Physics of Molecules and Materials Master Thesis

A Study on Dataset Distribution Impact: QuaLiKiz Neural Networks through Active Learning

Bram de Leeuw

Integrated Modeling and Transport Group,
Dutch Institute for Fundamental Energy Research

Radboud Universiteit

Nijmegen, April 2024

Daily supervisor:	Dr. Aaron Ho
Supervisor:	Prof. Sascha Caron
Second Corrector:	Prof. Nicolo de Groot

Abstract

Creation of a fast and high-fidelity tokamak plasma simulation through integrated modeling is being limited by turbulence simulations. Neural network surrogates explored in previous research have shown the ability to provide an incredible speedup. To generalize these results to higher fidelity simulations, the size of the dataset used for training the neural network surrogate must be reduced due to the higher computational cost of labeling. Active learning was used in this project and allowed for dataset size reduction by multiple orders of magnitude. This study investigated the effect of adapting the distributions of the unlabeled pool and training seed datasets. The matching of the distributions to the test dataset led to the least amount of points required. Similar performance using uniform datasets required twice the dataset size. Evolution of the distribution of the data acquired when using mixed distributions led to the discovery of a range of possible improvements. The exact acquisition function used as well as the active learning batch size proved to influence the results significantly, opening the door to further reduction of dataset sizes and higher fidelity neural network surrogates.

Acknowledgments

Although I have always been interested in physics and developments in science, and clearly remember the glow-in-the-dark stars and planets in my bedroom as a kid, it was only after a few years of studying in Delft that I realized that the bridge between theoretical physics and real world application is what really fascinated me. From choosing the correct assumptions, to choosing between a range of statistical methods to obtain results and determine uncertainty, a simple theoretical problem quickly becomes incredibly complex in practice.

Working on this project was very rewarding as it almost perfectly aligned with my interests. I would like to thank Jonathan Citrin and Aaron Ho for making it possible, and Aaron for also supervising me on a day to day basis.

The contrast between a perfect physical theory and the complexity of it's real world application was immense in this project. The turbulence simulation used, QuaLiKiz, uses a long list of assumptions that all had to be understood and considered throughout the project. Next to this, linking QuaLiKiz to the machine learning pipeline and ensuring correct results and compatibility with existing code led to endless debugging. I would therefore like to thank Lorenzo Zanisi, from the UKAEA, who not only provided me with a machine learning pipeline to start from, but who was always happy to answer any machine learning questions I had. Furthermore, I would also like to thank Sascha Caron for presenting differing opinions about machine learning, and helping me catch some errors in the code and pipeline design I would otherwise have overseen.

Lastly I would like to thank my friends from the DIFFER flex-office, as well as my family and friends for supporting me throughout!

Although many steps have been made in this project, I am happy to see that there still is a huge amount of room left for optimization of the pipeline and wish anybody working on it in the future the best of luck!

Contents

Chapter 1

Introduction

1.1 Introduction to Fusion

The demands on the energy sector today are unparalleled. Not only is the demand for energy increasing along with standards of living, but a worldwide transition towards sustainable energy is in full swing. Although energy can be generated sustainably through many ways such as solar, wind, and nuclear fission, these all have significant drawbacks. Solar and wind energy not only require a large amount of space but are also highly dependent on the weather. This requires energy storage when used in any significant capacity in an electricity network. Nuclear energy on the other hand, faces problems with safety from radioactivity during operation, the risk of accidents, and technology being used for nuclear weapon development. Fusion reactors can provide a way of generating sustainable energy without all these drawbacks. Indeed, fusion reactors would allow continuous, high-density, centralized energy production with negligible safety concerns [1]. The main drawbacks are a high initial research and development cost and the construction costs of reactors. Although fusion reactors are still far from consistent net energy production even after decades of research, renewed interest in the field has arisen as business and startups now see a realistic potential for return on investment. To better understand the challenges faced in developing fusion reactors, we must first understand the conditions under which fusion occur.

During fusion, two atoms collide to form a heavier atom. When colliding, the atoms need to have sufficient energy to overcome the electrostatic repulsion between the nuclei to get close enough to each other for the strong nuclear force to bind them together. Between small nuclei, as shown in figure 1.1, the binding energy per nucleon increases as the number of nucleons increases. For these nuclei, fusion will lead to a net energy gain as a small amount of mass is converted to energy.

Current experiments typically use a reaction between deuterium and tritium to achieve fusion, although the less energy dense deuterium-deuterium reactions are also commonly used as tritium is both rare and slightly radioactive. The reaction between deuterium and tritium,

$$\mathrm{H}^2 + \mathrm{H}^3 \longrightarrow \mathrm{He}^4 + \mathrm{n} \tag{1.1}$$

releases a neutron (n) and 17.6 MeV as kinetic energy of the products [3]. For fusion to occur, the reactants need to collide with enough kinetic energy. Different methods of achieving this exist, one of which is magnetic confinement in a tokamak. Here, a gas of deuterium and tritium is heated until the particles are completely ionized, forming a plasma. Through the use of strong magnets and field lines that close back on themselves, the particles are confined and can be heated up further with minimal loss of energy to the outside world. In tokamaks, fusion has been achieved for short periods of time, however there are still many issues to be solved before a net energy output can be achieved continuously [4].

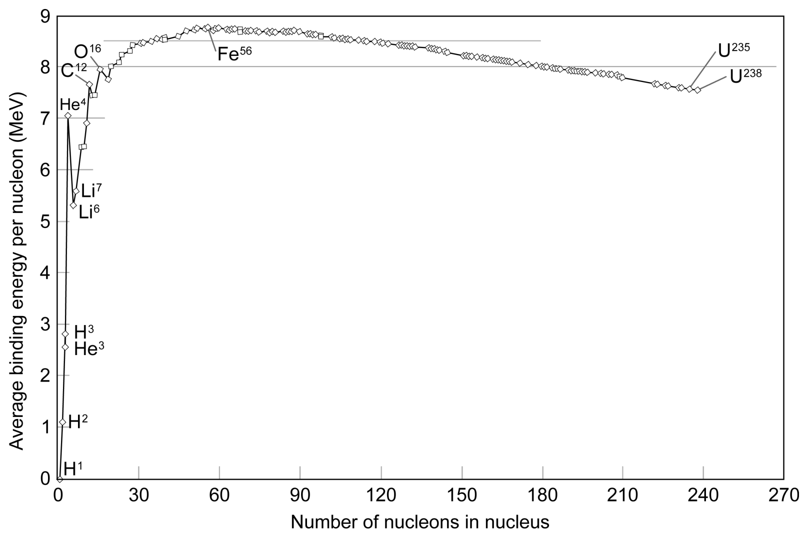


Figure 1.1: Binding energy per nucleon for a selection of nuclei. Energy is released through the conversion of mass to energy for fusion reactions from H^1 up to Fe^{56}. Conversely, energy is released through fission reactions for larger nuclei [2].

1.2 Introduction to Modeling

EUROfusion, the consortium of European research institutes and laboratories working on fusion, has created a roadmap to achieving commercial fusion reactors. One of EUROfusion's goals is to produce a "high-fidelity numerical tokamak" capable of simulating ITER operation and usable for designing DEMO, ITER's successor [1]. This numerical tokamak would consist of multiple models each simulating different physical processes. Plasma turbulence is calculated by one of these models, from which the outputs are then passed to other models as inputs. Currently calculating turbulence is one of the bottlenecks of integrated modeling as it needs to be recalculated frequently due to it's behavior changing on short time scales. QuaLiKiz is one of the codes simulating turbulence based on gyrokinetic theory, which considers spatial scales comparable to the charged particles' gyration radii and a much slower timescale. QuaLiKiz uses a long list of assumptions to reduce computation time. Higher-fidelity code using less approximations and assumptions, such as GENE, also exist but have a much longer computation time per point.

To achieve the goal of a "high-fidelity numerical tokamak", improving the speed of the current high-fidelity turbulence simulations will be essential. This will allow high-fidelity codes to become significantly more accessible for use in research to compare experimental results to simulations, as well as opening the door to "prediction first" experiments where fusion experiments are designed to confirm the optimal solutions found by simulations. Furthermore, it would allow for better optimized tokamak designs and make turbulence simulations available for real time control and prediction of plasma behavior during operation.

One way of speeding up turbulence simulation is through the use of machine learning. Obtaining a large enough dataset of inputs and outputs of the simulation over the relevant physical domain allows for the training of a neural network surrogate which can replace it. Once trained, a neural network surrogate can achieve near instantaneous results, as shown in table 1.1.

Table 1.1: Typical order of magnitude of calculation time to estimate 20 radial points and 10 wave numbers per flux[5].

Simulation	Computation time (s)
QLKNN-15D	10^{-3}
QuaLiKiz	10^{2}
Non-linear GENE	10^{8}

1.3 Machine and Active Learning

The choice of dataset determines the behavior of the surrogate and the domain upon which it can be used. Indeed, while neural networks perform well for interpolation, they are unusable for extrapolation. A smart choice of training, validation and test datasets is essential as to be able to capture the behavior of a model.

The end goal of this project is taking steps towards is the creation of a fast high-fidelity model usable for any fusion experiment. Before moving towards the use of a high-fidelity model, the methods used in creating a fast general model are tested on QuaLiKiz. Once the required number of points has been reduced to allow labeling within available computation time, a move to higher-fidelity codes can be considered.

In previous work, a neural network for QuaLiKiz with 15 inputs parameters was created with results within 10% averaged relative root mean squared error of 1D scans [6]. A dataset of $2 \cdot 10^7$ points was used to created this surrogate. Subsequently, active learning was used on this dataset to train a surrogate using just $2 \cdot 10^5$ points [7]. In this work we wish to not only increase the number of input parameters to 17 by including rotational plasma effects, but we also want to look at a larger input domain so that turbulence can be simulated for more than just the JET tokamak.

Active learning allows for the selection of only the points for which the predictions of a neural network are most uncertain. The neural network is pretrained on a small training seed dataset and batches of new highly uncertain points are added iteratively. Although active learning has been shown to be effective at reducing the number of points required for QuaLiKiz in literature [7], this was based on a fixed prelabeled dataset.

In this work QuaLiKiz will be coupled directly to the active learning pipeline from [7] for the creation of QLK17D model usable over a larger input domain. This removes the requirement of a large pre-labeled dataset and allow increased flexibility in the choice of points for the acquisition function.

1.4 Research Question

An initial unlabeled dataset is required from which the acquisition function can pick points to pass to QuaLiKiz. If points are not available in this dataset they clearly cannot be picked, affecting the final neural network trained. Similarly, for non-zero density it is expected that the distribution of the unlabeled pool affects the final neural network trained. It is also unclear to what degree the initial training seed used to pretrain the neural network affects the final neural network trained, and if a reduction in training points required for the same performance is possible by altering these distributions. This brings us to the research question:

How do the distributions of the unlabeled pool and training seed affect the neural networks trained through active learning?

This will be answered by training multiple models starting from different combinations of datasets. As we want to produce a QLKNN version that includes rotational effects for use in future projects, QLKNN-17D models are generated for this project.

In order to answer the research question we will first look at the theory behind turbulence modeling, machine learning and active learning in chapter 2. Following this we will describe the methodology used for this project in chapter 3. The results will then be presented in chapter 4 and discussed in chapter 5. Finally, the main findings of this report will be summarized in the conclusion in chapter 6.

Chapter 2

Theory

2.1 Plasma Physics

For fusion to occur inside a tokamak, a plasma needs to be formed and brought to a state where the particles have sufficiently high kinetic energy that collisions can lead to fusion. To do this, gas particles used are heated in the tokamak to such high temperatures that they become completely ionized, turning the gas into a plasma. This plasma needs to reach sufficiently high temperatures, density, and confinement time for fusion to occur. The requirements for fusion are expressed by the Lawson criterion,

$$n\tau_E T \tag{2.1}$$

the triple product of temperature (T), density (n) and confinement time (τ_E). For the particle species used, settings and design of the reactor are adjusted to maximize this figure and above a certain limit fusion reactions start to occur. For fusion deuterium and tritium are usually used as particle species. However, due to the radioactivity of tritium and similar dynamics of the particles, often only deuterium is used for experiments [8].

To achieve high temperature: To achieve the temperatures necessary for fusion, the plasma is in a primary instance heated through ohmic heating. Ohmic heating occurs when a current passes through the plasma. Collisions and interactions with the plasma will cause for resistance, leading to heating. This method raises the temperature of the gas, however, when taking into account electron-ion and electron-electron interactions a decrease in the resistance with temperature can be observed above a few keV [9]. The plasma can then be further heated using electromagnetic waves that resonate with either the electron or ion's cyclotron frequencies. Another method for further heating is to inject high energy neutral particles that transfer some of their energy to the gas through collisions. These methods are combined to produce high enough temperatures for fusion to occur [10].

To achieve high density: The product of density and temperature is defined as the plasma pressure. This metric follows from the kinetic theory of ideal gases and must be balanced by magnetic confinement so that the plasma stays in place [11]. It is difficult to increase both the density and temperature of the plasma, as many physical processes inside the tokamak limit the use of high densities [12]. As methods to increase heating are relatively efficient, low density, high temperature plasma's are used for fusion. Where the lower densities are achieved by creating a vacuum inside the tokamak before inserting the deuterium/tritium.

To achieve high confinement time: Tokamaks make use of a combination of toroidal and poloidal magnetic fields to create magnetic field lines that curve helically along the toroidal direction.

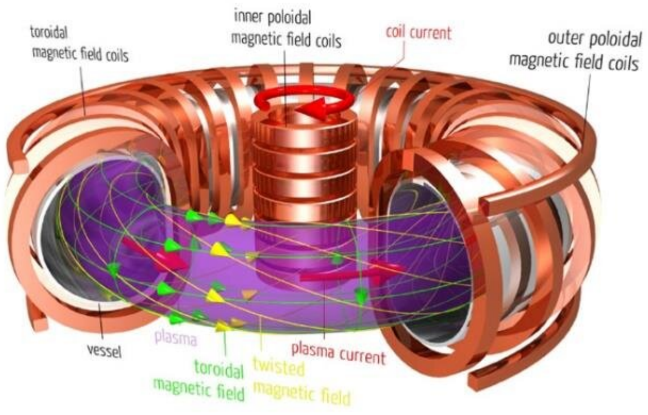

toroidal magnetic field coils
inner poloidal magnetic field coils
coil current
outer poloidal magnetic field coils
vessel
plasma
toroidal magnetic field
twisted magnetic field
plasma current

Figure 2.1: Internal view of a tokamak. The combination of poloidal and toroidal magnetic field coils create helical magnetic field lines throughout the torus which close back on themselves. Through $\boldsymbol{E} \times \boldsymbol{B}$ drift, the ionized particles in the plasma will be confined to within their Larmor radius about these field lines. Image from [14].

Through the Lorentz force, the ionized particles of the plasma are confined to move along these field lines, as shown in figure 2.1. The toroidal shape used together with poloidal and toroidal magnets allows for helical field lines which close back on themselves. The particles can theoretically move along these field lines forever, leaving them trapped. This works as follows, the charged particles will naturally gyrate about magnetic field lines with a Larmor radius,

$$\rho_L = \frac{mv_\perp}{e|\boldsymbol{B}|} \tag{2.2}$$

where the charged particle has a given non-zero initial velocity, $v_\perp$, m is the mass of the particle, e is it's charge, and $\boldsymbol{B}$ is the magnetic field. This confines the particles in the direction perpendicular to the field lines. In the direction along the field lines, the particles are then trapped due to the helical field lines closing back on themselves.

The charged particle can drift away from the field line it is orbiting through certain mechanisms. These should be taken into account as they cause loss of confinement. Drifts can occur due to a magnetic field gradient, $\nabla \boldsymbol{B}$, or due to the curvature of the magnetic field lines followed [3]. However, the mechanism most relevant for understanding turbulence is the $\boldsymbol{E} \times \boldsymbol{B}$ drift caused by the presence of an electric field, $\boldsymbol{E}$, perpendicular to $\boldsymbol{B}$. A force is exerted on the gyrating charged particle, causing it to move away from the field line it was originally orbiting [13]. The different mechanisms through which this electric field can occur will be covered later in this section. Minimizing or preventing the electric field from occurring is essential to reduce turbulence in the plasma.

Many other factors influence the confinement of a plasma, notably the toroidal geometry and the distribution of velocity of the particles which are considered in the neoclassical transport model. However, when using this model the actual transport of particles and heat out of a plasma is significantly larger than that measured in experiments. The best explanation for this anomalous transport is micro-instabilities in the plasma that lead to turbulence [3].

Gyrokinetic theory describes how turbulence occurs in a plasma. However this theory is not applicable to tokamaks as a whole due to the large amount of computing power required. Instead empirical scaling laws derived from experiments combined with numerical simulations based on multiple simplifying assumptions are used [15]. There are a range of different simulation codes available, each using different assumptions. To better understand their use case, the theory from which the simulation codes are derived is further discussed below.

As the effect of perturbations on the plasma are of interest, the plasma's dispersion relation were derived, starting from a statistical mechanical description. Only the steps needed for an intuitive understanding of the derivation are included, for a complete derivation please refer to [16].

The starting point of the derivation is the Vlasov equation for one species,

$$\frac{\partial f}{\partial t} + \boldsymbol{v} \cdot \frac{\partial f}{\partial \boldsymbol{r}} + \frac{e}{m}\left(\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B}\right) \cdot \frac{\partial f}{\partial \boldsymbol{v}} = \left(\frac{\partial f}{\partial t}\right)_C \tag{2.3}$$

which describes the evolution of the distribution function of a plasma over time. Here $f = f(x, y, z, p_x, p_y, p_z)$ is the distribution function, $\boldsymbol{v}$ is the velocity, r is the position, e_j is the charge, m_j is the mass, $\boldsymbol{E}$ is the electric field, $\boldsymbol{B}$ is the magnetic field, and $(\frac{\partial f}{\partial t})_C$ is a collisional term that we neglect for this first part of the derivation.

