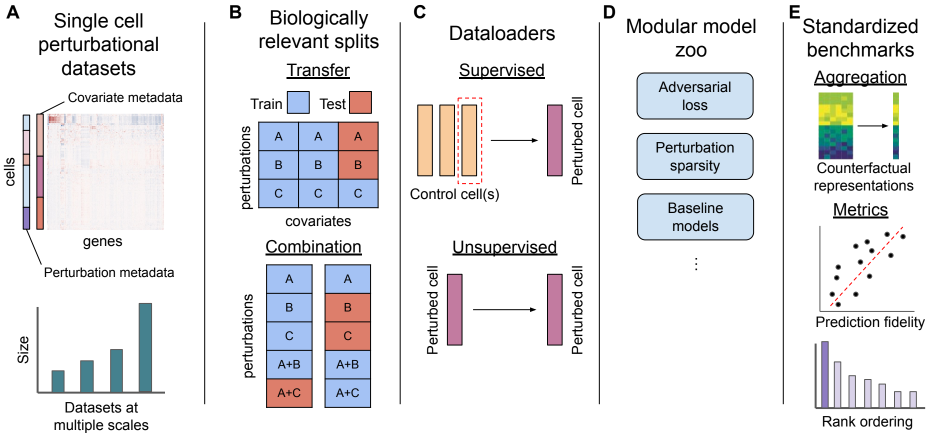


Figure 1: **A**) Single cell perturbational datasets at multiple scales. **B**) Biologically relevant covariate transfer and combinatorial prediction data splits. **C**) Dataloaders support two training modes: 1) supervised mode which involves mapping a sampled control cell to a perturbed cell and 2) unsupervised (or autoencoder) mode which maps a perturbed cell to itself. **D**) A model zoo with modular components such as relevant baseline models, adversarial loss, perturbation sparsity, and others. **E**) Standardized benchmarking suite supporting flexible pipelines and metrics for evaluating models

We reproduce key components of published models that cover a spectrum of architectures, and evaluate them against each other and strong baseline models. We specifically test the models on difficult tasks, simulating how they will be deployed in real-life contexts. Our findings reveal that some widely used models are prone to "mode" or "posterior" collapse (see Appendix C.1 for more details). Since a common use-case of these models is to run in-silico screens that rank perturbations by a desired effect (i.e. reversing a disease state) (Van de Sande et al., 2023), we propose rank metrics complementary to traditional measures of model fit (e.g. root mean squared error (RMSE)) that specifically capture model collapse. In addition, we demonstrate that models with simple architectures can outperform some of the more sophisticated models. In total, our benchmarking required more than 4 thousands GPU hours.

We anticipate that our code base, together with this benchmark and the accompanying metrics, will serve as a valuable framework for the community to develop robust and improved models.

2 Datasets and Tasks

Many published models have been evaluated on relatively small datasets, with a split where most of the data is used for training. However, in a real-world setting, we often have complex datasets that only contain a small fraction of the perturbation effects we are interested in predicting. Thus, we select datasets and tasks that mirror these real-world challenges. We select five published datasets, Norman et al. (2019), Srivatsan et al. (2020), Frangieh et al. (2021), McFaline-Figueroa et al. (2024), and Jiang et al. (2024a) which include at least 100 perturbations and cover a diversity of perturbation modalities (chemical vs genetic), combinatorial perturbations, dataset sizes, and covariates. We provide a cursory overview in Table 1 and more information in Appendix D.1. Here, we define a biological state as a unique set of covariates that we plan to model (i.e. cell line). Details of the preprocessing of the datasets are in the Appendix D.3.

Table 1: Summary of benchmarking datasets.

Dataset	Single perturbations	Dual perturbations	modality	biological states	cells	tasks
Srivatsan20	188	0	chemical	3	178,213	covariate transfer
Frangieh21	248	0	genetic	3	218,331	covariate transfer
Jiang24	219	0	genetic	30	1,628,476	covariate transfer
McFalineFigueroa23	525	0	genetic	15	892,800	covariate transfer
Norman19	287	131	genetic	1	91,168	combo prediction

We create a covariate transfer split for the Srivatsan20, Frangieh21 and Jiang24 datasets as well as a combo prediction task for the Norman19 dataset. In addition, we study two **scenarios**: data scaling and imbalanced data. In the former, we benchmark model performance with increasing number of additional data. In the later, we simulate a situation where data of some covariate type is more abundant than that of another. Details of the experiments are in sections 5.4 and 5.5 respectively. The aim of both scenarios is to simulate how models will be deployed in practice, where there are often complex covariates, imbalanced datasets, and/or large amounts of missing data (see e.g. Edwards, 2024). Additional details about data splitting implementation can be found in the Appendix D.4.

3 Perturbation Prediction Models

3.1 Modeling counterfactuals

Perturbation response modeling aims to predict out-of-sample effects of genetic or chemical interventions on cells. Here, we define out of sample as predicting effects in unobserved biological states or unobserved combinations of interventions. However, RNA sequencing technology destroys the cell, making it impossible to observe its gene expression state before and after perturbation. Building machine learning models that can predict counterfactual responses therefore requires modeling approaches that can identify causal or mechanistic features, meaning features that describe the direct effect of an intervention as opposed to attributes that are correlated but without a direct effect. Published models use two main strategies to learn representations of perturbation effects: *matching methods* to relate control and perturbed cells, or *disentanglement* strategies within autoencoder architectures to separate the effects of perturbations from the baseline cell state.

Matched Controls Matching treated outcomes to controls is a common approach to identify treatment effects (see e.g. Stuart, 2010, Section 1.3 for a historical summary). In the context of perturbation effect prediction, matching of perturbed cells with controls has been used by a variety of published models such as GEARS (Roohani et al., 2023), scGPT (Cui et al., 2024), and scFoundation (Hao et al., 2023). However, the effectiveness and validity of matched controls depend on fulfilling certain assumptions to ensure measuring a *causal* effect. One such assumption is ignorability, which posits that once covariates are controlled for, no residual confounding effects should influence the comparison between control and treatment groups (see e.g. Stuart, 2010). This can include, for instance, ensuring that the control cell is from the same cell type, experiment or batch. Some more complex methods involve using optimal transport to identify the control cell most likely to transition into a given perturbed cell (Jiang et al., 2024b) or even to use optimal transport to assist in perturbation prediction (Bunne et al., 2023). However, due to limited scalability of common algorithms to solve the optimal transport problem, these models have not been applied to larger datasets with more than 100 perturbations.

Disentanglement An alternative to *matching methods* involves *disentanglement* (Bengio et al., 2013), which enables models to learn separate representations for distinct concepts. In the context of perturbation models, the key disentanglement task is to separate the unperturbed cellular state and the perturbation effect. The compositional perturbation autoencoder (CPA) (Lotfollahi et al., 2023) uses an adversarial classifier to ensure that the unperturbed "basal" state is free of any perturbational information, forcing the perturbation encoder to learn a meaningful representation of the perturbation. These representations are added to control cell encodings during inference to generate counterfactual predictions. Biolord (Piran et al., 2024) partitions the latent space into subspaces and optimizes those latent spaces to represent covariates and perturbations, which can be recombined during inference to generate counterfactual predictions. sVAE (Lopez et al., 2022) leverages recent results by Lachapelle et al. (2022) demonstrating that enforcing a *sparsity* constraint can induce disentanglement. Bereket and Karaletsos (2024) build on sVAE using an additive conditioning for the perturbations and demonstrate that their model, SAMS-VAE, offers biological interpretability of the latent encodings.

3.2 Models for benchmarking

In this paper we implement a range of perturbation response models inspired by some of the state-of-the-art models in the field: CPA, Biolord, SAMS-VAE and GEARS. Our aim is to assess the core modeling component behind each model, such as the adversarial classifier for CPA, and thus there are differences between some of our implementations and the published versions, see Appendix D.5. We therefore refer to our implementations with a * following the model name (i.e. CPA*). We further encode gene expression values using scGPT, a single cell gene expression foundation model (Cui et al., 2024), and used the resulting embeddings as inputs to CPA and our latent additive baseline model. Our aim is to gauge whether using embeddings from a pretrained foundation model would improve performance. As many prior studies lacked strong baseline models, we also implement and benchmark the following baselines.

3.3 Baseline models

Linear The linear baseline model uses the *control matching* approach. Given a perturbed cell, x', we sample a random control cell with *matched* covariates, x, and reconstruct x' by applying one linear layer given the perturbation and covariates:

$$x' = f_{\text{linear}}(p_{\text{one\_hot}}, cov_{\text{one\_hot}}), \tag{1}$$

where $p_{\text{one\_hot}}$ denotes the one-hot encoding of the perturbation and $cov_{\text{one\_hot}}$ denotes one-hot encodings of covariates (e.g. cell type).

Latent Additive We extend the linear model into a baseline latent additive model by encoding expression values and perturbations into a latent space $\mathsf{Z} \subseteq \mathbb{R}^{d_z}$, i.e.

$$z_{\text{ctrl}} = f_{\text{ctrl}}(x), \quad \text{and} \quad z_{\text{pert}} = f_{\text{pert}}(p_{\text{one\_hot}}),$$

Subsequently, we reconstruct the expression value by decoding the added latent space representation $x' = f_{\text{dec}}(z_{\text{ctrl}} + z_{\text{pert}})$. All functions $f_{\text{dec}}, f_{\text{ctrl}}, f_{\text{pert}}$ are implemented as multilayer perceptrons (MLPs) with dropout (Srivastava et al., 2014) and layer normalization (Ba et al., 2016).

Decoder Only We also introduce a model class that does not use gene expression as an input and aims to predict the perturbation effect solely from covariates, $cov_{\text{one\_hot}}$, perturbation information,

$p_{\text{one\_hot}}$, or a mix of both. Consequently, prediction of the expression of a perturbed cell can be modelled as $x' = f_{\text{dec}}(z)$ for $z \in \{p_{\text{one\_hot}}\} \cup \{cov_{\text{one\_hot}}\} \cup \{(p_{\text{one\_hot}}, cov_{\text{one\_hot}})\}$ and we refer to them as *decoder only* models. For example, by training a covariate only model, we can assess how a model might perform if it completely ignored any perturbation information.

4 Benchmarking

Existing perturbation response modeling studies have used different metrics for different models, making comparison between models difficult. We develop a standardized benchmarking approach with a suite of metrics together with a simple API to facilitate evaluation and comparability of established and new perturbation models, see Appendix A for details.

4.1 Population Aggregation

If it was possible to measure the gene expression state of a cell before and after perturbation, a model could take as input the cell state before perturbation and predict the state after perturbation, making evaluation straightforward. Since this is not possible, a model takes as unperturbed cell states as input and predicts what their states *would* look like if they had been perturbed. To evaluate these counterfactual predictions, we thus compare the means of the predicted vs observed cell states and the predicted vs observed LogFCs. Using LogFCs helps focus the evaluation on the gene expression changes due to perturbation.

4.2 Metrics

We select metrics that capture the performance of perturbation response prediction models in real world applications, specifically using RMSE (as recommended by Ji et al. 2023) to compare the average predicted cell states to the observed cell states. We also use cosine similarity to assess the similarity of the predicted versus observed perturbation effects as measured by LogFCs. However, as we show in Appendix C.1, these "global" fit-based metrics do not fully capture all aspects of a model's performance. Hence, we introduce a set of rank-based metrics that can be seen as measures of *precision*.

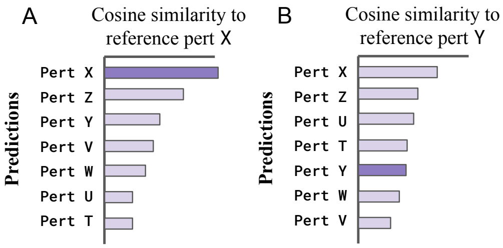


Figure 2: Visualization of the ranking approach. We measure which perturbation in the data is closest to the predicted perturbation as measured by the closeness of their transcriptomes.In case A the rank metric for prediction X is $\text{rank}(X) = \frac{0}{6} = 0$, in case B $\text{rank}(Y) = \frac{4}{6} = 0.67$.

