UNIVERSITÄT
LEIPZIG

Fakultät für Mathematik und Informatik

Meta hyperparameter optimization of autoencoders for multi-omics data integration

Ralf König
Matrikel-Nr. 3796977

Masterarbeit im Studiengang Data Science
zur Erlangung des akademischen Grades:
Master of Science

Universität Leipzig

1. Prüfer: Prof. Dr. Erhard Rahm
2. Prüfer: Dr. Aaron Klein

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere, dass ich alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe, und dass die eingereichte Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen ist.

Leipzig, 9. November 2025 Unterschrift

Abstract

This thesis investigates how to turn results on Autoencoder runs into data, that works as priors in Automated Hyperparameter optimization. Grounded in five concrete datasets from bioinformatics, the Autoencodix training stack, and Syne Tune as the HPO engine, we characterize which hyperparameters and value ranges most strongly affect performance, and how these choices interact. Using thousands of Autoencoder runs on random sampled hyperparameter combinations, we analyze main effects and interdependencies between architectural, optimization, and regularization parameters, linking intermediate training signals to down-stream utility. Building on these insights, we develop meta-learning warm-starts that encode prior evidence as initialization for HPO on the same task, and transfer mechanisms that leverage previous runs to accelerate tuning on related datasets.

To support rapid experimentation, we explore a surrogate-based simulation strategy within Syne Tune that augments limited observations with synthetic yet structure-preserving data. We then benchmark a portfolio of Syne Tune algorithms—including Bayesian and evolutionary methods—under budget constraints. The proposed warm-starts consistently reduce time-to-performance, and transfer methods yield gains when dataset similarity is adequately modeled; conditional search spaces derived from previous runs improve efficiency. Finally, we demonstrate that these methodological advances translate into better representations for multi-modal molecular genetic data, as measured by a set of down-stream classification tasks on clinical variables. The outcome is a set of principled defaults, reusable priors, and practical guidelines for efficient, transferable autoencoder tuning—grounded in specific artifacts yet designed to generalize to comparable representation-learning workflows.

Contents

List of Abbreviations

AE Autoencoder

ATAC Assay for Transposase-Accessible Chromatin

AutoHPO Automatic hyperparameter optimization

BO Bayesian Optimization

BORE Bayesian Optimization with Density-Ratio Estimation

CNN convolutional neural network

CQR Conformal Quantile Regression

DL Deep Learning

DNA Deoxyribonucleic acid

DNN Deep Neural Network

GeLU Gaussian Error Linear Unit function

HP hyperparameter

HPO hyperparameter optimization

KDE - Kernel Density Estimator

KL divergence - Kullback-Leibler divergence

ML Machine Learning

NN Neural Network (in the sense of an artificial neural network)

PCA Principal Component Analysis

REA Regularized Evolutionary Algorithm

ReLU Rectified Linear Unit function

RNA Ribonucleic acid

RS Random Search, Random Sampling

SCHC Single Cell Human Cortex

SGD Stochastic Gradient Descent

TPE Tree-Parzen Estimator

U-Net encoder-decoder architecture with a bottleneck and skip connections

UMAP Uniform Manifold Approximation and Projection for Dimension Reduction

TCGA The Cancer Genome Atlas

VAE Variational Autoencoder

YAML Yet Another Markup Language

1 Introduction

Autoencoders are a natural fit for representation learning in high-dimensional, multi-modal data, but their performance is notoriously sensitive to hyperparameter choices and training protocol. In practice, this leads to expensive trial-and-error cycles that are hard to reproduce and even harder to transfer across datasets and projects. By fixing concrete artifacts—five specific datasets from two multi-omics data scenarios in bioinformatics, the Autoencodix (Joas et al., 2024) training stack, and Syne Tune Salinas et al. (2022) for hyperparameter optimization (HPO)—we can study the space in a controlled yet realistic setting.

At the same time, we aim to go beyond one-off tuning by learning from prior runs so that good configurations become easier to find in the future. This raises methodological questions about which hyperparameters matter most, how they interact, and how prior evidence should be encoded and reused. It also raises questions about logging, comparability, and evaluation metrics that faithfully reflect down-stream utility. The overarching goal is to convert historical experimentation into actionable priors and robust procedures, improving both sample-efficiency and time-to-solution. While our conclusions are grounded in these specific tools and datasets, they can be generalized to other HPO on autoencoder workflows.

1.1 Problem Statement

Given five specific datasets, Autoencodix as a specific framework for training autoencoders, Syne Tune as a specific framework for HPO, explore the ways, to carry over knowledge from previous runs or from runs on other tasks. Consider the most relevant hyperparameters and evaluation metrics. Keep gathered insights with these specific artifacts (datasets, AE training software, HPO software) and specific exemplary results, but also explore generalization.

1.2 Research questions and motivation

This thesis deals with the following central research questions. For each question, a short motivation is given after the question.

RQ1: What are most relevant hyperparameters and sensible value ranges?

> Motivation: Selecting model hyperparameters such as input dimension, bottleneck shape, latent dimensionality and training algorithm hyperparameters like learning rate, batch size and drop probability often dominates final model quality. Without principled ranges, HPO wastes resources exploring implausible regions or missing viable ones. Establishing sensible default ranges and identifying the most influential hyperparameters reduces search space and improves reproducibility. Moreover, informative ranges are prerequisite inputs

for any meta-learning or warm-start strategy. They also help decouple model design from search budget, enabling fairer comparisons across datasets. This knowledge benefits both human practitioners and automated pipelines. Ultimately, a clear ranking of relevance and well-justified value ranges forms the foundation for efficient, transferable optimization.

RQ2: Given many thousand runs of Autoencoder training including down-stream task metrics, what can we learn about hyperparameter influence and interdependencies?

Motivation: Large-scale logs from thousands of autoencoder runs contain latent structure about what works, when, and why. Analyzing these data can reveal main effects (e.g., learning rate) and higher-order interactions (e.g., latent size $\times$ regularization) that are invisible in small studies. Understanding interdependencies helps avoid brittle configurations that perform well only under narrow conditions. It also enables conditional search spaces, where permissible values adapt to other choices, improving HPO efficiency. Correlating training metrics with down-stream task outcomes clarifies which early signals are actually predictive of utility. This analysis can further expose dataset-specific versus dataset-agnostic patterns, guiding generalization. The resulting empirical priors are the raw material for warm-starts and meta-models.

RQ3: How can a meta-learning approach be used to warm-start hyperparameter optimization for autoencoders for the same task on a different HP combination?

Motivation: Warm-starting HPO with meta-learning promises large savings by steering search toward promising regions from the first iteration. For repeated tasks or re-tuning the same model family, learned priors can replace naive random or grid initializations. The challenge lies in representing prior runs so they generalize across nearby configurations without overfitting to past idiosyncrasies. Designing such priors requires decisions about featureization (hyperparameters, dataset descriptors, interim metrics) and how to inject them into Syne Tune algorithms. Success here directly translates into faster convergence and more consistent outcomes in practice. This question is therefore central to operationalizing "learning to tune" within Autoencodix. It also sets up later transfer to new tasks and datasets.

RQ4: How can previous hyperparameter optimization runs be leveraged to improve performance on new tasks, like similar or other datasets?

Motivation: Across datasets, the value proposition shifts from reusing settings to transferring knowledge about tuning dynamics. If prior HPO runs on similar datasets can inform search on a new dataset, we reduce cold-start penalties. The key difficulty is measuring similarity and deciding how strongly to trust transferred information. Robust transfer mechanisms must provide benefits when datasets align while avoiding harm when they do not (negative transfer). This requires explicit dataset meta-features and uncertainty-aware

priors or multi-task surrogates. Demonstrating reliable improvements on new tasks would make autoencoder tuning more scalable in data-rich domains. Such methods could also support portfolio strategies that allocate budgets adaptively across algorithms. Ultimately, effective cross-task transfer turns isolated experiments into an accumulating asset.

RQ5: What strategies can be used to simulate training data for meta-models using the Autoencodix framework?

Motivation: Meta-learning methods often rely on surrogate models that approximate the response surface of performance over hyperparameters. However, real evaluations are costly; thus, simulated or augmented training data for surrogates can accelerate research and enable ablation studies. Within Autoencodix and Syne Tune, we can explore the strategy to fit an XGBoost surrogate to seed synthetic observations. The challenge is to produce synthetic data that preserves key structures—nonlinearities, heteroskedasticity, and interactions—without introducing artifacts. Simulation along simulated wall-clock time in Syne Tune allows controlled experiments on algorithmic choices, acquisition functions, and prior strength. It also enables stress-testing of warm-starts and transfer methods before committing large compute budgets.

RQ6: Among applicable HPO algorithms in Syne Tune, which one is most promising to find good HP combinations quickly for the given dataset use cases?

Motivation: Syne Tune offers a diverse toolbox——from random search baseline and Bayesian optimization to evolutionary methods——yet their relative performance is problem-dependent. For autoencoders, early-stopping signals, resource scaling, and noisy objectives complicate algorithm choice. A systematic benchmark across our five datasets provides evidence for which methods reach high-quality configurations quickly under realistic budget constraints. Recommendations reduce operational uncertainty for practitioners adopting Autoencodix. The result is a practical guide to "what to run first" in comparable representation-learning tasks.

RQ7: All in all, what can be derived on the effectiveness of approaches for integrating multi-modal molecular genetic data using autoencoder-based representations?

Motivation: Ultimately, autoencoders in our application domain are a means to an end: improving integration and analysis of multi-modal molecular genetic data. Beyond reconstruction losses, success must be judged by down-stream tasks such as classification, clustering, survival analysis, or association discovery. This question links tuning methodology to domain impact, asking whether better HPO and transfer actually yield better biological insights. It also probes the stability of learned representations across modalities and cohorts, which is crucial for reproducibility. By tying hyperparameter choices to end-task metrics, we can identify configurations that generalize rather than merely overfit.

2 Related Work

OMICS sequencing with its large amount of high-dimensional tabular numeric data overlaid by noise and influenced by many factors is a use case, where many skills are beneficial. Researchers in biology, medicine, chemistry, statistics, mathematics, computer science, as well as more recent interdisciplinary science such as biochemistry, bioinformatics, data science, medical computer science, should work together and contribute their individual expertise. Advances in each of these domains interacts with each other.

Multimodal results - tabular data (records), time series information (temporal), textual information (lingual, written), audio or radio frequencies (aural, ultrasonics), 2D and 3D images (visual, spatial) can all be important perspectives on the same studied organisms, cells, and genes; their interactions, diseases, and changes over time and species.

This thesis focuses on data for humans expressed as tabular numeric data from genomics, transcriptomics and epigenomics with two data sets, one on cancer research, one in cell development analytics for human cortex cells.

2.1 Data analysis workflows on high-dimensional tabular OMICS data

High-throughput sequencing technology involves complex processes on the generation and analysis of data from tissue samples and cells. The collection of sequencing data from genomics, transcriptomics, epigenomics, were major breakthroughs in their domains. All of which have come up with procedures on data generation and data normalization, as well as domain-specific data analysis. A biologically-inspired enumeration works along these "omics" disciplines in molecular biology:

Genomics - DNA sequencing (sometimes expressed as DNA-seq) is the process of determining the nucleic acid sequence – the order of nucleotides in DNA. See Shendure and Ji (2008) and Wang (2023) for a description of best practices and a common workflow.

Transcriptomics - RNA Sequencing (often abbreviated as RNA-seq) is a next-generation sequencing (NGS) technique used to identify and quantify RNA molecules in a biological sample, providing a snapshot of the transcriptome at a specific time. See Conesa et al. (2016), Luecken and Theis (2019), Wang (2023) for best practices and common workflows, focusing on single cell data.

Epigenomics - Assay for Transposase-Accessible Chromatin using sequencing (commonly abbreviated as ATAC-seq) is a laboratory technique used in molecular biology to assess genome-wide chromatin accessibility. See Heumos et al. (2023) for best practices and common workflows, focusing on single cell data.

This OMICS data (e.g. from DNA-seq, RNA-seq, ATAC-seq) is already considered multimodal itself. But as the basis sequencing techniques are further and further refined to e.g. isolation of cells and then single cell sequencing, or spatial analytics with the aid of high-resolution microscopy images (2D, 3D) and microarray carriers that add encoded spatial information (2D). It enables research into individual cells, cell-to-cell interaction to understand cellular function and heterogeneity, study genetic variations like mutations or genetic mosaicism.

These techniques are also extended to more and more "omics" domains, such as proteomics (e.g. CITE-seq - Cellular Indexing of Transcriptomes and Epitopes by Sequencing for RNA and surface proteins with available antibodies), metabolomics (adding influence of colon bacteria for example and digestion aspects in stomach), foodomics (adding food biochemistry data), to capture more and more aspects of biological interactions in the form of very detailed, low-level data.

From a data perspective, such omics workflows (whatever diverse they may be) result in high-dimensional vectors and matrices. Common dimensions are number of patients, number of tissue samples, number of cells, number of genes. The last dimension alone causes data along many thousands of genes. Thus, we have to cope with this high-dimensional data. We can assume, that the data (from roughly 20,000 protein-coding genes) contains a lot of redundancy.

Once normalized data is available after following a certain lab protocol, the workflows share common steps.

Two ways of analysis are common, that can also work in parallel:

1. Either, we work with this high-dimensional data directly. And exclude those, which are considered least relevant.
    - An extreme case is Random Feature Selection. It is not domain-specific at all. A machine decides what features to include based on a random process. Is unbiased, but hard to explain or to recreate. It may reduce noise, but it will also affect the actual signal. Thus only used for baseline comparisons.
    - Statistical property-based feature selection - statistical properties like correlation or variance can guide the (automated) feature selection process. It is based on the assumption, that the highest-variance features or most uncorrelated features are also the most relevant.
2. Or we get from high-dimensional data to fewer dimensions (lower rank approximation) with more sophistication that just filtering variables. So, subsequent work on such data aims to work on smaller-scale representations that still hold the essence of the high-dimensional data. We use dimension reduction techniques to also reduce noise or reduce sparsity (from sparse to more complete data) as an important step before downstream analysis.

2.2 High-dimensional labeled data: Statistical learning in multi-omics Workflows

Statistical Learning => provides the method of Lasso Regression (linear regression with L1 regularization) for feature selection on labeled data. It can reduce the assumed contribution of feature variables to zero, based on a parameter beta, thus excluding them altogether from further investigations. All influence from them is then either explained by other factors/features or considered irrelevant. However, the requirement of good quality labeled data sets limits on the applicability. At the same time, each feature selection will cover only this particular label-based use case.

2.2.1 High-dimensional parametric regression

In statistics and machine learning, the LASSO (least absolute shrinkage and selection operator—also called Lasso, LASSO, or L1 regularization) (see Hastie et al. (2009)) is a regression technique on labeled data that simultaneously selects variables and applies regularization to improve a model's predictive accuracy and interpretability. It was used in biomarker identification for example, see Ghosh and Chinnaiyan (2005).

Pro:

- can handle large amounts of data, fast and efficient
- can handle the case when number of variables (number of features) substantially exceeds the sample size (number of samples)
- relatively easy to comprehend/explain, like linear regression but with L1 regularization, that shrinks many coefficients to zero and therefore excludes these factors from a model
- needs only one hyperparameter: λ, that controls amount of regularization and hence the sparsity level of the fitted model
- returns coefficients of the remaining linear terms.

Contra:

- requires numerical labels/target variables (supervised learning), but also works with one-hot encoding on categorical target variables.
- averages all relationships (regardless of whether they are linear or nonlinear, i.e., quadratic, logarithmic, exponential, sigmoid) as linear relationships. (For an extension to generalized linear models, see below).
- Requires homoscedastic data (a prerequisite for linear models to deliver reliable results),

otherwise, the returned coefficients are either highly unstable or not very meaningful.

- With one hyperparameter, it is limited in adaptability.

2.2.2 Generalized Linear Models (GLM)

In Holmes and Huber (2018) in Ch. 8, the use of Generalized linear models is described for modern biology. Generalized Linear Models (GLM) make a non-linear model linear by a known transformation, based on expectation or observation/measurement. For example, if we take the square root of a quadratic relationship between variables or the log of an exponential relationship; then the new relationship is linear.

Pro: After this transformation, we can then use all of the existing methods and software for dealing with linear regression problems. The methods and software are mostly mature and stable. After back-transform to non-linear, this may be a way better fit than pure linear models. Apart from the specific parameters of this non-linear to linear transformation, there are no further hyperparameters that need tuning.

Contra: we must make assumptions or must "know" how which non-linear to linear transformation to apply. If the assumptions do not hold, then generalization will likely be poor.

2.3 High-dimensional unlabeled data: Dimensionality Reduction in Multi-Omics Workflows

The core topic of this thesis is work on unsupervised learning. Unsupervised learning methods include clustering and dimensionality reduction. See Hastie et al. (2009) for an overview. In dimensionality reduction, we learn lower-dimensional represantations of higher-dimensional data. Representation learning is motivated by the fact that downstream ML tasks such as classification often require input that is mathematically (a generic loss function can be used) and computationally (memory requirements, data types) convenient to process. However, real-world OMICS data, such as RNA data, methylation data, mutation data, it is hard to derive a generic feature compression suitable for a range of downstream ML tasks algorithmically.

This thesis works in the domain of unsupervised learning. This in an exploratory way of dealing with unlabeled data to identify new patterns/relationships, also to create new hypotheses.

In machine learning (ML), representation learning is a set of techniques that allow a system to automatically discover numeric representations (that keep certain required properties) needed for classification or regression from noisy raw data. This replaces manual feature engineering and allows a machine to both learn the features and use them to perform a specific task.

When we investigate recent literature in omics research on best practices for their dimensionality reduction steps, we can find a number of known methods. Here, we name some references in chronological order that also represent the development of dimension reduction on omics data over the last recent years:

Meng et al. (2016) gives an overview on dimension reduction techniques for the integrative analysis of multi-omics data for single-omics, two omics (paired omics), more than two omics. It includes t-SNE and UMAP for visualizing high-dimensional omics data.

Luecken and Theis (2019) recommends Principal Component Analysis (PCA) first (reduce to top k principal components (PCs)), then UMAP or t-SNE for visualization.

The Comparison for Dimensionality Reduction Methods of Single-Cell RNA-seq Data by Xiang et al. (2021) evaluated the stability, accuracy, and computing cost of 10 dimensionality reduction methods using 30 simulation datasets and five real datasets. As a result, as its top 3 methods along a compound measure on stability, accuracy, computing cost, it names: t-SNE, VAE, UMAP (all three of them non-linear techniques). They also looked at two aspects: 1) sensitivity to hyperparameter tuning and 2) whether users get appropriate recommendations. There, they claim that for dimensionality reduction methods based on non-linear model and neural network: users need to set hyperparameters appropriately according to the specific situation before. Citing from (Xiang et al. (2021)): "For those methods with multiple adjustable hyperparameters including (GrandPrix, scvis,) UMAP, and VAE, we noticed a dramatic change in the results when choosing different hyperparameter settings (Figures 6D–G). Therefore, we recommend that users consider the impact of hyperparameter settings before using these four methods."

Updated best practice as in Heumos et al. (2023) reserve a block generalized as dimensionality reduction. This generic block can then be filled with the best fitting method by researchers.

Let's look at the most common of these dimension reduction techniques in some more detail.

2.3.1 Principal Component Analysis (PCA)

Principal Component Analysis provides a linear method for dimension reduction and works by projection onto a lower subspace (along the eigenvectors). The user can then decide to keep only a number of main principal components for each sample in a lower dimensional space. See Holmes and Huber (2018) Ch, 7 - Multivariate Analysis and Hastie et al. (2009) for the use of PCA in general and on omics datasets.

Pro:

- It has relatively few requirements on input data (typical preprocessing: standard scaling). It is fast and scales well.

- It preserves global structure well. The PCs are orthogonal to each other.
- Does not require hyperparameters apart from choosing the number of principal components to keep. With this single hyperparameter, it needs no complex hyperparameter optimization.
- Few learned parameters (of the linear projection).
- Deterministic (apart from sign of eigenvectors), non-stochastic.