Typically the evolution of the distribution function is written in terms of Hamilton's equations, it can be rewritten as,

$$\frac{\partial f}{\partial t} + \dot{\boldsymbol{q}}\frac{\partial f}{\partial \boldsymbol{q}} + \dot{\boldsymbol{p}}\frac{\partial f}{\partial \boldsymbol{p}} = 0 \tag{2.4}$$

where $\boldsymbol{q}$ and $\boldsymbol{p}$ are respectively the position and canonical momentum. Their derivatives are given by Hamilton's equations of motion [17],

$$\frac{\partial H}{\partial \boldsymbol{p}} = \dot{\boldsymbol{q}} \tag{2.5}$$

$$\frac{\partial H}{\partial \boldsymbol{q}} = \dot{\boldsymbol{p}} \tag{2.6}$$

where H, the Hamiltonian, is the total energy of the system. A further transformation to action-angle coordinates is then made as it allows for conserved quantities to emerge, reducing the complexity of the problem,

$$\frac{\partial f}{\partial t} + \dot{\boldsymbol{\alpha}}\frac{\partial f}{\partial \boldsymbol{\alpha}} + \dot{\boldsymbol{J}}\frac{\partial f}{\partial \boldsymbol{J}} = 0 \tag{2.7}$$

where $\boldsymbol{J}$ and $\boldsymbol{\alpha}$ are vectors representing the action and angles. Each element in $\boldsymbol{J}$ is an adiabatic invariant quantity. These coordinates simplify the derivation of the dispersion relation [18]. Hamilton's equations become,

$$\frac{\partial H_0}{\partial \boldsymbol{\alpha}} = -\dot{\boldsymbol{J}} = 0 \tag{2.8}$$

$$\frac{\partial H_0}{\partial \boldsymbol{J}} = \dot{\boldsymbol{\alpha}} = \boldsymbol{\Omega} \tag{2.9}$$

where the constant angular frequencies are represented by $\boldsymbol{\Omega}$, and H_0 is the Hamiltonian at equilibrium.We assume the perturbations to the plasma can be represented as first order perturbations to both the Hamiltonian and distribution functions,

$$H = H_0 + \delta h \tag{2.10}$$

$$f = f_0 + \delta f \tag{2.11}$$

where f_0 is distribution functions at equilibrium and δh and δf are small perturbations. The unperturbed Hamiltonian for a charged particle e in an electromagnetic field is given by,

$$H_0 = \frac{1}{2m}(\boldsymbol{p}^2 - e\boldsymbol{A}_{\boldsymbol{0}}^2) + e\boldsymbol{\Phi} \tag{2.12}$$

where at equilibrium, A_0 is the vector potential, and Φ is the electrostatic potential, and assuming a purely electrostatic perturbation of the form,

$$\delta h = e\phi \tag{2.13}$$

in equation 2.10. This approximation is valid for plasma's that have a low ratio of plasma pressure to magnetic pressure, β. For low β plasma's, the movement is dominated by the magnetic field, and particles are strongly constrained to the field lines, while for high β plasma's the charged particles have enough kinetic energy that their movement causes a magnetic field (through the Biot-Savart law) large enough to significantly influence the background magnetic field [13].

This approximation is not completely valid anymore for some of the experiments being conducted today, notably simulation models based on this approximation will fail to account for extra stabilization from electromagnetic effects for ITG turbulence [19]. Solutions for this have been developed in literature [20], so that this assumption can be kept in this derivation. Continuing our derivation by deriving Hamilton's equations,

$$\frac{\partial H}{\partial \boldsymbol{\alpha}} = e\frac{\partial \phi}{\partial \alpha} \tag{2.14}$$

$$\frac{\partial H}{\partial \boldsymbol{J}} = \Omega = e\frac{\partial \phi}{\partial \boldsymbol{J}} \tag{2.15}$$

where we include the perturbations. Similarly substituting in the perturbations in 2.7 gives

$$\frac{\partial \delta f}{\partial t} + \boldsymbol{\Omega}\frac{\partial f}{\partial \boldsymbol{\alpha}} - e\frac{\partial \phi}{\partial \boldsymbol{\alpha}} \cdot \frac{\partial f_0}{\partial \boldsymbol{J}} = 0 \tag{2.16}$$

the Vlasov equation in terms of the equilibrium distribution and perturbation, where all terms higher than first order in δf have been neglected. These higher order perturbation terms are also neglected in further steps of the derivation.

Now defining the shape of the perturbations, it is assumed that the distribution function is sinusoidal and periodic with α. To keep the description of the perturbation as general as possible, it is defined as a discrete Fourier transform

$$\delta f = \sum_{\boldsymbol{\eta}} f_{\boldsymbol{\eta}}(\boldsymbol{J}) e^{i(\boldsymbol{\eta}\cdot\boldsymbol{\alpha} - \omega t)} \tag{2.17}$$

$$\phi = \sum_{\boldsymbol{\eta}} \phi_{\boldsymbol{\eta}}(\boldsymbol{J}) e^{i(\boldsymbol{\eta}\cdot\boldsymbol{\alpha} - \omega t)} \tag{2.18}$$

where we have a sum over the wavenumbers, η. Intuitively the sum over the different wavenumbers can be seen as the sum over the different scales over which turbulence occurs.

In the complete derivation, a number of assumptions are made, which are listed here for completeness before acknowledging the derived dispersion relation [16].

- The plasma is assumed to be at equilibrium with a Maxwellian distribution function, f_0.
- Any perturbation of the distribution function is assumed to be small enough that it's effect on the mean is negligible when equilibrium is reached again.
- The paths of passing particles are approximately toroidal instead of helical.
- The average velocity of the plasma parallel to B, $U_{||}$, is small compared to the speed of sound in the plasma.
- All derivatives of the parallel velocity are expected to be small in intermediate steps.
- Drift of charged particles along $\boldsymbol{B}$ can be decoupled from gyration about the field lines as they occur on very different time and spacial scales.

The electric field generated by the charged particles can be described by Poisson's equation,

$$\nabla^2 \phi = \sum_s -\frac{e_s n_s}{\epsilon_0} \tag{2.19}$$

where we introduce a description per species with the s subscripts and ϵ_0 is the permittivity of free space. A plasma consists of completely ionized particles, these are distributed such that the whole plasma appears neutral from an outside observer far away. As small variations in charge distribution do occur at small scale, a plasma is said to be quasi-neutral. As the resulting field in equation 2.19 must be zero for the plasma to be quasi-neutral,

$$\sum_s e_s n_s = \sum_s e_s \int d^3 v f_s = 0 \tag{2.20}$$

where the integral is over velocity space. Filling in the variables leads to the dispersion relation in action-angle variables, where any cross terms between modes are neglected, further intermediate steps can be found in literature [16] but are not relevant for this project,

$$\sum_s \sum_{\eta_1,\eta_2} \int d^3 r d^3 v \frac{e_s^2 |\phi_{\boldsymbol{\eta}}|^2}{T_s} f_{0,s} \left(1 - \frac{\omega - \boldsymbol{\eta} \cdot \omega_{*,s} - \boldsymbol{\eta} \cdot \omega_{E,s}}{\omega - \boldsymbol{\eta} \cdot \boldsymbol{\Omega_s}} \right) = 0 \tag{2.21}$$

where we sum over the wavenumbers η_1, and η_2, which appear from the use a Fourier transform to represent δf and ϕ. T_s is the temperature per species, $\omega_{E,s} = \frac{e}{T_s} \frac{d\phi}{d\boldsymbol{J}}$ and ω_* is

$$\omega_* = T \left(\frac{1}{n_{0,s}} \frac{\partial n_{0,s}}{\partial \boldsymbol{J}} + \left(\frac{E}{T_s} - \frac{3}{2} - \frac{U_{||}}{v_{T,s}^2} (2 v_{||,s} - U_{||}) \right) \frac{1}{T_s} \frac{\partial T_s}{\partial \boldsymbol{J}} \right) + \frac{2(v_{||,s} - U_{||})}{v_{T,s}^2} \frac{\partial U_{||}}{\partial \boldsymbol{J}} \tag{2.22}$$

with $n_{0,s}$ being the number density at equilibrium per species, E the total kinetic energy, $v_{||,s}$ the velocity parallel to $\boldsymbol{B}$ per species, $v_{T,s} = \sqrt{2T_s/m_s}$ and $U_{||}$ is the equilibrium plasma rotation velocity parallel to $\boldsymbol{B}$.

This integral can be divided into three functionals, each representing different physical phenomenon on which different approximations can be applied,

$$\sum_s \mathcal{L}_{0,s} - \mathcal{L}_{passing,s} - \mathcal{L}_{trapped,s} = 0 \tag{2.23}$$

where $\mathcal{L}_{trapped,s}$ corresponds to trapped particles, $\mathcal{L}_{passing,s}$ corresponds to particles traveling along the helical field lines and $\mathcal{L}_{0,s}$ is an adiabatic term reflecting the instantaneous response of the plasma to changes.

Further use of symmetry and approximations allow these integrals to be reduced in dimension far enough to be able to be computed in a reasonable time frame.

From the derivation above we obtain the dispersion relation of the plasma and through the use of a complex ω, we are able to get the growth rate of the different modes. As we are looking at the fastest departure from equilibrium, we can determine the dominant unstable mode in the linear solution from the largest growth rate. We then assume that it is dominant for both the nonlinear solution and stays dominant over time until saturation is reached.

To find the amplitude of the perturbations we must look at how the instabilities evolve over time. From nonlinear simulations, it is found that interactions between different modes become increasingly relevant as they grow in size. These interactions eventually lead to a saturation point at which the amplitude of the perturbations no longer increase. Using a quasilinear approach whereby linear and nonlinear simulations are compared, saturation rules have been derived that allow for the prediction of the amplitude of the perturbation at saturation, given the initial growth rates when departing from equilibrium [16].

In equation 2.4 we neglected all collisions. The contribution of collisions can be included through the use of a collision operator. In QuaLiKiz, the turbulence simulation model used, the collision operator is set as a Krook collision operator. Adding this collisional term to equation 2.16,

$$\frac{\partial \delta f_s}{\partial t} + \boldsymbol{\Omega}\frac{\partial f_s}{\partial \boldsymbol{\alpha}} - e_s\frac{\partial \phi}{\partial \boldsymbol{\alpha}} \cdot \frac{\partial f_{0_s}}{\partial \boldsymbol{J}} = -\nu\left(\delta f_s + \frac{e\phi f_{0_s}}{T_s}\right) \tag{2.24}$$

where ν is the collision frequency. This operator is a simplification of the Boltzmann operator and approximates the behavior of the distribution function moving towards equilibrium due to collisions between particles. The Krook operator is used as a trade-off between complexity of calculations and accuracy. Collisions are only considered for trapped electrons and their collisions with ions because the interactions are not strong enough to affect the passing particles significantly. The derivation to the dispersion relation is very similar and can be followed [16].

There are multiple ways in which perturbations can lead to a flux and it is useful to understand the physical mechanisms responsible. The perturbations that lead to turbulence are due to ion temperature gradient (ITG), electron temperature gradient (ETG) and trapped electron modes (TEM). Starting from a general description of the workings of a tokamak, an intuitive description of the source of these three instabilities can be sketched.

The perturbations originate from perturbations in gradients in the plasma. Taking temperature as an example, although the plasma is kept contained as much as possible, collisions with the walls still occur. As the walls are at a much lower temperature than the plasma, a temperature gradient, ∇T exists. A small perturbation of the electrostatic potential in the presence of the temperature gradient, δT, then leads to ITG instability.

For a sinusoidal perturbation, as shown in figure 2.2, the perturbation causes particles on the hotter side to drift faster than the particles on the colder side. The warmer particles (both electron and ions) will reach the edges of the perturbation faster than the cold particles on the other side. This leads to a net accumulation of charge, and thus an electric field. Due to the magnetic field lines out of the page and a periodic $\boldsymbol{E}$ field, a periodic $\boldsymbol{E} \times \boldsymbol{B}$ drift will occur, leading to the perturbation increasing in amplitude.

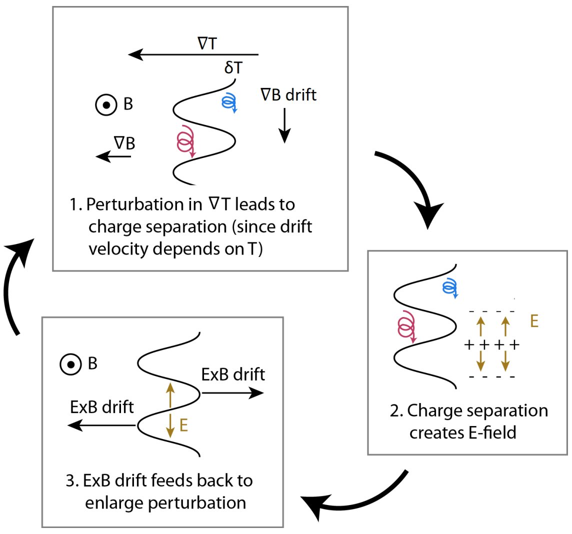

∇T
δT
B
∇B drift
∇B
1. Perturbation in ∇T leads to charge separation (since drift velocity depends on T)
E
2. Charge separation creates E-field
B
ExB drift
ExB drift
E
3. ExB drift feeds back to enlarge perturbation

Figure 2.2: Static view of a plasma perturbation and the mechanism through which it's growth is driven. In the top image, the sinusoidal perturbation in the temperature gradient leads to different drift speeds for charged particles. This causes a net accumulation of charge at different places along the perturbation. The resulting electric field causes $\boldsymbol{E} \times \boldsymbol{B}$ drift, enlarging the perturbation. Image adapted from [21].

Of course this picture is a simplification of what happens in a tokamak. In reality, the boundary of the perturbation is neither so clear cut, nor static. Indeed the boundary will constantly be affected by surrounding particle movements and changes to the electric field. For a moving perturbation, the ITG instability is a phase shift between the motion of the particles and the electric field around them. This phase shift in turn leads to an $\boldsymbol{E} \times \boldsymbol{B}$ force which further increases the perturbation.

ETG perturbations operate similarly to ITG perturbations but at both a faster timescale and at the scale of the electron gyroradius. Electrons in a plasma have a much smaller mass compared to ions, so changes to their movement have a negligible impact on ions. Only considering ETG, and for a moment considering a case where ions are fixed in space. For quasi-neutrality to hold, electron density must then also be fixed to match the density of ions. As the mass of electrons is so much smaller than that of ions, ETG only has a significant effect on electron heat transport, q_e [22].

Lastly, considering the TEM instability mechanism, electrons become trapped in orbits smaller than the circumference of the tokamak. This occurs when particles moving from the edge of the tokamak towards the core encounter a magnetic field gradient. Due to conservation of magnetic moment of the charged particle, $\mu = mv_\perp^2/2B$, if the velocity of the particle is too small, a negative $v_\perp^2$ can result as B increases. This means that the particle will have a negative $v_{||}$, effectively moving back to it's origin, in a trapped orbit [3]. During the orbit, the electrons cause density perturbations to the rest of the plasma, leading to similar perturbations for ITG and ETG. As TEM modes have similar time and spacial scales as ITG instabilities they strongly influence each other. Both should thus be considered when looking at resulting total turbulence.

Next to considering sources of turbulence of similar scale, other factors also influence if a given plasma scenario will lead to turbulence. There are other processes, such as the magnetic or flow shear in the plasma, which counteract the growth of perturbation. The driving gradient thus needs to be increased beyond a certain critical gradient point before turbulence will result, this will be referred to as the critical gradient.

In turbulent motion, particles will tend to move from high temperature regions to low temperature regions, following the second law of thermodynamics. Similarly, they will move from regions of high to low density. This movement is best described as a flux, and usually occurs from the center of the plasma to the edge following the temperature gradient. Although QuaLiKiz returns flux ratio's, telling us if a flux is present and it's direction, the magnitude of the flux needs to be calculated using a saturation rule derived from non-linear turbulence simulations.

2.2 QuaLiKiz Variables

The specific simulation model we will be using is QuaLiKiz. After inputting all the parameters of the plasma, the model calculates the turbulent heat, particle and angular momentum fluxes as outputs.

Many of the input parameters of the plasma will be kept constant throughout our simulations, these are listed in table 2.1. Notably the ion species used are fixed, as well as the range of different scales of turbulence covered by $k_\theta\rho$. For values of $k_\theta\rho$ lower or equal to 2, we are looking at ion scales, while for values above 2, we are looking at electron scales.

Table 2.1: QuaLiKiz variables kept constant throughout all simulations. R_0 and B_0 are only necessary as intermediate step between the dimensionless inputs and the dimensionless calculations within QuaLiKiz.

Variable	Value	Description
$k_\theta\rho$	0.1, 0.175, 0.25, 0.325, 0.4, 0.5, 0.6, 0.8, 1.0, 2.0, 3.5, 6.0, 10.5, 15.0, 19.5, 24.0, 30.0, 36.0	Characteristic wavelength of perturbation, k_θ being the poloidal wavenumber and ρ_s the Larmor radius. Their product tells us the scale of the turbulent structures compared to the gyration path
Z_{i0}	1	Charge of ion 0, Deuterium, normalized to elementary charge
Z_{i1}	4	Charge of impurity ion 1, Beryllium, when completely ionized, normalized to elementary charge
Z_{i2}	18	Charge of impurity ion 2, Argon, when completely ionized, normalized to elementary charge
A_{i0}	2	Ion mass of ion 0 normalized to proton mass
A_{i1}	9	Ion mass of ion 1 normalized to proton mass
A_{i2}	40	Ion mass of ion 2 normalized to proton mass
R_0	3	Averaged major radius of the torus
B_0	3	Magnetic field at the magnetic axis

The independent variables used are listed in table 2.2. All variables have either been normalized by division or by using natural logarithms so that values can be compared between different tokamaks. The gradients are normalized through division, in table 2.2, where L_x in the notation R/L_x refers to $x/\nabla x$.

Of the variables selected, R/L_{T_i}, R/L_{n_e}, R/L_{T_e} are the primary drivers of ITG, ETG and TEM[23]. The rest of the variables were chosen based on previous works. Simulations were performed observing the output fluxes while varying only one of the variables. The variables which showed either a stabilizing or destabilizing effect on the plasma modes were determined to influence turbulence and selected [6][24].

QuaLiKiz has a few different options for it's outputs. It can either output total particle, heat and angular momentum fluxes or the particle and heat fluxes can be subdivided into their respective diffusion and convection components [25]. Angular momentum flux subdivision has not been derived or added to QuaLiKiz yet. The respective contributions to the flux from ITG, ETG, and

TEM can also be returned. We will be using the fluxes subdivided by mode as output, and further using the returned diffusion and convection components which are calculated separately to check for consistency.

Table 2.2: QuaLiKiz variables varied throughout the simulations, this collection of variables will be called QLK17D. QLK15D refers to these variables with $\epsilon = 0.33$ and $R/Z_{eff} = 0$. All variables are dimensionless or normalized so results can be generalized to different tokamaks.

Variable	Description
R/L_{n_e}	Natural logarithm of the electron density radial derivative
R/L_{T_e}	Natural logarithm of the electron temperature radial derivative
$R/L_{T_{i_0}}$	Natural logarithm of the temperature radial derivative of ion 0
$R/L_{n_{i_0}}$	Natural logarithm of the density radial derivative of ion 0
q	Safety factor, a measure of how helical the magnetic field lines are
$\hat{s}$	Magnetic shear, a measure of how helicity changes radially
ϵ	Inverse aspect ratio, the ratio of the minor radius of the last closed flux surface to the major radius of the tokamak
T_{i_0}/T_e	The ion temperature of ion 0 normalized to the electron temperature
$\log_{10}(\nu^*)$	Log of collisionality normalized to inverse orbit time of trapped electrons. (A measure of how often trapped electrons are able to complete an orbit)
Z_{eff}	Effective charge of all the ions
α	Normalized pressure gradient
$c_s/\nabla u_{tor}$	Natural logarithm of the radial derivative of u_{tor}
$R/L_{Z_{eff}}$	Natural logarithm of the radial derivative of Z_{eff}
γ_e	Flow shearing rate perpendicular to $\boldsymbol{E} \times \boldsymbol{B}$
u_{tor}/c_s	The equilibrium plasma toroidal rotation velocity, normalized by the speed of sound in the plasma
n_{i_0}/n_e	The ion density of ion 0, normalized to the electron density
x	Normalized distance from the core of the tokamak to the last closed flux surface near the edge

As a first step this project will only be considering ITG dominant modes and we will therefore be considering the following fluxes,

- q_i: ion heat flux
- $\frac{q_e}{q_i}$: electron heat flux
- $\frac{\Gamma_e}{q_i}$: ion particle flux
- $\frac{\Gamma_e}{q_i}$: electron particle flux
- $\frac{\Pi_i}{q_i}$: ion angular momentum flux

$\frac{\Pi_e}{q_e}$ is very small due to the QuaLiKiz assumptions in section 2.1 and not considered.