4.2.1 Rank Metrics

Since the space of possible genetic or chemical perturbations is so massive, a common application of these models is to rank the predicted effects of these perturbations by their similarity to some desired gene expression effect to identify a smaller set of perturbations for experimental testing (Van de Sande et al., 2023). Thus, the degree to which the predicted perturbations are ordered correctly is critical, as the top ranked perturbations will be experimentally tested. This ordering has been overlooked in the field thus far resulting in models that generate predictions with high cosine similarity or low RMSE to the observed gene expression state, but fail to capture smaller but key changes that uniquely distinguish the effects of one perturbation from others.

We find it difficult to use existing information retrieval metrics such as the mean reciprocal rank (Radev et al., 2002) or mean average precision because we do not know in advance what the desired gene expression state and thus cannot partition perturbations into relevant and not relevant classes for these retrieval metrics. Hence, we introduce a rank-based metric that measures how well the predicted perturbations are ordered. Specifically, for a given observed perturbation, the prediction for that perturbation should be more similar than predictions for other perturbations. The rank metric is computed on a per perturbation basis:

$$\text{rank}_{\text{average}} := \frac{1}{p}\sum_{i=1}^{p} \text{rank}(\hat{x}_i), \quad \text{rank}(\hat{x}_i) := \frac{1}{p-1}\sum_{\substack{1\leq j\leq p\\ j\neq i}} \mathbb{I}(\text{dist}(\hat{x}_j, x_i) \leq \text{dist}(\hat{x}_i, x_i)), \quad (2)$$

where p is the number of perturbations in that are being modelled and $\hat{x}_i, x_i$ are the predicted and observed (average) expression value of perturbation i and dist is a generic distance. Figure 2 shows two examples of perturbations predictions and the respective computations of rank metrics. The rank metric is always a number between 0 and 1, where 0 is a perfect score and 0.5 is the expected score of a random prediction. Each fit-based metric has a corresponding rank metric.

4.3 Benchmarking Rules

The performance of each model critically depends on the judicious choice of hyperparameters. To identify suitable hyperparameters for each model, we run hyperparameter optimization (HPO) for each model on each dataset, task and scenario. Specifically, we use `optuna` (Akiba et al., 2019) with the default tree-structured Parzen estimator (Bergstra et al., 2011) to sample hyperparameters. Each model is run on a node with at least 64GB of RAM and one Nvidia A10 GPU; for the biggest dataset and models even stronger nodes are used. For every model we carry out training with at least 60 hyperparameter configurations and we describe the specific hyperparameters and their ranges in the Appendix D.6. For the best configuration we run model training four additional times with different seeds to assess stability of training. Error bars represent the standard deviation of model performance. To balance the two objectives, RMSE and corresponding ranking, we carried out a pilot HPO run and found that using a combination of the RMSE and $0.1\cdot$ rank results in good overall performance. We include pilot run details in Appendix D.6. Hence, for HPO we define the loss function $\mathcal{L}_{\text{HPO}} = \text{RMSE} + 0.1 \cdot \text{rank}_{\text{RMSE}}$.

5 Experiments

In this section, we summarize the main results of the two **tasks** (covariate transfer and combo prediction) and the two **scenarios** (data scaling and imbalanced data). Additional results, figures and implementation details can be found in the Appendix.

5.1 Predicting Drug Effects Across Cell Types

We begin with the covariate transfer task and assess the models' ability to predict the effects of drug treatment in cell types where the drugs have not been observed. To this end, for each cell line in the `Srivatsan20` dataset, we create a data split by holding out 30% of the perturbations for validation and testing. We ensure that each held out perturbation is observed in the two other cell types.

The results are summarized in Table 2. Interestingly, we see that the baseline models tend to outperform more sophisticated models. The latent additive model (LA) with scGPT embeddings performs strongest overall, indicating that scGPT cell embeddings provide rich representations for predicting perturbation effects. This is followed by the default latent additive model. SAMS-VAE* is the second best performing model, performing similar to or better than other published models on all metrics.

The decoder-only baseline (Decoder) that uses both perturbations and covariates does not achieve the same cosine LogFC but shows similar performance in terms of rank metrics. Both models outperform the more sophisticated CPA*, and BioLord* models, both in terms of the standard cosine and RMSE metrics and the ranking metrics. We investigate disentanglement in CPA* by removing the adversarial classifier from CPA* (CPA* noAdv), which increased the variability in the benchmarking metrics and seemed to slightly improve on the average performance. This is a surprising observation and suggests that CPA's perturbation encoder is able to learn a meaningful representation even without the adversarial classifier.

It is worth noting that the decoder-only baseline that only uses covariates and has no perturbation information, (Decoder (Cov)) achieves a fairly high cosine similarity and low RMSE. This suggests that it is possible for a model to find a single expression vector that is similar to all perturbations in

Table 2: Results of the first covariate transfer experiment (mean ± one standard deviation). Model performance generalizing across cell types in the Srivatsan20 dataset. Best performance per metric is indicated in bold.

Model	Cosine log fold change (LogFC)	RMSE mean	Cosine LogFC rank	RMSE mean rank
CPA*	$0.31 \pm 1 \times 10^{-2}$	$0.021 \pm 4 \times 10^{-4}$	$0.35 \pm 6 \times 10^{-3}$	$0.32 \pm 7 \times 10^{-3}$
CPA* (noAdv)	$0.37 \pm 4 \times 10^{-2}$	$0.020 \pm 8 \times 10^{-4}$	$0.33 \pm 3 \times 10^{-2}$	$0.29 \pm 7 \times 10^{-3}$
CPA* (scGPT)	$0.29 \pm 9 \times 10^{-4}$	$0.021 \pm 3 \times 10^{-4}$	$0.38 \pm 2 \times 10^{-2}$	$0.32 \pm 1 \times 10^{-2}$
SAMS-VAE*	$0.44 \pm 1 \times 3^{-3}$	$0.023 \pm 8 \times 10^{-5}$	$0.17 \pm 1 \times 10^{-2}$	$0.17 \pm 1 \times 10^{-2}$
Biolord*	$0.18 \pm 1 \times 10^{-1}$	$0.086 \pm 4 \times 10^{-2}$	$0.37 \pm 2 \times 10^{-2}$	$0.35 \pm 1 \times 10^{-1}$
LA	$0.45 \pm 2 \times 10^{-3}$	$0.018 \pm 6 \times 10^{-5}$	$\mathbf{0.13 \pm 4 \times 10^{-3}}$	$0.15 \pm 3 \times 10^{-3}$
LA (scGPT)	$\mathbf{0.50 \pm 4 \times 10^{-3}}$	$\mathbf{0.017 \pm 1 \times 10^{-4}}$	$\mathbf{0.13 \pm 7 \times 10^{-3}}$	$\mathbf{0.14 \pm 5 \times 10^{-3}}$
Decoder	$0.35 \pm 5 \times 10^{-3}$	$0.018 \pm 1 \times 10^{-4}$	$0.16 \pm 1 \times 10^{-2}$	$\mathbf{0.14 \pm 7 \times 10^{-3}}$
Decoder (Cov)	$0.30 \pm 1 \times 10^{-2}$	$0.023 \pm 3 \times 10^{-5}$	$0.47 \pm 9 \times 10^{-3}$	$0.50 \pm 4 \times 10^{-2}$
Linear	$0.16 \pm 1 \times 10^{-2}$	$0.030 \pm 5 \times 10^{-4}$	$0.28 \pm 5 \times 10^{-3}$	$0.27 \pm 2 \times 10^{-3}$

a given cell type, which again highlights the need for rank metrics that assess whether models can correctly order perturbations. See Appendix C.1 for a more detailed assessment.

CPA performs worse than the linear (Linear) baseline model in the rank metrics, suggesting that CPA* does not predict the unique effects of each perturbation as well. Biolord* shows high variance between training runs, suggesting the model initialization plays a large role in model performance for this task.

5.2 Generalizing from less complex to more complex biological systems

We then applied our model zoo and benchmarking suite to a highly relevant real world task: predicting perturbation effects in a more complex disease system using effects in less complex systems. The Frangieh21 dataset contains 3 biological systems: primary melanoma cells cultured alone, with IFNγ, and co-cultured with tumor infiltrating immune cells. We held out 70% of the perturbations in the co-culture system and used the remaining perturbations as well as all perturbations in the other systems as training.

The results are summarized in Table 3. The latent additive model, CPA* both with and without scGPT embeddings, and SAMS-VAE* are best at predicting the expression similarity as measured by the cosine LogFC and RMSE. However, CPA* and SAMS-VAE* do not perform well on either rank metric, while the latent additive model and BioLord* perform similarly well on the rank metric. The decoder only model performs best on both rank metrics while underperforming CPA* and the latent additive model on the cosine logFC and RMSE metrics.

5.3 Predicting Combinatorial Gene Overexpression Effects

In this section we discuss the results of the combo prediction experiment based on the Norman19 dataset, which contains both single and dual genetic perturbations. We use all of the single perturbations and randomly select 30% of the dual perturbations for training, and hold out the remaining 70% for validation and testing. We summarize the results in Table 4.

The linear model performs relatively well in predicting combinatorial perturbation effects, suggesting that most perturbation effects are approximately linearly additive in this dataset. The latent additive and decoder models were able to outperform the linear model in all metrics, suggesting they learned some non-linear interactions. The performance of the latent additive and decoder models are very similar, with the latent additive performing slightly better on rank metrics. This similarity suggests that the input gene expression values are not critical for strong model performance. CPA* and GEARS performs worse than the linear model in all metrics. Biolord* and SAMS-VAE* are worse than the linear model in the traditional cosine and RMSE metrics, but performs similarly with regards to the ranking metrics.

Table 3: Results of a covariate transfer experiment (mean ± one standard deviation) generalizing from less complex biological systems to a more complex co-culture system in the `Frangieh21` dataset. Results are reported as mean ± one standard deviation and best performance per metric is indicated in bold.

Model	Cosine LogFC	RMSE mean	Cosine LogFC rank	RMSE mean rank
CPA*	$0.17 \pm 1 \times 10^{-2}$	$0.027 \pm 6 \times 10^{-4}$	$0.41 \pm 2 \times 10^{-2}$	$0.42 \pm 1 \times 10^{-2}$
CPA* (scGPT)	$0.16 \pm 1 \times 10^{-2}$	$0.026 \pm 2 \times 10^{-3}$	$0.46 \pm 7 \times 10^{-3}$	$0.48 \pm 3 \times 10^{-2}$
SAMS-VAE*	$0.15 \pm 2 \times 10^{-2}$	$0.026 \pm 2 \times 10^{-4}$	$0.48 \pm 2 \times 10^{-2}$	$0.46 \pm 2 \times 10^{-2}$
BioLord*	$0.12 \pm 9 \times 10^{-3}$	$0.027 \pm 2 \times 10^{-4}$	$0.29 \pm 3 \times 10^{-2}$	$0.21 \pm 4 \times 10^{-2}$
LA	$\mathbf{0.17 \pm 6 \times 10^{-3}}$	$\mathbf{0.024 \pm 4 \times 10^{-4}}$	$0.26 \pm 1 \times 10^{-2}$	$0.21 \pm 2 \times 10^{-2}$
LA (scGPT)	$\mathbf{0.18 \pm 6 \times 10^{-3}}$	$\mathbf{0.024 \pm 6 \times 10^{-5}}$	$0.27 \pm 1 \times 10^{-2}$	$0.24 \pm 1 \times 10^{-2}$
Decoder	$0.10 \pm 2 \times 10^{-3}$	$\mathbf{0.025 \pm 4 \times 10^{-5}}$	$\mathbf{0.21 \pm 5 \times 10^{-3}}$	$\mathbf{0.15 \pm 4 \times 10^{-4}}$
Linear	$0.0081 \pm 4 \times 10^{-4}$	$0.043 \pm 7 \times 10^{-5}$	$0.24 \pm 9 \times 10^{-4}$	$0.30 \pm 2 \times 10^{-3}$

Table 4: Results of the first combo prediction experiment (mean ± one standard deviation). Model performance predicting dual perturbation effects in the `Norman19` dataset. Best performance per metric is indicated in bold.