Contra:

- Its linear nature cannot deal with nonlinearities in the underlying structures of the data.
- No decoder from latent space back to original space is trained explicitly. Once more and more PCs are excluded, reconstruction error rises.

To reduce these limitations, many derivations of PCA have been proposed. They include:

- Kernel-PCA (which can adapt to nonlinearities, but we need to supply an appropriate kernel, which adds hyperparameters), see Hastie et al. (2009).
- PCA with L1 norm, see Wu et al. (2015).
- Nonlinear PCA, see Liu and Yu (2025). This nonlinear embedding and integration of omics data is presented as a fast and tuning-free approach. But the type of nonlinearities it can handle are limited. There are no parameters to adapt ist explicitly to all sorts of nonlinear functions.
- Also see methods related to PCA: Holmes and Huber (2018) Ch. 9 for Multivariate methods for heterogeneous data, including Principal Coordinates Analysis (PCoA), Correspondence Analysis, Multidimensional Scaling (MDS).

2.3.2 Low rank approximation methods rooted in NLP but used for omics data

For the sake of broader coverage and the multidisciplinary research, we also list two dimension reduction methods that are used on multi-omics data with roots in natural language processing.

LSI - Latent Semantic Indexing / LSA = Latent Semantic Analysis Latent semantic analysis (LSA) is a distributional-semantics method in natural language processing that explores how documents and their terms relate by mapping them into a shared set of underlying concepts. Based on an occurence matrix and a document-term matrix, it works a low rank approximation (SVD) for topics, that then converge to concepts. According to Lance and Martens (2023) LsA is used in Python package Muon, and R packages Signac, ArchR.

LDA - Latent Dirichlet Allocation Latent Dirichlet Allocation (LDA) is a generative probabilistic model that treats each document as a mixture of topics and each topic as a distribution over words, using Dirichlet priors to infer those hidden topic–word and document–topic distributions.

According to Lance and Martens (2023) it is used in Python package PyCistopic to simultaneously identify cell states and cis-regulatory topics from single cell epigenomics data. It is an unsupervised approach that simultaneously clusters cells and co-accessible regions into regulatory topics. The authors claim it to be among the top performers in multiple independent single-cell epigenomics benchmarks.

2.3.3 t-distributed stochastic neighbor embedding (t-SNE)

t-SNE (see Holmes and Huber (2018)) is a nonlinear dimensionality-reduction algorithm that maps high-dimensional data to 2D/3D by converting pairwise similarities into probabilities and minimizing KL-divergence to preserve local structure for visualization.

Pro: Good for visualization in 2D and 3D. Evaluation on omics data (see Xiang et al. (2021)) resulted in good performance, good accuracy (ARI, NMI, and Silhouette score). One hyperparameter (perplexity) was considered there, authors did not find too much of a difference on their data sets.

Contra: Slow with growing dimensions, not invertible. Wattenberg et al. (2016) states and shows, how different hyperparameters of t-SNE make a big difference on artificial datasets. We also cannot work with the embeddings in downstream tasks as the stochastic nature brings in many artifacts.

2.3.4 Uniform manifold approximation and projection (UMAP)

UMAP (Uniform Manifold Approximation and Projection) (see Holmes and Huber (2018)) is a fast, scalable nonlinear dimensionality-reduction method that models high-dimensional relationships as a fuzzy graph and optimizes a low-dimensional embedding that preserves local—and some global—structure. It can be regarded topology-informed and locally topology-preserving. Results depend on hyperparameters and data:
$n_{neighbours}$: larger → more global structure (fewer topological breaks), smaller → very local, more potential tearing.
min_{dist}: affects crowding but not topology per se.
metric choice matters; wrong metric can scramble neighbourhoods.

Pro: The evaluation in Xiang et al. (2021) on their datasets and scenarios found good stability, moderate accuracy. UMAP well preserved the original cohesion and separation of cell popu-

lations. UMAP embeddings can also be used above 3 dimensions with a limit of around 50 dimensions for downstream tasks like clustering.

Contra: Slow on large dimensions. Not invertible. Stochastic randomness in the results. Coenen and Pearce (2019) shows the influence of hyperparameters on UMAP in an interactive way and the diversity of results, when used for visualization.

2.3.5 Standard Autoencoders (AE) and beta-Variational AE (beta-VAE)

Autoencoders are neural networks trained to compress data into a low-dimensional latent code (encoder) and reconstruct it back to the original space (decoder). By minimizing reconstruction error, they learn nonlinear features that can capture complex manifolds far beyond linear methods.

Variants like denoising AEs, sparse AEs, and variational AEs (see Prince (2023)) extend them for robustness, feature disentanglement, generative modeling, anomaly detection, and pretraining.

Pro Autoencoder:

- General: AE is extremely flexible due to many hyperparameters, flexible number of trained parameters (weights, biases), and many ways to modify the loss function.
- These embeddings (referred to as latent "representations") can be used for further calculations in down-stream tasks.
- Training DNN, among them AE, has shown remarkable results. They build on lower-level packages for neural networks training (such as Torch, Keras, TensorFlow). Together with GPU clusters and enough data, new levels of accuracy can be reached and new applications can be tackled.
- when compared to PCA (linear): Generalizes standard PCA and also Kernel-PCA (when architecture, loss function, weights set properly). AE has nonlinear representation power; flexible architectures (convolutions for images, sequences); easy out-of-sample mapping (the encoder); can optimize for task-specific losses.
- when compared to t-SNE (visualization-oriented): AE learns an explicit mapping for new data; preserves reconstructability (decoder). VAE supports generative workflows; scales to large data with mini-batches and GPUs.
- when compared to UMAP (topology-informed and locally topology-preserving): AE provides encoder/decoder for fast transforms and reconstructions; can be trained end-to-end with domain-specific objectives; handles different data types via tailored architectures.

Contra Autoencoder:

- General: Due to the large number of trainable parameters, it overfits very easily if not enough regularization is applied. It may also underfit the data, if the AE model architecture (created by network hyperparameters) is not extensive enough.
- The many hyperparameters must be set appropriately, which requires hyperparameter optimization (HPO).
- when compared to PCA (global structure): AE with risk of poor global structure if not regularized well.
- when compared to PCA (deterministic): stochastic nature of AE: Training doesn't always find a good optimum, but sometimes gets stuck in local minima or due to too few training epochs or slow convergence. Not always robust with respect to data splitting, and SGD/AdamW training. Needs robustness considerations like restarts or progress analysis on epochs until a good solution is found.
- when compared to PCA (explainable loadings): AE less interpretable. But there is recent work that works on explainability, see Kreller (2025) on explainable β-VAE.
- when compared to t-SNE (visualization-oriented): AE results can be worse than t-SNE at ultra-clean 2D/3D local neighborhood visualization; embeddings can mix clusters without careful losses/regularization; more hyperparameters to tune.
- when compared to UMAP (topology-informed and locally topology-preserving): UMAP often gives better local/global structure in very low dimensions for exploratory plots; autoencoders may distort manifold geometry without constraints; training adds complexity compared with use of UMAP.

Autoencodix as software package for training AE and VAE

Autoencodix (Joas et al., 2024) is Python software (based on PyTorch and Scikit-Learn) for training autoencoder architectures of varying AE architecture and neural network structure with different training hyperparameters on different training datasets. The version used in this thesis comes with make files and bash scripts for control flow.

The Autoencodix framework is used and the work on benchmarking and hyperparameter tuning that was begun there is continued and deepened on the same data sets.

- input: YAML config files to describe data input, network hyperparameters, training hyperparameters, configuration of downstream task (see Chapter 2.)
- output: log text files (to parse run times), loss progression (as Parquet file which serializes a pandas data frame), results on classification downstream task (as Parquet file).

Autoencodix supports different autoencoder architectures, of which this thesis covers "Vanillix" and "Varix". These two are also the base architectures for the other ones.

- **plain (or standard) autoencoder** (in Autoencodix: "Vanillix") with fully-connected feed forward layers in encoder and decoder
- **beta variational autoencoder** (in Autoencodix: "Varix") with latent distributions in its latent space. The VAE learns parameters of these distributions in its encoder, and then samples from then and recreates data close to input data in its decoder. An additional hyperparameter β allows to tune the ratio of VAE loss in the total loss formula.
- a variational autoencoder with fixed weights in the decoder, these weights based on a domain-specific ontology (in Autoencodix: "Ontix")
- **stacked variational autoencoder** (in Autoencodix: "Stackix") for multi-modal data, where each data modality gets its own encoder
- **cross-modal autoencoder** (in Autoencodix: "X-Modalix") to align the latent space of one modality with the latent space of another modality, which aids in cross-translations

Some autoencoder architectures are not yet supported in Autoencodix, but could be implemented in the future:

- **denoising autoencoder** - add noise to input, have AE recreate the original input data, otherwise like standard autoencoder
- **sparse autoencoder** - only a limited set of latent layer nodes is allowed to fire, otherwise like standard autoencoder
- **CNN autoencoders** - intended for images, the encoder has convolution layers, down-sampling and pooling layers for image feature extraction, the decoder has unpooling, up-sampling and deconvolution layers

More examples of AE in omics processing

More and more autoencoder based tools and frameworks have been published. This is only a small list of some of them.

- scVI (single-cell Variational Inference) by Lopez et al. (2018)
- scgen by Lotfollahi et al. (2019) - a model combining variational autoencoders and latent space vector arithmetics for high-dimensional single-cell gene expression data.
- scANVI (single-cell ANnotation using Variational Inference) by Xu et al. (2021)
- Python package totalVI (total Variational Inference) Gayoso et al. (2021) - posits a flexible

generative model of CITE-seq RNA and protein data that can subsequently be used for many common downstream tasks.

- scGLUE - conditional VAE, to integrate unpaired RNA and ATAC data by (Cao and Gao, 2022)
- MultiVI - conditional VAE for paired multi-omics integration by Ashuach et al. (2023)
- Autoencoder-based Batch Correction (ABC) by Danino et al. (2023) - for batch correction

2.4 Total set of configuration parameters in Autoencodix

The following set of tables explains the configuration parameters (which include hyperparameters, but are far more than these) with their canonical name in YAML configuration files for Autoencodix. It serves as an overview and also introduces the configuration options in preprocessing and postprocessing.

With analysis of the documentation and code, this is a list of all the parameters in Autoencodix for training a Vanillix autoencoder. They also all apply to Varix. Table 5 at the end of this series lists the additional configuration options for Varix.

Table 1: Configuration parameters for work with Vanillix AE

Parameter Category	**Parameter Name in Autoencodix**	**Options**	**Description**
Preprocessing	DATA_TYPE	(RNA, METH, DNA)	given name for modality, semantics not taken into account
	FILE_RAW	<file name>	input file name (typically parquet format)
	TYPE	Numeric, Annotation	numeric feature data; or label data (clinical annotations)
	FILTERING	on variance/on correlation	input feature selection
	SCALING	StandardScaler / RobustScaler	Standard scaler from scikit-learn without or with outlier removal (more robust)
	SPLIT - data split (train, validation, test)	70% : 20% : 10%	ratio of records in training, validation, test set. Can also be "pre-computed" for specific sub-workflows for predictions for test data on trained or tuned models.

Table 2: Configuration parameters for work with Vanillix AE (contd.)

Parameter Category	**Parameter Name in Autoencodix**	**Options**	**Description**
Neural network architecture	MODEL_TYPE	vanillix, (varix) for scope of this thesis	standard AE, or β variational AE
	K_FILTER	integer	per data modality: node count in input layer and output layer
	ENC_FACTOR	float	encoding factor: ratio of number of nodes in adjacent hidden layers
	N_LAYERS	integer	number of hidden layers in encoder/decoder
	LATENT_DIM_FIXED	integer	dimensionality of latent space
Training	BATCH_SIZE	integer	batch size of the mini batches in batch normalization. Batch normalization is always used in Autoencodix.
	EPOCHS	integer	number of repetitions to train
	DROP_P	float	drop probability in drop out layer of Fully Connected layers. Dropout is always used in Autoencodix, disable by value 0.
	WEIGHT_DECAY	float (default: 0.001)	weight decay in the weight updates for training with AdamW optimizer
	LR_FIXED	float	learning rate in the weight updates of AdamW optimizer, not in the total loss with AdamW

Table 3: Configuration parameters for work with Vanillix AE (contd.)

Parameter Category	**Parameter Name in Autoencodix**	**Options**	**Description**
Loss function	RECONSTR_LOSS	MSE, BCE	function for reconstruction loss - mean squared error or binary cross entropy
	LOSS_REDUCTION	sum, mean	indicates if the loss should be summed or averaged over all features in the loss function
Downstream task on either embeddings, PCA, UMAP, RandomFeature	ML_TYPE	Auto-detect, classification, regression	classification (with auc_roc_ovo metrics) or regression (with r^2 metrics)
	ML_ALG	Linear Regression, RF Regression, Logistic Regression for Classification, Random Forest Classification	ML algorithm to run in downstream ML task
	ML_SPLIT	use-split, CV-on-all-data	use split; or cross validation
	ML_TASKS	Latent, PCA, UMAP, RandomFeature	dimension reduction methods to collect ML results on
	CV	integer (default: 5)	number of cross-validation rounds (if ML_SPLIT: "CV-on-all-data")
	CLINIC_PARAM target variables	**TCGA:** CANCER_TYPE, SEX, AJCC_PATHOLOGIC_TUMOR_STAGE, GRADE, PATH_N_STAGE, DSS_STATUS, OS_STATUS **SCHC:** CELLTYPE_BY_AUTHOR, AGE_GROUP, SEX	name of label columns used in downstream task

Table 4: Configuration parameters for work with Vanillix AE (contd.)

Parameter Category	**Parameter Name in Autoencodix**	**Options**	**Description**
Global parameters on randomness	FIX_RANDOMNESS	"random"	"random" for no fixing, "all" fix all randomness, "data_split" fix train/test/valid split, "training" fix weight initialization and randomness in training
	GLOBAL_SEED	integer	seed used to fix randomness in all subsequent processes
Tuning with Optuna	OPTUNA_TRIALS		40
	PRUN_PATIENCE	float (default: 0.05)	controls pruning of runs. "0" is full pruning (no patience) and stops trials early. "1" leads to no pruning at all since all epochs are considered. Pruning is disabled when beta-annealing is active in Varix.
	LAYERS_LOWER_LIMIT		lower limit: 2
	LAYERS_UPPER_LIMIT		upper limit: 5
	ENCODING_FACTOR_LOWER_LIMIT		lower limit: 1
	ENCODING_FACTOR_UPPER_LIMIT		upper limit: 64
	LR_LOWER_LIMIT		lower limit: 0.0001
	LR_UPPER_LIMIT		upper limit: 0.05
	WEIGHT_DECAY_LOWER_LIMIT		lower limit: 0.00001
	WEIGHT_DECAY_UPPER_LIMIT		upper limit: 0.1
	DROPOUT_LOWER_LIMIT		lower limit: 0.00
	DROPOUT_UPPER_LIMIT		upper limit: 0.50

Table 5: Additional Configuration parameters for Varix

Parameter Category	**Parameter Name in Autoencodix**	**Options**	**Description**
For β-VAE (Varix)	BETA	float	scaling factor for VAE loss in total loss function
	VAE_LOSS	KL, MMD	in VAE loss function: Kullback-Leibler divergence or Maximum Mean Discrepancy
	ANNEAL	logistic-early, logistic-mid, logistic-late, 3 phase-linear, 3 phase-log or no-annealing	curve for beta-annealing (VAE-loss)
	ANNEAL_ PRETRAINING	True, False	use beta annealing in pretraining target modality or while using gamma annealing in pretraining

2.5 Necessity of Hyperparameter tuning for autoencoders on tabular omics data

As shown, the model architecture in Autoencodix (used as an example here) has many configuration parameters. Among them we can also find the:

- hyperparameters for the network structure (**model hyperparameters**). These settings on layers and widths must be set before training even starts. This part is commonly referred to as **neural (network) architecture search** (Prince, 2023).
- specific formulation of the loss function. (must be set in advance as well)
- hyperparameters for the training process (**training algorithm hyperparameters**). These are distinct from model parameters. Partially their are set in advance, some may be changed during training along a certain schedule.

All these hyperparameters can be varied. And only a good combination of these hyperparameters may lead to a good AE.

Neural network training has reacted with splitting source data into three partitions: training data, validation data and test data. The validation data set's purpose is to fit hyperparameters without ever seeing test data. See Prince (2023) for reference.

In an analysis of tuning VAE for scRNA-seq data, Hu and Greene (2018) already came to the conclusion:

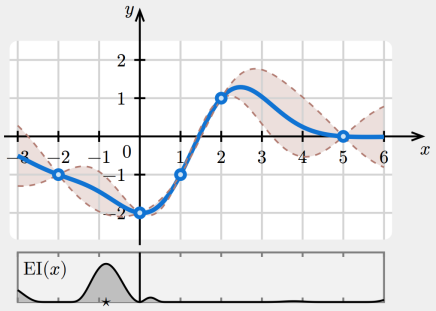
> "Parameter tuning is a **key part** of dimensionality reduction via deep variational autoencoders for single cell RNA transcriptomics." (Hu and Greene, 2018)

Prince (2023) writes in a section on training algorithm hyperparameters:

> "The choices of learning algorithm, batch size, learning rate schedule, and momentum coefficients are all considered hyperparameters of the training algorithm; these directly affect the final model performance but are distinct from the model parameters. Choosing these can be more art than science, and it's common to train many models with different hyperparameters and choose the best one. This is known as hyperparameter search." (Prince, 2023)

In the beginning, few authors actually published their settings for hyperparameters. This often caused excessive search for hyperparameters, which also led to enormous training resources being spent in AI in terms of GPUs and electric energy. When publishing hyperparameters (model parameters and training algorithm parameters) started in original publications, it led to the use of more recommendations and heuristics in various domains.

Example 1.1. To build intuition, let us consider the example depicted on the right. Five points (the blue dots) have been computed for f; the minimum value observed so far is $f_\star = -2$. Out of the prior over all functions, a posterior was computed by conditioning on these points. The **blue** solid curve represents the mean $\mu(x)$ of the posterior, while the **brown** shaded area represents one standard deviation $\pm\sigma(x)$ away from the mean.



The **expected improvement** (EI) is shown below, which is a heuristic that suggests where to query next. Specifically,

$$\mathrm{EI}(x) \coloneqq \mathbb{E}_{\tilde{f}}\big[\max\{0, f_\star - \tilde{f}(x)\}\big],$$

where at each x, the expectation is with respect to the conditional distribution of $f(x)$.

Figure 1: Intuitive example for Bayesian Optimization with known results at support points (here: deterministic function), but unknown regions in between. The expected improvement to minimize this function is then modeled a probability distribution (a more formal heuristic), where to sample next. Source: Farina (2024).

2.6 Automated hyperparameter optimization (AutoHPO) for Deep Neural Networks

Researchers and practitioners alike are not left alone with HP search. Prince (2023) in Ch. 8 also cites a number of influential academic papers on automating this process. It has become a very active research domain since then.

With manual HPO, the typical values for hyperparameters are either default values, or heuristics along recommendations, or resource intensive random sampling or grid search among HP combinations. Depending in resources, this can be minimal, trial-and-error up to excessive.

Automated hyperparameter optimization makes the step to the formulation of an optimization problem.

A common approach is Bayesian optimization (BO). This statement by Farina (2024) well summarizes, why BO is a very promising approach for HPO in neural network training:

> "(Bayesian optimization) is particularly useful when the function to be optimized is expensive to evaluate, and we have no information about its gradient. Bayesian optimization is a heuristic approach that is applicable to low-dimensional optimization problems. Since it avoids using gradient information altogether, it is a popular approach for hyperparameter tuning, architecture search, et cetera." Farina (2024)

In addition to BO, approaches from evolutionary algorithms or are other ways to tackle such

optimization problems.