Looking at fluxes resulting from ITG only, we expect q_i to be largest for most points. the rest of the fluxes are then divided by it so that when the critical gradient is reached, this is reflected in all the outputs at the same time [24]. Points with q_i flux of zero are labeled as stable, and any other points are labeled as unstable. The labels used may be somewhat confusing, to clarify, this does not tell us if the plasma state itself is stable as source terms of the transport equation have not been considered.

Calculations within QuaLiKiz are dimensionless, to convert the outputs to SI units, GyroBohm scaling formulas as listed in appendix A can be used. We will keep the outputs dimensionless for this project as it allows better generalization of the results and easier training of the neural networks.

QuaLiKiz is typically used within an integrated modeling platform to allow the full simulation of the behavior of a plasma. As calculating turbulence needs to be repeated for each time step, it is not surprising that turbulence calculations are the limiting factor to many simulations. Using a neural network surrogate for QuaLiKiz has been shown to be 10^4 times faster than QuaLiKiz, and within 10% RMS on 1D scans once trained [6], solving this issue.

2.3 Machine Learning

As in all neural networks, we will be using a biased perceptron for each node. In a perceptron the inputs are multiplied by weights and the results are then added together with a bias term. The weighted sum is then sent to an activation function, as shown figure 2.3.

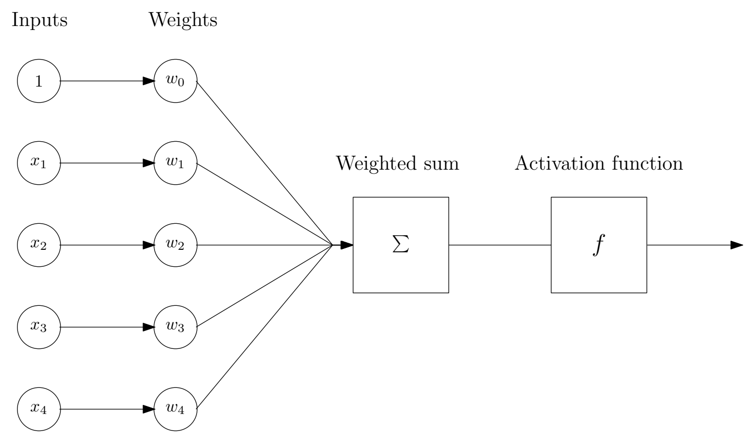


Figure 2.3: Simple perceptron, the input values are multiplied by weights and then added together with a bias term. The result is then passed through an activation function. The weights are adjusted during training, and the activation function can be tuned to model non-linear behaviors.

Coupling multiple perceptrons together into rows and layers allows for the handling of more inputs and outputs, and also increases the complexity of the model. We make use of a multilayered perceptron, MLP, which is a network consisting of different layers, each with multiple perceptrons. Between layers the perceptrons are all interconnected, as shown in figure 2.4.

Depending on which perceptron is considered, different activation functions are used. A linear activation function is used for the inputs, allowing them to pass through without change. For the "hidden layers" between the input and output layers, a non-linear activation function is used to capture non-linear behavior of the model. Typically ReLu is the activation function used for such problems, and as we don't have any special structure or data we will be using it as starting point [26].

In this project, both a classifier and a regressor will be used. For classifiers, we want a probability as output that can be assigned to match either of two classes, 0 and 1, depending on the threshold set. To achieve this a sigmoid activation function was used for the output layer. For regressors we will use a linear output activation function instead, to be able to get any range of output values.

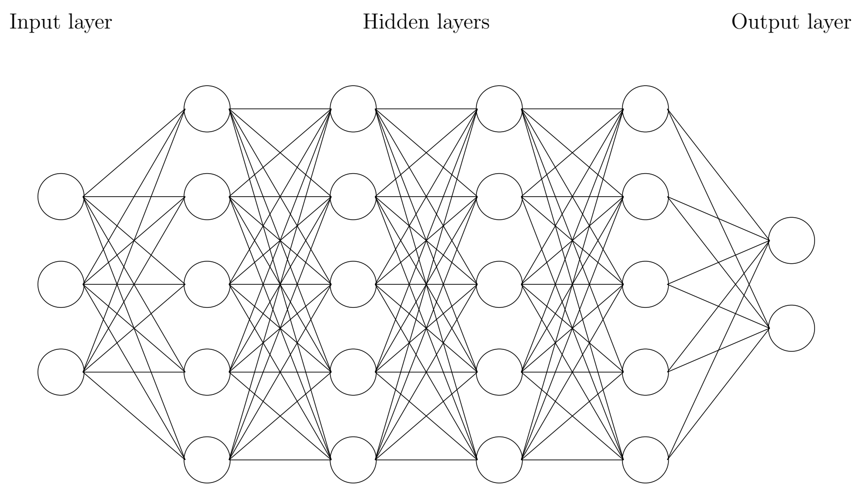


Figure 2.4: Deep multilayered perceptron. Each node represents a biased perceptron and is connected to all nodes from previous and next layers. The classifier and regressor models used will have 17 inputs matching the QLK17D variables. The classifier will have one output node, while the regressor will have two outputs .

In order to train a neural network, predicted values and the actual values are compared. The difference is quantified by a loss function. For a regressor, this can be achieved using the mean squared error. For a classifier, as the points trained on are all either 0 or 1, better performance is obtained when using binary-cross entropy.

A regressor model can be used to not only give an output value, but also a measure of uncertainty. To do so a Gaussian negative log-likelihood loss is used together with a second output for the regressor. The use of this structure allows for one of the outputs to be interpreted as the mean, and one to be interpreted as the variance [27].

When training, the weights are updated using backpropagation. Data is passed through the model to make a prediction, then the loss function is used to determine the loss of the prediction compared to the actual values. When the loss is minimized, the model's parameters are adjusted. An optimizer is used to adjust the weights. The learning rate and regularization terms of the optimizer are adjustable hyperparameters which can be adjusted to prevent overfitting of the model by penalizing large weights.

During training we will be using a training and validation dataset. Next to this, a test dataset will be kept completely separate for evaluation of the model's performance. The training dataset can be divided up into mini-batches which are passed through the model one by one, instead of training on the full dataset. This allows for more updates to the gradients, faster training, and better generalization [26]. The validation dataset is used for tuning of the model's hyperparameters and determining when to stop training.

To further improve performance, an ensemble of models can be used. One such method is bagging, whereby multiple models are trained in parallel. As these models all have different initialization parameters, this leads to slightly different trained weights, building noise resistance into the model. It has been shown that making each MLP in the ensembles "deep" by using many hidden layers, allows for better generalization when using adequate regularization. Using ensembles also allows us to get a measure of the uncertainty on the predicted flux [28].

For a deep ensemble, the mean of the ensemble is the mean of the individual networks,

$$\mu = \sum_{i}^{N} \mu_i \tag{2.25}$$

and the variance of the ensemble can be found using,

$$\sigma = \frac{1}{N}\sum_{i}^{N}\left(\sigma_i + \mu_i^2\right) - \frac{1}{N}\sum_{i}^{N}\mu_i^2 \tag{2.26}$$

where it is assumed that the ensemble is a uniformly-weighted mixture model [29]. For the ensemble of classifiers, the mean of the ensemble is the mean of the individual probability outputs of the classifiers.

As deep ensembles have much more parameters for fitting, they are more prone to overfitting. Use of proper regularization is therefore essential. Adding an L2 penalty term to the loss function to penalize large weights is one way of preventing overfitting. Another possible regularization technique is the use of dropout layers. During training a percentage of all weights in the hidden layers are randomly set to zero to add noise to the neural network. This has been shown to reduce overfitting and works better than using L2 weight regularization alone [30]. Once training is complete all weights are then used for evaluating the model.

To determine after how many epochs to stop training the model, early stopping can be used. The validation loss is computed for each training step and training is stopped after a model has not improved for a set number of consecutive iterations. The model with the lowest validation loss is then used.

In order to understand how a model is performing, and to be able to compare different models, using loss as metric is not sufficient. Comparing the neural network's predictions to the validation dataset allows us to calculate the coefficient of determination, R^2, for the mean output of the regressor.

The R^2 value is a statistical measure that shows the improved performance of your model compared to just using the mean of the actual values as your model. Ranging from 0 (using the mean as the model) to 1 (the model perfectly describes the data). It is calculated as

$$R^2 = 1 - \frac{\sum_{i}^{N}(y_i - f(\boldsymbol{x}_i))^2}{\sum_{i}^{N}(y_i - \bar{y})^2} \tag{2.27}$$

where y_i are the actual values, $f(\boldsymbol{x}_i)$ are the estimated values using the machine learning model, and $\bar{y}$ is the mean of the actual values [31]. The classifier will be evaluated using the f1 score,

$$\mathrm{f1} = \frac{2}{\frac{1}{\mathrm{Precision}} + \frac{1}{\mathrm{Recall}}} \tag{2.28}$$

where recall,

$$\mathrm{Recall} = \frac{\text{True positives}}{\text{True positives} + \text{False negatives}} \tag{2.29}$$

and precision,

$$\mathrm{Precision} = \frac{\text{True positives}}{\text{True positives} + \text{False positives}} \tag{2.30}$$

are derived from the confusion matrix. The f1 score gives us a general sense of how well the classifier performs, combining both precision and recall to prevent class imbalances from impacting the metric. This is relevant for active learning as an imbalanced class can result from the acquisition of new points. Next to the f1 score, we will be looking at recall separately. In our use case, any false negative points will lead to unstable points not being sent to the regressor at all, instead being labeled as stable. This is undesirable as we are specifically interested in unstable regions.

These performance metrics will be used for our models and be calculated for the validation and test datasets.

In practice, training of a QuaLiKiz neural network requires a lot of points. For example, the network in [6] used $2 \cdot 10^7$ points for a QLK15D network. Not only do we wish to use QLK17D inputs and preferably even more dimensions, but we also wish to use higher-fidelity codes such as GENE in future projects [32]. Reducing the number of points required for training is therefore essential as each point costs more computation time when using higher fidelity codes. It has been shown that the number of points needed for training can be reduced by using active learning [7].

2.4 Active Learning

Active learning is a technique used when labeling a large training dataset is computationally expensive. Instead, a pool of unlabeled data points is created, from which a batch of points most informative for training is selected. This batch is sent to an oracle, which then returns labeled data to the neural network for training.

There are different ways of selecting which points to acquire labels for. A common method is selecting the points from the unlabeled pool predicted to have the highest uncertainty. More complex methods exist, notably methods that find an optimal batch to send to the oracle, preventing the labeling of extra points that lie close to each other in parameter space [33].

The selected points are removed from the unlabeled pool and added to the training dataset. The neural network is then retrained again on the complete training dataset. After completion of one iteration, this process repeats until the performance of the model is sufficient. From literature it was found that R^2 and f1 scores of 0.95 were sufficient when modeling QuaLiKiz [7].

It is expected that the gains from active learning, compared to random sampling, will become increasingly small as additional points are added to the training dataset. To compensate for this, the number of points added to the training dataset through active learning will be increased.

From previous work it has been shown that the performance obtained when using the existing $2 \cdot 10^7$ point QLK15D dataset, can be achieved using just $2 \cdot 10^5$ points with active learning [7]. The model generated is however not trained over all parameter space we wish to consider. Coupling the active learning pipeline directly to QuaLiKiz will remove dependence on having an existing dataset. A matching large unlabeled pool can easily be generated over the space we wish to consider. This increased coverage and density of points allows the active learning method a lot more flexibility in it's choice of points.

We wish to obtain a model that is capable of predicting outputs in all relevant parameter space, however it is unclear how adjusting the distribution of the unlabeled pool and training seed affects the points selected through active learning. An idea to further reduce the amount of points needed is to give the active learning pipeline an unlabeled pool and training seed that have distributions matching typical input points for real plasma conditions. In this way the pipeline should start off with higher density of training points at typical plasma regimes, leading to faster convergence of the model. It must be verified that altering the distributions of the unlabeled pool and training seed in this way does indeed lead to a smaller dataset needed for similar performance.

Chapter 3

Methodology

Two different datasets were used for this project. The first was an x dependent multivariate Gaussian and the second was a uniform distribution. These inputs were then passed through QuaLiKiz and a series of filters to get usable output fluxes which were used as labels for training. In this section, we will consider the distributions of the datasets after all points have passed through QuaLiKiz and its filters, focusing on the training seed datasets.

3.1 Gaussian Dataset

This dataset is derived from a statistical analysis of experimental results. For each QLK17D variable the mean, standard deviation, bounds and correlations between variables are determined as functions of x, where x is the normalized distance from the core of the tokamak to the last closed flux surface near the edge. When x increases, and the edge is neared, many outside factors such as the engineering of the tokamak, impurities, and the cold wall of the tokamak increasingly affect the plasma. This leads to a larger variance of the plasma variables compared to the core. From analyzing experimental data, a multivariate Gaussian as a function of x was derived.

Data points were generated by randomly picking points within the bounds. Their probability of existing followed from the multivariate Gaussian distribution function, which is compared to a random number before selection. The result, after passing these points through QuaLiKiz and it's filters, is shown in figure 3.1. The distributions of the fluxes and the unlabeled pool are shown in appendix B. The points in the test dataset are filtered out beyond the 95% percentile. In this way a uniform test dataset could be constructed on the exact same domain with adequate density of points.

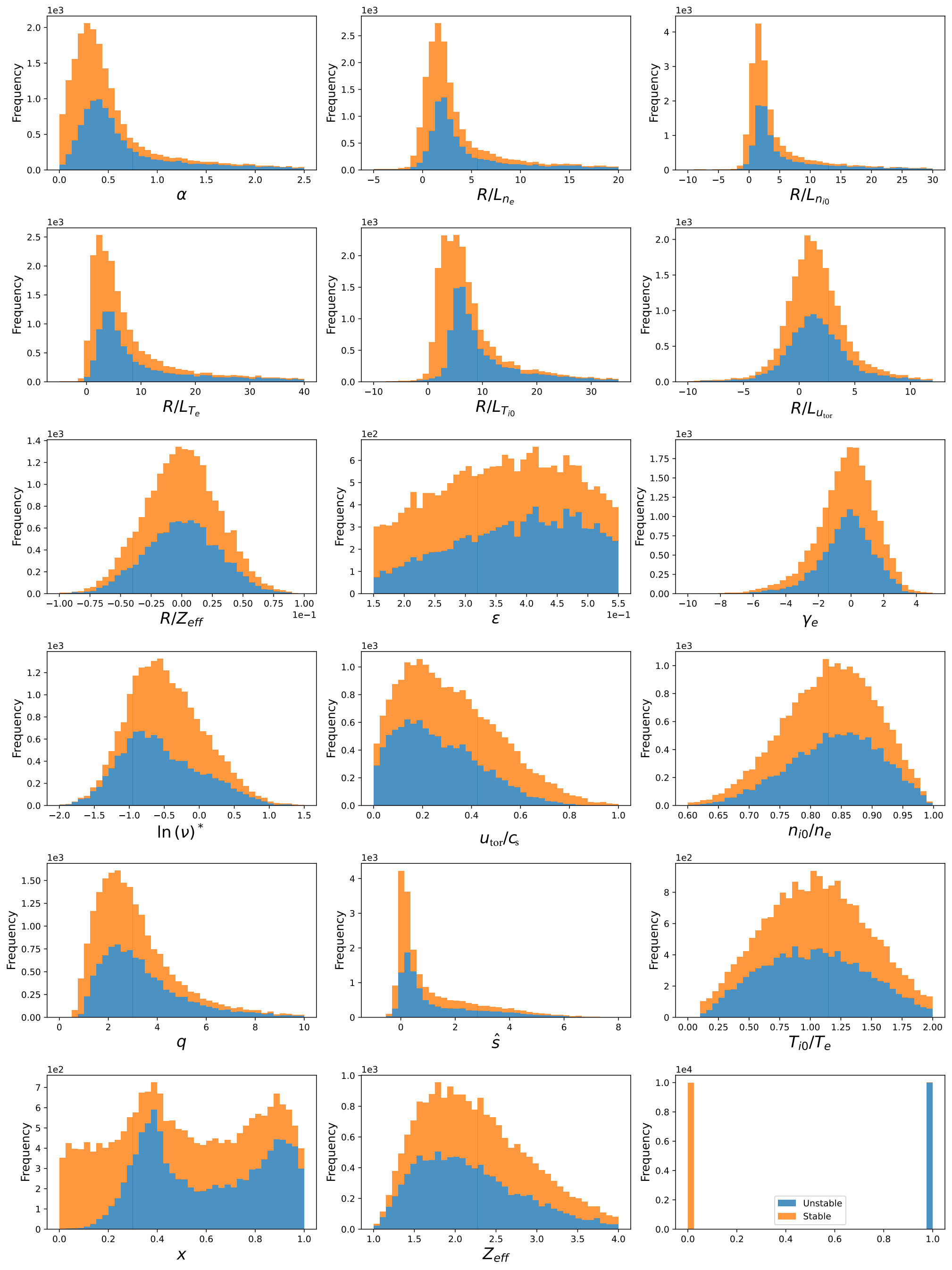


Figure 3.1: Gaussian training seed dataset, input variable 1D distributions after passing through all QuaLiKiz filters. The dataset was balanced between stable and unstable points using q_i from the output fluxes. Validation and test dataset distributions were visually identical.

3.2 Uniform Dataset

In order to compare the effect of using different distributions, a uniform dataset over a similar domain for all variables in 1D as the Gaussian dataset was generated. The domain was set to the 95% percentile limits of the Gaussian dataset, so that almost the full domain was covered, while still keeping adequate density in the regions relevant for fusion. Random points were then generated within this domain.

Unphysical combinations of points were filtered out using the pre-QuaLiKiz filter described in section 3.5. New points were then added iteratively, until a uniform distribution resulted, as observed through 1D histograms.

The resulting 1D distributions are shown in figure 3.2. Noticeable are the higher proportion of stable points at low x, near the core, as well as higher proportions of unstable points at higher values of $\hat{s}$ and $log_{10}(\nu^*)$, where turbulent behavior is disrupted.

Higher values of n_{i_0}/n_e led to a significant and abrupt decrease in the number of points. This behavior is also seen in the unlabeled pool in figure 6.2. This is due to points being filtered out due to negative densities of one of the ion species. For a more uniform distribution, more points can be sample in this region. However, as for this project we are interested in the effect of different distributions on active learning, it is only essential for the distributions to be noticeably different from each other.

3.3 Dataset Usage

For both the Gaussian and uniform datasets, unlabeled pools of 10 million points were generated. A training seed of 20000 points, and validation and test datasets of 10000 points were generated by passing points of both distributions through QuaLiKiz and it's filters.

It is important to note that these datasets have been artificially balanced and that most points are typically stable. Methods capable of using unbalanced datasets do exist. However, it we choose to start active learning with balanced datasets to give the classifier a good starting point for the locations of the stable-unstable boundary. As classifiers are further trained on points acquired for the regressor, this reduces the likelihood of unstable regions being mislabeled as stable and being missed completely.

Balancing of datasets can cause some regions to be relatively oversampled and other regions to be undersampled. This is not necessarily a problem. It is first and foremost important for QLKNN to distinguish stable and unstable points. From there, only the values predicted for the unstable points are relevant as the stable points all have zero output flux.