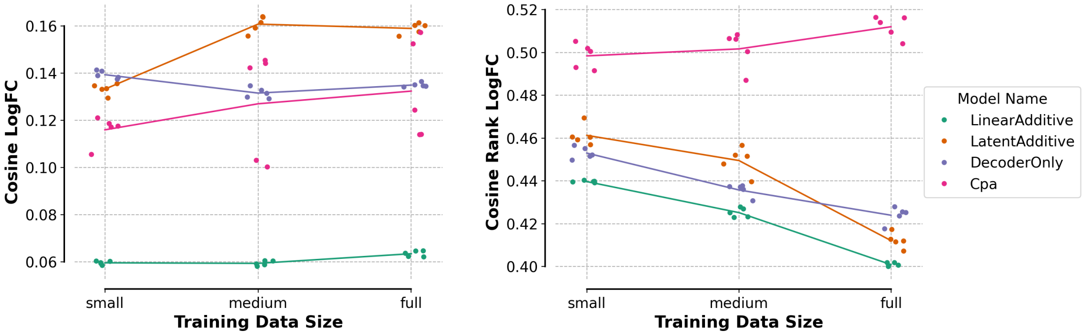
Model	Cosine LogFC	RMSE mean	Cosine LogFC rank	RMSE mean rank
CPA*	$0.52 \pm 6 \times 10^{-2}$	$0.079 \pm 5 \times 10^{-3}$	$0.12 \pm 2 \times 10^{-2}$	$0.17 \pm 3 \times 10^{-2}$
CPA* (noAdv)	$0.55 \pm 9 \times 10^{-2}$	$0.073 \pm 6 \times 10^{-3}$	$0.16 \pm 4 \times 10^{-2}$	$0.18 \pm 2 \times 10^{-2}$
CPA* (scGPT)	$0.70 \pm 1 \times 10^{-2}$	$0.061 \pm 2 \times 10^{-3}$	$0.064 \pm 1 \times 10^{-2}$	$0.13 \pm 2 \times 10^{-2}$
SAMS-VAE*	$0.45 \pm 2 \times 10^{-2}$	$0.084 \pm 7 \times 10^{-4}$	$0.021 \pm 5 \times 10^{-3}$	$0.026 \pm 2 \times 10^{-3}$
Biolord*	$0.41 \pm 2 \times 10^{-2}$	$0.086 \pm 6 \times 10^{-4}$	$0.027 \pm 1 \times 10^{-3}$	$0.028 \pm 1 \times 10^{-3}$
GEARS	$0.44 \pm 5 \times 10^{-3}$	$0.069 \pm 1 \times 10^{-3}$	$0.051 \pm 1 \times 10^{-2}$	$0.055 \pm 6 \times 10^{-3}$
LA	$\mathbf{0.79 \pm 1 \times 10^{-2}}$	$\mathbf{0.043 \pm 4 \times 10^{-4}}$	$\mathbf{0.005 \pm 2 \times 10^{-3}}$	$\mathbf{0.014 \pm 1 \times 10^{-3}}$
LA (scGPT)	$0.77 \pm 4 \times 10^{-3}$	$\mathbf{0.044 \pm 4 \times 10^{-4}}$	$0.0085 \pm 1 \times 10^{-3}$	$0.013 \pm 2 \times 10^{-3}$
Decoder	$0.73 \pm 2 \times 10^{-2}$	$\mathbf{0.043 \pm 3 \times 10^{-4}}$	$0.017 \pm 6 \times 10^{-3}$	$\mathbf{0.014 \pm 4 \times 10^{-4}}$
Linear	$0.60 \pm 2 \times 10^{-2}$	$0.057 \pm 3 \times 10^{-3}$	$0.035 \pm 4 \times 10^{-3}$	$0.016 \pm 8 \times 10^{-4}$

Ablating the adversarial classifier in CPA* again had little effect on average performance. However, using scGPT embeddings significantly improved CPA*'s performance, while having no significant effect on the `LatentAdditive` model's performance.

5.4 Effect of Data Scaling

In this section we report the results of the data scaling scenario by verifying if models can take advantage of additional training data to better generalize perturbation effects across biological states. We base the analysis on the `McFalineFigueroa23` dataset that includes both chemical and genetic perturbations across three cell lines. Since there were only 5 chemical perturbations, we consider each unique cell line and chemical perturbation a separate biological state, resulting in 15 total states with 525 genetic perturbations. To test whether adding biological states improves performance, we construct nested subsets of the dataset ($small \subset medium \subset full$), all sharing the same validation and test sets. Each of the subsets contains more biological states (details in Appendix D.4).

We find that model performance tends to improve with more training data in both cosine similarity and cosine rank (see Figure 3). A notable exception is our implementation of CPA*, which does not improve on the cosine rank. This model also has the highest variance. The latent additive model seems to have the most favourable balance of performance on both cosine similarity and rank. Surprisingly, the linear model performs best on the rank metric. For further details please refer to Appendix C.2.



(a) Cosine similarity of predicted and observed log fold changes.

(b) Rank of the cosine similarity of predicted and observed log fold changes.

Figure 3: Scaling of cosine similarity (left) and its rank (right) with increasing size of data included in the training process (x-axis) for several perturbation response models. Points represent results on test data for 5 different seeds, the line represent their average.

We further assessed how these models perform on large datasets with complex covariates by applying our benchmarking suite to the Jiang24 dataset, a large, 1.6 million cell dataset with complex covariates. The dataset contained 6 cell lines with 5 unique cytokine treatments, which we modeled as 30 distinct biological states and 219 genetic perturbations. However the set of perturbations applied was different for each cytokine treatment. We used a similar splitting strategy where we held out 70% of the covariates from 9 cell states for validation/testing.

The results are summarized in Table 6 where we see that the decoder model performs best for both the rank and RMSE/cosine metrics. CPA outperforms the Latent Additive model on the RMSE/cosine metrics and slightly underperforms on the rank metrics. The linear model is much worse than CPA and the latent additive model on the RMSE/cosine metrics, performs similarly on the RMSE rank metric, and is close to the decoder model on the cosine rank metric.

5.5 Effect of Data Imbalance

An important and overlooked consideration with perturbation response models is how robust they are to imbalanced data i.e. how evenly data is distributed across covariates. We quantify imbalance using normalized entropy as follows:

$$\text{Imbalance} := 1 - \frac{\sum_{i=1}^{k} \frac{n_i}{n} \log \frac{n_i}{n}}{\log k},$$

where $n_i, i = 1, \ldots, k$ denotes the number of samples in class i and $n = \sum_{i=1}^{n} n_i$ the overall number of observations. The Srivatsan20 data is perfectly balanced (Imbalance $= 0$), with every perturbation being observed in every cell type. However, *in-silico* machine learning perturbation models often aim to learn generalizable features by using data from multiple sources, which will invariably produce imbalanced datasets.

To test how different models' performance is affected by data imbalance we downsample perturbations per cell type from Srivatsan20 to construct three sub-datasets with different levels of imbalance (Appendix D.4). The results are summarized in Figure 4. We observe that when the data is highly balanced, the linear model performs acceptably well, but this does not hold as imbalance increases. Imbalance may therefore be an important criteria for deciding the suitability of a linear model. CPA both with and without scGPT embeddings, is more robust to changes in data balance than the the Linear or latent additive models, however even with fully balanced data, the cosine rank is high indicating collapse. Interestingly, whilst the latent additive model is more markedly affected by data imbalance than other models, using scGPT embeddings seems to confer some buffer against this. This could be a highly useful feature when working with imbalanced perturb-seq data. The extent to which performance is affected by data imbalance highlights the importance of curated datasets and oversampling strategies (Cui et al., 2019).

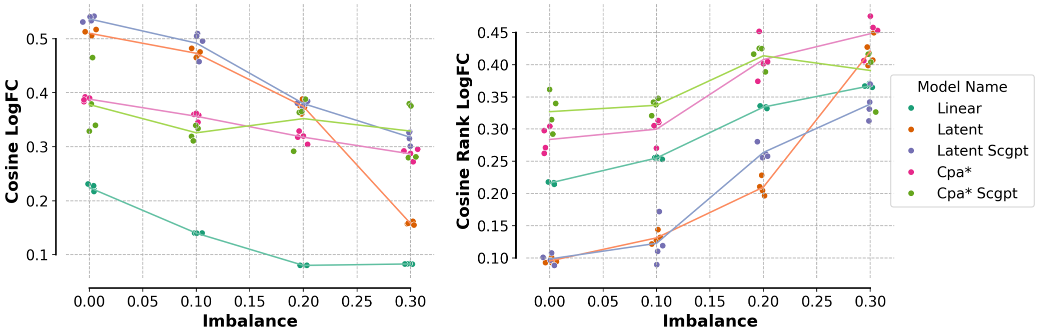


Figure 4: Cosine similarity of log fold changes (left) and its rank (right) of the models as a function of data balance.

6 Discussion

Summary Our study suggests that at least for estimating average treatment effects, *matching* methods perform better than methods that use *disentanglement*. In particular, the simple latent additive model seems to perform best across experiments. This suggests potential challenges, since matching is limited by our understanding of the relevant covariates for a dataset, making it difficult to accurately match controls if there are unknown sources of heterogeneity. For instance, single cell expression data is often confounded by *cell cycle* (referring to different stages in the cell division and proliferation cycle), which is usually not readily available as a covariate (Eberwine et al., 2014).

Further, we demonstrate that simple decoder-only models are able to explain much of the performance of perturbation models and for some datasets, are the best performing model. This is surprising since those models only have access to one-hot encoded perturbation and covariate information. The information present in the transriptomic readout hence has a smaller influence than the architecture and design choices of most models suggests (e.g. CPA and SAMS-VAE are designed around the expression data). Possible reasons include the lack of strong inductive biases (i.e. all neural networks are MLPs) and the noise inherent in either the single-cell RNA readouts or the perturbation labels. Specifically, CRISPR knockdown or knockout can be inefficient, resulting in some cells receiving the correct guide RNA but not having a robust gene knockdown or knockout (Liu et al., 2020). Additionally, high levels of heterogeneity in the control cells not captured by covariate labels could hinder the performance of the *matching* methods, as they randomly sample control cells to match to a given perturbed cell.

There is a wide range of performance across the different tasks and datasets. This could be due to both the intrinsic difficulty of the task, and the amount of measurement or biological noise present in a dataset. For example, the linear model performs well on the `Norman19` combination prediction task (see Table 4), which suggests that this task is relatively easy as most combinations are linearly additive. Most models perform worse on the `McFalineFigueroa23`, `Jiang24`, data scaling and imbalanced data tasks, especially on the rank metrics (see Table 6, Figure 3, and Table 4). For the data scaling task, most models performed better with more data, potentially suggesting perturbation models follow scaling laws (Kaplan et al., 2020). Thus, the `Jiang24`, data scaling and imbalanced data tasks may benefit from better modeling approaches, and we hope that these tasks will attract attention from the community.

Limitations Given the diversity of dataloaders and model frameworks among public models, we aimed to assess the core components of each model. Thus, our benchmarking results should be interpreted as an assessment of how these core components perform rather than a perfectly accurate recreation of the public model implementation (see Appendix D.5 for details).

Since our benchmarking metrics are defined on a population level, they may not fully capture heterogeneity in the perturbation response among cells. Future work in this area could include using distributional metrics such as maximum mean discrepancy (MMD) or Wasserstein distance to better capture response heterogeneity (Gretton et al., 2012; Ramdas et al., 2015).

Benchmarking codebase In our codebase, we provide three main components: datasets and dataloaders, a model development framework, and an evaluation API with metrics (Appendix A). Each component can be used together or individually. For example, a user who wants to benchmark predictions generated by an existing model can use the evaluation API alone. Whereas a different user who wants a develop a new model with our model framework can use the entire codebase. Each component is also extensible, making it easy for users to add datasets, models, and metrics.