The goal of HPO is to hand over *some* hyperparameters to automated hyperparameter tuning and to include previous results in the form of priors. Seldom, *all* hyperparameters are to be tuned by HPO, as configuration space expands with each new hyperparameter as a new dimension, unexpected combinations of hyperparameters can occur and new local minima in global configuration space can arise.

It is important to note, that HPO frameworks will introduce even new hyperparameters / configuration parameters themselves, so that the HPO can be tailored to the optimization problem and also to the amount of resources users are willing to spend on HPO. Typical ones are stopping criteria like number of trials, wall-clock time, or adaptive early stopping criteria. Also the amount of resources in terms of worker nodes can be configured and considered an HPO hyperparameter. Finally, if a surrogate model is used by the HPO, this surrogate model (e.g. XGBoost) will need its set of hyperparameters specified.

There is a number of packages for HPO on training of deep neural networks that all bring along pre-built functionality for this task. Here, we cover Optuna and Syne Tune.

2.6.1 Use of Optuna in Autoencodix

A popular HPO framework is Optuna by Akiba et al. (2019). The optional use of Optuna was already built into Autoencodix, and some of the results are published in Joas et al. (2024), with a reproducible data set and processing published in Ewald (2024) .

Use is limited, though: only for the hyperparameters N_LAYERS, ENCODING_FACTOR (that both alter the model architecture and number of trainable parameters) and WEIGHT_DECAY, LEARNING_RATE, DROPOUT (for training related hyperparameters), Autoencodix can optionally be configured to be tuned by Optuna within a lower and upper limit for each of these parameters.

The optimization target for Optuna is hard coded in Autoencodix as the total validation loss (which is affected by K_FILTER). A `TPESampler` (Tree-structured Parzen Estimator) is then used as a default sampler for this single-objective optimization.

An additional parameter PRUN_PATIENCE controls the pruning of runs during hyperparameter tuning with Optuna. It sets, after what percentage of epochs a pruning decision by a `MedianPruner` is to be made. For varix, pruning is disabled if beta-annealing is active in order to get the full epoch trajectory. Finally, the configuration parameter OPTUNA_TRIALS controls the number of Optuna trials.

Optuna (when enabled in Autoencodix) is used inline in the training process, but has only access to the validation loss, not the downstream performance. It was shown, that downstream

performance on the test set does only mildly correlate with total validation loss (see Joas et al. (2024)).

Inline in the training process means, that Optuna is used with running actual Autoencodix training runs. While this needs no pre-training, this approach cannot access prior knowledge. Convergence in tuning was slow and tuning was inefficient, as the tuning process restarts from zero knowledge each time when a new tuning run is started.

All in all, this approach can be seen as a first step towards HPO in Autoencodix, that has severe limits on the quality of the optimization results. Despite the limitations, it has shown that automatic HPO is valuable and convenient and motivated more intensive work on HPO frameworks and algorithms.

2.6.2 Syne Tune

This thesis uses Syne Tune by (Salinas et al., 2022), which is a library for large-scale distributed hyperparameter optimization (HPO). Although these two aspects of large-scale and distributed have not been used here, they can be very desirable features later.

Syne Tune (Python package) for hyperparameter optimization

- input: tabulated results (e.g. parquet files, CSV) loadable into Pandas data frame
- configuration: names of the hyperparameter columns as input, configuration space as a cross product of explicitly listed bounded distributions (e.g. uniform, log uniform, choice) on these hyperparameters, name of metrics columns, type of generic ML model to use to build the surrogate model, name of the schedulers/HPO algorithms to use, maximum wall-clock time for a simulated HPO runs.
- output: new recommended hyperparameter configurations. This is accompanied by a predicted performance (metrics) estimate, if a surrogate model is given to Syne Tune.

Syne Tune offers many different execution backends with different searchers, single-objective or multi-objective optimization, single-fidelity or multi-fidelity optimization. This thesis uses the following searchers, all on single-objective, single-fidelity optimization problems:

RS (RandomSearch): Selects hyperparameters randomly from the search space, providing a simple baseline method that does not use prior knowledge. As such it is unbiased, but also slow in improvement, and only used as a baseline.

KDE (Kernel Density estimator): similar to TPE (Tree-Parzen Estimator) as used in Optuna described before.

REA (Regularized Evolution Algorithm): , a population-based evolutionary approach that mu-

tates and selects the best-performing hyperparameter sets over time.

BORE (Bayesian Optimization with Density-Ratio Estimation): an adaptive method that models the probability of improvement using density estimation.

CQR (Conformal Quantile Regression): a robust uncertainty-aware optimization method from Bayesian optimization that uses quantile regression for reliable performance estimation.

The individual HPO algorithms are all cited in Syne Tune itself, either in the paper by Salinas et al. (2022) or in the source code from Github.

Out of the extensive feature set of Syne Tune, transfer learning as well as searching in constrained configuration spaces were used in combination with the optimization strategies. See Chapter 3 on page 27 for the actual approach.

2.7 Datasets used for Training, Validation, Test

2.7.1 Dataset 1: The Cancer Genome Atlas (TCGA)

Authors describe the data set "Pan Cancer Atlas 2018" Liu et al. (2018) from TCGA as a harmonized cancer dataset for 11,160 human patients with over 33 cancer types. It is intended for large-scale translational research.

The dataset contains data from transcriptomics (bulk cell RNA sequencing results from tissue); epigenomics (bulk cell ATAC (methylation) sequencing results); genomics (bulk cell DNA mutations sequencing results).

Such bulk cell sequencing results are less data intensive on subsequent data processing, but contain averages over cells in the tissue sample and therefore limit in-depth research.

Clinical labels are available on patient meta-data as well as disease-specific survival and overall survival outcome.

2.7.2 Dataset 2: Single cell from Human cortex

This data set originated from Zhu et al. (2023) and contains *single cell* sequencing results from samples of human cortex from various development stages.

The dataset contains data from transcriptomics (single cell RNA sequencing results); and epigenomics (single cell ATAC sequencing results on DNA methylation) from a 10x Multiome platform.

It includes two meta-data labels (sex, age group) at the patient level, and one label (author-given cell type) at the cell level, suitable for downstream classification tasks. This way, latent

space embeddings can be evaluated, wether they capture and separate well on these variables in a meaningful manner as a first step towards domain specific properties.

3 Approach

This thesis takes a data science approach that is aimed at HPO for autoencoders in general. In data science, data sets are processed and analyzed more based on their statistical distributions and with cross-domain applicability in mind.

This thesis combines Autoencodix and Syne Tune. Autoencodix writes result files, these are modeled into a data format for processing with Syne Tune. Regarding data sets: this thesis works on data, that had already been studied before by authors of Autoencodix (TCGA, SC human cortex) with OMICS data, see section 2.7 on page 25. This allows to compare the results. We mainly abstract from the domain-specific aspects (domain: epigenomics, genomics, transcriptomics, biology, bio-informatics, medical, cancer diagnostics, molecular biology), as we cannot really judge on the implications. Basic plausibility was checked however.

3.1 General approach to a hyperparameter optimization problem

The basic properties of a hyperparameter optimization problem are:

- a list of hyperparameter names to vary (either discrete or continuous) - both **model hyperparameters** (for (neural) network architecture search) and **training algorithm hyperparameters**. Please keep in mind, that both of these searches to not actually mean the SGD/AdamW optimization that optimizes the networks weights and biases (commonly called ϕ_t or θ_t.) in the training loop.
- the ranges or choices of these hyperparameters (i.e. a distribution to sample from and get a value for each hyperparameter) – this forms a search space
- an objective function, dependent on these parameters. This objective function can either give a single value metric (single-objective optimization) or multiple values (multi-objective optimization).

As we do not know the objective function (in closed form, this is next to impossible given the many influence and stochastic methods), the optimization problem is regarded as a black box. In Syne Tune this is implemented in a "blackbox tabular" data structure with these attributes:

- configuration space of hyperparameters
- hyperparameter combinations - a grid: $n_{features}$ features - the hyperparameter names, and $n_{samples}$ samples - the HP combinations of values
- objective names - the pure names of the objectives
- objective evaluations - a grid of results for each hyperparameter combination and each objective (eventually with two more dimensions: fidelities ($n_{fidelities}$) and seeds n_{seeds})

This is simplified here, because we use only a single fidelity (training budget: EPOCHS=300). We do not use multiple fidelities, which are a concept in Syne Tune to store the development of objectives over the training epochs. See blackbox `lcbench` (Zimmer et al., 2021) in Syne Tune for an example, where the results for the first 50 epochs of 35 tasks with one seed are stored explicitly.

We also use only a single seed (which we set as random, so no fix seed). A Syne Tune Blackbox Tabular in contrast is prepared to store the objective evaluations of multiple runs with different seeds. This was not done here however due to resource constraints.

This whole Blackbox Tabular data structure is then filled on each of the 5 tasks (TCGA RNA, TCGA METH, TCGA DNA, SCHC RNA, SCHC METH).

Indeed, some of the blackboxes in Syne Tune were created with more than one seed and/or more than one fidelity:

- NASBench201 - 3 seeds, 201 fidelities (each epoch: 1 to 201)
- FCNet - 4 seeds, 100 fidelities (each epoch: 1 to 100)
- LCBench - 1 seed, 50 fidelities (each epoch: 1 to 50)
- TabRepo - 3 seeds, 1 fidelity (exactly 100 epochs)
- The blackboxes for PD1 and HPOB have a different structure.

For the references to these data sets, see Syne Tune software from Github directly.

3.2 Hyperparameter Configuration space

RQ1: What are most relevant hyperparameters and sensible value ranges?

For the subsequent experiments, we want to run Autoencodix on many autoencoder models (different neural network architecture) of different sizes (and as such: with different numbers of trainable parameters) and with many different values for training algorithm hyperparameters. This necessitates a configuration space of hyperparameters as a list of hyperparameter names, that are each associated with a distribution from which hyperparameter values are then sampled.

In Chapter 2 on Related work, Table 1 on page 16 to Table 5 listed *all* configuration parameters available in Autoencodix for Vanillix and Varix. In Autoencodix by Joas et al. (2024), hyperparameters and ranges for tuning with Optuna had already been tested to some degree.

This thesis is follow-up work on these previous efforts. For HPO, we want to focus on a manageable subset of these configuration parameters and focus on the *hyperparameters* that either affect the autoencoder model size (model parameters) and hyperparameters in the loss function,

and training algorithm hyperparameters during training and validation. This way, we can first sample HP combinations for actual Autoencodix runs for Vanillix and Varix on the 5 data sets. And we can later hand over a hyperparameter configuration space to Syne Tune.

The chosen hyperparameters, with their parameter name in Autoencodix, a common mathematical symbol for easy reference with literature or other papers, and distributions/ranges are shown in Table 6. The values were slightly modified from the original search space.

Smallest and largest network in this configuration space To get an imagination of the coverage of the model parameters, smallest, average, and largest networks are given here.

Smallest network size: ($D = 128, n_{layers} = 2, r = 4, d_z = 2$): (128, 32), [2], (32, 128) nodes in 5 layers total. This results in 8,514 trainable parameters (weights and biases).

An average network ($D = 512$ or $1024, n_{layers} = 3, r = 2, d_z = 8$ or 16) would have 7 layers and 331k to 1.3M trainable parameters, or 776 to 1,544 hidden nodes.

Largest network size: ($D = 4096, n_{layers} = 4, r = 1, d_z = 64$): (4096, 4096, 4096, 4096), [64], (4096, 4096, 4096, 4096) nodes in 9 layers total. This results in 101,216,320 trainable parameters (weights and biases).

So, at the extremes, autoencoder models with about 8k to 100M (600k on average) trainable parameters, and a network capacity (see definition in Prince (2023)) of 66 to 24,640 hidden nodes (1000 on average) are covered within this HP configuration space. We could also calculate the number of linear regions (see Prince (2023)) here as a property of the DNN with ReLU.

Maximum-to-minimum ratio. For each hyperparameter, we can compute a max/min ratio to get a better understanding of the variability. It is shown in Table 7. Large min/max ratios then either get a log uniform distribution or a choice of values from an exponential series to better explore multiplicative developments instead of linear ones.

Approximate size of HP configuration space. The HP configuration space has some discrete hyperparameters sampled from a fix set of choices (D, n_{layers}, d_z, bs) and some continuous hyperparameters sampled from uniform ($1/r$, p_{drop}) or log uniform (lr, β) distributions. With these continuous distributions, the space is essentially endless. In case we would imagine these continuous variables as being discretized to 5 values each (min, 25%, mid, 75%, max) in a grid, then the HP configuration space would have about $540 \cdot 100 = 54,000$ HP configurations in total for Vanillix and $270,000$ HP configurations for Varix. See Table 7 for deduction of these numbers.

As Vanillix and Varix differ in their hyperparameter sets (Varix adds β), we need to setup HPO

Table 6: Hyperparameter configuration space for subsequent experiments.

Configuration parameter name in Autoencodix	Symbol	Distribution to sample from	Description
Network architecture hyperparameters			
K_FILTER	D (also N in literature)	choice([128, 256, 512, 1024, 2048, 4096])	input features per data modality, width of input and output layers
N_LAYERS	n_{layers} (also K in literature)	choice([2, 3, 4])	number of layers for the encoder, respectively decoder, always symmetric in Autoencodix
ENC_FACTOR	$1/r$	uniform(1, 4)	factor r by which the width of input layer is divided in encoder to get the width of the subsequent layer. $r > 1$: compressing AE; $r < 1$: overcomplete AE.
LATENT_DIM_FIXED	d_z (also n_z or k in literature)	choice([2, 4, 8, 16, 32, 64])	dimension of latent space, width of latent space layer
Hyperparameters in training process with AdamW optimizer			
LR_FIXED	lr (also γ, η, α in literature)	log_uniform(1.0e-5, 1.0e-1)	learning rate in AdamW optimizer in the weight updates
BATCH_SIZE	bs (also B, m in literature)	choice([32, 64, 128, 256]	implicit regularization via batch normalization layer; also governs mini-batch processing on GPU
DROP_P	p_{drop}	uniform(0, 0.9)	drop probability in dropout layer
Hyperparameters in loss function			
BETA (Varix only)	β	log_uniform(0.001, 10)	scaling factor of VAE loss in total loss

Table 7: Analysis of hyperparameter range

Variable network architecture hyperparameters			
Symbol	Distribution	Max/Min ratio	#values
D	choice([128, 256, 512, 1024, 2048, 4096])	32	6 values
n_{layers}	choice([2, 3, 4])	2	3 values
$1/r$	uniform(1, 4)	4	5 values (if discretized)
d_z	choice([2, 4, 8, 16, 32, 64])	32	6 values
		$6 \cdot 3 \cdot 5 \cdot 6 = 540$ network architectures for the Autoencoder model.	
Variable hyperparameters in loss function or training process			
lr	log_uniform(1.0e-5, 1.0e-1)	10,000	5 values (if discretized)
bs	choice([32, 64, 128, 256]	8	4 values
p_{drop}	uniform(0, 0.9)	—	5 values (if discretized)
β (Varix only)	log_uniform(0.001, 10)	10,000	5 values (if discretized)
		$5 \cdot 4 \cdot 5 = 100$ training parameter combinations for Vanillix. 500 for Varix.	

separately for Vanillix and Varix. This will be addressed again later in data import into Syne Tune objects.

3.3 Identification of objectives/target variables/metrics

An optimization problem needs an objective function, that outputs a target variable or metric based on input.

3.3.1 List of objectives for general analysis

There are many potential objectives, that were divided into three target variable groups:

- different runtime metrics of the steps in Autoencodix
  - data preprocessing runtime
  - training runtime
  - visualization runtime
  - ml task runtime
  - total runtime (sum of the above)

- different loss metrics:
  - reconstruction loss in training and or validation or test (absolute sum of squared error) or mean squared error (MSE)
  - training R2
  - validation R2
  - test R2
  - varix: vae loss = KL loss
  - varix: total loss: reconstruction loss + beta * vae loss
- different downstream performance results per clinical variable:
  - The downstream tasks on the embeddings of OMICS data (DNA mutations, Methylation, RNA transcription) are multiclass classification problems on clinical variables using logistic regression. (RandomForest classification or SVM classification were not used, they could overfit more easily and not give a representative result on quality of the latent space.) The ML downstream task in Autoencodix returns values for area under curve (AUC) for receiver operating characteristic (ROC) in a one versus one (OVO) comparison for the classes. "One versus one" splits a multiclass classification task (except for sex, which has binary values, all other clinical variables are multiclass) in many binary classifications with a one versus one classification of each combination of two classes. AUC for ROC then reflects a potential trade-off between true positive rate (TPR), also known as sensitivity or recall, and true negative rate (TNR), also known as specificity.
  - In the downstream tasks of all runs, We also compute baseline downstream performances on latent embeddings (representations) by PCA, UMAP and RandomFeature representations and create plots for easy reference.

3.3.2 Focused subset for Syne Tune

A focused subset of target variables for Syne Tune must be made in a way, so that 1) knowledge transfer is possible and 2) simulation on wall-clock time get the most appropriate metric. We chose:

- training runtime - *time metric* for simulation in Syne Tune later
- validation R^2 - as it is independent of input dimension D
- (weighted) average downstream performance on AUC ROC OVO results. (weighted) average downstream performance for TCGA is the mean of of AUC ROC OVO values on

mean(cancer type, cancer sub type, oncotree code—these are highly correlated—), sex, AJCC pathologic tumor stage, grade, path N stage, disease specific survival status and overall survival status). For SCHC, it is simply the mean of AUC ROC OVO values on of sex, age group, author cell type.

3.4 Getting the objectives evaluations

To get results on the target metrics (runtime, validation R^2, weighted avg downstream performance) for the objective calculations, actual Autoencodix runs are performed and these metrics are measured.

3.4.1 Objectives in Autoencodix are results of random experiments

It is important to remark, that an Autoencodix run with FIX_RANDOMNESS: random (no seed set) is a random experiment. So running Autoencodix in such a configuration will return different results. Within the target variables:

- Runtimes are a random variable in any case, as they are affected by outside influence like network delay to network storage devices.
- Losses (Training, Validation, Test) are random variables due to the nature of Stochastic Gradient Descent (SGD) used in training and due to a random sample split.
- Downstream Task performance are random variables due to random data split what particular records are in training (batches), validation, test set.

The latent representations are also random variables as they depend on the trained weights and biases which are part of SGD. UMAP introduces randomness into its results by design. PCA would be deterministic (if the data were fix), if a classical method is used, apart from the sign and order of the eigenvectors. The used method of Logistic Regression depends on the implementation, whether it is deterministic (given the same input) or non-deterministic.

3.4.2 Number of trials/hyperparameter combinations

The Number of runs is limited by compute resources that we are willing to spend. We could spend a budget of runs differently, here are two options for 30,000 runs:

30,000 runs = 5 datasets, 2 AE architecture, 3000 HP combinations, 1 seed (or no seed)
30,000 runs = 5 datasets, 2 AE architecture, 1000 HP combinations, 3 seeds.

Important constraint: If we want to create the Syne Tune Blackbox Tabular data structure

for multiple *tasks* (like on TCGA RNA data, TCGA METH data, TCGA DNA data, SCHC RNA data, SCHC METH data), then these tasks must share the exactly same hyperparameter combinations for transfer learning later. So, we use the same HP combinations across the 5 datasets.

3.4.3 Sampling method for HP combinations

Grid search is prohibitive given the space of 54,000 or 270,000 combinations (if uniform distributions were discretized to 5 values, see Ch. 2). It is also not considered necessary, as Syne Tune shall pick up on a much smaller set already.

Random Sampling (i.i.d. = independent and identically distributed) – with the specified log scales – was chosen here to get 3000 HP parameter combinations.

While a Sobol sequence of 3000 configurations would cover the space more evenly by avoiding local agglomerations of the random samples, this would introduce internal dependencies between the samples.