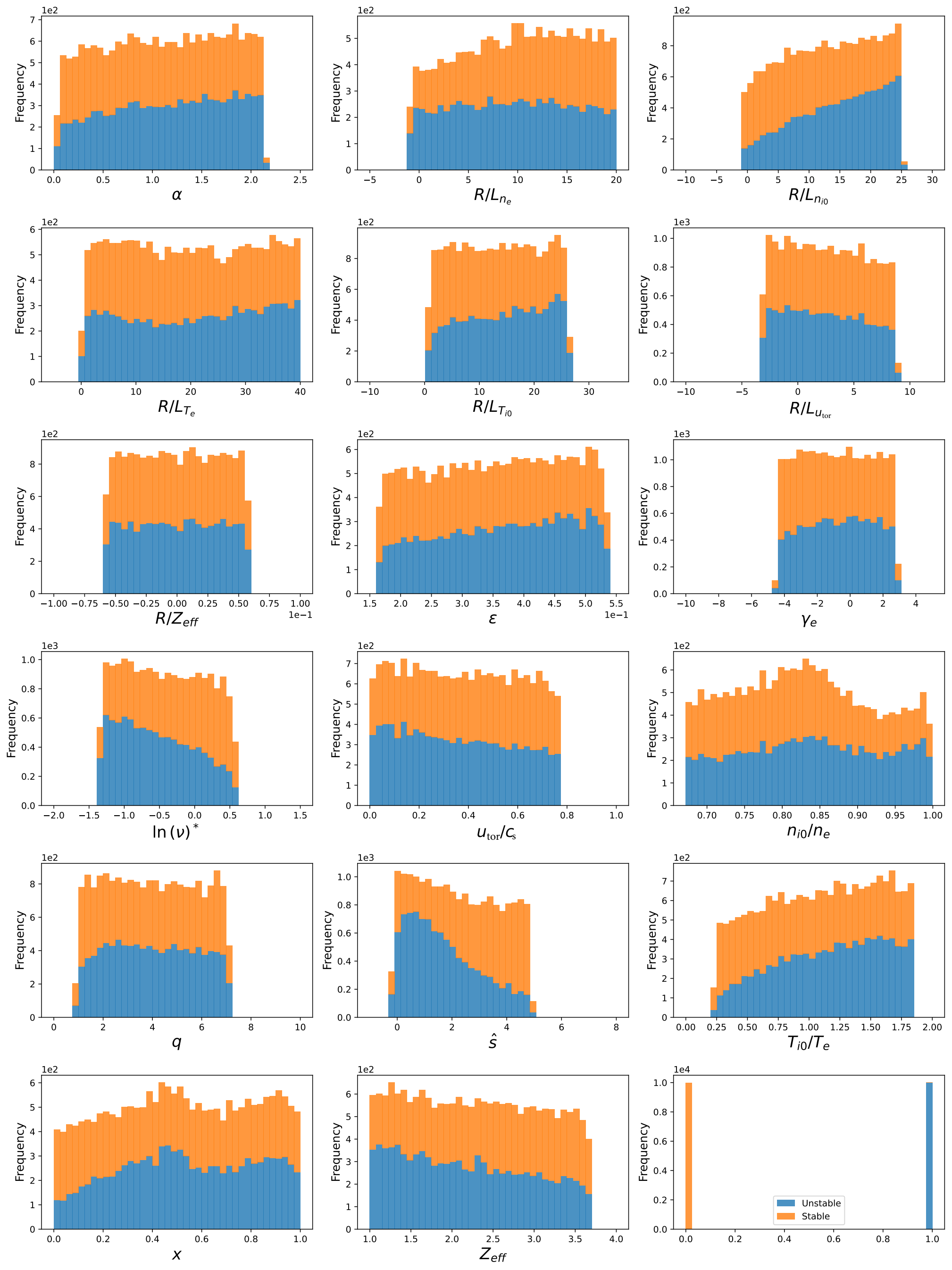


Figure 3.2: Uniform training seed dataset, input variable 1D distributions after passing through all QuaLiKiz filters. The dataset was balanced between stable and unstable points using q_i from the output fluxes. Validation and test dataset distributions were visually identical.

3.4 Machine and Active Learning

Both the machine and active learning implemented in this project used the pipeline developed in [7] as starting point.

For each neural network, a network with six hidden layers with 512 nodes per layer was used. An ensemble of five of these models was used for each output flux for the regressor One extra ensemble was used for the classifier, which was trained on the largest output flux, q_i.

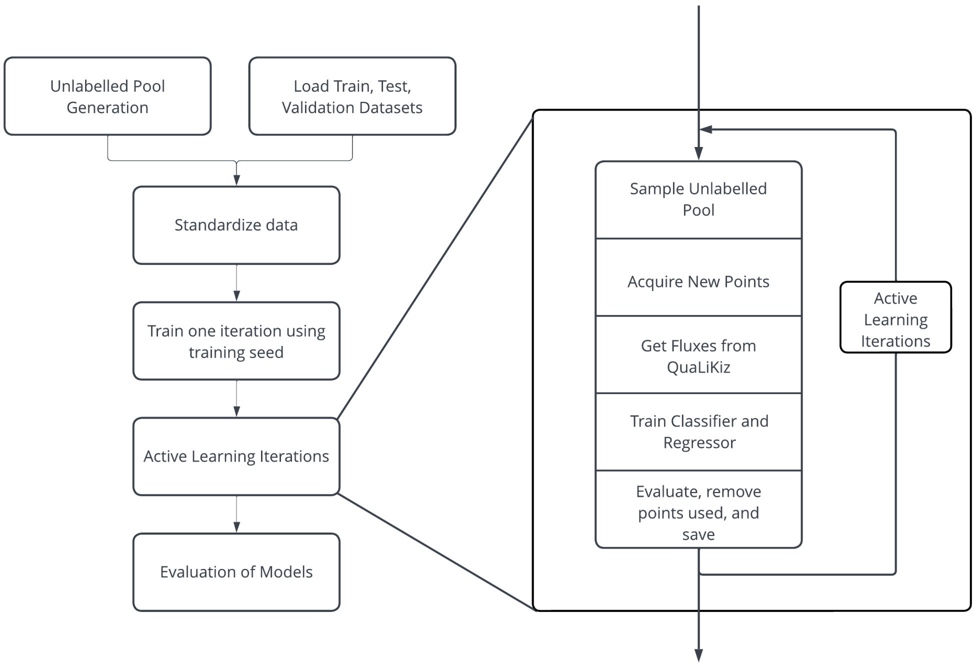

Unlabelled Pool Generation
Load Train, Test, Validation Datasets
Standardize data
Train one iteration using training seed
Active Learning Iterations
Evaluation of Models
Sample Unlabelled Pool
Acquire New Points
Get Fluxes from QuaLiKiz
Train Classifier and Regressor
Evaluate, remove points used, and save
Active Learning Iterations

Figure 3.3: Structural overview of the active learning pipeline. In this project code was written to get fluxes from QuaLiKiz, and for data generation. Other sections, such as the training routine, were adjusted from [7].

After loading in the datasets, they were standardized by removing the mean and scaling to unit variance, as shown in figure 3.3. Both the classifier and regressor were trained for a first iteration on the training seed of labeled data. Once trained, the models were reloaded for the next iteration and used to find points to acquire labels for through QuaLiKiz. To reduce excessive calculations in the acquisition phase, a sample of $5 \cdot 10^5$ points was taken from the unlabeled pool. The regressor from the previous iteration was used to find highest uncertainty on predicted fluxes. Although [7] used the sum of the maximum variance from the regressor over the fluxes as the measure of uncertainty, this led to points only being selected near the outside bounds when using the uniform datasets. Instead, the sum of the highest relative error in the output flux,

$$\epsilon_{rel} = \frac{1}{5} \sum_{i}^{5} \left| \frac{\sigma_i}{\phi_i} \right| \tag{3.1}$$

was used, where σ is the variance from the ensemble (including the variance from each individual regressor), ϕ_i is an output flux, and the summation is over the number of output fluxes. The points with the largest variance were kept.

Table 3.1: Overview of relevant machine learning settings used for all neural network training performed.

Parameter	Value
Ensemble size	5
MLP hidden layers	6x512
Hidden layer activation function	ReLU
Dropout per hidden layer	10%
Optimizer	AdamW
Learning rate	$1 \cdot 10^{-4}$
L2 regularization	$1 \cdot 10^{-4}$
Early stopping patience	20 steps
Mini-batch size	128

Table 3.2: Active learning settings used for acquiring new points from QuaLiKiz.

Parameter	Value
Acquisition	through equation 3.1
Points acquired	512 regressor, 1024 classifier
Doubling of points acquired	30 iterations

A batch of points with the largest uncertainty were sent to QuaLiKiz. Before and after QuaLiKiz, they were filtered as described in section 3.5. To compensate for the points filtered out, extra points were sent into QuaLiKiz.

Although not specifically tailored for the classifier, these points were also used to train it. As many sampled points for active learning ended up being stable due to poor performance of the classifier at the start of the pipeline, these stable points were passed to the classifier for a total active learning batch size of 1024.

The number of points acquired was doubled every 30 iterations as active learning was only expected to have a significant effect at the start of training [7]. These points were removed from the unlabeled pool and added to the training dataset on which the regressor and classifier models were then trained. Both the regressor and classifier used AdamW as the optimizer. This is an extension of the popular Adam optimizer that improves the implementation of L2 regularization which allows the final model to generalize better [34]. Early stopping was applied to determine when a model should stop training. As each model within the deep ensembles has a different initialization, it was applied to each individual model within the deep ensemble. The criteria for stopping training was set to a maximum of consecutive training steps in which no improvement of the validation loss occurred [26]. The threshold of the classifier was set for the ensemble as whole after each active learning iteration using Youden's J statistic to find the ideal threshold.

The maximum number of training steps was set at 350 epochs but this was never reached. Dropout layers were included during training and the size of mini-batches was decreased from 512 to 128, which adds noise to the training process and provides a regularizing effect [26]. The pipeline already included an L2 regularization term of $1 \cdot 10^{-4}$ which was left in. An overview of the settings used for machine learning and active learning are given in tables 3.1 and 3.2. After training, the deep ensembles were evaluated separately on both test datasets. The models, the training dataset and unlabeled pool were saved to be used in the next iteration.

Table 3.3: QuaLiKiz filters applied before sending points to QuaLiKiz. These check that the combination of input variables are physically possible.

Filter	Value	Description
Ion density filter	-	Filters out points if any ions have negative density
α filter	-	Checks if $\lvert\boldsymbol{B}\rvert$ calculated from α is real

Table 3.4: QuaLiKiz output filters, applied after QuaLiKiz provided labels. The results calculated internally by QuaLiKiz are verified to be consistent through two different methods. The remaining filters check the points are physically possible.

Filter	Value	Description
Γ_e filter	$\pm$ 5%	Flux filters compare convergence between flux directly calculated and calculated from convection and diffusion terms
Γ_i filter	$\pm$ 10%	
q_e filter	$\pm$ 5%	
q_i filter	$\pm$ 10%	
Negative heat flux filter	-	Negative heat flux is filtered out
Sum of modes flux filter	$\pm$ 50%	Max factor between ITG+ETG+TEM and total flux.
Ambipolar filter	$\pm$ 10%	Checks that the transport of ions and electrons is balanced and that movement of charge is conserved,
Rounding error filter	10^{-4}	Limit under which small fluxes are assumed to be zero.

3.5 QuaLiKiz Integration

The latest version of QuaLiKiz, 2.8.4, was used to acquire labels. This version improved impurity rotation physics, the collision operator, integration methods, and fixed a variety of bugs compared to version 2.6.2. which was used in previous works [25].

During integration of QuaLiKiz into the active learning pipeline, many points caused failures within QuaLiKiz. By adding two filters, depicted in table 3.3, that check the combination of input parameters sent into QuaLiKiz, clearly unphysical combinations of input parameters were filtered out beforehand using a density and α filter. The density filter checks if no negative ion densities result from the combination of inputs. The α filter makes use of the equation for α,

$$\alpha = -\frac{2}{\mu_0}\frac{q^2 R}{B^2}\sum_s \left(T_s \nabla n_s + n_s \nabla T_s\right) \tag{3.2}$$

and calculates $\lvert\boldsymbol{B}\rvert$ from the value of α in the input dataset. As ∇n_s and ∇T_s can also be negative, some combinations of variables lead to an imaginary $\lvert\boldsymbol{B}\rvert$, which are filtered out.

After QuaLiKiz, the same filters were used as in [6]. The capped heat flux filter, which removed points with high output fluxes, was removed for this project. We now want to generate a QLKNN version that can simulate points everywhere in the given 17D space. Points with large output flux were undesirable in previous projects due to their high absolute variance and were filtered out. As this project opted for using relative variance, this is not expected to be a problem and these points can be kept. An overview of the output filters is given in table 3.4. For future projects, all points generated were automatically appended to a large database of QuaLiKiz points.

It was found that including more input parameters allows for a better performing model [6]. As active learning allows for the use of less input data, while still keeping high performance, it is now feasible to construct a 17D model and expand the input domain to include more plasma experiments. In order to compare the effect of different distributions on the trained model the following experiments were performed.

3.6 Mixed Distributions

A uniform distribution is typically used for machine learning applications. Information about the application can be added to the training seed and unlabeled pool through adjustment of their distributions. To see how this affects the number of points to acquire, we generated models through active learning using different combinations of datasets.

Table 3.5: Distributions of the unlabeled pool and training seed datasets used for the models trained for this project.

Run name	Unlabeled pool	Training seed	Validation dataset
Full Gaussian configuration	Gaussian	Gaussian	Gaussian
Full uniform configuration	Uniform	Uniform	Uniform
Uniform seed configuration	Gaussian	Uniform	Gaussian
Uniform unlabeled pool configuration	Uniform	Gaussian	Gaussian

If the active learning pipeline is not affected by changes in the distribution of the training seed and unlabeled pool, similar points will be selected, leading to similar models. Each dataset covers the same domain and has points covering the same regions in one dimension, but with different densities. First a model was generated using uniform unlabeled pool, training seed and validation datasets, as shown in table 3.5, which we will refer to as the full uniform configuration. For comparison a model with a full Gaussian configuration was also generated, where all distributions used were Gaussian.

Although there are eight possible combinations, only four will be tested due to the large amount of computing time needed to train a model through active learning. The effect of altering the distribution of the unlabeled pool used will be considered by training a model using a uniform unlabeled pool and a Gaussian training seed and validation dataset.

The effect of altering the distribution of the training seed will also be considered by training a model using a Gaussian unlabeled pool, uniform training seed and Gaussian validation dataset, and comparing it's performance to the model trained using a full Gaussian configuration.

3.7 Dataset Comparison Techniques

After training all models, the distribution of the data selected through active learning will analyzed both visually and through the use of a distance metric between distributions. First 1D histograms will be generated, then the distance between the distribution of the cumulative training dataset of acquired points and the distributions of both the unlabeled pool and the training seed of the marginal distributions will be calculated. For each iteration this is repeated. This not only gives us information on the influence of either the unlabeled pool or the training seed on the data selected, but it also allows us to understand how the data selected changes from early to late iterations.

The training dataset at each iteration, as well as the unlabeled pool and training seed datasets were standardized according to the uniform dataset. A comparison is made to the cumulative training dataset as the active learning batch size used led to noisy distributions.

3.7.1 Jensen-Shannon Divergence

Many different distance metrics exist, each with their own benefits and drawbacks. The first considered is the Jensen-Shannon divergence. It is a symmetrical version of the Kullback–Leibler (KL) divergence and always has positive values. The KL-divergence is defined as,

$$D_{KL}(P||Q) = \sum_{x \in \chi} P(x) \log\left(\frac{P(x)}{Q(x)}\right) \tag{3.3}$$

where χ represents all sample space. As this measure is not symmetric and doesn't work with zero bin values, it isn't well suited as a distance metric. The Jensen-Shannon divergence can be derived from the KL-divergence through,

$$D_{JS}(P||Q) = \frac{1}{2} D_{KL}\left(P||M\right) + \frac{1}{2} D_{KL}\left(Q||M\right) \tag{3.4}$$

where $M = (P + Q)/2$. This results in a symmetric distance metric. The benefits of using the Jensen-Shannon divergence are that it works well with Monte-Carlo methods and can be extended to higher dimensions. However, it does not give a good measure when distributions are not overlapping.

3.7.2 Wasserstein Distance

For each 1D histogram, the distance between two datasets will also be calculated using the Wasserstein (Earth mover's) distance as it takes into account the distance between sections of data when they are not overlapping. The Wasserstein distance measures how much one distribution has to be moved to fit the other [35], as visualized in figure 3.4.

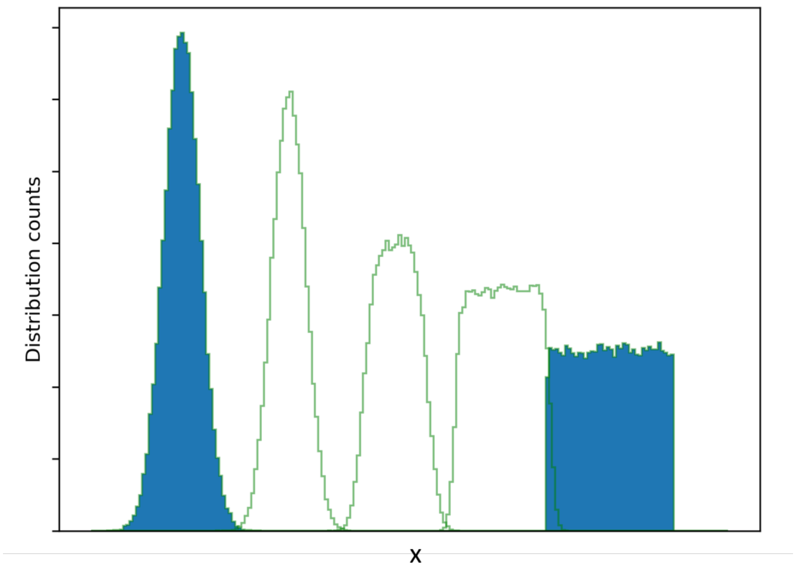


Figure 3.4: Visual representation of the Wasserstein distance, the minimal distance required to move from one distribution to another. This can be illustrated as the transforming from one distribution into another, for example from a normal to a uniform distribution where intermediate steps are shown as outlines.

When moving a normalized probability distribution $\mu(x)$ at point x to a distribution $\nu(y)$ at point y, we can set up a cost function $c(x, y)$ that quantifies the cost of moving a unit of the probability

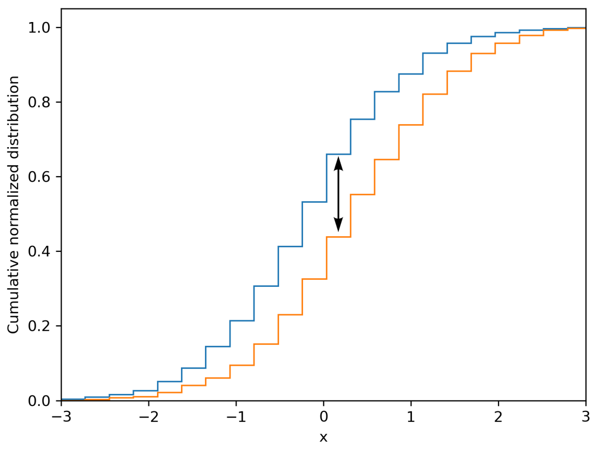


Figure 3.5: Kolmogorov-Smirnov distance measured between two normalized cumulative distribution functions. x is used as an example.

distribution from x to y. For a moving path $\gamma(x, y)$, the Wasserstein distance is given by,

$$C = \inf_{\gamma \in \Gamma(u,v)} \int c(x, y) d\gamma(x, y) \tag{3.5}$$

where the cost function is the distance moved and $\gamma(x, y)$ is the amount of distribution moved. $\Gamma(u, v)$ represents all possible transport plans, within which γ is the optimal one. Through linear algebra and using this cost function, the calculation of all possible transport plans is greatly simplified, allowing the optimal plan to be found.