Conclusion The perturbation response modeling field holds great promise in complementing experimental approaches to identify novel disease targets and potential therapeutics. In this work we bring together some of the state-of-the-art models in a unified framework with thorough evaluation. We identify three key considerations in regards to benchmarking these models: **(1)** model components (which we tested through ablation studies), **(2)** performance on different datasets (especially at different scales and levels of imbalance), and **(3)** benchmarking metrics that fully characterize the effects of perturbations. We demonstrate the necessity of rank metrics and propose they become the standard in the field. Finally, we anticipate that our modular codebase will prove valuable in future model development and benchmarking efforts, ensuring meaningful discovery of potential targets to treat diseases.

References

Adamson, B., Norman, T. M., Jost, M., Cho, M. Y., Nuñez, J. K., Chen, Y., Villalta, J. E., Gilbert, L. A., Horlbeck, M. A., Hein, M. Y., et al. (2016). A multiplexed single-cell crispr screening platform enables systematic dissection of the unfolded protein response. *Cell*, 167(7):1867–1882.

Ahlmann-Eltze, C., Huber, W., and Anders, S. (2024). Deep learning-based predictions of gene perturbation effects do not yet outperform simple linear methods. *bioRxiv*, page 2024.09.16.613342.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.

Bereket, M. and Karaletsos, T. (2024). Modelling cellular perturbations with the sparse additive mechanism shift variational autoencoder. *Advances in Neural Information Processing Systems*, 36.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24.

Bock, C., Datlinger, P., Chardon, F., Coelho, M. A., Dong, M. B., Lawson, K. A., Lu, T., Maroc, L., Norman, T. M., Song, B., Stanley, G., Chen, S., Garnett, M., Li, W., Moffat, J., Qi, L. S., Shapiro, R. S., Shendure, J., Weissman, J. S., and Zhuang, X. (2022). High-content CRISPR screening. *Nat. Rev. Methods Primers*, 2(8):1–23.

Bunne, C., Stark, S. G., Gut, G., del Castillo, J. S., Levesque, M., Lehmann, K.-V., Pelkmans, L., Krause, A., and Rätsch, G. (2023). Learning single-cell perturbation responses using neural optimal transport. *Nat. Methods*, 20(11):1759–1768.

Burkhardt, D., Benz, A., Cannoodt, R., Cortes, M., Gigante, S., Lance, C., Lieberman, R., Luecken, M., and Pisco, A. (2023). Single-cell perturbation prediction: generalizing experimental interventions to unseen contexts. In *Single-cell perturbation prediction competition*.

Cui, H., Wang, C., Maan, H., Pang, K., Luo, F., Duan, N., and Wang, B. (2024). scGPT: toward building a foundation model for single-cell multi-omics using generative AI. *Nat. Methods*, pages 1–11.

Cui, Y., Jia, M., Lin, T.-Y., Song, Y., and Belongie, S. (2019). Class-balanced loss based on effective number of samples. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9268–9277.

Dixit, A., Parnas, O., Li, B., Chen, J., Fulco, C. P., Jerby-Arnon, L., Marjanovic, N. D., Dionne, D., Burks, T., Raychowdhury, R., et al. (2016). Perturb-seq: dissecting molecular circuits with scalable single-cell rna profiling of pooled genetic screens. *cell*, 167(7):1853–1866.

Eberwine, J., Sul, J.-Y., Bartfai, T., and Kim, J. (2014). The promise of single-cell sequencing. *Nat. Methods*, 11(1):25–27.

Edwards, L. (2024). *Reflections on ICLR24 (or 'Why is AI for drug discovery so difficult to get right?')*. Medium, Münster, Germany.

Falcon, W. and The PyTorch Lightning team (2019). PyTorch Lightning.

Frangieh, C. J., Melms, J. C., Thakore, P. I., Geiger-Schuller, K. R., Ho, P., Luoma, A. M., Cleary, B., Jerby-Arnon, L., Malu, S., Cuoco, M. S., Zhao, M., Ager, C. R., Rogava, M., Hovey, L., Rotem, A., Bernatchez, C., Wucherpfennig, K. W., Johnson, B. E., Rozenblatt-Rosen, O., Schadendorf, D., Regev, A., and Izar, B. (2021). Multimodal pooled Perturb-CITE-seq screens in patient models define mechanisms of cancer immune evasion. *Nat. Genet.*, 53(3):332–341.

Gavriilidis, G. I., Vasileiou, V., Orfanou, A., Ishaque, N., and Psomopoulos, F. (2024). A mini-review on perturbation modelling across single-cell omic modalities. *Comput. Struct. Biotechnol. J.*, 23:1886.

Gayoso, A., Lopez, R., Xing, G., Boyeau, P., Valiollah Pour Amiri, V., Hong, J., Wu, K., Jayasuriya, M., Mehlman, E., Langevin, M., Liu, Y., Samaran, J., Misrachi, G., Nazaret, A., Clivio, O., Xu, C., Ashuach, T., Gabitto, M., Lotfollahi, M., Svensson, V., da Veiga Beltrame, E., Kleshchevnikov, V., Talavera-López, C., Pachter, L., Theis, F. J., Streets, A., Jordan, M. I., Regier, J., and Yosef, N. (2022). A Python library for probabilistic analysis of single-cell omics data. *Nat. Biotechnol.*, 40(2):163–166.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. (2012). A Kernel Two-Sample Test. *Journal of Machine Learning Research*, 13(25):723–773.

Hao, M., Gong, J., Zeng, X., Liu, C., Guo, Y., Cheng, X., Wang, T., Ma, J., Song, L., and Zhang, X. (2023). Large Scale Foundation Model on Single-cell Transcriptomics. *bioRxiv*, page 2023.05.29.542705.

Jaeger, S., Fulle, S., and Turk, S. (2018). Mol2vec: Unsupervised Machine Learning Approach with Chemical Intuition. *J. Chem. Inf. Model.*, 58(1):27–35.

Ji, Y., Green, T., Peidli, S., Bahrami, M., Liu, M., Zappia, L., Hrovatin, K., Sander, C., and Theis, F. (2023). Optimal distance metrics for single-cell rna-seq populations. *bioRxiv*, pages 2023–12.

Jiang, L., Dalgarno, C., Papalexi, E., Mascio, I., Wessels, H.-H., Yun, H., Iremadze, N., Lithwick-Yanai, G., Lipson, D., and Satija, R. (2024a). Systematic reconstruction of molecular pathway signatures using scalable single-cell perturbation screens. *bioRxiv*, page 2024.01.29.576933.

Jiang, Q., Chen, S., Chen, X., and Jiang, R. (2024b). scPRAM accurately predicts single-cell gene expression perturbation response based on attention mechanism. *Bioinformatics*, page btae265.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Lachapelle, S., Rodriguez, P., Sharma, Y., Everett, K. E., Le Priol, R., Lacoste, A., and Lacoste-Julien, S. (2022). Disentanglement via mechanism sparsity regularization: A new principle for nonlinear ica. In *Conference on Causal Learning and Reasoning*, pages 428–484. PMLR.

Liu, G., Zhang, Y., and Zhang, T. (2020). Computational approaches for effective CRISPR guide RNA design and evaluation. *Comput. Struct. Biotechnol. J.*, 18:35–44.

Lopez, R., Tagasovska, N., Ra, S., Cho, K., Pritchard, J. K., and Regev, A. (2022). Learning Causal Representations of Single Cells via Sparse Mechanism Shift Modeling. *arXiv*.

Lotfollahi, M., Susmelj, A. K., De Donno, C., Hetzel, L., Ji, Y., Ibarra, I. L., Srivatsan, S. R., Naghipourfar, M., Daza, R. M., Martin, B., Shendure, J., McFaline-Figueroa, J. L., Boyeau, P., Wolf, F. A., Yakubova, N., Günnemann, S., Trapnell, C., Lopez-Paz, D., and Theis, F. J. (2023). Predicting cellular responses to complex perturbations in high-throughput screens. *Mol. Syst. Biol.*

Lowe, R., Shirley, N., Bleackley, M., Dolan, S., and Shafee, T. (2017). Transcriptomics technologies. *PLoS computational biology*, 13(5):e1005457.

Macosko, E. Z., Basu, A., Satija, R., Nemesh, J., Shekhar, K., Goldman, M., Tirosh, I., Bialas, A. R., Kamitaki, N., Martersteck, E. M., Trombetta, J. J., Weitz, D. A., Sanes, J. R., Shalek, A. K., Regev, A., and McCarroll, S. A. (2015). Highly Parallel Genome-wide Expression Profiling of Individual Cells Using Nanoliter Droplets. *Cell*, 161(5):1202–1214.

McFaline-Figueroa, J. L., Srivatsan, S., Hill, A. J., Gasperini, M., Jackson, D. L., Saunders, L., Domcke, S., Regalado, S. G., Lazarchuck, P., Alvarez, S., et al. (2024). Multiplex single-cell chemical genomics reveals the kinase dependence of the response to targeted therapy. *Cell Genomics*, 4(2).

Norman, T. M., Horlbeck, M. A., Replogle, J. M., Ge, A. Y., Xu, A., Jost, M., Gilbert, L. A., and Weissman, J. S. (2019). Exploring genetic interaction manifolds constructed from rich single-cell phenotypes. *Science*, 365(6455):786–793.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E. Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, abs/1912.01703.

Peidli, S., Green, T. D., Shen, C., Gross, T., Min, J., Garda, S., Yuan, B., Schumacher, L. J., Taylor-King, J. P., Marks, D. S., Luna, A., Blüthgen, N., and Sander, C. (2024). scPerturb: harmonized single-cell perturbation data. *Nat. Methods*, 21(3):531–540.

Piran, Z., Cohen, N., Hoshen, Y., and Nitzan, M. (2024). Disentanglement of single-cell data with biolord. *Nat. Biotechnol.*, pages 1–6.

Radev, D. R., Qi, H., Wu, H., and Fan, W. (2002). Evaluating web-based question answering systems. In González Rodríguez, M. and Suarez Araujo, C. P., editors, *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'02)*, Las Palmas, Canary Islands - Spain. European Language Resources Association (ELRA).

Ramdas, A., Garcia, N., and Cuturi, M. (2015). On Wasserstein Two Sample Testing and Related Families of Nonparametric Tests. *arXiv*.

Replogle, J. M., Saunders, R. A., Pogson, A. N., Hussmann, J. A., Lenail, A., Guna, A., Mascibroda, L., Wagner, E. J., Adelman, K., Lithwick-Yanai, G., Iremadze, N., Oberstrass, F., Lipson, D., Bonnar, J. L., Jost, M., Norman, T. M., and Weissman, J. S. (2022). Mapping information-rich genotype-phenotype landscapes with genome-scale Perturb-seq. *Cell*, 185(14):2559–2575.e28.

Reymond, J.-L. (2015). The Chemical Space Project. *Acc. Chem. Res.*, 48(3):722–730.

Rives, A., Meier, J., Sercu, T., Goyal, S., Lin, Z., Liu, J., Guo, D., Ott, M., Zitnick, C. L., Ma, J., and Fergus, R. (2021). Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proc. Natl. Acad. Sci. U.S.A.*, 118(15):e2016239118.

Roohani, Y., Huang, K., and Leskovec, J. (2023). Predicting transcriptional outcomes of novel multigene perturbations with GEARS. *Nat. Biotechnol.*, pages 1–9.

Ross, J., Belgodere, B., Chenthamarakshan, V., Padhi, I., Mroueh, Y., and Das, P. (2022). Large-scale chemical language representations capture molecular structure and properties. *Nat. Mach. Intell.*, 4(12):1256–1264.

Shalem, O., Sanjana, N. E., Hartenian, E., Shi, X., Scott, D. A., Mikkelsen, T. S., Heckl, D., Ebert, B. L., Root, D. E., Doench, J. G., and Zhang, F. (2014). Genome-Scale CRISPR-Cas9 Knockout Screening in Human Cells. *Science*, 343(6166):84–87.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Srivatsan, S. R., McFaline-Figueroa, J. L., Ramani, V., Saunders, L., Cao, J., Packer, J., Pliner, H. A., Jackson, D. L., Daza, R. M., Christiansen, L., Zhang, F., Steemers, F., Shendure, J., and Trapnell, C. (2020). Massively multiplex chemical transcriptomics at single-cell resolution. *Science*, 367(6473):45–51.