3.4.4 Configuration on data preprocessing in the Autoencodix runs

- input feature selection: Select the K-FILTER (or D) ...
- filter selection criteria - DATA FILTERING: ... most variable features ...
- data scaling - DATA SCALING: ... and run them through a standard scaler to bring features into common mean $\mu = 0$ and variance $\sigma^2 = 1$.

The datasets are assumed to have few outliers, which would encourage the use of a RobustScaler.

3.4.5 Run these configurations on a compute cluster on data

These 30,000 runs are full, computationally expensive runs of the AE framework on a single hyperparameter configuration each. Distributed execution on GPU cluster was used to do the the heavy lifting.

3.5 Gather results, analyze and visualize results

RQ2: Given many thousand runs of Autoencoder training including down-stream task metrics, what can we learn about hyperparameter influence and interdependencies?

To collect the data, target variables from training runs need to be gathered from the YAML

condifuration files, logfiles (time stamp evaluation to get runtimes), parquet files (losses), text files (downstream performance) and compiled into tabular data. On this basis, we then need many scatter plots for variable to variable relationships and correlation matrices, laid out by groups or hierarchical clustering along similarity. This compiled information can then be saved as a data frame in Parquet format.

3.6 Prepare for AutoHPO and transfer learning

RQ3: How can a meta-learning approach be used to warm-start hyperparameter optimization for autoencoders for the **same task** on a **different HP combination**?

RQ4: How can previous hyperparameter optimization runs be leveraged to improve performance on **new tasks**, like similar or other datasets?

AutoHPO by Syne Tune needs all the data in its own object-oriented format. Syne Tune has internal function to support creating import routines into syne tune blackboxes.

As set up above, all 5 data sets (modeled as tasks in Syne Tune, specifically: TCGA RNA data, TCGA METH data, TCGA DNA data, SCHC RNA data, SCHC METH) share the same support points (HP combinations).

Two Syne Tune black boxes are set up, one for Vanillix and one for Varix, as Vanillix and Varix differ in their hyperparameter sets (Varix adds β). But by filling them with the 5 tasks each with the same hyperparameter combinations, we can still later use transfer evaluations.

Two types of transfer can later be used:

1. transfer from previous runs on the same task. This just loads previous results and can warm-start the BO (or candidate population in the case of evolutionary algorithms). This enables an informed Bayesian optimization approach, which can take into account prior distributions.
2. transfer of promising hyperparameter configurations across tasks (transfer learning), where the HP search process can be initialized with previous results from other tasks.

RQ5: What strategies can be used to simulate training data for meta-models using the AUTOENCODIX framework?

The 3000 support points with real Autoencodix results are only a fraction of the total space. So, in order to enable Syne Tune to get metrics for every HP combinations, we need to train a surrogate tabular model for Vanillix and Varix from previous batch runs. This will not only return estimates on validation R^2, training time and an estimate on average downstream performance, but also substantially speed up response of the Autoencodix process. XGBoost is

a method that is quick in training and evaluation and can serve this purpose.

Then, we can run many algorithms of HPO framework on this surrogate model. This combination significantly speeds up search process for potentially better hyperparameter configurations, as the actual autoencoder network does not have to be trained each time.

3.7 Run HPO algorithms on surrogate model to get new recommended configurations

RQ6: Among HPO algorithms in Syne Tune, which one is most promising to find good HP combinations quickly for the given dataset use cases?

From Syne Tune source code, the applicable searchers/schedulers were identified. In our case, single fidelity, single objective searchers were appropriate choice.

Supported optimization strategies are: CQR, BORE (both of them methods in Bayesian optimization), KDE (like TPE), REA (method from evolutionary algorithms). See Section 2.6.2 on page 24 for a description. We can run them either on a single task (repeatedly with a number of worker nodes, with an overall stopping criteria along simulated wall-clock time, where wall-clock time itself runs on simulated training time). We can also use the same scheme and run on task transfer evaluations.

We no longer use grid search or random sampling here; they are too basic and do not leverage the previous knowledge. Random sampling will be useful as a baseline, however.

3.8 Evaluate new recommended configurations in actual Autoencodix

RQ7: All in all, what can be derived on the effectiveness of approaches for integrating multimodal molecular genetic data using autoencoder-based representations?

Running Syne Tune, it shall give us new better HP combinations along with a prediction on training time, prediction of validation R^2 and prediction of downstream performance. If we then take these HP combinations and feed them into real Autoencodix, we can get actual "true" metrics, still keeping in mind, these are values for random variables. A high correlation of predicted metrics and actual metrics would mean, the surrogate model was close enough to the real results. Getting new higher values on actual downstream task ML performance (within a reasonable number of tries) would indicate, that the whole setup works.

4 Results

4.1 TCGA and SCHC as multi-omics datasets – analyzed with their modalities separately (single-omics)

This thesis uses the same data input as previous runs of Autoencodix in Joas et al. (2024), so that many results on this data are already available and the hyperparameter configuration space for two multi-omics data scenarios (TCGA, SCHC) have been explored to some degree. This will allow better evaluation based on previous results and findings.

4.1.1 Dataset 1: Bulk cell sequencing results from Pan Cancer Atlas from TCGA

We extend previous analyses shown in Joas et al. (2024) and use preprocessed datasets. Preprocessing of data from tab-delimited text files to Parquet files is documented in the reproducible dataset by Ewald (2024).

As input, we use one Parquet file for each of the three modalities and one for clinical metadata. Descriptive statistics are shown in Table 8.

- **bulk** cell sequencing results from **mRNA-seq** (RSEM results)
- **bulk** cell sequencing results from **ATAC** (methylation per gene as percentage)
- **bulk** cell sequencing results from **DNA** mutations (CNA: copy number alterations)
- **clinical metadata** at the level of tissue samples (n=10,953). 54 clinical variables, a subset of which (9 clinical variables) we will use as labels in downstream classification tasks to evaluate embeddings on multi-class classification tasks using logistic regression.

Table 8: Statistical properties of preprocessed TCGA data set

Dataset	**mRNA-seq**	**ATAC-seq**	**DNA mut**
Number of samples (tissue samples)	10,071	10,741	9,870
Number of features (Entrez gene IDs)	16,313	9,228	20,304
data type	float64	float64	float64
count	164,288,223	99,117.948	189,701,400 (5% null)
mean	1,197	0.26	0.20
standard deviation	6,862	0.29	0.30
min	-0.98	0.00	0.00
25%	56	0.04	0.00
50%	362	0.08	0.03
75%	1065	0.45	0.28
max	7,092,440	1.00	96.34
Compressed data size	1,278 MB	972 MB	363 MB

4.1.2 Dataset 2: Single cell sequencing results from Human Cortex

Like the dataset on TCGA, this original dataset on single cell sequencing byZhu et al. (2023) was already preprocessed in Joas et al. (2024). It was also analyzed there. Reproducible data preprocessing is published in the same Zenodo repository by Ewald (2024).

We use the same published two preprocessed Parquet files for

- single cell sequencing results from **scRNA-seq** and
- single cell sequencing results from **scATAC-seq**

modalities plus one preprocessed Parquet file for the **clinical variables** in this thesis for training standard feed-forward autoencoders (Vanillix) and β-Variational autoencoders (Varix). Descriptive statistics on these two datasets are shown in Table 9. The associated Parquet file with clinical metadata has n=45,549 samples and 30 features, of which sex, age_group and author_cell_type are used in this thesis. 10x Genomics Multiome single-cell barcodes / cell IDs serve as an index for the samples.

Table 9: Statistical properties of preprocessed SCHC data set

Dataset	**scRNA-seq**	**scATAC-seq**
Number of samples (cells)	24,437	45,549
Number of features (Ensembl Gene IDs)	6,000	6,000
data type	float64	float64
count	146,622,000	273,294,000
mean	0.22	0.30
standard deviation	0.46	0.50
min	0.00	0.00
25%	0.00	0.00
50%	0.00	0.00
75%	0.00	0.69
max	6.06	5.85
Compressed data size	48 MB	107 MB

4.2 Results of the 30,000 Autoencodix runs

30,000 Autoencodix runs were completed on the cross product of:

- 3+2=5 single-omics modalities (TCGA: RNA, METH, DNA), (SCHC: RNA, METH) on these two multi-omics datasets.
- 2 autoencoder architectures (Vanillix = standard feed-forward autoencoder, Varix = β-VAE).
- 3000 random sampled HP configurations from configuration space described in Chapter 3.

Number of seeds/repetitions: 1 repetition each was performed on the 30,000 runs, despite known randomness of the Autoencodix training and ML downstream task. Reason: spending more runs on data sets and AE architectures was prioritized over repetitions.

Figure 2 shows the best 7 results (in terms of highest average downstream performance as stated in Chapter 3) for TCGA for each of the three modalities and both autoencoder architectures.

Figure 3 shows the best 10 results (in terms of highest average downstream performance as stated in Chapter 3) for SCHC for each of the two modalities and both autoencoder architectures.

Total compute: about 5 weekends on a variable number of about 25-150 GPUs. See GPU cluster spec at the end of this chapter.

Total output size: about 10 MB per run (with saved plots as PNG images, small tables on the result data, log files), about 300 GB in total.

4.2.1 Correlation plots

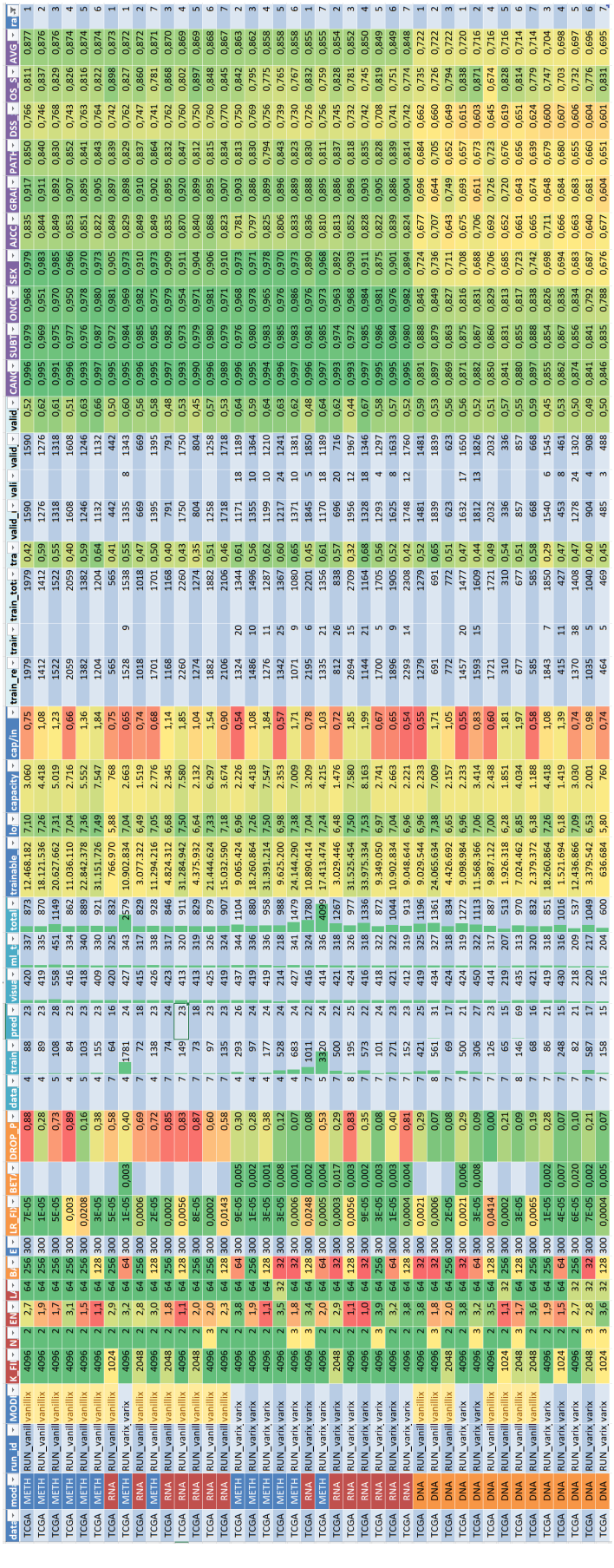
See correlation plots for the target groups in Fig. 4 (correlation of hyperparameters with runtimes), 5 (correlation of hyperparameters with train R^2 and validation R^2) and 6 (correlation of hyperparameters with downstream performance on test set). All correlations have been determined as Spearman rank correlations.

4.2.2 Varix - influence of beta

For Varix as a β-VAE we tested the effect of the scaling coefficient β which was log-uniform sampled from (0.001, 10) on a particular example.

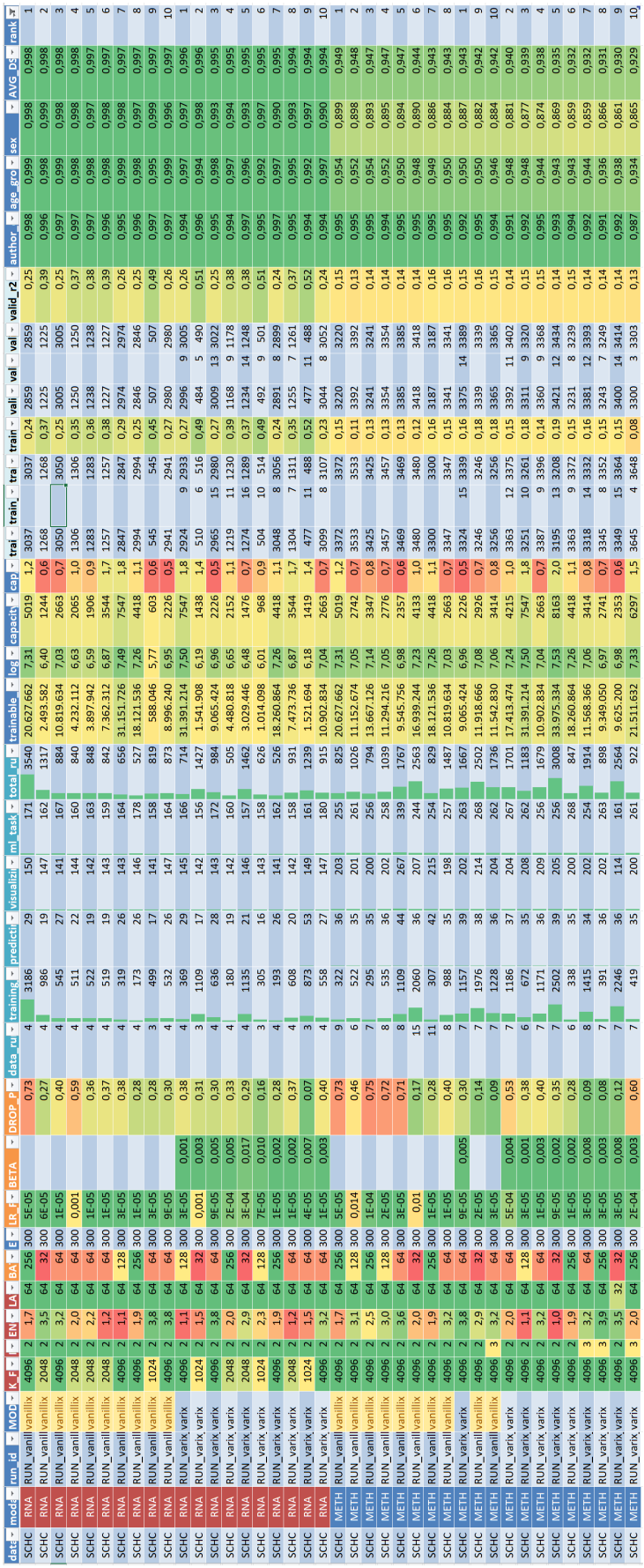
We did a beta sweep over all values for latent dimension d_z (2, 4, 8, 16, 32, 64) on SCHC RNA on one HP combination that was recommended before by Syne Tune vanillix for this data set SCHC RNA.

This also has a manual standard AE to β VAE transfer aspect (all parameters except beta were