Although the Wasserstein distance has many benefits, one of it's drawbacks is that it generalizes poorly to higher dimensions.

3.7.3 Kolmogorov–Smirnov Statistic

The third distance metric, the two-sample Kolmogorov-Smirnov test, was used to confirm the result found using the previous two methods. This test takes a completely different approach and compares the cumulative probability curves of two distribution samples. It returns the Kolmogorov-Smirnov statistic, which is the maximum absolute difference between the two curves. It is important that the distributions cover the same domain along the x-axis, as shown in figure 3.5. A benefit of this technique is that the probability that the two distributions are samples of the same underlying probability distribution is also returned. However, next to the limitation that the distributions must lie along the same axis, the ks-statistic is difficult to generalize to higher dimensions due to the different combinations available for computing the cumulative distributions.

For each iteration, these distance measures were recorded giving an evolution of the distance between the distributions. To reduce the 17 resulting values to one, the sum over the normalized distance between the two distance metrics for all the variables was calculated. As they were both uniformly or close to uniformly distributed, x and ϵ were excluded.

Chapter 4

Results

The results obtained throughout this project are presented in this section. Starting with how the pipeline interacts with QuaLiKiz and it's filters, we then look at the training of deep ensembles, before moving on to the main focus of this study: the effect of using different distributions on the performance of the trained models. The points acquired through active learning are then further analyzed by considering their distribution. The distance between this distribution and the distributions of the unlabeled pool and training seed datasets are then quantified and used in further analysis.

4.1 QuaLiKiz Filters

While generating the labels for the uniform training seed dataset, the response of QuaLiKiz and it's filters was recorded. The distribution of the unlabeled dataset is given in figure 4.1 and the distribution of the output points is given in figure 4.2.

The strong filtering of points at high values of n_{i_0}/n_e and at low values of Z_{eff} is due to the input ion density filter (see appendix C, figure 6.5). With the fixed choice of impurity species, some combinations of points led to negative densities which were filtered out. The resulting loss of points was compensated by generating more points. The output filters are responsible for filtering out unstable points for high values of $\hat{s}$. High values of shear prevent turbulent eddy currents by breaking them up before they can fully form, making unstable points very unlikely to exist. From these figures we also see that points near the core are more likely to be stable. A large number of generated points were filtered out as shown in table 4.1. Notably the amount of points filtered out by the consistency filters was much lower compared to preceding work [6]. This was expected with the use of the much newer QuaLiKiz 2.8.2. However, a large number of points were now being filtered out by the ambipolar filter following them. The calculation of impurity fluxes when using non-zero rotation does not always work as expected within QuaLiKiz and could be the reason these points were generated. Overall the total number of points filtered out is similar to previous work [6].

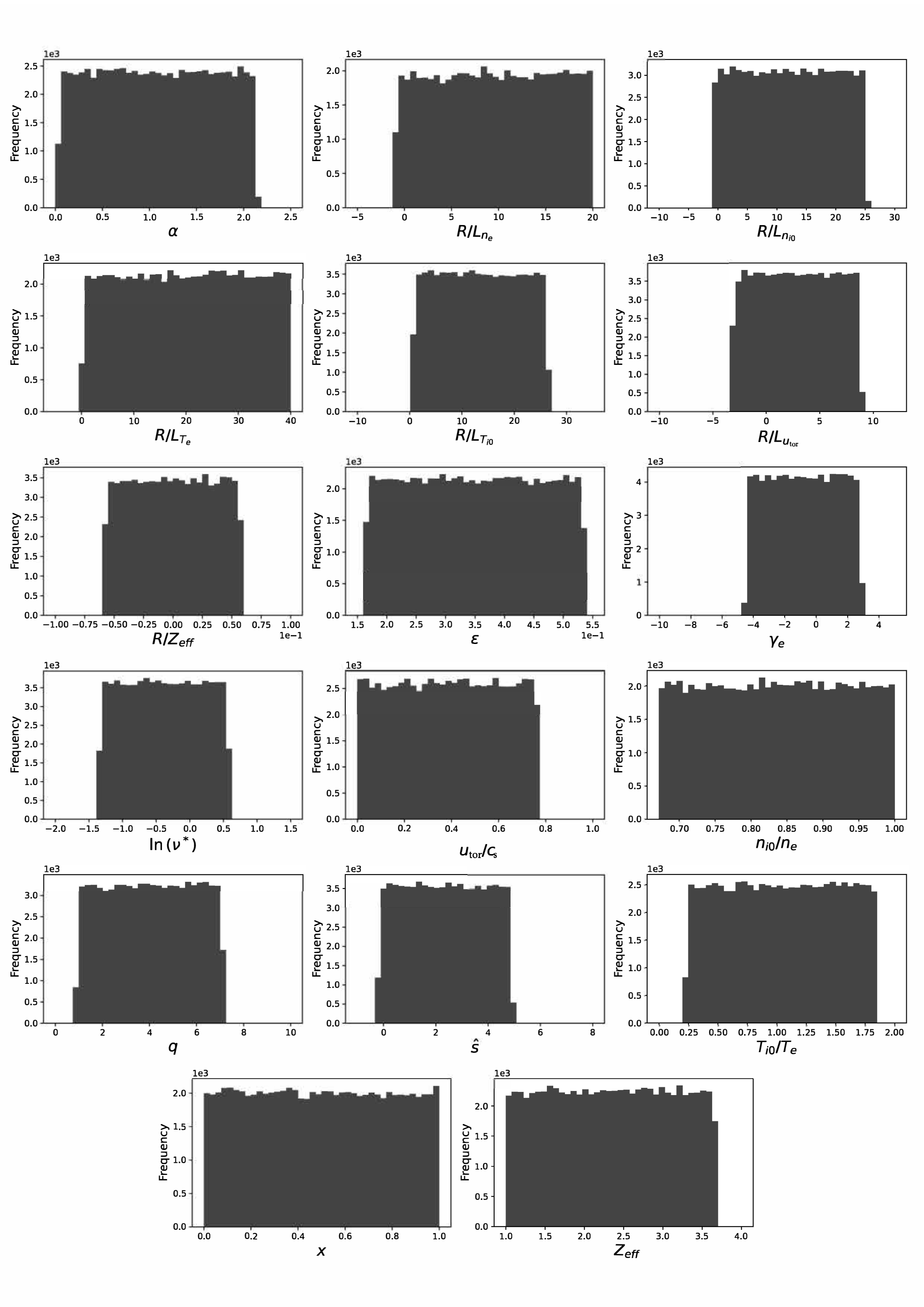


Figure 4.1: Uniform 1D distributed training seed dataset before passing through the input filters, QuaLiKiz and the output filters. QuaLiKiz allows for stable and unstable labeling so differentiating between classes is not yet possible in this figure. After setting bounds based on JET data, points were randomly generated within them.

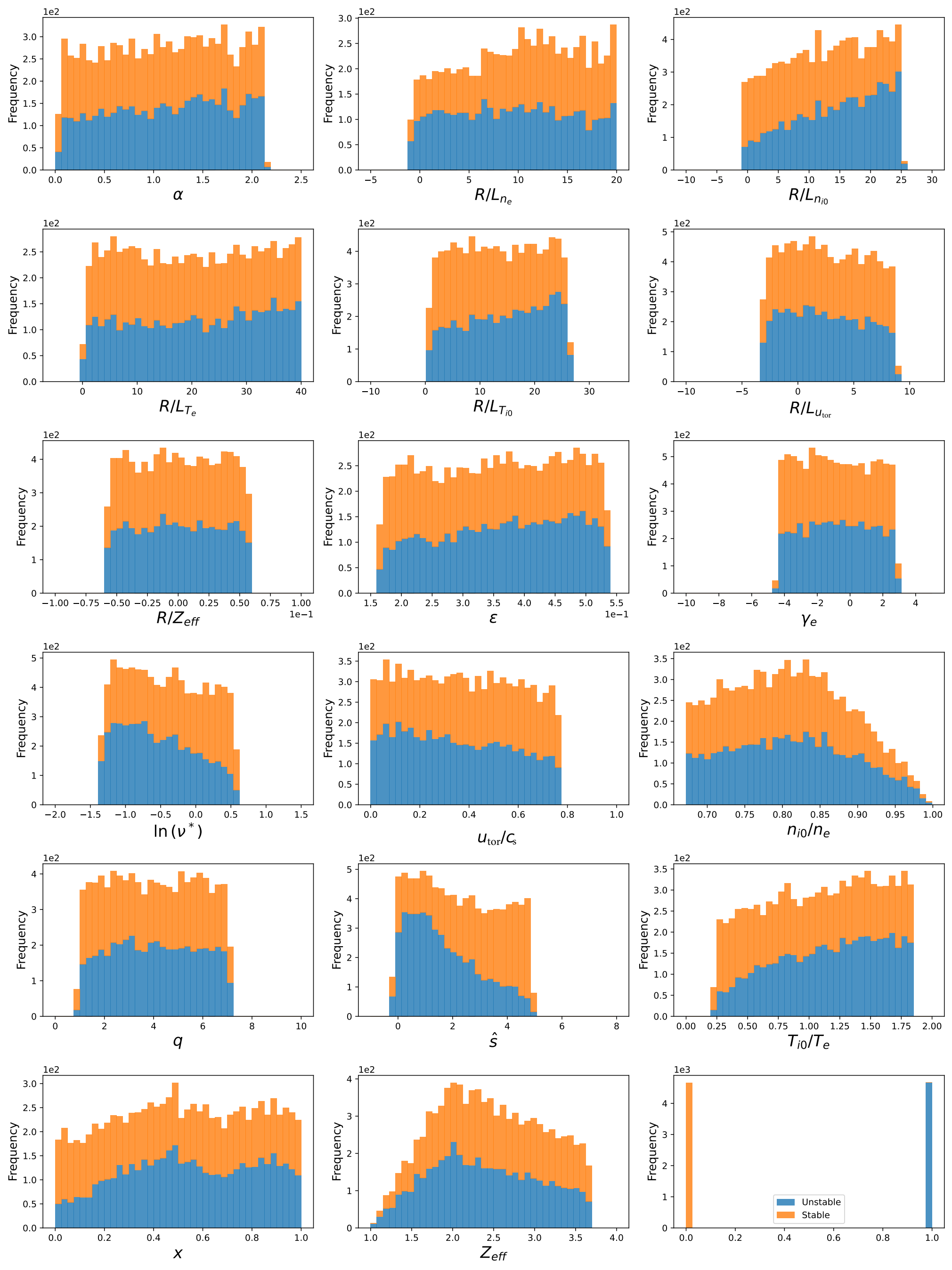


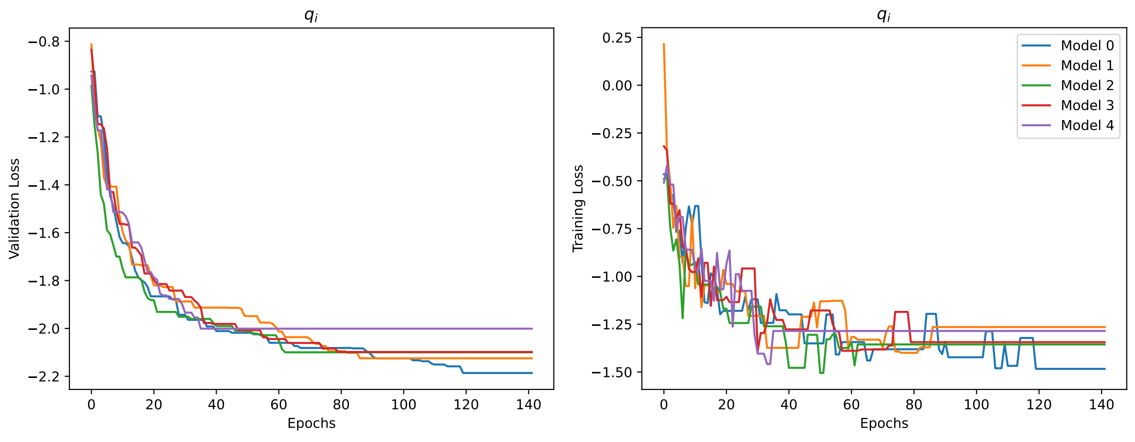
Figure 4.2: Uniform 1D distributed training seed after passing the dataset through QuaLiKiz and it's filters. The filtering at high values of n_{i_0}/n_e and at low values of Z_{eff} was due to the input ion density filter (see appendix C, figure 6.5). The α input filter rarely filters out any points so any other changes are due to the output filters.

Table 4.1: Average loss of data due to the use of QuaLiKiz filters when using uniformly distributed datasets. Similar data loss was observed when using datasets with different distributions, with total losses up to 44% for some runs. Density and α filters were already applied to datasets generated and therefore did not filter out extra points.

Filter	Data lost	Total lost
Γ_e filter	6.3%	6.3%
Γ_i filter	1.5%	7.7%
q_e filter	1.4%	9.0%
q_i filter	6.4%	14.8%
Negative heat flux filter	6.2%	20.1%
Sum of modes flux filter	1.5%	20.3%
Ambipolar filter	15.7%	33.7%
Rounding error filter	0.6%	34.0%

4.2 Machine Learning

A new deep ensemble was trained on the full training dataset after each iteration. Early stopping was implemented on each of the individual models within the ensemble. Taking a look at the loss curves of the last model trained by the pipeline, the loss curves for the each model within the ensemble as well as the loss curves of the deep ensemble itself are shown in figures 4.3a and 4.3b. From these figures it is clear that the neural networks are learning as the loss is decreasing for 60 to 70 epochs before early stopping is triggered.



(a) Validation loss **(b)** Training loss

Figure 4.3: Loss of regressor deep ensemble per model in the ensemble. Early stopping can be seen to trigger separately for each model. The noise in the training curves originates from the small mini-batch size used. Data from iteration 100 of the full Gaussian configuration.

4.3 Effect of Dataset Distributions

In this subsection we look at the results obtained when training deep ensembles based off of different distributions. The four combinations of datasets used are given in table 3.5. For each trained model we obtain both classifier and regressor deep ensembles from the pipeline. The classifiers already performed well being trained on just the training seed dataset as shown in figure 4.4 for the full uniform configuration evaluated on the uniform test dataset. Only the regressor will be considered in depth in this section as the changes to the distribution used led to very similar performance changes for the classifier (relevant figures 6.7 and 6.8 can be found in appendix D).

The improvement of the R^2 metric for the full uniform configuration evaluated on the uniform test dataset is shown in figure 4.5. From this figure we observe that the points acquired through active learning improve the model as expected.

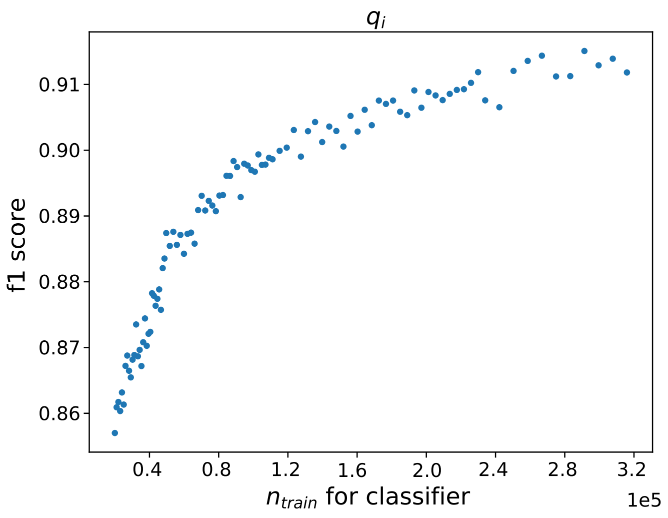


Figure 4.4: Improvement of the f1 score of the classifier deep ensemble when evaluated on the uniform test dataset. Points were acquired through active learning on the regressor of the full uniform configuration and were passed to the classifier as extra data training. Starting from an initial seed of 20000 points, a new deep ensemble was initialized and trained after each active learning iteration.

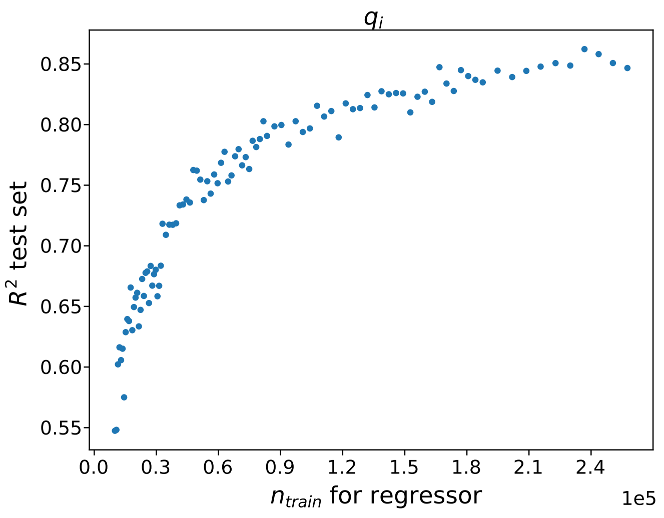
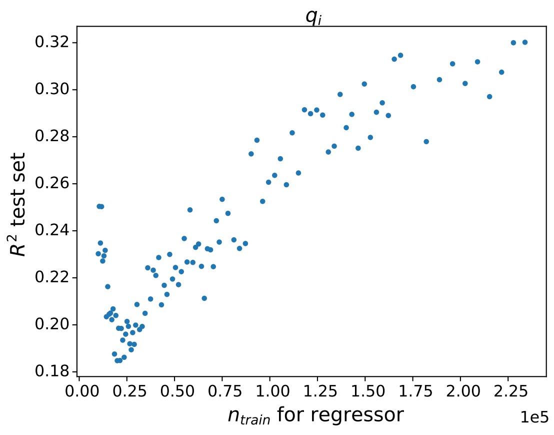


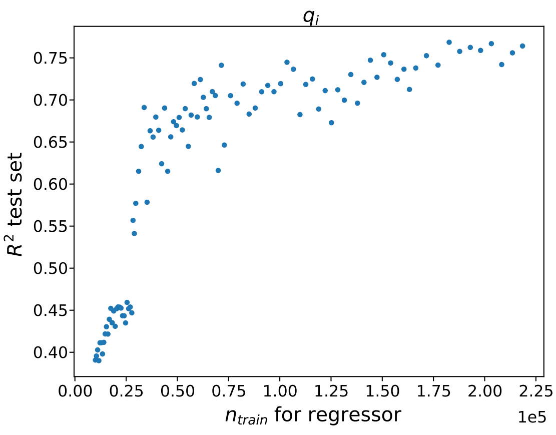
Figure 4.5: Improvement of the R^2 score of the regressor on the uniform test dataset as points were acquired through active learning for the full uniform configuration. Starting from an initial seed of 10000 points, a new deep ensemble was initialized and trained after each active learning iteration.