Stuart, E. A. (2010). Matching methods for causal inference: A review and a look forward. *Statistical science: a review journal of the Institute of Mathematical Statistics*, 25(1):1.

Tang, F., Barbacioru, C., Wang, Y., Nordman, E., Lee, C., Xu, N., Wang, X., Bodeau, J., Tuch, B. B., Siddiqui, A., et al. (2009). mrna-seq whole-transcriptome analysis of a single cell. *Nature methods*, 6(5):377–382.

Van de Sande, B., Lee, J. S., Mutasa-Gottgens, E., Naughton, B., Bacon, W., Manning, J., Wang, Y., Pollard, J., Mendez, M., Hill, J., Kumar, N., Cao, X., Chen, X., Khaladkar, M., Wen, J., Leach, A., and Ferran, E. (2023). Applications of single-cell RNA sequencing in drug discovery and development. *Nat. Rev. Drug Discovery*, 22(6):496–520.

Wenteler, A., Occhetta, M., Branson, N., Huebner, M., Curean, V., Dee, W. T., Connell, W. T., Hawkins-Hooker, A., Chung, S. P., Ektefaie, Y., Gallagher-Syed, A., and Córdova, C. M. V. (2024). PertEval-scFM: Benchmarking Single-Cell Foundation Models for Perturbation Effect Prediction. *bioRxiv*, page 2024.10.02.616248.

Wolf, F. A., Angerer, P., and Theis, F. J. (2018). Scanpy: large-scale single-cell gene expression data analysis. *Genome biology*, 19:1–5.

Yadan, O. (2019). Hydra - a framework for elegantly configuring complex applications. Github.

Yu, H. and Welch, J. D. (2022). PerturbNet predicts single-cell responses to unseen chemical and genetic perturbations. *bioRxiv*, page 2022.07.20.500854.

Appendix

A Software Framework: The PerturBench Library

The PerturBench library is designed to encapsulate the essential elements of perturbation modeling, prioritizing reusability and flexibility for researchers. It integrates seamlessly with leading Python-based machine learning frameworks including PyTorch, PyTorch Lightning, and Hydra, as well as cutting-edge single-cell analysis libraries like Scanpy and AnnData. Our design choices enhance the ease of training innovative model architectures and the assessment of both existing and novel techniques across a comprehensive range of benchmarks and datasets. The library is structured into three core modules: `data`, `model`, and `analysis`. These modules are engineered to work together to facilitate a variety of application scenarios, from complete model development to modular use for streamlined integration with other tools and analytical assessments. Subsequent sections will detail the primary abstractions each module offers, illustrating their practicality and adaptability for diverse research tasks.

A.1 Foundational Frameworks

PerturBench leverages contemporary machine learning and single-cell analysis libraries that are prevalent within their respective research communities. This strategic choice is intended to lower the adoption barrier for the proposed benchmark. Additionally, these libraries offer comprehensive guidelines on usage patterns and best practices, which serve to inform the organizational structure of the code.

Pytorch: is one the most widely spread neural network libraries (Paszke et al., 2019). Its core functionality is to build computational graphs with support for efficient auto differentiation. Using this autodiff engine, Pytorch then provides abstractions to build and optimize various neural networks and ML algorithms. In addition, it provides utilities to load and serve data under different training regimes. These concepts are captured within the `torch.utils.data.Dataset` and `torch.utils.data.DataLoader` abstractions. We use these to implement our `perturbench.data` module.

Pytorch Lightning: While Pytorch provides most of the functionality to train any neural network model, it could still be a challenging task to write training and evaluation code that can be portable across different platforms, have minimal bioler plate code, and be easy to read and understand. Pytorch Lighting is a library that builds on top Pytorch Lighting to provide (Falcon and The PyTorch Lightning team, 2019): 1) hardware agnostic model implementations, 2) clear easy to read codebase with minimal boiler plate code, 3) reproducible experiments, and 4) integration with popular machine learning tools. We wrap our models and data into Lightning's `LightningModule` and `LightningDataModule` to abstract away most of the code for managing model training and serving data. Then we leverage Lightning's `Trainer` that abstracts the various traning loops to write generic train and evaluation scripts. Furthermore, Lightning's `Callback` to integrate various logging libraries such as TensorBoard.

Hydra: A complex benchmark suite needs to configure its large number of components and to provide a simple summary for reproducing any experiment. In `pertubench`, we utilize Hydra (Yadan, 2019) for managing configuration. Hydra provides a hierarchical configuration sytems that can be composed based on the components of the system being configured. In addition, it provides conveniet tools such as a command line interface (cli) with auto-completion, support for hpo via optuna, and basic type checking.

AnnData We use AnnData as our primary format for storing and interacting with single cell RNA-seq datasets (Wolf et al., 2018). Each AnnData object contains a single cell gene expression matrix with associated cell level metadata such as perturbation and covariates, as well as gene level metadata such as gene name and ID. Our data module expects single cell datasets stored as AnnData h5ad files and our analysis module expects model predictions in the form of AnnData objects.

A.2 Data Abstractions

The library is built around the `Example` abstraction given in Listing 1 that represents a single datum and its batched version. This data structure contains the necessary fields required for the training and evaluation of perturbation prediction models and serves to unify the model/data API. Each example has two required fields: a `gene_expression` 1D tensor that contains the gene expression levels, and the `perturbations` that has the list perturbation names that has been applied to this cell. In addition, the example contains some optional fields that support more complex functionality like using pre-computed embeddings in the `extra` field, or control matching via `controls`. An ordered list of gene names can be provided in `gene_names` such that it is ordered according to the provided gene counts in `gene_expression`.

```python
class Example(NamedTuple):
    """Single Cell Expression Example."""

    gene_expression: Tensor
    perturbations: list[str]
    covariates: dict[str, str] | None = None
    controls: Tensor | None = None
    gene_names: list[str]
    extra: dict[str, Any]
```

Listing 1: Data structure representing a single example.

For training, we provide two types of pytorch datasets Listing 3. The `SingleCellPerturbation` class represents a single cell RNA-seq dataset and the `SingleCellPerturbationWithControls` class adds control matching functionality, sampling a matched control cell for every perturbed cell.

```python
class SingleCellPerturbation(Dataset):
    """Single Cell Perturbation Dataset."""

    gene_expression: Tensor
    perturbations: list[list[str]]
    covariates: dict[str, list[str]] | None = None
    cell_ids: list[str] | None = None
    gene_names: list[str] | None = None
    transforms: Callable | None = None
    extra: dict[str, Any]

    # factory method
    @staticmethod
    def from_anndata(
        adata: ad.AnnData,
        perturbation_key: str,
        perturbation_combination_delimiter: str | None,
        covariate_keys: list[str] | None = None,
        perturbation_control_value: str | None = None,
        embedding_key: str | None = None,
    ) -> tuple[SingleCellPerturbation, dict[str, Any]]:
    ...
```

Listing 2: Pytorch dataset classes for training.

```python
class SingleCellPerturbationWithControls(SingleCellPerturbation):
    """Single Cell Perturbation Dataset with matched controls."""

    control_ids: Sequence[str] | None = None
    control_indexes: Map(CovariateDict, list[int])
    control_expression: Tensor

    # factory method
    @staticmethod
    def from_anndata(
        adata: ad.AnnData,
        perturbation_key: str,
        perturbation_combination_delimiter: str | None,
        covariate_keys: list[str] | None = None,
        perturbation_control_value: str | None = None,
        embedding_key: str | None = None,
    ) -> tuple[SingleCellPerturbation, dict[str, Any]]:
    ...
```

Listing 3: Pytorch dataset classes for training.

For inference, we provide an additional two types of pytorch datasets. The `Counterfactual` dataset represents a desired set of counterfactual predictions. Since these counterfactual predictions are applied to unperturbed control cells, we only need to store control cell expression values. A single item of this dataset is a counterfactual perturbation applied to set of control cells with will return a `Batch` of data with the control cell expression, control covariates, and desired perturbation.

To evaluate counterfactual predictions, we also provide the `CounterfactualWithReference` class which inherits from the `Counterfactual` class. In additional to providing a `Batch` of control cells with covariates and the desired perturbation, this class also provides an AnnData object with the gene expression values for the perturbed cells corresponding to the covariates and desired perturbations. This enables us to use our suite of benchmarking metrics to compare the model predictions with the observed data.

```python
class Counterfactual(Dataset):
    """Counterfactual Dataset."""
    # Desired counterfactual perturbations
    perturbations: Sequence[list[str]]
    covariates: dict[str, Sequence[str]]
    control_expression: SparseMatrix
    control_indexes: FrozenDictKeyMap
    gene_names: Sequence[str] | None = None
    transforms: InitVar[Callable | Sequence[Callable] | None] = field(default=None)
    info: dict[str, Any] | None = None
    control_embeddings: np.ndarray | None = None

class CounterfactualWithReference(Counterfactual):
    """Counterfactual Dataset with matched Reference Data."""
    # A map from a unique perturbation and set of covariates to indexes
    # in the reference_adata (i.e. all indexes that contain k562 cells
    # with AGR2 knocked down)
    reference_indexes: dict[str, FrozenDictKeyMap] | None = None
    # An AnnData object containing the observed perturbational dataset
    # matching the desired counterfactual predictions
    reference_adata: ad.AnnData | None = None
```

Listing 4: Pytorch dataset classes for inference.

A.3 Data Splitting

We implement a datasplitter class that can generate three types of datasplits:

1. Cross covariate splits that ask a model to predict a perturbation's effect in covariate(s) that were not in the training split. The model will have seen other perturbation in the covariate(s).
2. Combinatorial splits that ask a model to predict the effect of multiple perturbations. The model will have seen the individual perturbations and some other combinations.
3. Inverse combinatorial splits that ask a model to predict the effect of a single perturbation when it has seen a dual perturbation and the other single perturbation.

We design a data splitter with two parameters that allow us to curate the splits: (1) The maximum number, m, of cell types (covariates) to hold out. We randomly hold out between one and m cell types (sampled uniformly). The more cell types held out, the more challenging the task becomes due to fewer training cell types. (2) The total fraction of perturbations held out per cell type, f. A larger fraction makes it more difficult for the model to generate accurate predictions. The datasplitter can also read in custom splits from disk as a csv file.

A.4 Model Abstraction: Model Base Class

We implement a base model class, `PerturbationModel`, that abstracts common model components 5. Specifically,

- A default optimizer
- A training record that contains the data transforms and other key metadata needed for training and inference
- Methods for generating and evaluating counterfactual predictions

```
class PerturbationModel(L.LightningModule, ABC):
    """A base model class for perturbation prediction models."""
    training_record: dict = {
        'transform': None,
        'train_context': None,
        'n_total_covs': None,
    }
    evaluation_config: DictConfig | None = None
    summary_metrics: pd.DataFrame | None = None
    prediction_output_path: str | None = None

    def configure_optimizers(self):
        """Base optimizer for lightning Trainer."""

    def predict_step(
        self,
        data_tuple: tuple[Batch, pd.DataFrame],
        batch_idx: int,
    ) -> ad.AnnData | None:
        """Given a batch of data, predict the counterfactual perturbed """

    def test_step(
        self,
        data_tuple: tuple[Batch, pd.DataFrame, ad.AnnData],
        batch_idx: int,
    ):
        """Run evaluation on a Batch of counterfactual predictions and
        matched observed predictions."""

    def on_test_end(self) -> None:
        """Run rank evaluations (if specified) and summarize benchmarking
        metrics."""

    @abstractmethod
    def predict(self, counterfactual_batch: Batch) -> torch.Tensor:
        """Given a counterfactual_batch of data,
        predict the counterfactual perturbed expression.
        """
```

Listing 5: Pytorch dataset classes for inference.