data	mod	run_id	MOD	K_cat	N	EN	L	B	E	LR_fi	BET	DROP_P	date	train	pred	visua	ml_t	total	trainable	lo	capacity	cap/in	train_re	train	train_tot	tra	valid_l	vali	valid	CAN	SUBT	ONCT	SEX	AECO	GRAI	PATH	DSS	OS_S	AVG	ra
TCGA	METH	RUN_vanill	vanillix	4096	2	2,7	64	256	300	7E-05		0,88	4	88	23	420	337	873	12.468.182	7,10	3.060	0,75	1979		1979	1979	0,42	1590		1590	0,52	0,996	0,979	0,968	0,979	0,835	0,917	0,850	0,766	0,811	0,877	1
TCGA	METH	RUN_vanill	vanillix	4096	2	1,9	64	256	300	1E-05		0,28	4	89	23	419	335	870	18.121.536	7,26	4.418	1,08	1412		1412	1412	0,59	1276		1276	0,62	0,995	0,969	0,951	0,983	0,844	0,911	0,840	0,746	0,837	0,876	2
TCGA	METH	RUN_vanill	vanillix	4096	2	1,7	64	256	300	5E-05		0,73	5	108	28	558	451	1149	20.627.662	7,31	5.019	1,23	1522		1522	1522	0,55	1318		1318	0,61	0,991	0,975	0,970	0,985	0,849	0,892	0,830	0,768	0,829	0,876	3
TCGA	METH	RUN_vanill	vanillix	4096	2	3,1	64	256	300	0,003		0,89	4	84	23	416	334	862	11.036.110	7,04	2.716	0,66	2059		2059	2059	0,40	1608		1608	0,51	0,996	0,977	0,950	0,966	0,853	0,907	0,852	0,743	0,826	0,874	4
TCGA	METH	RUN_vanill	vanillix	4096	2	1,5	64	256	300	0,0208		0,16	5	103	23	418	340	889	22.842.378	7,36	5.552	1,36	1382		1382	1382	0,59	1246		1246	0,63	0,993	0,976	0,978	0,970	0,851	0,895	0,841	0,763	0,816	0,874	5
TCGA	METH	RUN_vanill	vanillix	4096	2	1,1	64	128	300	3E-05		0,38	4	155	23	409	330	921	31.151.726	7,49	7.547	1,84	1204		1204	1204	0,64	1132		1132	0,66	0,997	0,987	0,980	0,973	0,822	0,905	0,843	0,764	0,822	0,874	6
TCGA	RNA	RUN_vanill	vanillix	1024	2	2,9	64	256	300	3E-05		0,58	7	64	16	420	325	832	766.970	5,88	768	0,75	565		565	565	0,41	442		442	0,50	0,995	0,972	0,981	0,905	0,849	0,897	0,839	0,742	0,898	0,873	1
TCGA	METH	RUN_varix_	varix	4096	2	3,2	64	64	300	1E-05	0,003	0,40	4	1781	24	427	343	2579	10.902.834	7,04	2.663	0,65	1528	9	1538	1538	0,55	1335	8	1343	0,60	0,995	0,984	0,969	0,973	0,829	0,898	0,829	0,762	0,827	0,872	1
TCGA	RNA	RUN_vanill	vanillix	2048	2	2,8	64	256	300	0,0006		0,69	7	72	18	415	317	829	3.077.322	6,49	1.519	0,74	1018		1018	1018	0,47	669		669	0,56	0,996	0,985	0,982	0,910	0,849	0,910	0,837	0,747	0,860	0,872	2
TCGA	METH	RUN_vanill	vanillix	4096	2	3,0	64	128	300	2E-05		0,72	4	138	23	426	338	928	11.294.216	7,05	2.776	0,68	1701		1701	1701	0,50	1395		1395	0,58	0,995	0,985	0,975	0,973	0,849	0,902	0,864	0,741	0,781	0,871	7
TCGA	RNA	RUN_vanill	vanillix	2048	2	1,8	64	256	300	0,0002		0,85	7	74	24	423	317	846	4.824.312	6,68	2.345	1,14	1168		1168	1168	0,40	791		791	0,48	0,997	0,982	0,979	0,909	0,835	0,895	0,832	0,762	0,868	0,870	3
TCGA	RNA	RUN_vanill	vanillix	4096	2	1,1	64	128	300	0,0056		0,83	7	149	23	413	320	911	31.284.942	7,50	7.580	1,85	2260		2260	2260	0,43	1750		1750	0,53	0,993	0,973	0,954	0,911	0,870	0,920	0,847	0,760	0,802	0,869	4
TCGA	RNA	RUN_vanill	vanillix	2048	2	2,0	64	256	300	8E-05		0,87	7	73	18	413	319	829	4.375.932	6,64	2.132	1,04	1274		1274	1274	0,35	804		804	0,45	0,990	0,979	0,971	0,904	0,840	0,899	0,812	0,750	0,897	0,869	5
TCGA	RNA	RUN_vanill	vanillix	4096	3	2,0	64	256	300	0,0002		0,60	7	97	23	425	326	879	21.444.624	7,33	6.297	1,54	1882		1882	1882	0,51	1258		1258	0,57	0,996	0,980	0,981	0,906	0,868	0,895	0,815	0,760	0,848	0,868	6
TCGA	RNA	RUN_vanill	vanillix	4096	2	2,3	64	128	300	0,0143		0,58	7	135	23	419	324	907	15.032.590	7,18	3.674	0,90	2106		2106	2106	0,46	1718		1718	0,53	0,989	0,979	0,971	0,910	0,823	0,907	0,834	0,770	0,845	0,867	7
TCGA	METH	RUN_varix_	varix	4096	2	3,8	64	64	300	9E-05	0,005	0,30	4	293	26	437	344	1104	9.065.424	6,96	2.226	0,54	1324	20	1344	1344	0,61	1171	18	1189	0,64	0,996	0,976	0,968	0,973	0,781	0,903	0,813	0,750	0,842	0,863	2
TCGA	METH	RUN_varix_	varix	4096	2	1,9	64	256	300	1E-05	0,002	0,28	4	97	24	419	336	880	18.260.864	7,26	4.418	1,08	1486	10	1496	1496	0,56	1355	10	1364	0,59	0,995	0,980	0,978	0,971	0,797	0,886	0,830	0,769	0,795	0,862	3
TCGA	METH	RUN_varix_	varix	4096	2	1,1	64	128	300	3E-05	0,001	0,38	4	177	24	419	336	958	31.391.214	7,50	7.547	1,84	1276	11	1287	1287	0,62	1199	10	1210	0,64	0,994	0,983	0,965	0,978	0,825	0,899	0,794	0,756	0,775	0,858	4
TCGA	METH	RUN_varix_	varix	4096	2	3,5	32	32	300	8E-05	0,008	0,12	4	528	24	214	218	988	9.625.200	6,98	2.353	0,57	1342	25	1367	1367	0,60	1217	24	1241	0,63	0,996	0,985	0,976	0,970	0,806	0,896	0,843	0,739	0,765	0,858	5
TCGA	METH	RUN_varix_	varix	4096	3	1,8	64	32	300	0,0006	0,001	0,07	4	683	24	427	341	1479	24.144.290	7,38	7.009	1,71	1071	9	1080	1080	0,65	1371	10	1381	0,62	0,997	0,983	0,986	0,973	0,833	0,889	0,823	0,730	0,767	0,858	6
TCGA	RNA	RUN_varix_	varix	4096	3	3,4	64	128	300	0,0248	0,002	0,08	7	1011	22	416	324	1780	10.890.414	7,04	3.209	0,78	2195	6	2201	2201	0,45	1845	5	1850	0,48	0,996	0,981	0,976	0,890	0,836	0,888	0,830	0,726	0,832	0,855	1
TCGA	METH	RUN_varix_	varix	4096	2	2,0	64	64	300	0,0005	0,004	0,53	5	3320	24	414	336	4099	17.413.474	7,24	4.215	1,03	1335	21	1356	1356	0,61	1170	18	1189	0,64	0,997	0,985	0,973	0,968	0,810	0,895	0,811	0,756	0,759	0,855	7
TCGA	RNA	RUN_varix_	varix	2048	2	2,9	64	32	300	0,0003	0,017	0,29	7	500	22	421	318	1267	3.029.446	6,48	1.476	0,72	812	26	838	838	0,57	696	20	716	0,62	0,993	0,974	0,963	0,892	0,813	0,886	0,837	0,745	0,828	0,854	2
TCGA	RNA	RUN_varix_	varix	4096	2	1,1	64	128	300	0,0056	0,003	0,83	8	195	25	424	326	977	31.525.454	7,50	7.580	1,85	2694	15	2709	2709	0,32	1956	12	1967	0,44	0,993	0,972	0,968	0,903	0,852	0,896	0,818	0,732	0,781	0,852	3
TCGA	RNA	RUN_varix_	varix	4096	2	1,0	64	32	300	9E-05	0,002	0,35	7	573	22	416	318	1336	33.975.334	7,53	8.163	1,99	1144	21	1164	1164	0,68	1328	18	1346	0,67	0,997	0,985	0,984	0,911	0,828	0,903	0,835	0,742	0,745	0,850	4
TCGA	RNA	RUN_varix_	varix	4096	3	3,9	64	256	300	3E-05	0,003	0,08	7	101	24	418	322	872	9.349.050	6,97	2.741	0,67	1700	5	1705	1705	0,56	1293	4	1297	0,58	0,995	0,986	0,981	0,875	0,822	0,905	0,828	0,708	0,819	0,849	5
TCGA	RNA	RUN_varix_	varix	4096	2	3,2	64	64	300	1E-05	0,003	0,40	7	271	23	421	322	1044	10.902.834	7,04	2.663	0,65	1896	9	1905	1905	0,52	1625	8	1633	0,57	0,995	0,984	0,976	0,901	0,839	0,886	0,839	0,741	0,751	0,849	6
TCGA	RNA	RUN_varix_	varix	4096	2	3,8	64	128	300	0,0004	0,004	0,81	7	152	23	412	319	913	9.048.644	6,96	2.221	0,54	2293	14	2308	2308	0,42	1748	12	1760	0,52	0,995	0,980	0,982	0,894	0,824	0,904	0,814	0,742	0,774	0,848	7
TCGA	DNA	RUN_vanill	vanillix	4096	2	3,8	64	32	300	0,0021		0,29	7	421	25	419	325	1196	9.029.544	6,96	2.233	0,55	1279		1279	1279	0,52	1481		1481	0,59	0,891	0,888	0,845	0,724	0,677	0,696	0,684	0,662	0,735	0,722	1
TCGA	DNA	RUN_vanill	vanillix	4096	3	1,8	64	32	300	0,0006		0,07	8	561	31	434	327	1361	24.065.634	7,38	7.009	1,71	691		691	691	0,65	1839		1839	0,53	0,897	0,879	0,849	0,736	0,707	0,644	0,705	0,660	0,726	0,722	2
TCGA	DNA	RUN_vanill	vanillix	2048	2	2,0	64	256	300	2E-05		0,08	7	69	17	424	318	834	4.426.692	6,65	2.157	1,05	772		772	772	0,51	623		623	0,56	0,869	0,863	0,827	0,711	0,643	0,749	0,652	0,649	0,794	0,722	3
TCGA	DNA	RUN_varix_	varix	4096	2	3,8	64	32	300	0,0021	0,006	0,29	7	500	21	424	319	1272	9.098.984	6,96	2.233	0,55	1457	20	1477	1477	0,47	1632	17	1650	0,56	0,871	0,875	0,816	0,708	0,675	0,693	0,657	0,615	0,838	0,720	1
TCGA	DNA	RUN_varix_	varix	4096	3	3,2	64	64	300	3E-05	0,008	0,09	7	306	27	450	322	1113	11.568.366	7,06	3.414	0,83	1593	15	1609	1609	0,44	1812	13	1826	0,52	0,882	0,867	0,831	0,688	0,706	0,611	0,673	0,603	0,871	0,716	2
TCGA	DNA	RUN_vanill	vanillix	4096	2	3,5	64	128	300	0,0414		0,00	7	126	23	414	317	887	9.887.122	7,00	2.438	0,60	1721		1721	1721	0,49	2032		2032	0,51	0,850	0,860	0,829	0,706	0,692	0,726	0,723	0,645	0,674	0,716	4
TCGA	DNA	RUN_vanill	vanillix	1024	2	1,1	32	256	300	0,0002		0,21	7	65	15	219	207	513	1.926.318	6,28	1.851	1,81	310		310	310	0,54	336		336	0,57	0,841	0,831	0,813	0,685	0,652	0,720	0,676	0,619	0,828	0,716	5
TCGA	DNA	RUN_vanill	vanillix	2048	3	1,7	64	128	300	3E-05		0,09	8	146	69	435	313	970	7.024.462	6,85	4.034	1,97	677		677	677	0,51	857		857	0,55	0,880	0,855	0,817	0,723	0,661	0,643	0,656	0,651	0,814	0,714	6
TCGA	DNA	RUN_vanill	vanillix	2048	3	3,6	64	256	300	0,0065		0,19	7	68	16	421	320	832	2.379.372	6,38	1.188	0,58	585	58	585	585	0,58	668		668	0,59	0,897	0,888	0,838	0,742	0,665	0,674	0,639	0,624	0,779	0,714	7
TCGA	DNA	RUN_varix_	varix	4096	2	1,9	64	256	300	1E-05	0,002	0,28	7	86	21	419	318	851	18.260.864	7,26	4.418	1,08	1843	7	1850	1850	0,29	1540	6	1545	0,45	0,855	0,854	0,826	0,698	0,711	0,648	0,679	0,600	0,747	0,704	3
TCGA	DNA	RUN_varix_	varix	1024	2	1,5	64	64	300	4E-05	0,007	0,07	7	248	15	430	316	1016	1.521.694	6,18	1.419	1,39	415	11	427	427	0,47	453	8	461	0,53	0,862	0,867	0,836	0,694	0,666	0,684	0,680	0,607	0,703	0,698	4
TCGA	DNA	RUN_varix_	varix	4096	2	2,7	32	256	300	6E-05	0,020	0,10	7	82	21	218	209	537	12.436.866	7,09	3.030	0,74	1370	38	1408	1408	0,47	1278	24	1302	0,50	0,874	0,856	0,834	0,683	0,663	0,683	0,655	0,606	0,732	0,697	5
TCGA	DNA	RUN_varix_	varix	2048	3	2,8	32	32	300	7E-05	0,002	0,21	7	587	17	220	217	1049	3.379.542	6,53	2.001	0,98	1035	5	1040	1040	0,40	904	4	908	0,49	0,841	0,841	0,792	0,687	0,640	0,681	0,660	0,604	0,776	0,696	6
TCGA	DNA	RUN_varix_	varix	1024	3	3,6	32	128	300	0,0004	0,005	0,07	7	158	15	216	204	600	636.684	5,80	760	0,74	464	5	469	469	0,45	485	3	488	0,50	0,846	0,835	0,788	0,676	0,677	0,604	0,651	0,601	0,831	0,695	7

Figure 2: Best 7 results (highest avg downstream performance) for TCGA for each modality and AE architecture from 18,000 Autoencodix runs in total.



dat	mod	run_id	MOD	K_F	EN	LA	BA	E	LR_F	BETA	DROP_P	data_ru	training	predicti	visualisi	ml_task	total_ru	trainable_	log	capacit	cap	trai	train_	tra	train	vali	val	valid_r2	author_	age_gro	sex	AVG_p5	rank
SCHC	RNA	RUN_vanil	vanilix	4096	2	1,7	64	256	300	5E-05		0,73	4	3186	29	150	171	3540	20.627.662	7,31	5019	1,2	3037		3037	0,24	2859		2859	0,25	0,998	0,999	0,998	0,998	1
SCHC	RNA	RUN_vanil	vanilix	2048	2	3,5	64	32	300	6E-05		0,27	4	986	19	147	162	1317	2.493.582	6,40	1244	0,6	1268		1268	0,37	1225		1225	0,39	0,996	0,998	0,999	0,998	2
SCHC	RNA	RUN_vanil	vanilix	4096	2	3,2	64	64	300	1E-05		0,40	4	545	27	141	167	884	10.819.634	7,03	2663	0,7	3050		3050	0,25	3005		3005	0,25	0,997	0,999	0,998	0,998	3
SCHC	RNA	RUN_vanil	vanilix	2048	2	2,0	64	64	300	0,001		0,59	4	511	22	144	160	840	4.232.112	6,63	2065	1,0	1306		1306	0,35	1250		1250	0,37	0,997	0,998	0,998	0,998	4
SCHC	RNA	RUN_vanil	vanilix	2048	2	2,2	64	64	300	1E-05		0,36	4	522	19	142	163	848	3.897.942	6,59	1906	0,9	1283		1283	0,36	1238		1238	0,38	0,997	0,998	0,997	0,997	5
SCHC	RNA	RUN_vanil	vanilix	2048	2	1,2	64	64	300	1E-05		0,37	4	519	19	143	159	842	7.362.312	6,87	3544	1,7	1257		1257	0,38	1227		1227	0,39	0,996	0,998	0,998	0,997	6
SCHC	RNA	RUN_vanil	vanilix	4096	2	1,1	64	128	300	3E-05		0,38	4	319	26	143	164	656	31.151.726	7,49	7547	1,8	2847		2847	0,29	2974		2974	0,26	0,995	0,999	0,998	0,997	7
SCHC	RNA	RUN_vanil	vanilix	4096	2	1,9	64	256	300	1E-05		0,28	4	173	26	146	178	527	18.121.536	7,26	4418	1,1	2994		2994	0,25	2846		2846	0,25	0,996	0,998	0,997	0,997	8
SCHC	RNA	RUN_vanil	vanilix	1024	2	3,8	64	64	300	3E-05		0,28	3	499	17	141	158	819	588.046	5,77	603	0,6	545		545	0,45	507		507	0,49	0,997	0,995	0,999	0,997	9
SCHC	RNA	RUN_vanil	vanilix	4096	2	3,8	64	64	300	9E-05		0,30	4	532	26	147	164	873	8.996.240	6,95	2226	0,5	2941		2941	0,27	2980		2980	0,26	0,997	0,999	0,996	0,997	10
SCHC	RNA	RUN_varix	varix	4096	2	1,1	64	128	300	3E-05	0,001		4	369	29	145	166	714	31.391.214	7,50	7547	1,8	2924	9	2933	0,27	2996	9	3005	0,26	0,994	0,997	0,997	0,996	1
SCHC	RNA	RUN_varix	varix	1024	2	1,5	64	32	300	0,001	0,003	0,31	3	1109	17	142	156	1427	1.541.908	6,19	1438	1,4	510	6	516	0,49	484	5	490	0,51	0,996	0,998	0,998	0,996	2
SCHC	RNA	RUN_varix	varix	4096	2	3,8	64	64	300	9E-05	0,005	0,30	4	636	28	143	172	984	9.065.424	6,96	2226	0,5	2965	15	2980	0,27	3009	13	3022	0,25	0,995	0,998	0,993	0,995	3
SCHC	RNA	RUN_varix	varix	2048	2	2,0	64	256	300	2E-04	0,005	0,33	4	180	19	142	160	505	4.480.818	6,65	2152	1,1	1219	11	1230	0,39	1168	9	1178	0,38	0,994	0,997	0,994	0,995	4
SCHC	RNA	RUN_varix	varix	2048	2	2,9	64	32	300	3E-04	0,017	0,29	4	1135	21	146	157	1462	3.029.446	6,48	1476	0,7	1274	16	1289	0,37	1234	14	1248	0,38	0,997	0,996	0,993	0,995	5
SCHC	RNA	RUN_varix	varix	1024	2	2,3	64	128	300	7E-05	0,010	0,16	3	305	16	143	158	626	1.014.098	6,01	968	0,9	504	10	514	0,49	492	9	501	0,51	0,995	0,992	0,997	0,995	6
SCHC	RNA	RUN_varix	varix	4096	2	1,9	64	256	300	1E-05	0,002	0,28	4	193	26	141	162	526	18.260.864	7,26	4418	1,1	3048	8	3056	0,24	2891	8	2899	0,24	0,997	0,997	0,990	0,995	7
SCHC	RNA	RUN_varix	varix	2048	2	1,2	64	64	300	1E-05	0,002	0,37	4	608	20	142	158	931	7.473.736	6,87	3544	1,7	1304	7	1311	0,35	1255	7	1261	0,37	0,995	0,995	0,993	0,994	8
SCHC	RNA	RUN_varix	varix	1024	2	1,5	64	64	300	4E-05	0,007	0,07	3	873	53	149	161	1239	1.521.694	6,18	1419	1,4	477	11	488	0,52	477	11	488	0,52	0,994	0,992	0,997	0,994	9
SCHC	RNA	RUN_varix	varix	4096	2	3,2	64	64	300	1E-05	0,003	0,40	4	558	27	147	180	915	10.902.834	7,04	2663	0,7	3099	8	3107	0,23	3044	8	3052	0,24	0,994	0,997	0,990	0,994	10
SCHC	METH	RUN_vanil	vanilix	4096	2	1,7	64	256	300	5E-05		0,73	9	322	36	203	255	825	20.627.662	7,31	5019	1,2	3372		3372	0,15	3220		3220	0,15	0,995	0,954	0,899	0,949	1
SCHC	METH	RUN_vanil	vanilix	4096	2	3,1	64	128	300	0,014		0,46	6	522	35	201	261	1026	11.152.674	7,05	2742	0,7	3533		3533	0,11	3392		3392	0,13	0,995	0,952	0,898	0,948	2
SCHC	METH	RUN_vanil	vanilix	4096	2	2,5	64	256	300	1E-04		0,75	7	295	35	200	256	794	13.667.126	7,14	3347	0,8	3425		3425	0,13	3241		3241	0,14	0,995	0,954	0,893	0,947	3
SCHC	METH	RUN_vanil	vanilix	4096	2	3,0	64	128	300	2E-05		0,72	8	535	36	202	258	1039	11.294.216	7,05	2776	0,7	3457		3457	0,13	3354		3354	0,14	0,994	0,952	0,895	0,947	4
SCHC	METH	RUN_vanil	vanilix	4096	2	3,6	64	64	300	3E-05		0,71	8	1109	44	267	339	1767	9.545.756	6,98	2357	0,6	3469		3469	0,13	3385		3385	0,14	0,995	0,950	0,894	0,947	5
SCHC	METH	RUN_vanil	vanilix	4096	2	2,0	64	32	300	0,01		0,17	15	2060	36	207	244	2563	16.939.244	7,23	4133	1,0	3480		3480	0,12	3418		3418	0,14	0,995	0,948	0,890	0,944	6
SCHC	METH	RUN_vanil	vanilix	4096	2	1,9	64	256	300	1E-05		0,28	11	307	42	215	254	829	18.121.536	7,26	4418	1,1	3300		3300	0,16	3187		3187	0,16	0,995	0,949	0,886	0,943	7
SCHC	METH	RUN_vanil	vanilix	4096	2	3,2	64	64	300	1E-05		0,40	8	988	35	198	257	1487	18.019.634	7,03	2663	0,7	3347		3347	0,15	3341		3341	0,16	0,995	0,950	0,884	0,943	8
SCHC	METH	RUN_varix	varix	4096	2	3,8	64	64	300	9E-05	0,005	0,30	7	1157	39	202	263	1667	9.065.424	6,96	2226	0,5	3324	15	3339	0,16	3375	14	3389	0,15	0,992	0,950	0,887	0,943	1
SCHC	METH	RUN_varix	varix	4096	2	2,9	64	32	300	2E-05		0,14	7	1976	38	214	268	2502	11.918.666	7,08	2926	0,7	3246		3246	0,18	3339		3339	0,16	0,995	0,950	0,882	0,942	9
SCHC	METH	RUN_varix	varix	4096	3	3,2	64	64	300	3E-05		0,09	7	1228	36	204	262	1736	11.542.830	7,06	3414	0,8	3256		3256	0,18	3365		3365	0,15	0,994	0,946	0,884	0,942	10
SCHC	METH	RUN_varix	varix	4096	2	2,0	64	64	300	5E-04	0,004	0,53	7	1186	37	204	267	1701	17.413.474	7,24	4215	1,0	3363	12	3375	0,15	3392	11	3402	0,14	0,991	0,948	0,881	0,940	2
SCHC	METH	RUN_varix	varix	4096	2	1,1	64	128	300	3E-05	0,001	0,38	6	672	35	208	262	1183	31.391.214	7,50	7547	1,8	3251	10	3261	0,18	3311	9	3320	0,15	0,992	0,948	0,877	0,939	3
SCHC	METH	RUN_varix	varix	4096	2	3,2	64	64	300	1E-05	0,003	0,40	7	1171	36	209	256	1679	10.902.834	7,04	2663	0,7	3387	9	3396	0,14	3360	9	3368	0,15	0,995	0,944	0,874	0,938	4
SCHC	METH	RUN_varix	varix	4096	2	1,0	64	32	300	9E-05	0,002	0,35	7	2502	39	205	256	3008	33.975.334	7,53	8163	2,0	3195	13	3208	0,19	3421	12	3434	0,14	0,993	0,943	0,869	0,935	5
SCHC	METH	RUN_varix	varix	4096	2	1,9	64	256	300	4E-05	0,002	0,28	6	338	35	200	268	847	18.260.864	7,26	4418	1,1	3363	9	3372	0,15	3231	8	3239	0,15	0,994	0,943	0,859	0,932	6
SCHC	METH	RUN_varix	varix	4096	3	3,2	64	64	300	3E-05	0,008	0,09	8	1415	34	202	254	1914	11.568.366	7,06	3414	0,8	3318	14	3332	0,16	3381	12	3393	0,14	0,992	0,944	0,859	0,932	7
SCHC	METH	RUN_varix	varix	4096	3	3,9	64	256	300	3E-05	0,003	0,08	7	391	36	202	263	898	9.349.050	6,97	2741	0,7	3345	8	3352	0,15	3243	7	3249	0,14	0,991	0,936	0,866	0,931	8
SCHC	METH	RUN_varix	varix	4096	2	3,5	32	32	300	3E-05	0,008	0,12	7	2246	36	114	161	2564	9.625.200	6,98	2353	0,6	3349	15	3364	0,15	3400	14	3414	0,14	0,992	0,938	0,861	0,930	9
SCHC	METH	RUN_varix	varix	4096	3	2,0	64	256	300	2E-04	0,003	0,60	7	419	35	200	261	922	21.511.632	7,33	6297	1,5	3645	4	3648	0,08	3300	3	3303	0,13	0,987	0,934	0,865	0,929	10

Figure 3: Best 10 results (highest avg downstream performance) for SCHC for each modality and AE architecture from 12,000 Autoencodix runs in total.