However, when using different combinations of datasets, the improvement in the R^2 score was neither as large nor as consistent, as shown in figure 4.6. The full Gaussian configuration had the worst performance. This was expected as the Gaussian distribution only covers the regions in the tails with very sparse density. However, in the uniform test dataset these regions were equally represented. This led to reduced performance of the model in the regions inadequately covered by the training data. From figures 4.6b and 4.6c we see that for some iterations the addition of points didn't lead to an improvement in performance. To understand this, a more in-depth analysis of the distribution of the points acquired per iteration is provided in section 4.4.

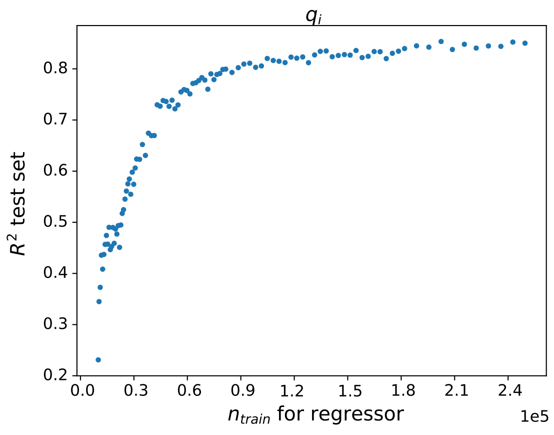
A Gaussian test dataset bounded was also used for evaluation of model performance in figure 4.7. The full Gaussian configuration performed best, showing that matching the distribution of both the unlabeled pool and training seed to the test dataset leads to the best performance. Although the performance of the full uniform configuration increased with the addition of points, it's performance didn't reach the level of the full Gaussian configuration. While a model that performs well over the full input domain is desired, these results show that an application specific model performs better for the same number of points labeled. From the uniform seed configuration and uniform unlabeled pool configuration we can see that providing information on the distribution



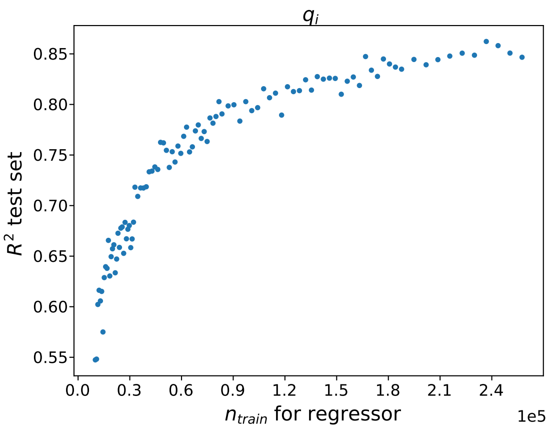
(a) Full Gaussian Configuration



(b) Uniform Seed Configuration



(c) Uniform Unlabeled Pool Configuration



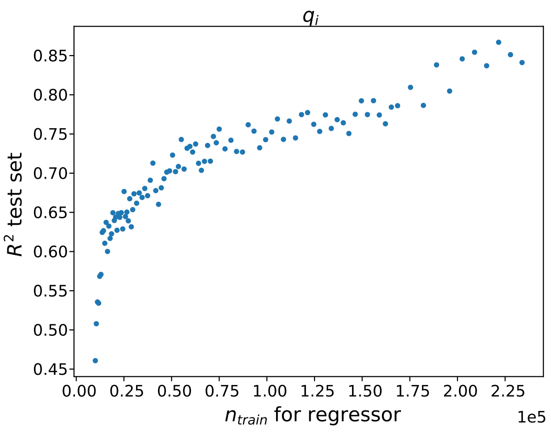
(d) Full Uniform Configuration

Figure 4.6: Comparison of R^2 score of the regressor when evaluating on the uniform test dataset. New points are acquired through active learning, starting from an initial seed of 10000 points, a new deep ensemble was initialized and trained after acquiring each set of points.

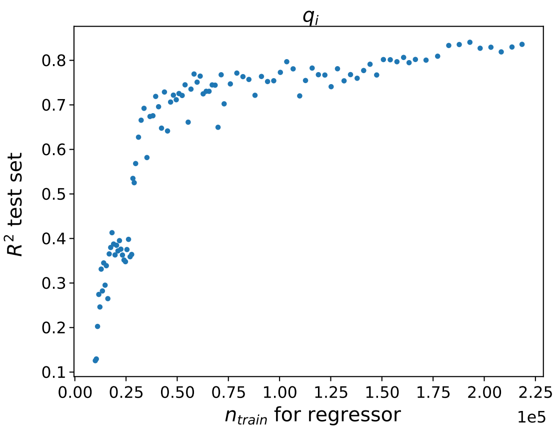
of the test dataset, either in the form of a training seed or as the distribution of the unlabeled pool, leads to improved performance.

To better understand the performance of the models with increased number of points added, their performance against points acquired was overlaid in figure 4.8. We notice that the initial performance of the models varies wildly. This is partly due to the dataset distribution used and partly due to the random initialization. Looking at the evaluation of the models on a uniform test dataset in figure 4.8a, we notice that the models trained through the uniform pool configuration and the full uniform configuration converge to the same performance.

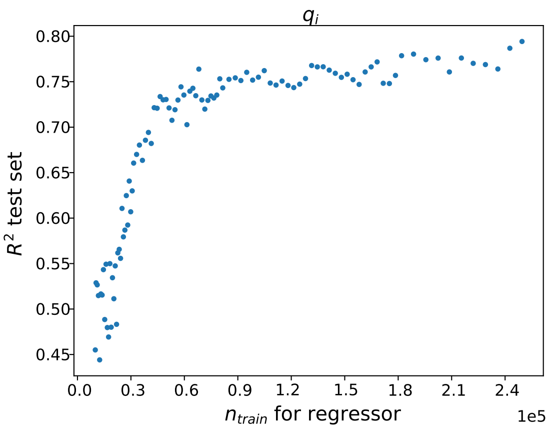
Similarly, when considering performance on the Gaussian test dataset in figure 4.8b, we notice that the all models converge to similar performance between $0.5 \cdot 10^5$ and $1.5 \cdot 10^5$ training points. After this point the performance of the models diverge this is suspected to be due to the large amount of new points added each iteration, as the active learning batch size doubles every 30 iterations. At iteration 30 (25390 points), a significant increase in performance can be seen for multiple models. This suggests that the relation between the amount of points acquired per iteration and the acquisition function used is not as trivial as assumed in previous work, and can further be optimized.



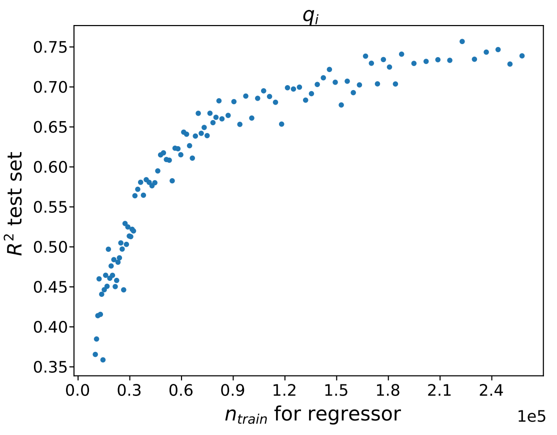
(a) Full Gaussian Configuration



(b) Uniform Seed Configuration

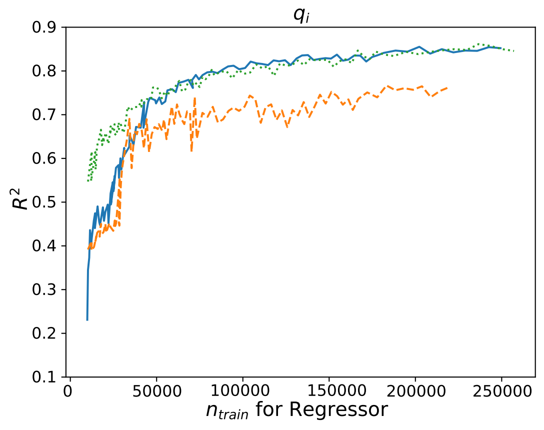


(c) Uniform Unlabeled Pool Configuration

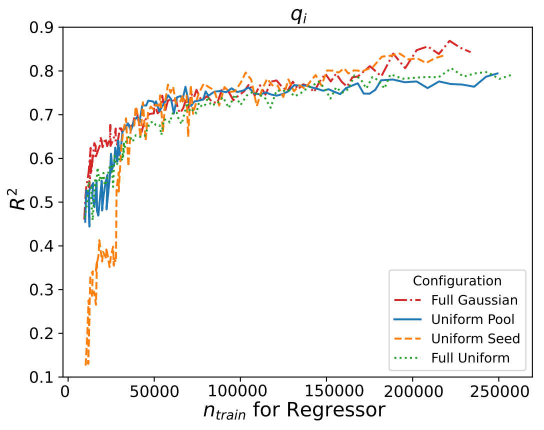


(d) Full Uniform Configuration

Figure 4.7: Comparison of R^2 score of the regressor when evaluating on the Gaussian test dataset. New points were acquired through active learning, starting from an initial seed of 10000 points, a new deep ensemble was initialized and trained after each active learning iteration.



(a) Uniform Test Dataset



(b) Gaussian Test Dataset

Figure 4.8: Comparison of model performance evaluated on two different test datasets. The model trained through a full Gaussian configuration is only shown for the Gaussian test dataset due to poor performance.

Although no large difference in performance is noticeable at iteration 60 (56170 points), after iteration 90 (117730 points) noticeable divergence in performance between models is observed when evaluating on the Gaussian test dataset. To understand why this divergence is occurring, we will look at the points acquired per iteration in the section 4.4 more closely.

Due to lack of time on the cluster, the training of the models was not continued further. Acquiring QuaLiKiz points took about 0.7s per point and the time required to train the models started at 1 hour for the first iteration. However the addition of more data caused the training time of the neural networks to increase up to 5 hours. Although an attempt was made in this project and in previous work to retrain the models each iteration instead of initializing a new model, this quickly led to the models getting stuck in a local minima and not training further.

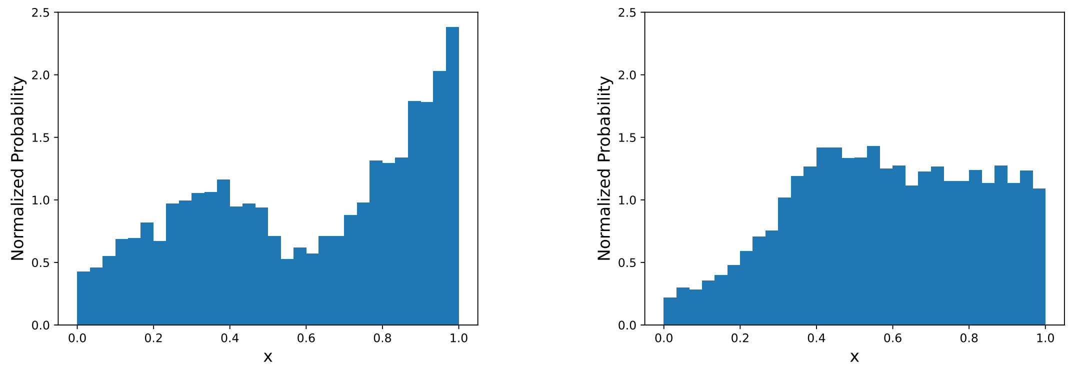
An estimated number of points needed to reach R^2 and f1 score performance of 0.95 can be found through extrapolation of the results. An estimated $3.6 \cdot 10^5$ points will be needed to reach a performance of 0.95 for the regressor on the Gaussian test dataset with the full Gaussian configuration. In previous work, this performance was reached using just $2 \cdot 10^5$ points, however, as two input variables were added in this project and the domain was expanded, it was expected that much more points would be needed to reach the same performance. The full uniform configuration is expected to require about twice, $6 \cdot 10^5$, as many points to reach the same performance on the Gaussian test dataset.

As active learning was not applied on the classifier in this project, the improvement in it's performance was much smaller. To reach an f1 score performance of 0.95, an estimated $1.5 \cdot 10^6$ points are required for the full uniform configuration evaluated on the uniform test dataset. Applying active learning, or separately training a classifier first, should be considered in the future to reduce the number of unnecessary stable points sent to QuaLiKiz for labeling.

To better understand the effect of using different distributions and why this leads to changes in performance, we take a look at the points selected by active learning.

4.4 Acquired Data

The acquisition function used in [7] proved to not acquire the desired points after coupling QuaLiKiz to the active learning pipeline. The points acquired were strongly skewed towards points at the plasma edge. Not only does QuaLiKiz not perform well at the edge due to the assumptions made in chapter 2, but the higher variance of plasma variables leads to larger fluxes. As we are more interested in points near the critical gradient, our model should perform well for small fluxes and stable points. Using the maximum relative error instead of the maximum error of the output flux led to the acquisition of points more evenly spread out over x, as shown in figure 4.9.



(a) Maximum absolute error acquisition **(b)** Maximum relative error acquisition

Figure 4.9: Comparison of distributions of acquired points using different acquisition functions. An acquisition strategy which samples points everywhere along x is desirable as it leads to a model with good performance over all space.

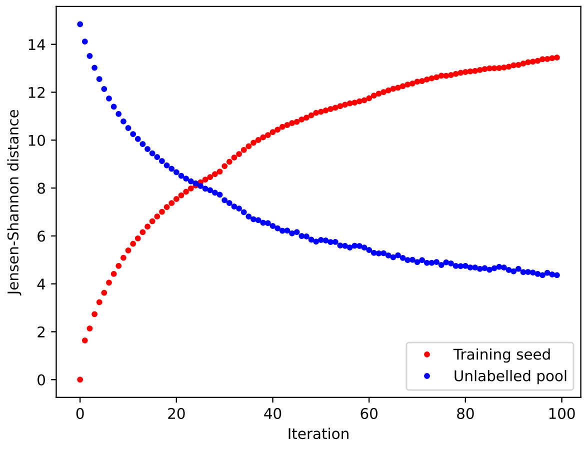
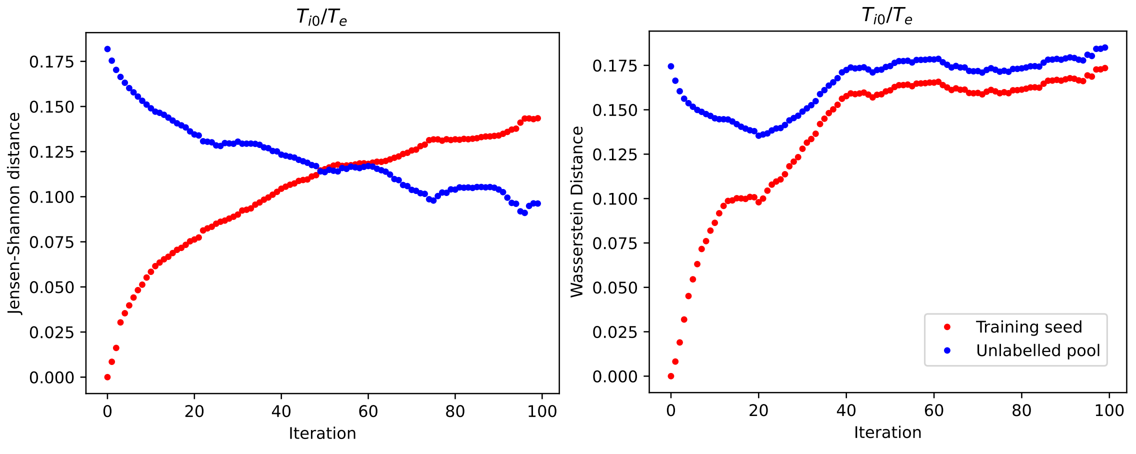


Figure 4.10: Normalized sum of the Jensen-Shannon divergence between the 1D distributions of the classifier training dataset to either the training seed or unlabeled pool for the uniform seed configuration.

Considering the uniform seed and uniform unlabeled pool configurations, to understand how these models lead to different performance, we consider how the points acquired are distributed. Following this we consider how these distributions shift with the number of active learning iterations. To do this we will be using the distance metrics discussed in section 3.7. First starting by looking at the Jensen-Shannon divergence between the uniform unlabeled pool configuration's training dataset and the unlabeled pool and training seed used datasets used. We find that the training dataset initially has zero divergence to the training seed. As an increasing number of points are added through active learning, the divergence between the training dataset and the training seed increases. Simultaneously, the divergence between the training dataset and the unlabeled pool decreases.

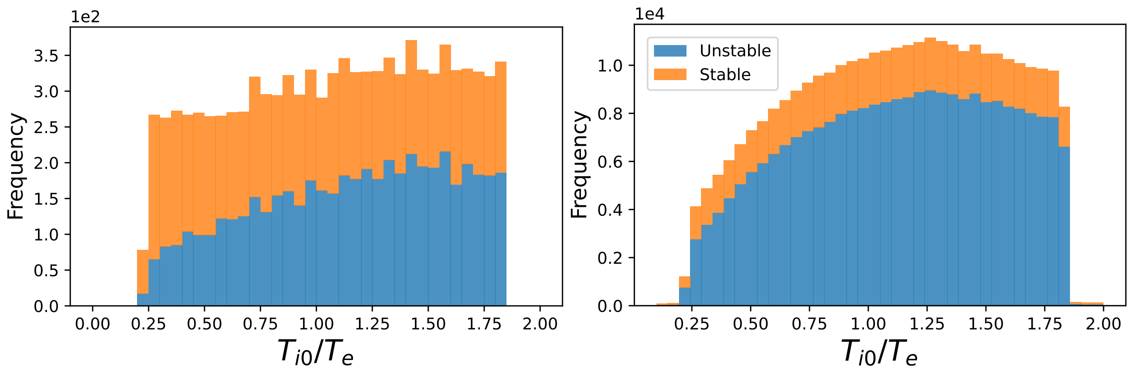
Of the 17 input variables, x and ϵ were excluded. In the Gaussian unlabeled pool, x is uniformly distributed as it was used to create the covariance matrices from which all other variables were acquired. ϵ is given a very large standard deviation so that it appears almost uniform within the bounds to it's domain. The distances between distributions for all variables are equally weighted and summed so that a single distance for all remaining 15 variables results as shown in figure 4.10.

One of the drawbacks of the Jenson-Shannon divergence is that it is unable to account for bins that do not overlap. As the Jensen-Shannon divergence is unable to consider non-overlapping sections of two distributions along the x-axis, two extra distance metrics are considered, the Wasserstein distance and the KS-statistic.



(a) Jensen-Shannon divergence **(b)** Wasserstein distance

Figure 4.11: View of T_{i_0}/T_e from before the summation leading to figure 4.10. This is an example of a variable showing behavior inconsistent with the general trend observed.



(a) Uniform training seed dataset **(b)** Iteration 100 training dataset

Figure 4.12: Inconsistent behavior in figure 4.11 is shown to be due to using the distance to the unlabeled pool and training seed as a whole. Points acquired are almost exclusively unstable at high iterations, causing comparison to an unlabeled dataset to give different results compared to using the distance between the unstable distributions only.