A.5 Evaluation

All models that inherit from the base `PerturbationModel` class will be able to run evaluation using the Pytorch Lightning trainer test step. These evaluations can be configured via Hydra if using our `train.py` script and evaluations can be run automatically after training completes. For users who only want to use our evaluation metrics, we offer a kaggle style evaluation API that takes as input model predictions as an AnnData object 6. We also offer an adaptor for models built using scvi-tools Gayoso et al. (2022).

```
from perturbench.analysis.benchmarks.evaluator import Evaluator

# List available tasks
print(Evaluator.list_tasks())

# Select an evaluation task
evaluator = Evaluator(
    task = "sciplex3-transfer",
)
# The input format of the Evaluator class is a
# dictionary of model predictions stored as AnnData objects
input_dict = {"CPA_pred": cpa_pred} # cpa_pred is an AnnData object
result_df = evaluator.evaluate(input_dict)
print(result_df) # Summary dataframe with evaluation metrics
```

Listing 6: Evaluation API usage example.

B Additional Modeling Background

Perturbation response modeling aims to predict out-of-sample transcriptomic effects of perturbations in new cellular contexts or combinations, by applying counterfactuals to unperturbed cells. A key challenge is the destructive nature of transcriptomic technologies, which preclude measuring gene expression before and after perturbation in the same cell. To overcome this, published models employ different training strategies. We identify two prominent strategies: *matching methods* to relate control and perturbed cells, or *disentanglement* strategies within autoencoder architectures to separate the effects of perturbations from the baseline cell state.

Matched Controls The matching of perturbed cells with controls is the ostensibly simpler approach and has been used by a variety of published models such as GEARS (Roohani et al., 2023), scGPT (Cui et al., 2024), and scFoundation (Hao et al., 2023).

The absence of a direct correspondence between unperturbed and perturbed cells necessitates a sampling or matching strategy to align control cells with their perturbed counterparts. This task is non-trivial, as the effectiveness and validity of matched controls depend on fulfilling certain assumptions to ensure measuring a *causal* effect. One such assumption is ignorability, which posits that once covariates are controlled for, no residual confounding effects should influence the comparison between control and treatment groups (see e.g. Stuart, 2010).

This can include, for instance, ensuring that the control cell is from the same cell type, experiment or batch. More complex methods involve using optimal transport to identify the control cell most likely to transition into a given perturbed cell (Jiang et al., 2024b) or even to use optimal transport to assist in perturbation prediction (Bunne et al., 2023).

Disentanglement An alternative approach to explicitly matching controls with perturbed cells involves *disentanglement* (Bengio et al., 2013), which enables models to learn separate representations for distinct, meaningful concepts. In the context of perturbation models, the key disentanglement task is to separate the unperturbed cellular state and the perturbation effect.

The CPA (Lotfollahi et al., 2023) uses an adversarial classifier to ensure that the unperturbed 'basal' state is free of any perturbational information. The perturbed state is generated from the basal state by adding and encoding of the perturbation information. The adversarial classifier ensures that during training, the perturbation encoder is forced to learn a meaningful representation of the perturbation. These learned representations can then be added to control cell encodings during inference to generate counterfactual predictions. Biolord (Piran et al., 2024) partitions the latent space into subspaces and optimizes those latent spaces to represent concepts covariates and perturbations. The perturbation and covariate subspaces can then be recombined during inference to generate counterfactual predictions.

sVAE (Lopez et al., 2022) leverages recent results by Lachapelle et al. (2022) demonstrating that *sparsity* can induce disentanglement. Hence, the latent representation that is common to many models introduced here is augmented with a binary mask which in turn is regularised towards a very low activation rate (in the order of 1%). Bereket and Karaletsos (2024) build on sVAE using an additive conditioning for the perturbations and demonstrate that their model, SAMS-VAE, offers biological interpretability of the sparse mechanism framework in the context of perturbation modelling. This sparsity may also improve the additive conditioning mechanism in the context of predicting combinatorial effects, as it enables the decoder to recognize when multiple perturbations have been added together.

B.1 Perturbation embeddings

Drug Embeddings It can be beneficial to use pre-trained embeddings to enable or enhance predictive performance of perturbation models, for instance, ESM embeddings for gene expression (Rives et al., 2021). The performance of these models in predicting unseen perturbations is dependent on the quality of the perturbation representation, which is itself a complex task (Jaeger et al., 2018; Ross et al., 2022; Rives et al., 2021) and outside the scope of this study. GEARS uses gene co-expression to build a gene to gene graph (Roohani et al., 2023), PerturbNet uses a perturbation encoder network to encode perturbations into a lower dimensional embedding (Yu and Welch, 2022). For drug perturbations, PerturbNet uses a structure encoder and for genetic perturbations, it models the gene as a multi-hot vector over all gene ontology annotations. The authors of CPA include a variation to their original model that embeds drugs into a lower dimensional space using molecular features (Lotfollahi et al., 2023). scFoundation leverages GEARS but instead of constructing the graph using static gene coexpression, it uses the gene embeddings for a given cell to create a gene-gene graph (Hao et al., 2023). The performance of these models in predicting unseen perturbations is dependent on the quality of the perturbation representation, which is itself a complex task (Jaeger et al., 2018; Ross et al., 2022; Rives et al., 2021) and outside the scope of this study.

C Further Results

C.1 Collapse and Rank Metrics

The phenomenon of *mode or posterior collapse* represents a significant failure mode in generative models, notably in Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs). The problem arises when a generative model inadequately captures the diversity of its input data, resulting in the production of a restricted array of outputs. Such limitations are particularly detrimental in areas such as single-cell perturbation prediction, where the accurate representation of a wide range of cell types and states is essential. However, as we will demonstrate, relatively good performance as measured by common metrics, such as RMSE, can already be obtained by simply predicting the average expression level for a particular cell type.

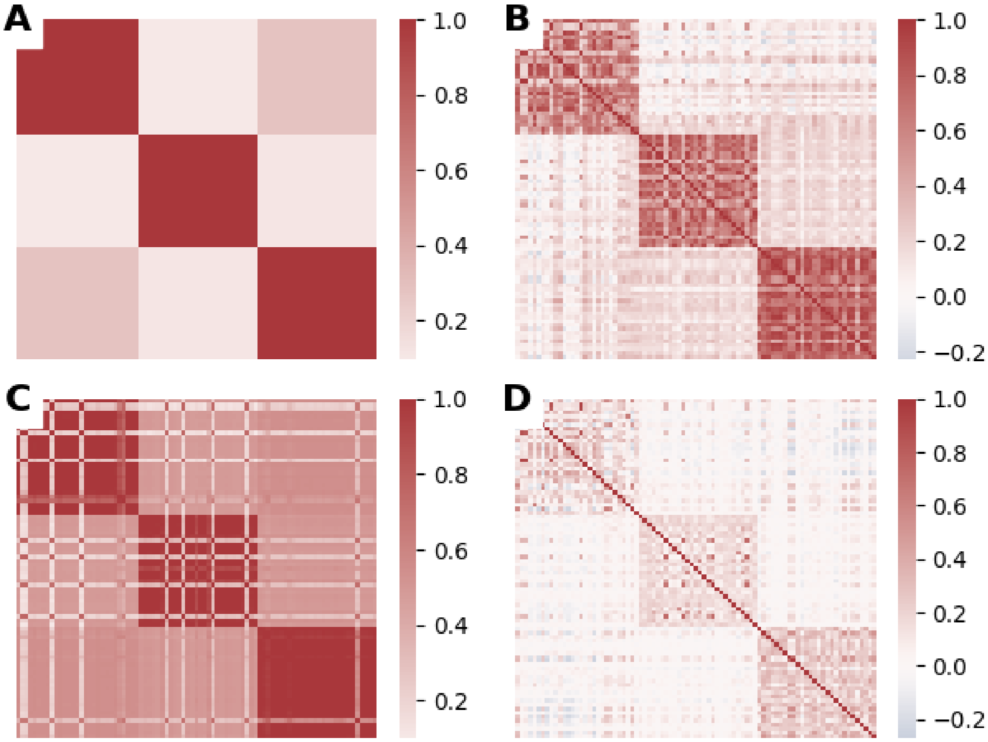


Figure C.1: Cosine similarity matrix based on log-fold changes induced by perturbations **A**) Hyperparameter optimised `DecoderOnly` model with only cell type as covariates. **B**) Hyperparameter optimised `DecoderOnly` model with covariates and perturbations as input. **C**) Hyperparameter optimised version of our CPA fork. **D**) true log-fold changes in the underlying dataset. Clusters correspond to cell types A549, K562, MCF-7.

To further demonstrate this problem and the need for the *rank metrics* introduced in this paper, we will consider a simple example based on the Srivatsan20 dataset. This dataset contains the outcome of small molecules applied to three cancer cell lines: A549, K562, MCF-7. To simulate the impact mode collapse, we will investigate three models: 1) our `DecoderOnly` taking as input *only* the cell type (as a one-hot encoded vector), 2) our `DecoderOnly` taking as input covariates *and* perturbations, and 3) CPA taking as input covariates, perturbation and expression of perturbed cell. All models have been hyperparameter optimized with `optuna`, using 60 iterations. In addition, we will compare the model prediction to the data derived from the experimentally observed expression values. The result is visualized in Figure C.1 using matrix heatmaps and numerical results are presented in Table 5. Higher values (darker color) in the heatmaps correspond to higher similarity between predicted log-fold changes of different perturbation and cell type combinations on the validation data. Plot **A**) shows the `DecoderOnly` model based on the cell type alone. The large squares correspond to the three cell

Table 5: Summary of the model performance of the three perturbation models in the mode collapse example.

metric	rmse_average	rmse_rank_average	matrix_distance
DecoderOnly	0.027	0.534	41.823
DecoderOnly (+ perturbations)	0.020	0.092	25.765
CPA	0.025	0.311	47.512

types and unsurprisingly, the prediction is identical for all perturbation based on the same cell type (as this information is not used by the model.) The true data, **D**), in contrast shows clusters, some of which are related to cell type, but overall different perturbations have different impacts. However, the model visualized in **A**) provides a valuable baseline, because it demonstrates what performance can be achieved without being able to provide any insight into the underlying perturbation modelling task. Notably, the RMSE are of roughly similar order of magnitude for all models, making it challenging to discern which level of performance is adequate, that is, there is no intrinsic quality in the value 0.027 in the `DecoderOnly` model that would suggest that the model is not capabale to predict perturbation effect *at all*.

Turning to Figures **B**) and **C**) we can observe that both models show some correlation by cell type, but the `DecoderOnly` model that ignores the expression values is much better able to predict the nuances of different perturbations than CPA, which looks much more similar to **A**). This indicates that CPA suffers from *mode collapse*.

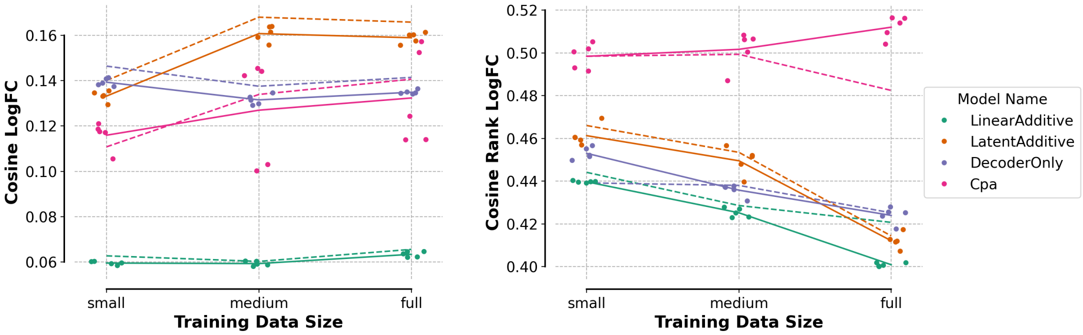
In Table 5, we show three metrics: RMSE, our new rank measure (2), here based on RMSE, and the matrix distance between true similarity matrix and predicted similarity matrix as measured by the Frobenius norm

$$\text{distance}(\hat{S}_{\text{cosine}}, S_{\text{cosine}}) = \|\hat{S}_{\text{cosine}} - S\|_{\text{Frobenius}} = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}(\hat{s}_{ij} - s_{ij})^2},$$

where $\hat{S}$ is the similarity matrix between log-fold changes as predicted by the model and S the corresponding matrix for the observed data. As can be seen in Table 5, out of these three metrics, the rank statistics are best suited to indicate mode collapse. The `DecoderOnly` model based on covariates only has an average rank of 0.534 which is close a rank of 0.5 expected for a model that is unable to distinguish between perturbations. On the one hand, the `DecoderOnly` model has a low average rank, indicating that its “precision” is good. On the other hand, CPA’s average rank of 0.311 shows some mode collapse. Interestingly, the Frobenius norm related to CPA is even worse than the that of the `DecoderOnly` model based on the covariates.