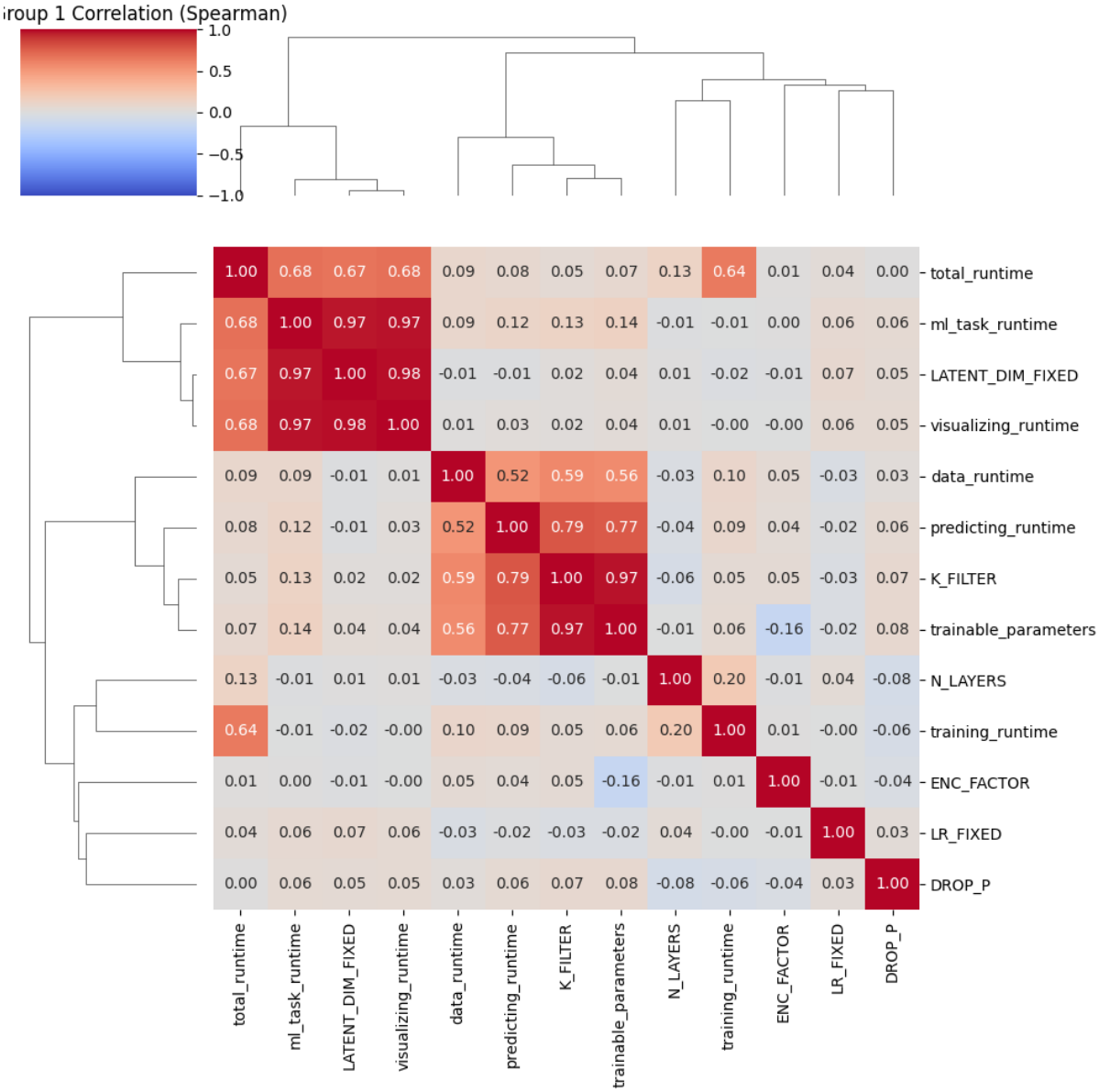


Figure 4: Correlation (Spearman) of **hyperparameter variables** (uppercase) and **run times** (scenario: TCGA RNA) (n=500)

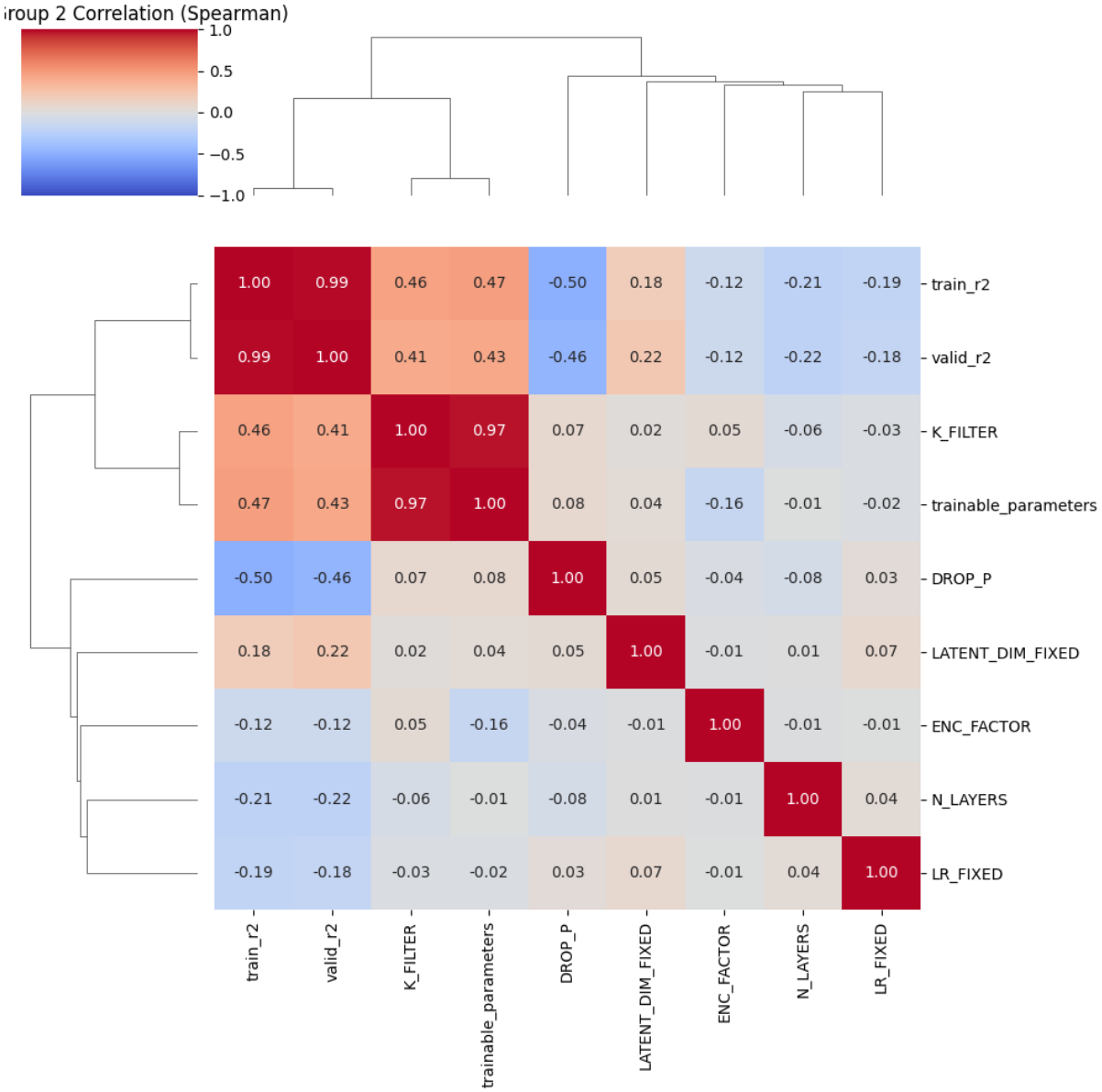


Figure 5: Correlation (Spearman) of **hyperparameter variables** (uppercase) and **losses** (scenario: TCGA RNA) (n=500)

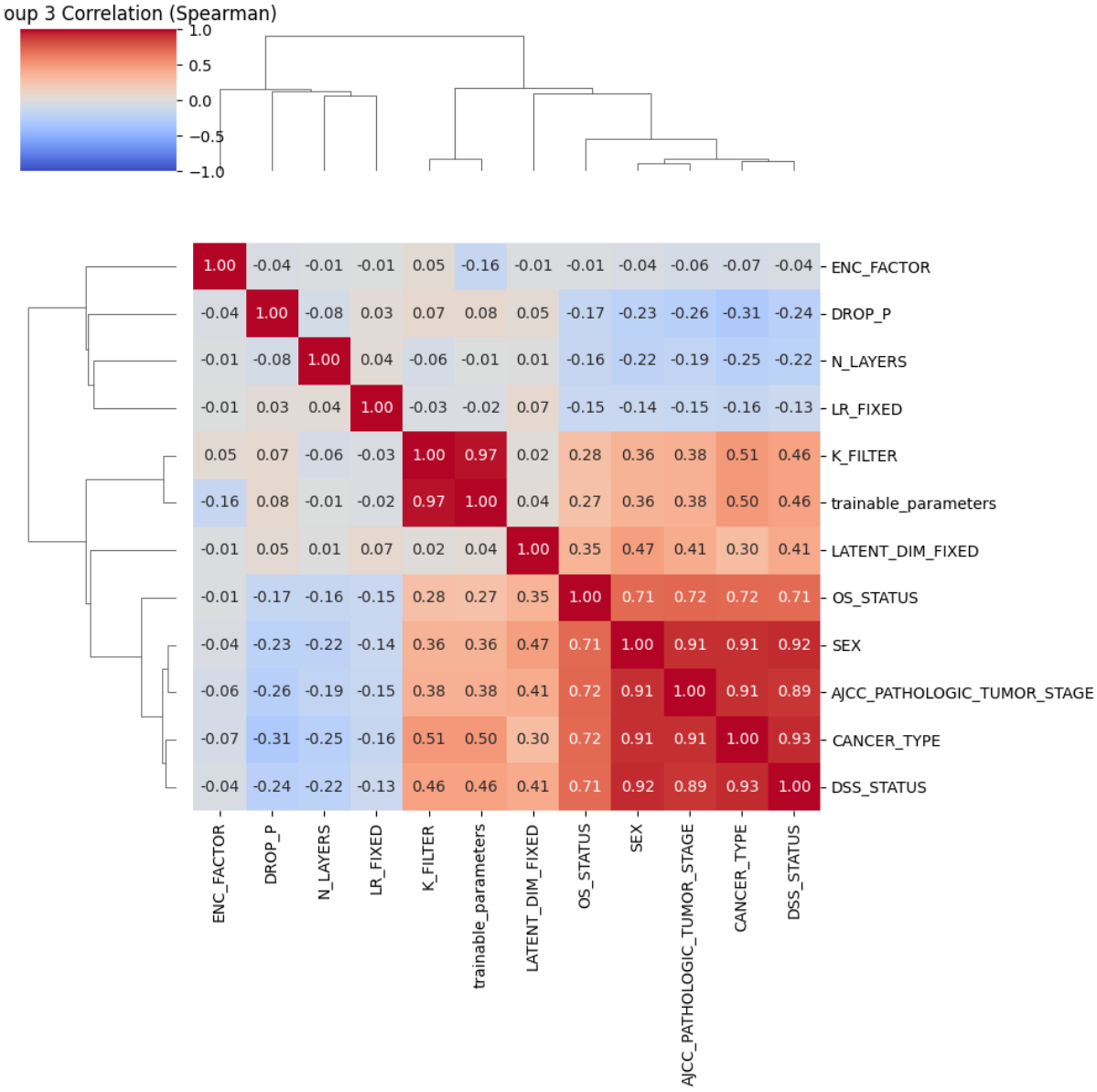


Figure 6: Correlation (Spearman) of **hyperparameter variables** (uppercase) and **downstream performance (AUC)** (scenario: TCGA RNA) (n=500)

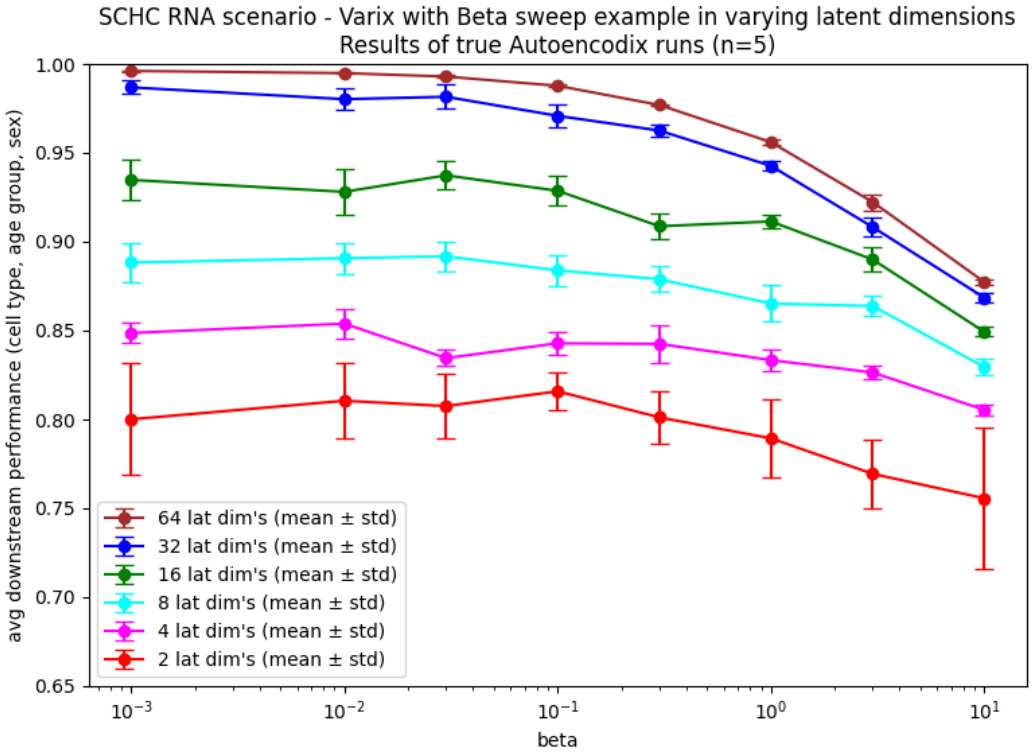


Figure 7: SCHC RNA in Varix - Beta Sweep on multiple latent dimensions d_z. Results of true Autoencodix runs (mean and standard deviation of downstream task performance) vary over ranges of d_z and β. Smaller values of β (edge optimum) are typically better in large d_z (32, 64), but can go to middle value for $d_z = 4, 8$ or 16. β= 0.1 looks best here for $d_z = 2$ (center optimum with low variance). Autoencodix was run 5 times with equal HP configuration to get each data point and small distribution.

tuned by Syne Tune on Vanillix SCHC RNA). We measured true Autoencodix results on this beta sweep. Autoencodix was run 5 times with equal HP configuration to get each data point and a small distribution.

We measured AUC ROC OVO for all three clinical variables (author cell type, age group, sex) and their average. The result is shown in Fig. 7.

Downstream performance degrades with less latent dimensions. Best downstream results with small beta, but not always smallest beta. Beta around 0.1 can be better and also have less variance. Beta at 1 or higher degrades performance in most cases.

The influence on latent space structure was also evaluated along UMAP visualizations (despite their known limitations). UMAP visualizations of well structured latent spaces for $d_z = 64, 8,$ and 2, are shown in Fig. 8.