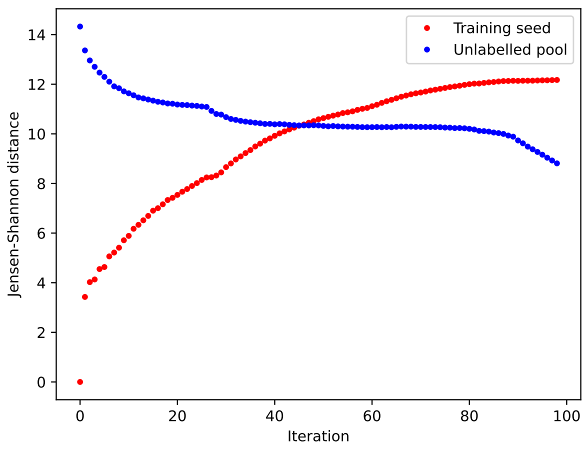


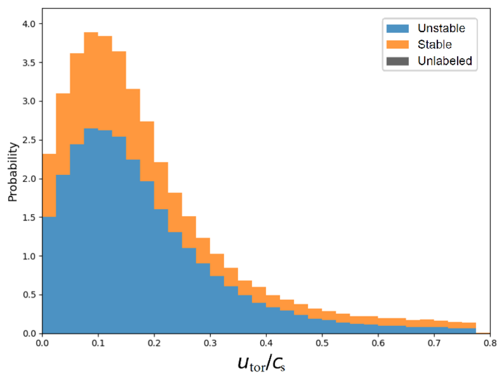
Figure 4.13: Normalized sum of the Jensen-Shannon divergence between the 1D distributions of the classifier training dataset to either the training seed or unlabeled pool for the uniform seed configuration.

For these metrics we see that although the same behavior occurs for the large majority of variables, for some columns, notably $\frac{T_{i0}}{T_e}$, different behavior is observed, as shown in figure 4.11. In this specific case this was due to almost exclusively unstable points being selected as shown in figure 4.12. Although the unstable points show similar distributions, the large proportion of stable points in the training seed dataset cause for a larger distance between the distributions compared to considering just the distribution of unstable points. While considering the distributions in one dimension yields the general trends needed for this project, considering how the distributions change in 17 dimensions should yield better insights into the active learning process. The Jensen-Shannon divergence could be used to achieve this in future projects.

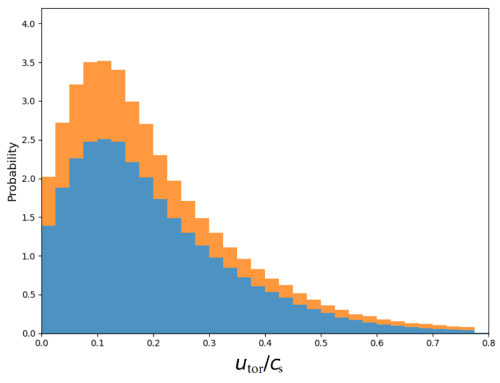
The uniform unlabeled pool configuration acquires points as we would expect. However, in the final distributions we do not see any clear sign that active learning has taken place. The most probable reason for this is that due to the large active learning batch size used at iteration 100, the large amount of points added are distributed as the unlabeled pool, slowly overwhelming any sign that active learning has taken place. This is even more visible when considering the uniform seed configuration. The Jensen-Shannon divergence of the classifier training dataset with both the unlabeled pool and training seed datasets is shown in figure 4.13 for this configuration. We notice a sharp decrease of the divergence to the unlabeled pool right at iteration 90, where the active learning batch size doubles to 4096.

This decrease is explained in figure 4.14 as the distribution of unlabeled pool leading to a shift of the distribution of the training dataset once points are added in large enough batches. For a uniform unlabeled pool this causes the acquisition to increasingly resemble random sampling, which is the intended effect as active leaning was expected to be most effective at the start of training. However it has unintended consequences when using an unlabeled pool dataset with a distribution different from the test dataset. In the case where the application and the test dataset's distribution is known, using a configuration where all distributions match will lead to the best performance.

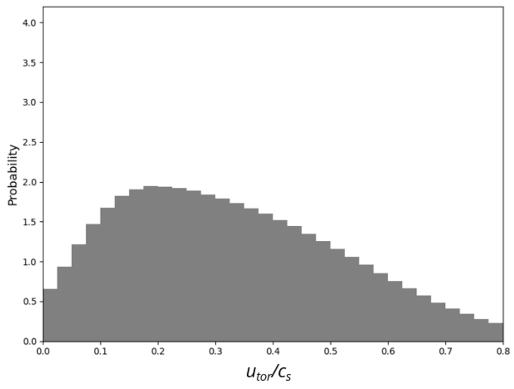
For completeness the distance metrics were also considered for training datasets acquired using the full uniform and full Gaussian configurations. The distance of the classifier training dataset to the unlabeled pool and training seed was the same everywhere with just a normalization factor difference.



(a) Classifier training dataset, iteration 80



(b) Classifier training dataset, iteration 100



(c) Gaussian unlabeled pool dataset

Figure 4.14: Evolution of u_{tor}/c_s of the classifier training dataset with the addition of points through active learning. A clear shift towards higher u_{tor}/c_s can be seen past iteration 90, in line with the peak of the unlabeled pool at higher u_{tor}/c_s values.

The active learning batch size clearly has a large effect on the outcome of active learning and suggestions for further improvement to it's implementation are given in section 5. Due to an incorrect implementation of the relative acquisition function, the points selected through active learning were not adjusted for the division by the leading flux, q_i, this led to a skewing of the results towards points with high uncertainty in the leading flux. As a result, the final trained models for the other fluxes had varying performance and should be retrained using the now corrected implementation before use.

Chapter 5

Discussion

5.1 Optimization

There is clear room for optimization of the pipeline through the choice of acquisition function and batch size. The batch size of 512, with batch doubling every 30 iterations, was chosen when working with relative and absolute maximum error, as the number of iterations could not be too large due to the neural networks being reinitialized each iteration. Retraining neural networks was attempted both for this project and during previous works [7]. Specifically, the models were updated on the combined dataset of old training data and newly acquired points. The training loss continued to decrease for several active learning iterations, however, after approximately ten iterations a minima was quickly reached. An alternative approach would be to retrain the neural networks on just the newly acquired points. This would likely allow for more active learning iterations before a minimum is reached, due to the models being updated with iterations from only new data points.

When retraining only on new data it is possible that the network "forgets" the initial relations learned, known as catastrophic forgetting [36]. This would lead to points similar to those in the initial training seed from being reacquired through active learning. To prevent this issue, we suggest still retraining the neural networks from scratch every few active learning iterations. Further adjustments will need to be made to the hyperparameters used for retraining to be successful. Techniques to find the optimal hyperparameters will be discussed later in this chapter.

5.2 Acquisition Functions

Further research into batch strategies found that the "N-best" strategy used in this project does not typically lead to the best performance when used for batch active learning. Indeed it usually doesn't outperform random sampling [37], although previous work did show an improvement for our specific use case [7]. Many different options exist that would be more suitable and a selection will be covered in this section.

Batch acquisition functions are used to prevent points close to each other in parameter space from both being acquired for labeling, which would occur when just basing the selection on an uncertainty estimate. Current batch methods therefore predominantly make use of both an uncertainty estimate and a measure of the diversity of the data. An uncertainty estimate for points is typically achieved through using Bayesian neural networks [38]. Another way of approaching this problem, which is what was used for this project, is to use an ensemble of frequentist deep neural networks together with a proper scoring rule to obtain an uncertainty estimate [29]. Getting an accurate estimate of uncertainty remains difficult for neural networks and recent developments in the field should be followed to see if better methods are developed or a clear consensus is reached on the best methods to use.

Most batch active learning methods are optimized for small batches and their computation time rapidly increases with batch size [39][40]. Due to the issue of needing to retrain the neural networks completely each iteration, large active learning batches were required for this project. Implementing a solution to this issue could allow for smaller batches and the use of better active learning methods.

To consider a few different acquisition functions, we will be mostly limiting this discussion to well-known algorithms that have been proven to work in well-cited literature and include both measures of uncertainty and diversity such as BatchBALD, DMBAL and BADGE.

BatchBALD: A very popular batch acquisition method that works effectively with very small batches. As it has to be implemented with a Bayesian neural network, it is only relevant for future works if uncertainty estimates are also shown to be more accurate using Bayesian neural networks [41]. Furthermore, it doesn't generalize well to large batch-sizes, although alternative algorithms exist that approximate BatchBALD [39].

DMBAL: An algorithm initially developed for classification, it uses the uncertainty prediction of all points in the unlabeled pool and keeps only the most uncertain points. From there, using k-means clustering these data points are grouped into specific clusters and the points closest to the cluster centers are selected for acquisition [42].

BADGE: An algorithm designed for classification that uses the magnitude of the gradients in the last output layer as a weight for the uncertainties. The idea being that regions where small changes to the inputs of the neural network lead to large changes in the outputs are regions where points will be more informative to the model [43]. Although this was designed for classification with Bayesian neural networks it has been adapted to be used for regression with any neural network structure using kernel density approximation [44]. However, as a large amount of input dimensions are used, a kernel density approximation will quickly become too computationally intensive for this project.

Implementing DMBAL for future versions of this project is believed to be the most appropriate choice. Furthermore it can also be implemented using existing packages. Many more algorithms exist although they quickly become increasingly complex in their implementation. An example of an algorithm designed for selection of large batches of data is the cluster-margin algorithm.

Cluster-Margin: Clusters of points based on their location are found. Following this high uncertainty points in each cluster starting from the smallest clusters [40]. This allows for the creation of batches with thousands of data points.

Research into large batches for active learning is however still sparse, with no readily made python packages available that can easily be applied to this project. Many active learning strategies do not generalize well to different machine learning architectures [45]. This prevents any conclusion from a literature study alone to be made on which acquisition function will work best for our architecture.

Batch size doubling was implemented in previous work with the underlying idea being that points selected by active learning would increasingly contain less extra information compared to random sampling. It's use caused for the distribution of the unlabeled pool to strongly affect the points acquired for training, which is undesirable with a non-uniform unlabeled pool. Batch size doubling would have to be removed when switching to more efficient active learning algorithms in future projects as these techniques do not scale well with increased batch size.

5.3 Hyperparameter Tuning

Although a few adjustments were made to the hyperparameters in this project, no extensive hyperparameter tuning was performed, leaving room for performance improvement of the machine learning models. This will in turn lead to improved uncertainty estimates at each iteration, leading to fewer points necessary to reach the same performance.

As hyperparameter optimization on the active learning pipeline as a whole would cost too much computation time, the optimization can be done on the datasets acquired through active learning. For our networks, the training datasets for the full uniform and Gaussian configurations from iteration 90 should be used to prevent the large batch size from affecting the results. Once the required datasets are loaded, there are multiple ways to adjust hyperparameters. Specifically, models can be trained using a grid search or a random search, however, Bayesian optimization has been shown to be most efficient [46]. The optimal hyperparameters found can be used as the starting point for future projects, although further small adjustments may be required as the data acquired can be distributed differently.

Due to the large number of adjustable hyperparameters, those which are most likely to have a large impact are listed to reduce the size of the optimization problem. It is suggested to reduce early stopping patience during optimization, as it strongly affects the duration of model training.

- Learning rate: Different learning rates may be needed for training on the full dataset and on newly acquired batches of data.
- L2 regularization: Affects the ability of the model in finding an optimum and in generalizing.
- Dropout: Also provides regularization, the total amount of regularization can be adjusted together with the L2 hyperparameter.
- Acquisition function: Different parameters will work best with different acquisition functions.
- Active learning batch size: Dependent on the acquisition function used.
- Neural network batch size: Batches smaller than the full dataset provide some noise in the network, leading to a regularizing effect. Although using small batches also allows for faster training, batches too small can lead to too much noise and suboptimal training.
- Neural network layers: Different architectures have been used in previous works, however the effect due to layer size and number of layers has not been evaluated systematically.
- Training seed dataset size: Dependent on the acquisition function used an optimal point exists between sampled points for initialization and points acquired through active learning.

As this pipeline will be used for generating neural network surrogates of higher fidelity codes in the future, a benefit of optimizing the pipeline on QuaLiKiz is that the loss landscape of higher fidelity codes will be very similar. This will allow the same settings to be used, as starting point.

5.4 Classifier Architecture

Although the classifiers improve when training on the points acquired for the regressor, this still leads to an approximately 15% of data acquired to be stable for both the full uniform and full Gaussian configurations. The classification and regression problems have already been partly separated in this pipeline, however it is advisable to separate them further so that problems can more easily be identified and different techniques can be used. This can be done by training a classifier prior to using it in the active learning pipeline before the regressor.

Although preliminary tests on applying active learning for the classifier did not lead to significant improvement during this project, many of the acquisition functions listed above have specifically been developed for classification and are sure to lead to significant improvement compared to the acquisition method used.

The same dataset seed can be used for training both the classifier and regressor through active learning. In this project it was shown that the dataset seed only has a large effect on the performance of the initial models trained, but not on the final distribution of points. The extra points acquired through active learning for the classifier can thus be added to the training seed for the regressor in order to provide a better starting point for active learning.

Another issue with the classifier used for this project, is that it only considered q_i for training. This caused errors in labeling due to some points being stable in q_i in ITG mode, but being unstable in one of the other turbulence modes (ETG or TEM). For information about all modes to be considered a classification with multiple outputs would seem to be the best solution. Previous work has however shown that for regression, multiple independent neural networks lead to improved performance when compared to a single neural network with multiple outputs [6]. As a classifier is now being used, both options should be considered to see which obtains the best performance.

These changes to the classifier would allow for more information about the underlying physics to be provided, improving the fundamental limit on it's performance.

Chapter 6

Conclusion

During this project a QuaLiKiz surrogate was developed making use of 17 input dimensions, two more than previous works, while increasing the available input domain. This was achieved through the use of active learning to reduce the number of data points needed to train a neural network with similar performance. The research question, **"How do the distributions of the unlabeled pool and training seed affect the neural networks trained through active learning?"**, was tackled by applying active learning on different starting scenarios using two different distributions.

It was found that the distributions used for the training seed and unlabeled pool datasets did have a large effect on performance. Matching the distribution of these datasets to the application led to the smallest number of points needed, for similar performance when evaluated on the application (Gaussian) dataset. Comparable performance using all uniform datasets required approximately twice as many points.

Although it was shown that using a training seed dataset with a distribution matching the application gives an initial boost in performance, using an unlabeled pool with active learning batch doubling led to the effect of active learning becoming negligible in the final training dataset. As much more suitable acquisition functions exist, these should be investigated in future projects. This would allow for the removal of batch doubling as these acquisition functions outperform random sampling.

A few suitable acquisition functions were discussed, and the most suitable alternatives were presented in chapter 5.

Switching to continuing the training of neural networks instead of reinitialization between iterations would allow for the use of smaller active learning batch sizes needed for many acquisition functions. Further hyperparameter tuning and configuration of the classifier to account for different turbulence modes, show that large improvements in efficiency of the pipeline are still possible.

Bibliography

[1] EUROfusion. European research roadmap to the realisation of fusion energy. https://euro-fusion.org/eurofusion/roadmap/, 2018. 1, 2

[2] B. Nouhi, N. Darabi, P. Sareh, H. Bayazidi, F. Darabi, and S. Talatahari. The fusion–fission optimization (fufio) algorithm. *Scientific Reports*, 12(1):12396, 2022. ID: Nouhi2022. 2

[3] J. Wesson. *Tokamaks.* Oxford University Press, Oxford, UK, 3 edition, 2004. 1, 6, 11

[4] M. Keilhacker, A. Gibson, C. Gormezano, P.J. Lomas, P.R. Thomas, M.L. Watkins, P. Andrew, B. Balet, D. Borba, C.D. Challis, I. Coffey, G.A. Cottrell, H.P.L. De Esch, N. Deliyanakis, A. Fasoli, C.W. Gowers, H.Y. Guo, G.T.A. Huysmans, T.T.C. Jones, W. Kerner, R.W.T. König, M.J. Loughlin, A. Maas, F.B. Marcus, M.F.F. Nave, F.G. Rimini, G.J. Sadler, S.E. Sharapov, G. Sips, P. Smeulders, F.X. Söldner, A. Taroni, B.J.D. Tubbing, M.G. von Hellermann, D.J. Ward, and JET Team. High fusion performance from deuterium-tritium plasmas in jet. *Nuclear Fusion*, 39(2):209, 1999. 1

[5] C. Bourdelle, J. Citrin, B. Baiocchi, A. Casati, P. Cottier, X. Garbet, F. Imbeaux, and JET Contributors. Core turbulent transport in tokamak plasmas: bridging theory and experiment with qualikiz. *Plasma Physics and Controlled Fusion*, 58(1):014036, 2015. 3

[6] A. Ho, J. Citrin, C. Bourdelle, Y. Camenen, F.J. Casson, K.L. van de Plassche, H. Weisen, and JET Contributors. Neural network surrogate of qualikiz using jet experimental data to populate training space. *Physics of Plasmas*, 28, 2021. 3, 12, 14, 17, 24, 25, 28, 43

[7] L. Zanisi, A. Ho, T. Madula, J. Barr, J. Citrin, S. Pamela, J. Buchanan, F. Casson, V. Gopakumar, and JET contributors. Efficient training sets for surrogate models of tokamak turbulence with active deep ensembles, 2023. 3, 17, 22, 23, 35, 40

[8] EUROfusion. Would a sustainable deuterium-deuterium (d-d) fusion reaction require much more energy compared to deuterium-tritium (d-t) fusion? https://euro-fusion.org/faq/deuterium-deuterium-fusion-reaction-energy/, 2023. Last accessed 12 January 2024. 5

[9] R.S. Cohen, Jr. L. Spitzer, and P. Routly. The Electrical Conductivity of an Ionized Gas. *The Computer Journal*, 80(2), 1950. 5

[10] ITER. External heating. https://www.iter.org/sci/PlasmaHeating, 2024. Last accessed 11 January 2024. 5

[11] M.R. Gordinier, J.W. Davis, F.R.Scott, and K.R.Schultz. Nuclear Fusion Power. In R.A. Meyers, editor, *Encyclopedia of Physical Science and Technologys*, pages 671–699. Academic Press, 2001. 5

[12] M. Greenwald, J.L. Terry, S.M. Wolfe, S. Ejima, M.G. Bell, S.M. Kaye, and G.H. Neilson. A new look at density limits in tokamaks. *Nuclear Fusion*, 28(12):2199, 1988. 5

[13] J.D. Jackson. *Classical Electrodynamics.* John Wiley and Sons, 1998. 6, 8

[14] M. Knölker. *Investigation of the nonlinear phase of edge-localized-modes on the DIII-D tokamak*. PhD thesis, LMU Munchen, 2019. 6

[15] J.P. Freidberg. *PLasma Physics and Fusion Energy*, chapter 14. Cambridge University Press, Boston, USA, 2 edition, 2010. 6

[16] C.D. Stephens, X. Garbet, J. Citrin, C. Bourdelle, K.L. van de Plassche, and F. Jenko. Quasilinear gyrokinetic theory: A derivation of qualikiz. *Journal of Plasma Physics*, 2021. 7, 9, 10

[17] J. Thijssen. Lecture notes on classical mechanics. 2018. 7

[18] D. Cline. Action-angle variables. `https://phys.libretexts.org/Bookshelves/Classical_Mechanics/Variational_Principles_in_Classical_Mechanics_(Cline)/`, 2024. Last accessed 12 January 2024. 7

[19] M.E. Smedberg. Reducing the qualikiz-neural-network 10-dimensional dataset: sampling and correlation techniques. Master's thesis, Eindhoven Univerisy of Technology, 2021. 8