C.2 More scaling results

Most models have similar performance on the validation and test splits. However, CPA notably overfits to the validation split on the full data split and the linear model actually performs better on the test split than the validation split (see Figure C.2).



(a) Cosine similarity of predicted and observed log fold changes.

(b) Rank of the cosine similarity of predicted and observed log fold changes.

Figure C.2: Scaling of cosine similarity (left) and its rank (right) with increasing size of data included in the training process (x-axis) for several perturbation response models. Points represent results on test data for 5 different seeds, the line represent their average. Dotted lines are results on validation, solid lines are results on test.

C.3 Additional results on a large perturbation response dataset

Table 6: Results of the Jiang24 experiment which contains 30 different cell states (6 cell lines & 5 cytokine treatments), with 70% of perturbations from 9 cell states held out for validation/testing. Results are reported as mean ± one standard deviation. Best performance per metric is indicated in bold.

Model	Cosine LogFC	RMSE mean	Cosine LogFC rank	RMSE mean rank
CPA*	$0.58 \pm 4 \times 10^{-2}$	$0.017 \pm 5 \times 10^{-4}$	$0.40 \pm 8 \times 10^{-3}$	$0.42 \pm 9 \times 10^{-3}$
LA	$0.47 \pm 1 \times 10^{-3}$	$\mathbf{0.015 \pm 7 \times 10^{-5}}$	$0.38 \pm 6 \times 10^{-3}$	$0.38 \pm 7 \times 10^{-3}$
Decoder	$\mathbf{0.64 \pm 8 \times 10^{-4}}$	$\mathbf{0.015 \pm 3 \times 10^{-5}}$	$\mathbf{0.32 \pm 8 \times 10^{-3}}$	$\mathbf{0.32 \pm 5 \times 10^{-3}}$
Linear	$0.17 \pm 8 \times 10^{-5}$	$0.038 \pm 5 \times 10^{-5}$	$0.34 \pm 2 \times 10^{-3}$	$0.43 \pm 1 \times 10^{-3}$

D Implementation Details

D.1 Dataset Summary

Dataset 1 (Norman19) *This datasets (Norman et al., 2019) contains 287 gene overexpression perturbations with 131 containing multiple perturbations in K562 cells. We selected this dataset as it is the largest perturb-seq dataset with combinatorial perturbations so far. This dataset was also used in existing perturbation prediction studies including, e.g. CPA (Lotfollahi et al., 2023), scGPT (Cui et al., 2024), SAMS-VAE (Bereket and Karaletsos, 2024) and Biolord (Piran et al., 2024).*

Dataset 2 (Srivatsan20) *This dataset (Srivatsan et al., 2020) includes 188 chemical perturbations across the K562, A549, and MCF-7 cell lines. The chemical perturbations were applied at 4 doses but for the purposes of this study, we subset to highest dose only since most of the models we are benchmarking do not have dose response modeling capacity. We selected this dataset to benchmark prediction of chemical perturbations. Additionally, this dataset was used as a benchmark in multiple perturbation prediction studies including CPA (Lotfollahi et al., 2023) and Biolord (Piran et al., 2024).*

Dataset 3 (Frangieh21) *This dataset (Frangieh et al., 2021) includes 248 genetic perturbations across 3 melanoma cell conditions that simulate interaction with immune cells. The conditions are: 1) melanoma cells cultured alone, 2) melanoma cells with IFNγ, and 3) melanoma cells co-cultured with tumor infiltrating immune cells. We selected this dataset to benchmark whether a perturbation response prediction model could predict perturbation effects in the more complex co-culture condition using training data from the simpler conditions.*

Dataset 4 (Jiang24) *This dataset (Jiang et al., 2024a) includes 219 genetic perturbations across 6 cell lines and 5 cytokine treatments (which can be seen as 30 unique biological states). We selected this dataset due to the large number of biological states and the fact that the perturbations were chosen because they had been reported to modulate cytokine signaling. This dataset has not been previously used to benchmark perturbation prediction models.*

Dataset 5 (McFalineFigueroa23) *This dataset (McFaline-Figueroa et al., 2024) includes 525 genetic perturbations across 3 cell lines and 5 chemical treatments (which can be seen as 15 unique biological states). We selected this dataset due to the large number of perturbations and the fact that it contains multiple covariates (cell lines and chemical treatments). This dataset has not been previously used to benchmark perturbation prediction models.*

D.2 Dataset Curation

We downloaded the gene expression counts matrices for these datasets as is from the original sources and mapped key metadata columns (perturbation, cell line, chemical treatment) to a standardized set of columns.

D.3 Dataset Preprocessing

It is common practice to pre-process perturbation datasets before ingesting them into a machine learning training pipeline for training and prediction. In the following we describe the data processing that this benchmark is based on.

To ensure we are capturing the most biologically relevant features, we subset to highly variable or differentially expressed genes. Specifically, we keep the top 4000 variable genes using the scanpy `pp.highly_variable_genes` method with `flavor='seurat_v3'`. We also keep the top 25 top differentially expressed genes for every perturbation in every unique set of covariates, using scanpy's `tl.rank_genes_groups` method with default parameters. For datasets with genetic perturbations, we also ensure that the perturbed gene is included in the feature set as well.

For the models that require log-normalization, we apply the default `scanpy` (Wolf et al., 2018) preprocessing pipeline. Specifically, we divide the counts by the total counts in each cell, multiply by a scaling factor of 10,000, and apply a log-transform with a pseudocount of 1, i.e.

$$x_{i,\text{normalized}} = \log\left(1 + \frac{x_i}{\sum_j x_j} \cdot 10^4\right).$$

D.4 Data Splitting

McFalineFigueroa23 splits We manually generate the data scaling splits for the McFalineFigueroa23 dataset by first selecting 3 covariates to hold out perturbations in. McFalineFigueroa23 has 3 cell type (a172, t98g, u87mg) and 5 treatments (control, nintedanib, zstk474, lapatinib, trametinib). We specifically selected covariates 3 different cell types and chemical treatments (a172 with nintedanib, t98g with lapatinib, and u87mg with control). Within each of these "heldout covariates", we randomly hold out 70% of perturbations for validation and testing. Some perturbations may be held out across multiple covariates. To build the small version of the dataset, we select 3 additional covariates that match the cell type and chemical treatment of the "heldout covariates" to add to the training split (a172 with control treatment, t98g with nintedanib, u87mg with lapatinib). We built the medium version of the dataset by adding the remaining 3 covariates that match cell type and chemical treatment to the training split. The large version contains the full dataset, 15 covariates.

The attached codebase has a python notebook responsible for generating this split: `notebooks/neurips2024/build_data_scaling_splits.ipynb`.

Jiang24 splits We held out 70% of perturbations in all 12 combinations of the following cytokines: IFNG, INS, TGFB and cell lines: k562, mcf7, ht29, hap1. The remaining perturbations were used for training.

The attached codebase has a python notebook responsible for generating this split: `notebooks/neurips2024/build_jiang24_frangieh21_splits.ipynb`.

Frangieh21 splits We held out 70% of the perturbations in the co-culture condition and trained on the remaining perturbations.

The attached codebase has a python notebook responsible for generating this split: `notebooks/neurips2024/build_jiang24_frangieh21_splits.ipynb`.

Srivatsan20 Data Imbalance splits To generate the imbalanced Srivatsan20 datasets for Figure 4, we set three different desired level of imbalance, which we quantified via normalized entropy computed on the number of perturbations per cell type:

$$\text{Imbalance} := 1 - \frac{\sum_{i=1}^{k} \frac{n_i}{n} \log \frac{n_i}{n}}{\log k},$$

where $n_i, i = 1, \ldots, k$ denotes the number of samples in class i and $n = \sum_{i=1}^{n} n_i$ the overall number of observations. The full Srivatsan20 data is fully balanced with 188 perturbations seen in all three cell types. For the three subsequent imbalanced data sets, we fix the first cell type to always see all 188 perturbations, and then randomly choose the number of seen perturbations for the other two cell types that will result in the desired level of balance (distributions given in Table 7). Control cells are always seen in training for each cell type. We then randomly downsampled each cell type to the desired number of perturbations, and used our datasplitter with default parameters to generate a cross cell type split. We set the minimum number of perturbations to 30 per cell type.

Unseen perturbation splits Some models such as scGPT, GEARS, and PerturbNet create an embedding over the perturbation space which enables prediction of the effect of perturbations that were never seen during training in any context. Since this task is very complex and most likely highly dependent on the quality of the perturbation embedding/representation, we choose not to address it the scope of this study.

Table 7: Number of perturbations in each cell type for downsampled subsets of Srivatsan20 with different levels of data balance.

Balance	# Perts Cell Type 1	# Perts Cell Type 2	# Perts Cell Type 3
1	188	188	188
0.9	188	50	117
0.8	188	81	30
0.7	188	33	33

D.5 Models

CPA We implemented a version of CPA using the published Theis lab model (forked 02/23). However, the Theis lab codebase has been updated since publication which we have not incorporated into our implementation. Thus, we refer to our implementation as CPA*.

To ensure that we correctly implemented CPA, we verified that our implementation achieved similar or better performance on all metrics compared to the published versions. To this end we trained a CPA model using the published Theis lab model (forked 02/23) and our implementation using the exact same hyperparameters identified to be optimal by the authors, on the same data split and assessed the performance. Our implementation of CPA obtained comparable (and indeed, slightly better) results than the original codebase.

SAMS-VAE SAMS-VAE is available under a restrictive licence. For this reason, we implemented a version of the model carefully following the authors' description. Since, the model is re-implementation, we refer to it as SAMS-VAE*.

BioLord For modelling the effect of perturbations, Biolord as presented by the authors requires the use of embeddings. either using the GEARS GO graph for genes or RDKIT embeddings for small molecules. To be able to compare models on an equal footing, we have excluded the use of embeddings and have therefore implemented a slight variation of Biolord, henceforth referred to as Biolord*, where instead of neighbourhoods based on embeddings, we use the same one hot representation for perturbations as for all other models.

GEARS Since the GEARS model differs from the other models in its use of GNNs to encode gene expression values and perturbations and since the authors did not recommend applying it to the covariate transfer task, we chose not to reimplement GEARS in our repo. We instead wrote a helper function and HPO script for training and evaluating GEARS using the publicly available version on the same Norman19 split we used for the other models.

scGPT Embeddings To generate scGPT embeddings, we used the pretrained whole human model and generated embeddings with no fine-tuning on our processed datasets.

Linear The simple linear baseline model uses the *control matching* approach. Given a perturbed cell, x', we sample a random control cell with *matched* covariates, x, and reconstruct x' by applying one linear layer given the perturbation and covariates:

$$x' = f_{\text{linear}}(p_{\text{one\_hot}}, cov_{\text{one\_hot}}), \tag{3}$$

where $p_{\text{one\_hot}}$ denotes the one-hot encoding of the perturbation and $cov_{\text{one\_hot}}$ denotes one-hot encodings of covariates (i.e. cell type).

Latent Additive We extended the linear model into a baseline latent additive model by encoding expression values and perturbations into a latent space $\mathsf{Z} \subseteq \mathbb{R}^{d_z}$, i.e.

$$z_{\text{ctrl}} = f_{\text{ctrl}}(x), \quad \text{and} \quad z_{\text{pert}} = f_{\text{pert}}(p_{\text{one\_hot}}),$$

where $p_{\text{one\_hot}}$ denotes the one-hot encoding of the perturbation. Subsequently, we reconstruct the expression value by decoding the added latent space representation $x' = f_{\text{dec}}(z_{\text{ctrl}} + z_{\text{pert}})$.