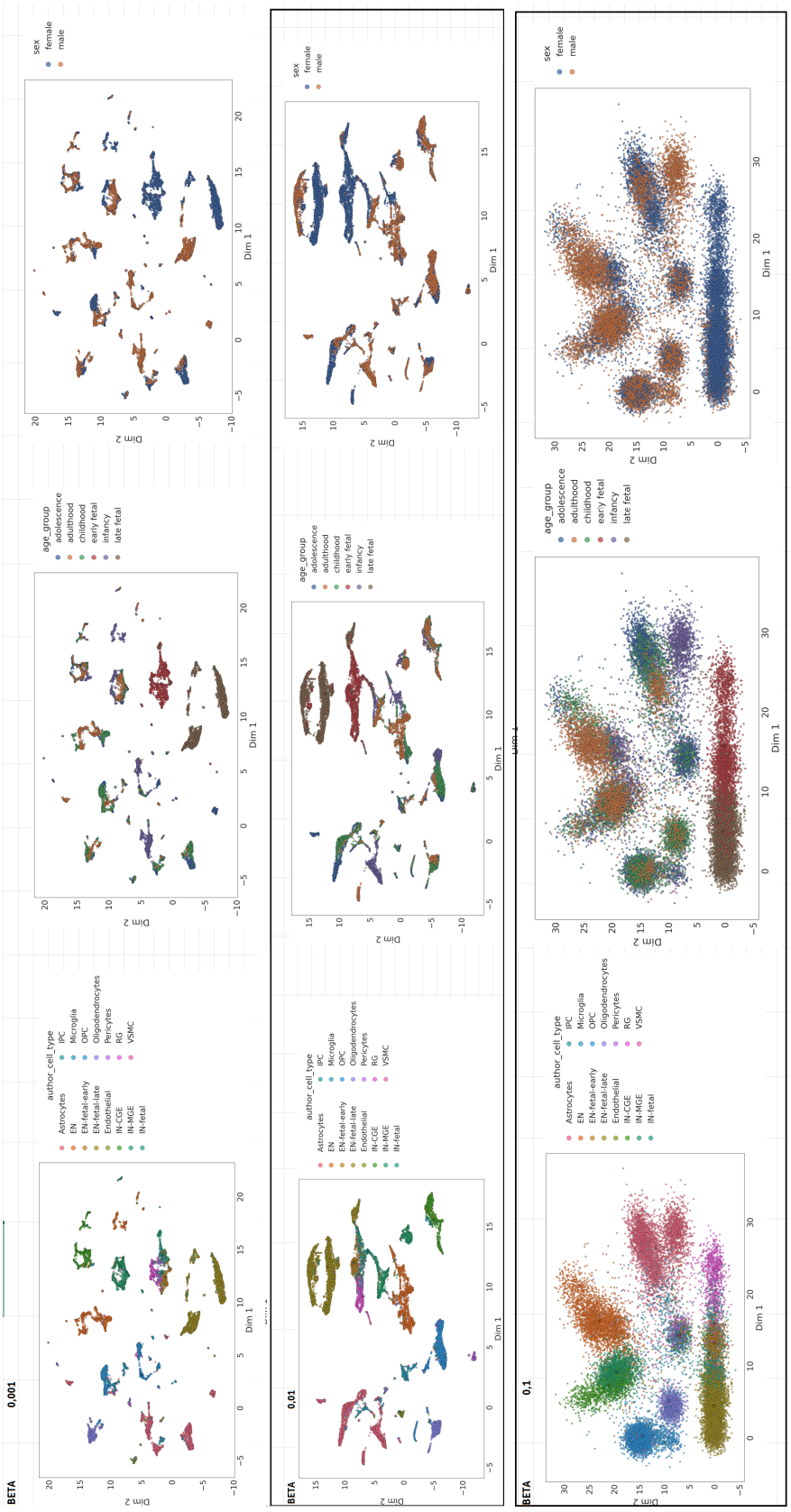


Figure 8: UMAP projections for well structured latent spaces at:
($d_z = 64, \beta = 0.001$, avg ds perf=0.9959), ($d_z = 8, \beta = 0.01$, avg ds perf=0.8931),
($d_z = 2, \beta = 0.1$, avg ds perf=0.8189) (top, middle, bottom)

4.3 Warmstart HPO on the same task

We started CQR, BORE, REA, KDE (a kernel density estimator, similar to TPE) an all 5 tasks for Vanillix to come up with new HP combinations with higher predicted downstream performance. Each algorithm was given the surrogate model that was trained on 1000 Autoencodix HP combinations.

	2023 CQR	2021 BORE	2019 REA	2011 KDE
pred. training runtime	61.7	516.3	300.6	496.5
pred. valid R^2	0.641	0.617	0.502	0.592
pred. wgtd avg ds perf	0.8799(+.0066)	0.8800 (+.0067)	**0.8822 (+.0089)**	0.8814 (+.0081)
D	2048	2048	1024	4096
n_{layers}	2	3	4	3
r	2.85	3.37	3.09	3.64
d_z	64	64	64	64
bs	256	32	64	32
lr (fix)	9.71-04	3.59e-05	1.32e-03	1.74e-04
p_{drop}	0.348	0.015	0.231	0.326
epochs	300	300	300	300

Table 10: Best predicted HP combinations by 4 searchers in Syne Tune, **TCGA RNA** with **Vanillix**, (all: 8 h simulated wall-clock time). For reference: best average downstream performance found in previous 3000 HP (random sampled) combinations was: **0.8733**.

	2023 CQR	2019 REA
pred. training runtime	133.3	84.5
pred. valid R^2	0.68	0.62
pred. wgtd avg ds perf	**0.8914 (+.0144)**	0.8889 (+.0119)
D	4096	4096
n_{layers}	2	2
r	1.66	1.73
d_z	64	64
bs	256	256
lr (fix)	1.49e-05	3.88e-05
p_{drop}	0.69	0.52
epochs	300	300

Table 11: Best predicted HP combinations by 2 searchers in Syne Tune, **TCGA METH Vanillix**, (all: 4 h simulated wall-clock time). For reference: best average downstream performance found in 3000 HP combinations was: **0.8770**.

	2023 CQR	2019 REA
pred. training runtime	80.8	129.5
pred. valid R^2	0.45	0.55
pred. wgtd avg ds perf	0.7140 (-.0077)	**0.7337 (+.0120)**
D	2048	2048
n_{layers}	2	2
r	2.88	1.73
d_z	32	32
bs	256	128
lr (fix)	1.63e-05	2.76e-05
p_{drop}	0.24	0.27
epochs	300	300

Table 12: Best predicted HP combinations by 2 searchers in Syne Tune, **TCGA DNA Vanillix**, (all: 4 h simulated wall-clock time). For reference: best average downstream performance found in 3000 HP combinations was: **0.7217**.

	2023 CQR	2019 REA
pred. training runtime	179.1	138.0
pred. valid R^2	0.37	0.39
pred. wgtd avg ds perf	1.0102 (+0.0122)	**1.0122 (+0.0142)**
D	2048	2048
n_{layers}	2	2
r	1.89	3.78
d_z	64	64
bs	256	256
lr (fix)	1.04e-04	1.57e-04
p_{drop}	0.0017	0.40
epochs	300	300

Table 13: Best predicted HP combinations by 2 searchers in Syne Tune, **SCHC RNA Vanillix**, (all: 4 h simulated wall-clock time). For reference: best average downstream performance found in 3000 HP combinations was: **0.9980**.

	2023 CQR	2019 REA
pred. training runtime	1935.6	622.4
pred. valid R^2	0.15	0.15
pred. wgtd avg ds perf	0.9463 (-.0029)	**0.9551 (+.0059)**
D	4096	4096
n_{layers}	2	3
r	2.61	2.27
d_z	64	64
bs	32	128
lr (fix)	1.35e-05	2.24e-04
p_{drop}	0.25	0.12
epochs	300	300

Table 14: Best predicted HP combinations by 2 searchers in Syne Tune, **SCHC METH Vanillix**, (all: 4 h simulated wall-clock time). For reference: best average downstream performance found in 3000 HP combinations was: **0.9492**.

4.4 Warmstart HPO on new tasks (transfer learning)

Table 15 shows results on transfer learning. Here, the other 4 tasks have been used with transfer evaluations. Predicted improvements on downstream performance are given in the table.

	TCGA RNA	TCGA METH	TCGA DNA	SCHC RNA	SCHC METH
pred. training runtime	486.2	73.9	502.5	296.8	2132.8
pred. valid R^2	0.58	0.67	0.47	0.22	0.17
pred. wgtd avg ds perf	0.9028 (+.0295)	0.8966 (+0.0196)	0.7337 (+.0120)	1.0249 (+0.0269)	0.9559 (+0.0067)
D	4096	4096	2048	4096	4096
n_{layers} 2	2	3	2	2	
r	1.22	1.83	1.73	1.71	1.65
d_z	64	64	32	64	64
bs	32	256	32	128	32
lr (fix)	2.17e-04	2.03e-04	1.42-05	1.60e-03	4.05e-04
p_{drop}	0.60	0.55	0.25	0.85	0.55
epochs	300	300	300	300	300

Table 15: Best predicted HP combinations by REA searcher in Syne Tune, all 5 datasets on **Vanillix**, (all: Transfer Learning (100 loops, 4 h wall-clock time each). References on best average downstream performance found in previous 3000 HP (random sampled) were listed in the tables above.

4.5 General remarks on implementation and used hardware

AutoEncondix was checked out from Github, then virtual environment created, dependencies installed. It was mainly run on an GPU cluster at TU Dresden for deep neural network training.

GPU cluster specifics for the runs of Autoencodix:
GPU Compute Cluster at TU Dresden: Alpha Centauri
CPU: AMD EPYC 7352, 12 cores reserved per task
RAM: 16 GB reserved per task
GPU model: Nvidia A100-SXM4, 40 GB
OS: Linux

Syne Tune was installed on a desktop PC in a conda environment, results from Autoencodix runs imported, new "blackboxes" created (this is the internal name in Syne Tune for a data structure, that encapsulates all necessary information to run an HPO algorithm on a data set) and then different HPO algorithms run.

For training the XGBoost surrogate model and running HPO in Syne Tune on the surrogate models, CPU-based computation was mainly used, but a GPU was also available:
CPU: Intel(R) Core(TM) i5-10500 CPU @ 3.10 GHz
GPU: Nvidia RTX 4000 Ada SFF edition (75W TDP), 20 GB VRAM
32 GB of RAM (DDR4)
SSD storage
OS: Windows 11

Glue code to combine these two software packages was implemented in Python code - to write configurations for Autoencodix, gather results from files, run data analysis, run HPO methods in Syne Tune, create plots.

5 Discussion

This thesis deals with questions, how hyperparameter selection and tuning can be improved.

We documented the configuration options of Autoencodix and default values, did a random sampling of HP combinations with value ranges in configuration space.

With these results, we can continue to heuristics: What are promising regions or values? What is data set specific? What is common across data sets?

The expectation then is, that Automated HPO with BO and evolutionary methods will later formalize such heuristics (findings from observations) in its model. And learn with more and more data, keeping track of the many interdependencies itself.

Important: Any AutoHPO mechanism will have no access to results from the test set during optimization for the current run to be optimized. But in training with updated or extended training data, then previous results from the test set or former phases could lead the way in validation phase.

Essentially, we need links from predictors available during training and validation:

- architecture properties (model capacity, trainable parameters, number of linear regions in DNN with ReLU)
- training error, train R^2, validation error, validation R^2

to

- (target) downstream performance (which is very relevant metric in the application). Either as a single-objective metric or multi-objective.

Validation R^2 and downstream performance correlate loosely, but optimizing for validation R^2 alone will often degrade downstream performance. Better: take an acceptable lower threshold value for validation R^2 and continue training as long as this threshold holds. Such thresholds are documented below.

5.1 Dataset specific findings on Vanillix

We start with the data set specific findings. In multi-modal datasets, each additional modality shall bring additional insight. For this, it can either

- support findings/hypotheses from other modalities, so that confidence in certain pattern rises. In this case, it can also serve as a proxy for other modalities in other data sets, that lack a certain information-rich modality.
- add new information that was not present in other modalities.

If it does not contribute in either of those two ways, it can be regarded as irrelevant. The following tables give a better impression on these modality specific results.

5.1.1 TCGA: Heuristics on HP values and expected downstream performance

TCGA is the harder problem for the autoencoders. It is interesting, that METH has better classification capability on target variable sex (this is different from SCHC, where RNA is the better predictor.) RNA seems the best information for Overall Survival Status. It is also interesting, that DNA mutations fall behind the other two modalities in classifying all downstream clinical variables, including OS and DSS. Finally, it is interesting, that RNA has a wide range in good p_{drop} values, while METH and DNA seem relatively constrained to a small region.

Data set	TCGA RNA	TCGA METH	TCGA DNA
avg ds perf (Top 50):	0.85 – 0.87	0.85 – 0.87	0.70 – 0.72
ONCOTREE code	0.97 – 0.98	0.97 – 0.98	0.78 – 0.84
SEX	0.90	0.97	0.66 – 0.74
Overall Survival Status	0.84 – 0.89	0.80	0.70 – 0.85
Disease Specific Survival Status	0.72 – 0.76	0.75	0.60 – 0.66
p_{drop} - promising region	$1 - 60\%$	$52 - 70\%$	$24 - 26\%$
good model capacity (hidden nodes):	$2000 - 6000$	$2000 - 6000$	$1000 - 5000$
good validation R^2	valid. $R^2 > 0.45$. (Pure optimization for max validation R^2 will degrade downstream performance!)	validation $R^2 > 0{,}5$	validation $R^2 > 0.5$

Table 16: TCGA : Heuristics on downstream clinical AUC values, HP ranges and necessary validation R^2 to get good downstream results.

5.1.2 SCHC: Heuristics on HP values and expected downstream performance

RNA with better results for all three clinical variables. Relatively strong correlation to METH results, but METH always with lower values among these three clinical variables.

A stacked model or an inter-modal model can improve results, especially on more downstream tasks, when new clinical variables are introduced that have a stronger biological tie to ATAC sequencing than to RNA sequencing data.

Data set	SCHC RNA	SCHC METH
avg ds perf (Top 50):	0.995 – 0.997	0.932 – 0.949
author cell type age group sex	0.993 – 0.998 0.994 – 0.999 0.993 – 0.998	0.990 – 0.995 0.937 – 0.954 0.867 – 0.899
p_{drop} - promising region	$0\% - 80\%$	$0 - 75\%$
good model capacity (hidden nodes):	$2000 - 4000$	$4000 - 5000$
good validation R^2	min 0.25 (D=4096) or 0.35 (D=2048) or 0.5 (D=1024)	> 0.10 (D=4096). Pure optimization on validation R^2 leads to worse ds results!

Table 17: SCHC : Heuristics on downstream clinical AUC values, HP ranges and necessary validation R^2 to get top downstream results

5.2 Cross-dataset findings on Vanillix

In the results section, there were also many commonalities among the dataset. These are summarized here. It makes clear, that transfer between TCGA and SCHC also makes sense, despite them being bulk cell vs single cell and despite being from different tissue (TCGA: cancer tissue, SCHC: human cortex tissue).

Across all 5 data sets (TCGA RNA, METH, DNA, SCHC RNA, SCHC METH), knowledge transfer on good hyperparameter regions is possible and promising. Finds better solutions.

Best HP values for highest downstream performance (on Vanillix) have shown the following patterns or heuristics:

Results on Neural network architecture hyperparameters

Input dimension: D config space: choice([128, 256, 512, 1024, 2048, 4096]).

D will typically be set by domain specialists. It can depend on device, protocol, preprocessing (data cleaning). Higher D brings more hidden nodes even with small n_{layers}. To investigate: it can make a difference whether these are most variable features (as done by Autoencodix preprocessing) or principal components (PCs) among the input matrix. With PCs, D could be much lower, but for explainability, it would be needed to go back from PCs to actual input variables.

⇒ Best downstream results at the upper region of the values: **4096 and 2048**.

Number of Encoder/decoder layers: n_{layers} config space: choice([2,3,4]).

Has influence on trainable parameters once D is fixed. Has important influence on number of piecewise linear, see Prince (2023).

⇒ Best downstream results at **2** or rarely **3** – otherwise quality in test degrades, likely due to overfitting.

Encoding factor: $1/r$ config space: uniform(1,4). Has major influence on trainable parameters once D is fixed.

⇒ Best downstream results for **no particular value**. Values seem equally likely despite their influence on model capacity.

Latent dimension width: d_z config space: choice([2,4, 8, 16, 32, 64])

if smaller (D=2 or 4), more layers (3,4) to get more hidden nodes; if larger (32, 64), less layers (2); reason: not get too many hidden nodes (risk of overfitting).

⇒ Best downstream results clearly at **64**. Very few occasions with 32.

hidden nodes: for D=1024, 2048, 4096: at least $D/2$ (equivalent to enc factor = 4), maximum: $2 \cdot D$ (equivalent to enc factor $r = 2$)

number of hidden nodes (=DNN capacity) Prince (2023), minimum capacity here:
-> small D like 128: $2 \cdot D$ as capacity
-> large D like 4096: $0.5 \cdot D$ as capacity
at least 1000 hidden nodes (without input and output layer)

Results on Training algorithm hyperparameters

Learning rate (in AdamW): lr config space: log uniform(1.0e-5, 1.0e-1)
⇒ Best downstream results **often at small values 1e-5 or 1e-4**. Only rarely at 1e-2. Larger values: very poor performance much more likely.

Batch size in BatchNorm1d: bs config space: choice([32, 64, 128, 256]
⇒ Best downstream results at **no particular value**. Every value can find good solutions.

Batch size 128 and 256 more likely in HPO as they speed up training (and allow more trials then). High influence on training runtime, larger values speed up training time roughly linearly: small batch size 32 (6-12 min) is about factor 6 however in training time than batch size 256 (1-2 min). The smaller networks can use bs 64 or 32 easily. Essentially, all the networks with capacity: 100 to 10.000 (latent nodes) train in under 10 min, besides outliers likely caused by cluster file system.

Drop probability in Dropout: p_{drop} config space: uniform(0, 0.9)
$\Rightarrow$ Most downstream results at **dropout up to 50%**. Good values from as little as 1% to 50% by HPO. Rare occasions of high values between 70-90%. For small networks: capacity under 3000 hidden nodes choose $p_{drop} < 75\%$. Very small networks: $p_drop < 0.2$ as a good rule of thumb. For larger networks (like > 10,000 hidden nodes): p_drop can be higher than 50%. But then eventually reduce layer widths.

High $p_{drop} > 60\%$ is more likely to cause negative training R^2 and bad validation R^2 and also to get bad downstream performance.

5.3 Results on Varix

In general, predicted results on Vanillix are higher than on varix, but Vanillix lacks the nice property of a VAE to learn mean and variance and the opportunity to sample from a latent distribution to get new similar data.

For Varix, solutions with high predicted downstream task performance usually had very small beta. Fig.7 showed that a transfer of hyperparameters from Vanillix to Varix is still possible and promising. It also gives an indication that high d_z and small β will maximize downstream performance.

5.4 HPO algorithms in Syne Tune

CQR achieves very good early results, but not ones with highest predicted downstream performance. Among the tested algorithms (all single objective and single fidelity), its execution time was considerable slower.

REA achieves better values at a later time (more exploration, step changes). It was very fast could be repeated many times in short time.

As seen in results on SCHC, HPO on the surrogate model can overshoot metrics. While a true value for AUC ROC OVO cannot be better than 1, extrapolations in the XGBoost model can cause predicted metrics to be above 1 as for SCHC.

Syne tune worked well, the concept of blackboxes needed some work first for data import into the blackboxes, as some constraints, such as shared hyperparameter value combinations on the tasks of a blackbox do not seem to be documented.

Transfer learning On all 5 datasets: Running HPO with transfer learning has better downstream task performance prediction on the XGBoost surrogate model than without transfer learning.

Regularized Evolution Algorithm (REA) gave best results in many runs. So it was chosen for transfer learning. It was very fast on the surrogate model, and seemed more exploratory in nature to find better new optima candidates.

In transfer learning, Syne Tune can be adapted to a top-k set of hyperparameter evaluations with a parameter `num_hyperparameters_per_task` on a `SingleObjectiveScheduler`.

This changes, that only the hyperparameter ranges of best (top k) previous configurations are evaluated and the search space is shrunk. Using this parameter, improved results. We found that a value of 10 was too limiting (will shrink the search space), but 30 seemed better to leave enough terrain to still explore.

5.5 Evaluation

Syne Tune predictions have partially been evaluated for real results in Autoencodix. The results in Fig. 7 are actual Autoencodix runs on Syne Tune recommended HP combinations.

5.6 Unexplored configuration space within Hyperparameter configuration space

This is a discussion of what remains unexplored with the previous experiments.

The random sampling of the 3,000 HP combinations came from a certain space of hyperparameter values. Naturally, all the HP combinations that Autoencodix was run on, were sampled from this space.

HPO also received this configuration space for values of the hyperparameters. It will not consider HP values beyond min or max in these value domains. (It will not extrapolate hyperparameter values.)

- **choice** parameters - are relatively easy to evaluate with few choices like up to 6 like chosen here, few computations, but no gradient descent.

Aspect to keep in mind: choice parameters with many choices can cause many evaluations for every choice value. Syne Tune adapts choice ranges to min and max values of the top-k evaluations. So, the numeric values are still important, such choices are not simply replaced by one-hot encoding. The top-k evaluations however introduce a new hyperparameter to tune when using Syne Tune.