Decoder Only As a further ablation study, we introduce a model class that aims to predict the transcriptome solely from covariates, $cov_{\text{one\_hot}}$, perturbation information, $p_{\text{one\_hot}}$, or a mix of both. This model takes as an input neither the transcriptome of a control cell nor the transcriptome of a perturbed cell. Consequently, prediction of the expression of a perturbed cell can be modelled as $x' = f_{\text{dec}}(z)$ for $z \in \{p_{\text{one\_hot}}\} \cup \{cov_{\text{one\_hot}}\} \cup \{(p_{\text{one\_hot}}, cov_{\text{one\_hot}})\}$ and we refer to them as *decoder only* models. This class of models provides a range of baselines:

- Firstly, a model decoding only from covariates provides a lower bound on the performance of acceptable models and a sense of what performance can be expected when a model collapses to its mode(s). For instance, if the covariates contain only the cell type, this model will only learn the average expression value for each cell type. Since no perturbation information is used, the model is completely collapsed for every class of covariates.
- Secondly, a model that decodes only from perturbations offers a baseline that illustrates the extent to which expression levels resulting from perturbations can be predicted, disregarding any information about cell type or expression levels in control cells.
- Thirdly, a model that decodes information from both cell type and perturbations provides a baseline for understanding the additional information that the transcriptome could offer, which is not already captured by the covariates or inherently present in the perturbation data.

D.6 Hyperparameter Optimization

D.6.1 Identifying a Hyperparameter Metric

In order to carry out HPO, we need to define a performance metric that can be taken as an objective function for `optuna`. The model loss calculated on the validation data can in many cases be unsuitable for such a task, as some hyperparameters are part of the loss itself and aim, for instance, to find a balancing factor between different loss terms. In such scenarios, the objective would induce `optuna` to simply set a scaling factor to 0. Hence, we require an alternative metric as an HPO objective function.

To define an objective functions we set out the following requirements:

- To make our models comparable and to avoid confounding issues, we compare all models based on the same metric for the purposes of HPO.
- Considering the results of Section C.1 hyperparameter optimization can not simply be carried out on one metric, such as RMSE, as we have established that this metric alone does not cover all aspects of model performance.

To identify suitable hyperparameter metrics, we carried out several HPO runs with linear combinations of cosine similarity and the respective rank metric, as well as RMSE and its respective rank metric. In a few pilot hpo runs we observed that

$$\mathcal{L}_{\text{HPO}} = \text{RMSE} + 0.1 \cdot \text{rank}_{\text{RMSE}}$$

results in models that perform well on both aspects, traditional model fit as well as ranking metrics.

D.6.2 Hyperparameter Ranges

For hyperparameter optimization we used `optuna` (Akiba et al., 2019). Hence, we can define all hyperparameter ranges as `optuna` distributions, either in the form of `categorical`, `int` or `float`. We describe the seed and the specific `optuna` hyperparameter ranges as well as their distribution classes in Tables 8 to 12 and 14.

Table 8: CPA hyperparamter range.

Hyperparameter	**Distribution**
Number of layers in the encoder part of the model:	
`n_layers_encoder`	`Int: 1 to 7, step=2`
Number of perturbation embedding layers:	
`n_layers_pert_emb`	`Int: 1 to 5, step=1`
Number of layers in the adversarial classifier:	
`adv_classifier_n_layers`	`Int: 1 to 5, step=1`
Hidden dimension size:	
`hidden_dim`	`Int: 256 to 5376, step=1024`
Hidden dimension size of the adversarial classifier:	
`adv_classifier_hidden_dim`	`Int: 128 to 1024, log=True`
Number of adversarial steps:	
`adv_steps`	`Categorical: [2, 3, 5, 7, 10, 20, 30]`
Number of latent variables:	
`n_latent`	`Categorical: [64, 128, 192, 256, 512]`
Learning rate:	
`lr`	`Float: 5e-6 to 1e-3, log=True`
Weight decay:	
`wd`	`Float: 1e-8 to 1e-3, log=True`
Dropout rate:	
`dropout`	`Float: 0.0 to 0.8, step=0.1`
KL divergence weight:	
`kl_weight`	`Float: 0.1 to 20, log=True`
Adversarial weight:	
`adv_weight`	`Float: 0.1 to 20, log=True`
Penalty weight:	
`penalty_weight`	`Float: 0.1 to 20, log=True`

Table 9: Latent additive model hyperparameter range.

Hyperparameter	**Distribution**
Number of layers in the encoder part of the model:	
`n_layers`	`Int: 1 to 7, step=2`
Width of the encoder layers in the model:	
`encoder_width`	`Int: 256 to 5376, step=1024`
Dimensionality of the latent space:	
`latent_dim`	`Categorical: [64, 128, 192, 256, 512]`
Learning rate:	
`lr`	`Float: 5e-6 to 5e-3, log=True`
Weight decay:	
`wd`	`Float: 1e-8 to 1e-3, log=True`
Dropout rate:	
`dropout`	`Float: 0.0 to 0.8, step=0.1`

Table 10: Linear additive model hyperparameter range.

Hyperparameter	**Distribution**
Learning rate:	
lr	Float: 5e-6 to 5e-3, log=True
Weight decay:	
wd	Float: 1e-8 to 1e-3, log=True

Table 11: Biolord hyperparameter range.

Hyperparameter	**Distribution**
Weight of the penalty term in the loss function:	
penalty_weight	Float: 1e1 to 1e5, log=True
Number of layers in the encoder part of the model:	
n_layers	Int: 1 to 7, step=2
Width of the encoder layers in the model:	
encoder_width	Int: 256 to 5376, step=1024
Dimensionality of the latent space:	
latent_dim	Categorical: [64, 128, 192, 256, 512]
Learning rate:	
lr	Float: 5e-6 to 5e-3, log=True
Weight decay:	
wd	Float: 1e-8 to 1e-3, log=True
Dropout rate:	
dropout	Float: 0.0 to 0.8, step=0.1

Table 12: SamsVae hyperparameter range.

Hyperparameter	**Distribution**
Number of layers in the encoder part of the model:	
`n_layers_encoder_x`	`Int: 1 to 7, step=2`
Number of layers in the encoder part of the model:	
`n_layers_encoder_e`	`Int: 1 to 7, step=2`
Number of layers in the decoder part of the model:	
`n_layers_decoder`	`Int: 1 to 7, step=2`
Width of the encoder layers in the model:	
`latent_dim`	`Categorical: [64, 128, 192, 256, 512]`
Hidden dimension for x:	
`hidden_dim_x`	`Int: 256 to 5376, step=1024`
Hidden dimension for the conditional input:	
`hidden_dim_cond`	`Int: 50 to 500, step=50`
Whether to use sparse additive mechanism:	
`sparse_additive_mechanism`	`Categorical: [True, False]`
Whether to use mean field encoding:	
`mean_field_encoding`	`Categorical: [True, False]`
Learning rate:	
`lr`	`Float: 5e-6 to 1e-3, log=True`
Weight decay:	
`wd`	`Float: 1e-8 to 1e-3, log=True`
The target probability for the masks:	
`mask_prior_probability`	`Float: 1e-4 to 0.99, log=True`
Dropout rate:	
`dropout`	`Float: 0.0 to 0.8, step=0.1`

Table 13: DecoderOnly hyperparameter range.

Hyperparameter	**Distribution**
Number of layers in encoder/decoder:	
`n_layers`	`Int: 1 to 7, step=2`
Width of the encoder:	
`encoder_width`	`Int: 256 to 5376, step=1024`
Learning rate:	
`lr`	`Float: 5e-6 to 5e-3, log=True`
Weight decay:	
`wd`	`Float: 1e-8 to 1e-3, log=True`
Whether to apply a softplus activation to the output of the decoder to enforce non-negativity:	
`softplus_output`	`Categorical: [True, False]`

Table 14: GEARS hyperparameter range.

Hyperparameter	**Distribution**
Number of layers in perturbation GNN: `num_go_gnn_layers`	`Int:  1 to 3, , step=1`
Number of layers in gene GNN: `num_gene_gnn_layers`	`Int:  1 to 3, step=1`
Number of neighboring perturbations in GO graph: `num_similar_genes_go_graph`	`Int:  10 to 30, step=10`
Number of neighboring genes in gene co-expression graph: `num_similar_genes_co_express_graph`	`Int:  10 to 30, step=10`
Width of the encoder: `hidden_size`	`Int:  32 to 512, step=32`
Minimum coexpression threshold: `co_express_threshold_graph`	`Float:  0.2 to 0.5, step=0.1`
Learning rate: `lr`	`Float:  5e-6 to 5e-3, log=True`
Weight decay: `wd`	`Float:  1e-8 to 1e-3, log=True`

D.7 Compute Resources

For the Norman19 and Srivatsan20, and data imbalance tasks, we used nodes with one Nvidia A10G GPU each. We ran 60 hyperparameter optimization trials for each model, and assessed 10 models on the Srivatsan20 task and 9 models on the Norman19 task. We also ran 4 training runs with the best hyperparameters for stability analysis. We also ran an additional 5 models on the 4 different data imbalance splits, again with 4 runs for stability. For the HPO runs we used 813 hours for Srivatsan20 and 399 hours for Norman19. See details in Table 15.

For the McFalineFigueroa23 data scaling task, we used nodes with Nvidia A10G GPUs for most of the combinations of models and subsets. We used A100G GPUs for all deep learning model for the biggest split, and for all datasets on CPA (which required the most GPU memory). We again used 60 hyperparameter optimization trials across 4 models with an additional 4 runs for stability. In total for this experiments we used 2517 hours of servers with GPUs, see details in Table 15.

Table 15: Total runtime of HPO for different models and datasets

dataset	model	runtime	A100
mcfaline23-full	cpa	171.97	yes
mcfaline23-full	decoder-only	136.91	yes
mcfaline23-full	latent-additive	150.36	yes
mcfaline23-full	linear-additive	321.08	
mcfaline23-medium	cpa	127.44	yes
mcfaline23-medium	decoder-only	225.24	
mcfaline23-medium	latent-additive	280.12	
mcfaline23-medium	linear-additive	359.33	
mcfaline23-small	cpa	105.12	yes
mcfaline23-small	decoder-only	135.38	
mcfaline23-small	latent-additive	186.14	
mcfaline23-small	linear-additive	317.91	
norman19	biolord	129.71	
norman19	cpa	42.98	
norman19	cpa-no-adversary	48.08	
norman19	cpa-scgpt	25.20	
norman19	decoder	20.48	
norman19	latent	32.69	
norman19	latent-scgpt	21.42	
norman19	linear	30.17	
norman19	sams	48.06	
sciplex3	biolord	312.49	
sciplex3	cpa	41.66	
sciplex3	cpa-no-adversary	51.83	
sciplex3	cpa-scgpt	38.54	
sciplex3	decoder	40.41	
sciplex3	decoder-cov	36.42	
sciplex3	latent	56.59	
sciplex3	latent-scgpt	76.25	
sciplex3	linear	70.48	
sciplex3	sams	88.76	

D.8 Benchmarking metrics

A common approach is to report metrics that are associated with the "global" fit or the *accuracy* of the model. These metrics include RMSE and *cosine similarity*

$$\mathrm{S}_{\text{cosine}}(x, y) = \frac{\sum_i^n x_i y_i}{\sqrt{\sum_{i=1}^n x_i^2}\sqrt{\sum_{i=1}^n y_i^2}}$$

between predicted and observed perturbations since the correlation metrics (whether in the flavour of Spearman or Pearson) are invariant with respect to shifts in mean expression values.

We have two different classes of *accuracy* related metrics. One class of metrics evaluates whether predicted and observed aggregates have similar shapes (pearson, cosine). Another class evaluates whether predicted and observed aggregates have similar values (RMSE, MAE, MSE, R2 score).