- **uniform** parameters - min and max from config space are replaced by Syne Tune with the min and max of the actual top-k evaluations. Example: drop probability: uniform distribution(0, 0.9)) but replaced by (0.02, 0.89) if top-k evaluations only fill this interval.

  Uniform parameters could enable gradient descent for minima search, but not in conjunction with other choice variables. Only then along subspaces with all choice values fixed.

  Bayesian Optimization on domains over uniform intervals can be covered with Gaussians or with intervals and step functions. Gaussians not fittable directly in case of multi-seed data, but Gaussians could work on better averages then.

  Optimizing uniform parameters can cause many evaluations in very small regions.

- **log uniform:** allows a better investigation of large ranges, but no longer i.i.d. on linear scale, but i.i.d. on log-scale.

Epochs All training in this thesis ended at 300 epochs as a fixed boundary as a tradeoff between quality and training resources . The development of the weights, models, model performance beyond this was not explored. Very likely, results can improve there.

5.7 Unexplored configuration space

There are more *potential* configuration aspects (not commonly considered as hyperparameters) on training autoencoders.

Symmetry / Asymmetry of Autoencoders In Autoencodix, encoder and decoder for Vanillix and Varix are symmetrical in number of layers and number of nodes per hidden layer. This keeps hyperparameters on the model lower as (n_{layers} and encoding factor r are applied in both of them. Other autoencoders can be set up differently, such as totalVI (Gayoso et al., 2021).

Activation functions Different activation functions could be tried, such as Gaussian Error Linear Unit (GeLU) (non-parametric) or Leaky ReLU (ReLU with leakage parameter). Autoencodix is hard coded to use ReLU, likely with a pragmatic "Good defaults beats endless knobs." approach by the authors.

Optimizer Different optimizers could be chosen: standard Stochastic Gradient Descent (SGD), or SGD with adaptive momentum (Adam).

Autoencodix is hard coded to use "Adaptive Momentum with separate weight decay" (AdamW), where weight decay does not accumulate in the momentum nor variance.

Datatype There would also be choices on more technical aspects, such as used data types (e.g. float64, float32, float16, "brain floating point" bfloat16) for processed data and weights.

In Autoencodix, float32 (4 bytes) is fixed in the source code for processed data and weights.

Quantization Finally, quantization could be a configuration option, either in quantization-ware training or as post-training quantization. Such technical aspects, mainly intended for either running large models on edge devices or increasing model throughput, are not exposed to be configurable to Autoencodix users. As outlined in Chapter 3, the size of Autoencodix models is generally manageable, thus hardware limitations pose a smaller concern.

6 Future work/Outlook

6.1 More work within existing Autoencodix and Syne Tune

6.1.1 Different ranges of the hyperparameters in Vanillix and Varix

Investigate use of a learning rate scheduler. The name of the Autoencodix hyperparameter LR_FIXED indicates the use of a fixed learning rate. As seen in the results, in case of very good downstream performance, often very small learning rates can be found. Yet, large learning rates are more likely to lead to high reconstruction error and essentially a collapse of the model. When investigating the source code of Autoencodix, no use of a learning rate scheduler was found. This seems contrary to recommendations in Loshchilov and Hutter (2016), Goyal et al. (2017), Prince (2023), where learning rate is recommended to be shaped by a learning rate scheduler. It could improve results and stability of the training, becoming less dependent on the data split. This can also be explored by closer examination of the progress in the training loop and therefore by a multi-fidelity sampling of the intermediate results.

Experiment with sparse overcomplete autoencoder for OMICS data In LLMs, a Token ID (integer number) is translated to a very long embedding vector (length e.g. 4,096 floats, 12,000 floats or even higher, depending on model size) for that input (overcomplete embedding). Often it's combined with a constraint (e.g., sparsity) to avoid learning a trivial identity mapping. While more memory and processing are needed with this latent representation, relationships (such as distances or similarities) between the embeddings of two token IDs can be better preserved (in those many dimensions of the token embeddings) and the overall quality of the results improves.

A similar approach could be done in Autoencodix, setting ENC_FACTOR r smaller than 1 (the hidden layers get *more* nodes than the input layer) and setting d_z larger than D. The encoder now had ample nodes for new intermediate approximations, and would not compress, but expand; the decoder would reduce the embedding to the appropriate output size. This would introduce way more trainable parameters, but also offer more derived latent space features for downstream tasks.

Consider more available hyperparameters in Vanillix and Varix This thesis concentrated on a fixed subset of the available hyperparameters. All others were fixed, like the training hyperparamters on EPOCHS = 300 and WEIGHT_DECAY = 0.001, and in varix: LOSS_REDUCTION = sum (for aggregation on KL divergence), ANNEAL = logistic_mid, ANNEAL_PRETRAINING = False.

Future work may include more of these additional hyperparameters like EPOCHS, WEIGHT_DECAY, LOSS_REDUCTION, ANNEAL, ANNEAL_PRETRAINING.

For example, considering EPOCHS = choice (10, 50, 150, 300, 600, 1200) would enable the multi-fidelity HPO algorithms in Syne Tune to work on these multiple target metrics values for a different number of epochs with better early stopping. It would also enable to find better solutions later in the epoch history than mere 300 epochs. However, more training resources (computations and time; GPU hours) would be needed first, that could pay off however with better HPO results (based on a multi-fidelity model) with a good compromise of early stopping for unpromising HP combinations but a longer epoch trajectory for those HP combinations, that have a promising development on the first epochs.

6.1.2 Experiments on other AE architectures

Consider Ontix, the AE architecture specifically prepared for weights based on application domain ontologies Autoencodix also offers Ontix Joas et al. (2024). Ontix is a version of Varix (VAE), that takes into account a domain-specific ontology in the decoder for better explainability by constraining the decoder work along known pathways. At the same time, it serves as a test whether even due to randomness of data splitting and training by SGD/AdamW expected weights in the AE network, and therefore explainable embeddings, are really found robustly and reliably. The role of hyperparameters should be investigated there more deeply, such as a learning rate scheduler. Also the activation function, where Leaky ReLU and GeLU could guide AdamW better in the training process than ReLu.

Consider Stackix and X-Modalix for multi-modal input Autoencodix also offers Stackix and X-Modalix. These are built on varix and work on combining at least two data modalities for compound or joint embeddings. This is made for tasks on modality translation, conversion or alignment. Chances are, that these interactions are less covered in existing research and are therefore a promising path for knowledge discovery and new methods.

6.1.3 Use multi-objective optimization instead of single-objective in Syne Tune

This thesis used the single-objective optimization algorithms in Syne Tune. For this, it created an averaging metric for downstream task performance on a number of clinical variables and looked at this one target metric at a time. This keeps things simple and explainable, as in one metric two results can easily be decided for which one is better. Yet, when we actually look at the downstream task variables, then a multiobjective optimization would likely be closer to the domain goal. This would introduce Pareto fronts of good solutions (good hyperparameter combinations) that are harder to act upon in subsequent steps. When such Pareto fronts as optimization results would be evaluated for robustness, evaluation and metrics are still a research problem.

6.1.4 More/new downstream tasks on the found embeddings

Training an autoencoder can be relatively expensive (about 10 min on a GPU like the Nvidia A100 on the TUD cluster), even for these small Autoencoder models (roughly 8k to 100M parameters) and small datasets (like roughly 11k records in TCGA and 45k records in SCHC).

But inference on the encoder (encoding new datasets into latent space) is cheap. Also, ML downstream tasks on embeddings are cheap. Thus, more new down-stream tasks (provided, that high-quality labels are available) could be performed to evaluate the general applicability of the latent space embeddings. This evaluation on latent space embeddings will give more insight on their necessary properties such as width of the latent space layer when the number of downstream tasks rises.

6.1.5 To improve target metric *validation coefficient of determination* - introduce more domain expertise

Classify and detect posterior collapse in varix When evaluating the XGBoost surrogate model, it became apparent, that the surrogate had not learned well, under which circumstances posterior collapse would occur in Varix. One way would be to avoid such collapse better with other choices of hyperparameters. In addition, as stated before, a minimum validation R^2 thresholds (which is task-dependent!) can be used to not run downstream tasks on low-performing models. This might already reduce the problem.

Otherwise, a separate binary classifier on posterior collapse may deal with this problem and be trained on the results from the 15,000 runs on varix. This may then be used as a filter when defining a reasonable hyperparameter configuration space or be checked after drawing samples for individual HP combinations.

At the same time, it s not really a major problem, because as soon as HPO searchers find better spots and then regions in configuration space, they continue to work around those better solutions.

Improve quality by domain-specific review This thesis treats a cross-domain problem (data science, bio-informatics, genomic pathways, cancer research in the medical domain, single cell analytics, OMICS data for the prediction of complex outcomes): domain expert in cancer diagnostics or single cell diagnostics. They could work on, what the encoder actually encodes (explainability) and if the results are feasible. Can identify gaps, if the embedding cannot explain certain tasks that are relevant. Eventually, the latent dimension should be more than 128, like 256, 512 or 1024. A rough analysis on the number of biomarker genes for SCHC's 15 cell types, 6 age groups and 2 sexes brings up a number on excess of 60 marker genes. TCGA with

its way higher number of classes along the clinical variables may necessitate such an approach even more, especially to get good disentanglement.

6.1.6 To improve target metric *runtime*

Track the use of more/all hyperparameters in source code Both Autoencodix (and also Syne Tune) are open source software and therefore a white box. The used libraries are open-source software as well. An in-depth call graph analysis along the software could return a call graph of functions and parameters, generated while running the software, would allow better white-box analysis, where a function call stack of each hyperparameter would be tracked. Software execution can also be profiled at function level for a deeper understanding of runtimes and Big O analysis in terms of hyperparameters.

Profiling Analysis close to the RAM, caches, CPUs, GPUs, caching, storage access, network would enable better analysis closer to execution hardware. This would likely speed up training and therefore enable quicker HPO with actual Autoencodix calls.

6.1.7 More work on meta-features like dataset features to help Syne Tune in transfer learning from relevant data sets

This work has shown empirically with 5 data sets from two scenarios (The Cancer Genome Atlas, Single Cell Human Cortex) that the properties of the data set influence the results, which was to be expected.

Data sets can be turned to descriptive vectors themselves with dataset-related descriptive metrics like: number of features, number of samples, feature to sample ratio, min and max, mean and std variance, sparsity metrics up to landmark features.

Existing work such as (Feurer et al., 2015) has shown that these can improve convergence in HPO.

Some aspects of a dataset are domain-specific. While statistic properties could be used to classify outliers, or remove low variance features, domain-specific aspects for OMICS data include:

- noise profile of OMICS data => outliers, spread in measurements on devices, transmission errors, spread due to process variation, human or machine errors in executing a protocol, controls in the process (control samples, control markers)
- preprocessing profile of OMICS data => domain-specific, e.g. log1p for sequencing data
- different methods of normalization

A domain-specific approach could attach metadata, that is considered most relevant from the domain experts. This metadata might either be numeric or text data. With text data then leading to conditional VAEs.

Such dataset feature vectors to describe a whole dataset can be used to identify more relationships between dataset features and hyperparameters for even faster convergence in HPO or better results.

One idea/hypothesis would be, that datasets with a similar dataset description vector (based on some distance metric or similarity metric) would better warm-start HPO with their individual distributions of promising hyperparameter value ranges.

6.1.8 Let Syne Tune do actual Autoencodix runs instead of surrogate model alone

This thesis had a large random scan of the hyperparameter space and Autoencodix runs as a starting point. This gives us a good global view but is a large investment of necessary computation resources up front. As a reward however, Syne Tune can then fit a surrogate model on these results. This is fast, but the correlation of the surrogate model and actual results is not perfect.

If Syne Tune would do actual Autoencodix calls instead of working on a surrogate model, this would take more time per Autoencodix run, but would require less time/effort for checking if a predicted performance metric (like from a surrogate) is actually realistic. This approach would give no global view, but could clearly cut compute times for precomputed results from runs and still find very promising HP combinations.

These runs however should use the knowledge gained during training and validation of other runs on similar hyperparameters.

Bibliography

T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019. URL https://dl.acm.org/doi/abs/10.1145/3292500.3330701.

T. Ashuach, M. I. Gabitto, R. V. Koodli, G.-A. Saldi, M. I. Jordan, and N. Yosef. Multivi: deep generative model for the integration of multimodal data. *Nature Methods*, 20(8):1222–1231, 2023.

Z.-J. Cao and G. Gao. Multi-omics single-cell data integration and regulatory inference with graph-linked embedding. *Nature Biotechnology*, 40(10):1458–1466, 2022.

A. Coenen and A. Pearce. Understanding umap. https://pair-code.github.io/understanding-umap/, 2019. Google PAIR, Accessed on 2025 Nov 2nd.

A. Conesa, P. Madrigal, S. Tarazona, D. Gomez-Cabrero, A. Cervera, A. McPherson, M. W. Szcześniak, D. J. Gaffney, L. L. Elo, X. Zhang, et al. A survey of best practices for rna-seq data analysis. *Genome biology*, 17(1):13, 2016.

R. Danino, I. Nachman, and R. Sharan. Batch correction of single-cell sequencing data via an autoencoder architecture. *Bioinformatics Advances*, 4(1):vbad186, 12 2023. ISSN 2635-0041. doi: 10.1093/bioadv/vbad186. URL https://doi.org/10.1093/bioadv/vbad186.

J. Ewald. Autoencodix raw data for reproducibility, Sept. 2024. URL https://zenodo.org/records/13691753. Dataset; Licensed under CC BY-NC-ND 4.0.

G. Farina. Bayesian optimization. Lecture notes for MIT 6.7220/15.084 Nonlinear Optimization Lecture 16, Massachusetts Institute of Technology, Apr. 2024. URL https://www.mit.edu/~gfarina/2024/67220s24_L16_bayesian_optimization/L16.pdf. Class notes.

M. Feurer, J. Springenberg, and F. Hutter. Initializing bayesian hyperparameter optimization via meta-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 29, 2015. URL https://ojs.aaai.org/index.php/AAAI/article/download/9354/9213.

A. Gayoso, Z. Steier, R. Lopez, J. Regier, K. L. Nazor, A. Streets, and N. Yosef. Joint probabilistic modeling of single-cell multi-omic data with totalvi. *Nature methods*, 18(3):272–282, 2021.

D. Ghosh and A. M. Chinnaiyan. Classification and selection of biomarkers in genomic data using lasso. *BioMed Research International*, 2005(2):147–154, 2005.

P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

T. Hastie, R. Tibshirani, J. Friedman, et al. The elements of statistical learning, 2009.

L. Heumos, A. C. Schaar, C. Lance, A. Litinetskaya, F. Drost, L. Zappia, M. D. Lücken, D. C. Strobl, J. Henao, F. Curion, et al. Best practices for single-cell analysis across modalities. *Nature Reviews Genetics*, 24(8):550–572, 2023.

S. H. Holmes and W. Huber. *Modern statistics for modern biology*. Cambridge university press, 2018.

Q. Hu and C. S. Greene. Parameter tuning is a key part of dimensionality reduction via deep variational autoencoders for single cell rna transcriptomics. In *BIOCOMPUTING 2019: proceedings of the Pacific symposium*, pages 362–373. World Scientific, 2018.

M. Joas, N. Jurenaite, D. Prascevic, N. Scherf, and J. Ewald. A generalized and versatile framework to train and evaluate autoencoders for biological representation learning and beyond: Autoencodix. *bioRxiv*, pages 2024–12, 2024. URL `https://www.biorxiv.org/content/biorxiv/early/2024/12/20/2024.12.17.628906.full.pdf`.

I. Kreller. Explainable beta-VAE for biomarker discovery: A comparative study of post-hoc XAI methods. Master's thesis, Universität Leipzig, Leipzig, Germany, Mar. 2025. Submitted on March 13, 2025.

C. Lance and L. Martens. Single-cell atac sequencing, 2023. URL `https://www.sc-best-practices.org/chromatin_accessibility/introduction.html`. In *Single-cell best practices*. Brought to you by Theislab; single-cell best practices consortium.

J. Liu, T. Lichtenberg, K. A. Hoadley, L. M. Poisson, A. J. Lazar, A. D. Cherniack, A. J. Kovatich, C. C. Benz, D. A. Levine, A. V. Lee, L. Omberg, D. M. Wolf, C. D. Shriver, and V. Thorsson. An integrated tcga pan-cancer clinical data resource to drive high-quality survival outcome analytics. *Cell*, 173(2):400–416.e11, 2018. ISSN 0092-8674. doi: https://doi.org/10.1016/j.cell.2018.02.052. URL `https://www.sciencedirect.com/science/article/pii/S0092867418302290`.

S. Liu and T. Yu. Nonlinear embedding and integration of omics data: a fast and tuning-free approach. *Briefings in Bioinformatics*, 26(2):bbaf184, 04 2025. ISSN 1477-4054. doi: 10.1093/bib/bbaf184. URL `https://doi.org/10.1093/bib/bbaf184`.

R. Lopez, J. Regier, M. B. Cole, M. I. Jordan, and N. Yosef. Deep generative modeling for single-cell transcriptomics. *Nature methods*, 15(12):1053–1058, 2018.

I. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

M. Lotfollahi, F. A. Wolf, and F. J. Theis. scgen predicts single-cell perturbation responses. *Nature methods*, 16(8):715–721, 2019.

M. D. Luecken and F. J. Theis. Current best practices in single-cell rna-seq analysis: a tutorial. *Molecular systems biology*, 15(6):e8746, 2019.

C. Meng, O. A. Zeleznik, G. G. Thallinger, B. Kuster, A. M. Gholami, and A. C. Culhane. Dimension reduction techniques for the integrative analysis of multi-omics data. *Briefings in Bioinformatics*, 17(4):628–641, 03 2016. ISSN 1467-5463. doi: 10.1093/bib/bbv108. URL `https://doi.org/10.1093/bib/bbv108`.

S. J. Prince. *Understanding Deep Learning*. The MIT Press, 2023. URL `http://udlbook.com`.

D. Salinas, M. Seeger, A. Klein, V. Perrone, M. Wistuba, and C. Archambeau. Syne tune: A library for large scale hyperparameter tuning and reproducible research. In *International Conference on Automated Machine Learning*, pages 16–1. PMLR, 2022. URL `https://proceedings.mlr.press/v188/salinas22a.html`.

J. Shendure and H. Ji. Next-generation dna sequencing. *Nature biotechnology*, 26(10):1135–1145, 2008.

X. Wang. *Next-Generation Sequencing Data Analysis*. CRC Press, Boca Raton, FL, 2 edition, 2023. ISBN 9780367349899.

M. Wattenberg, F. Viégas, and I. Johnson. How to use t-sne effectively. *Distill*, 1(10):e2, 2016.

D. Wu, D. Wang, M. Q. Zhang, and J. Gu. Fast dimension reduction and integrative clustering of multi-omics data using low-rank approximation: application to cancer molecular classification. *BMC genomics*, 16(1):1022, 2015.

R. Xiang, W. Wang, L. Yang, S. Wang, C. Xu, and X. Chen. A comparison for dimensionality reduction methods of single-cell rna-seq data. *Frontiers in genetics*, 12:646936, 2021.

C. Xu, R. Lopez, E. Mehlman, J. Regier, M. I. Jordan, and N. Yosef. Probabilistic harmonization and annotation of single-cell transcriptomics data with deep generative models. *Molecular systems biology*, 17(1):e9620, 2021.

K. Zhu, J. Bendl, S. Rahman, J. M. Vicari, C. Coleman, T. Clarence, O. Latouche, N. M. Tsankova, A. Li, K. J. Brennand, et al. Multi-omic profiling of the developing human cerebral cortex at the single-cell level. *Science advances*, 9(41):eadg3754, 2023.

L. Zimmer, M. Lindauer, and F. Hutter. Auto-pytorch: Multi-fidelity metalearning for efficient and robust autodl. *IEEE transactions on pattern analysis and machine intelligence*, 43(9): 3079–3090, 2021.