[20] F.J. Casson, H. Patten, C. Bourdelle, S. Breton, J. Citrin, F. Koechl, M. Sertoli, C. Angioni, Y. Baranov, R. Bilato, E.A. Belli, C.D. Challis, G. Corrigan, A. Czarnecka, O. Ficker, L. Frassinetti, L. Garzotti, M. Goniche, J.P. Graves, T. Johnson, K. Kirov, P. Knight, E. Lerche, M. Mantsinen, J. Mylnar, M. Valisa, , and JET contributors. Predictive multi-channel flux-driven modelling to optimise icrh tungsten control and fusion performance in jet. *Nuclear Fusion*, 60(6), 2020. 8

[21] Wiki Fusion. Ion temperature gradient instability. `http://fusionwiki.ciemat.es/wiki/Ion_Temperature_Gradient_instability`, 2021. Last accessed 12 January 2024. 11

[22] S.F.C. Marechal. Expanding the qualikiz neural network surrogate model with generalized ion particle transport coefficient networks. Master's thesis, Eindhoven University of Technology, 2021. 11

[23] K.L. van de Plassche. Realtime capable turbulent transport modelling using neural networks. Master's thesis, Eindhoven University of Technology, 2017. 12

[24] K. L. van de Plassche, J. Citrin, C. Bourdelle, Y. Camenen, F.J. Casson, V.I. Dagnelie, F. Felici, A. Ho, S. van Mulders, and JET Contributors. Fast modeling of turbulent transport in fusion plasmas using neural networks. *Physics of Plasmas*, 27, 2020. 12, 13

[25] QuaLiKiz-group. Input and output variables. `https://gitlab.com/qualikiz-group/QuaLiKiz/-/wikis/QuaLiKiz`, 2023. Last accessed 14 January 2024. 12, 24, 48

[26] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Boston, USA, 1 edition, 2015. 14, 15, 23

[27] D.A. Nix and A.S. Weigend. Estimating the mean and variance of the target probability distribution. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 1, pages 55–60 vol.1, 1994. 15

[28] M.A. Ganaie, M. Hu, A.K. Malik, M. Tanveer, and P.N. Suganthan. Ensemble deep learning: A review. *Engineering Applications of Artificial Intelligence*, 115:105151, 2022. 15

[29] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6405–6416, Red Hook, NY, USA, 2017. Curran Associates Inc. 16, 40

[30] R. Moradi, R. Berangi, and B. Minaei. A survey of regularization strategies for deep models. *Artificial Intelligence Review*, 53(6):3947, 3986. 16

[31] John A. Rice. *Mathematical Statistics and Data Analysis*, chapter Linear Least Squares, pages 583–584. Duxbury, third edition, 2007. 16

[32] T. Görler, X. Lapillonne, S. Brunner, T. Dannert, F. Jenko, F. Merz, and D. Told. The global version of the gyrokinetic turbulence code gene. *Journal of Computational Physics*, 230(18):7053–7071, 2011. 17

[33] D. Holzmüller, V. Zaverkin, J. Kästner, and I. Steinwart. A framework and benchmark for deep batch active learning for regression, 2023. 17

[34] I. Loshchilov and F. Hutter. Decoupled weight decay regularization, 2019. 23

[35] A. Lipp and P. Vermeesch. Short communication: The wasserstein distance as a dissimilarity metric for comparing detrital age spectra and other geological distributions. *Geochronology*, 5:263–270, 2023. 26

[36] R. Kemker, M. McClure, A. Abitino, Hayes T, and C. Kanan. Measuring catastrophic forgetting in neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018. 40

[37] B. Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison, 2009. 40

[38] J. Gawlikowski, C.R.N. Tassi, M. Ali, J. Lee, M. Humt, J. Feng, A. Kruspe, R. Triebel, P. Jung, R. Roscher, M. Shahzad, W. Yang, R. Bamler, and X.X. Zhu. A survey of uncertainty in deep neural networks, 2022. 40

[39] A. Kirsch. Speeding up batchbald: A k-bald family of approximations for active learning, 2023. 41

[40] G. Citovsky, G. DeSalvo, C. Gentile, L. Karydas, A. Rajagopalan, A. Rostamizadeh, and S. Kumar. Batch active learning at scale. *Advances in Neural Information Processing Systems*, 2021. 41

[41] A. Kirsch, J. van Amersfoort, and Y. Gal. Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning. *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing System*, 2019. 41

[42] F. Zhdanov. Diverse mini-batch active learning, 2019. 41

[43] P. Ren, Y. Xiao, X. Chang, P.Y. Huang, Z. Li, B.B. Gupta, X. Chen, and X. Wang. A survey of deep active learning. *ACM Computing Surveys*, 54, 2021. 41

[44] Andreas Kirsch. Black-box batch active learning for regression, 2023. 41

[45] P. Munjal, N. Hayat, M. Hayat, J. Sourati, and S. Khan. Towards robust and reproducible active learning using neural networks. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022. 41

[46] J. Snoek, H. Larochelle, and R.P. Adams. Practical bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems 25*, 2012. 42

Appendix

A: GyroBohm units

To convert the fluxes from GyroBohm to SI units, the following equations are used:

Defining the GyroBohm scaling factor:

$$x_{GB} = \frac{\sqrt{A_{i,0} m_p} T_e^{1.5}}{q_e^2 B_0^2 a} \tag{6.1}$$

For q_s:

$$x_{GB} = \frac{a}{n_s T_s \chi_{GB}} x_{SI} \tag{6.2}$$

For Γ_s:

$$x_{GB} = \frac{a}{n_s \chi_{GB}} x_{SI} \tag{6.3}$$

For Π_s:

$$x_{GB} = \frac{a}{\sqrt{2T_s/m_s} n_s m_s R_0 \chi_{GB}} x_{SI} \tag{6.4}$$

Further conversions for the individual convection and diffusion terms as well as growth rates and frequencies can be found in [25].

B: Dataset Distributions

Unlabeled Pool Distributions

The distributions of the unlabeled pools are included to show the effect of balancing and filtering on the datasets used and their differences compared to the training seed and test datasets. Notably the change in distribution of x, which starts off as almost uniform in figure 6.1, is significantly different in figure 3.1. This can be explained by almost all points being stable near the core, leading to less stable points elsewhere in the distribution.

In figure 6.2 the effect of the input filters had been partially compensated by sampling additional points in the regions where points are filtered out. Nevertheless, this still led to an imperfection in the n_{i0}/n_e dataset. Although this process makes the 1D distributions appear more uniform and ensures that points are available everywhere in the given domain, it does lead to oversampling of some specific regions.

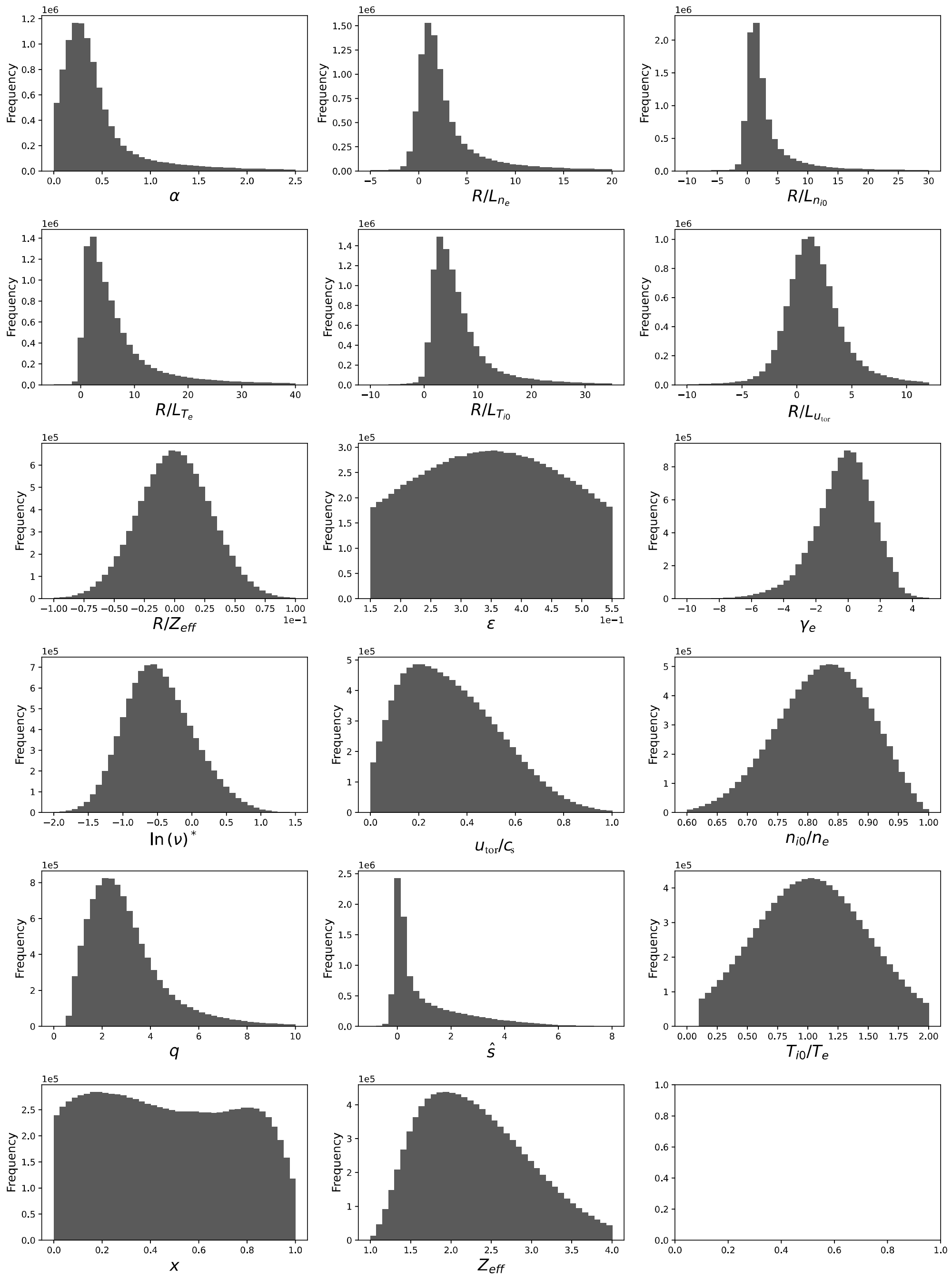


Figure 6.1: Gaussian unlabeled pool dataset input variable distribution. The y-axis is limited to 1/5 the dataset size.

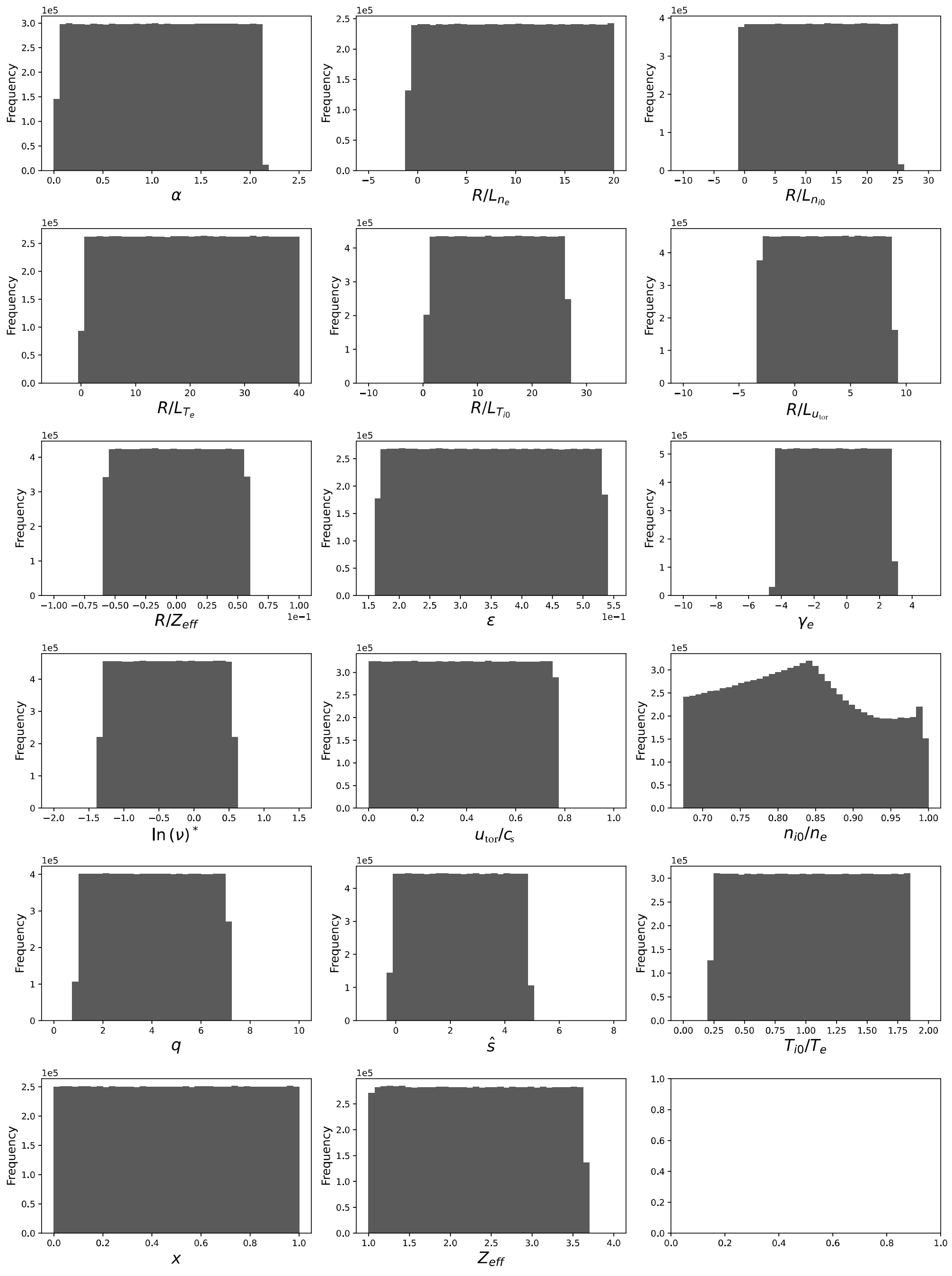


Figure 6.2: Uniform unlabeled pool dataset input variable distribution. Extra points were sampled in the regions where the input filters filtered out points in figure 6.5. The y-axis is limited to 1/5 the dataset size.

Training Seed Dataset Fluxes

The flux labels of the training seed datasets are shown in figures 6.3 and 6.4. As expected, we see that when looking at ITG as the source of turbulence, q_i mostly has the largest values, and all other fluxes are much smaller. From these figures we see that negative fluxes also exist, these represent fluxes towards the core of the tokamak. As the temperature and density decrease as you move from the core to the edge, particles will tend to move towards the edge as well, which is reflected by the proportion of fluxes which are negative being small.

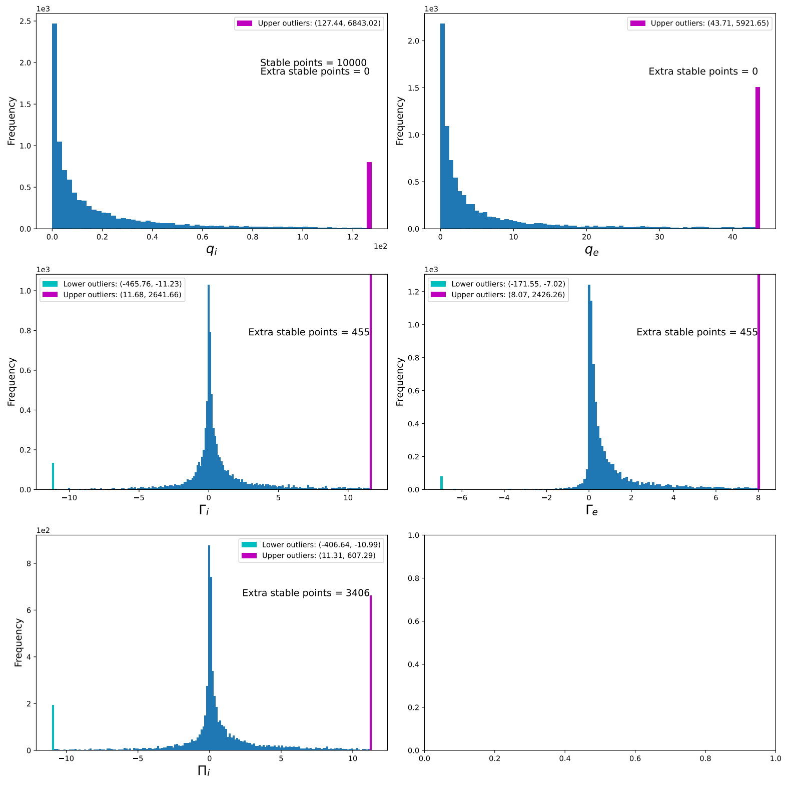


Figure 6.3: Gaussian training seed dataset output fluxes. Only unstable fluxes in q_i are shown. Outliers were detected and binned separately for a better view of the distributions. Points unstable in q_i but stable in another flux were removed and displayed in text.

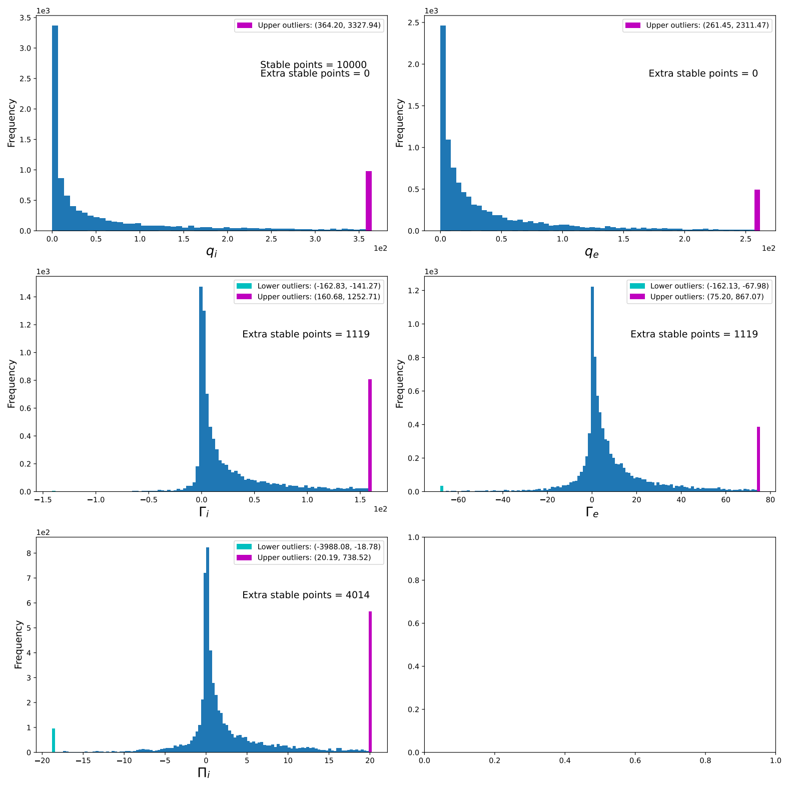


Figure 6.4: Uniform training seed dataset output fluxes. Only unstable fluxes in q_i are shown. Outliers were detected and binned separately for a better view of the distributions. Points unstable in q_i but stable in another flux were removed and displayed in text.

C: Input Filter Response

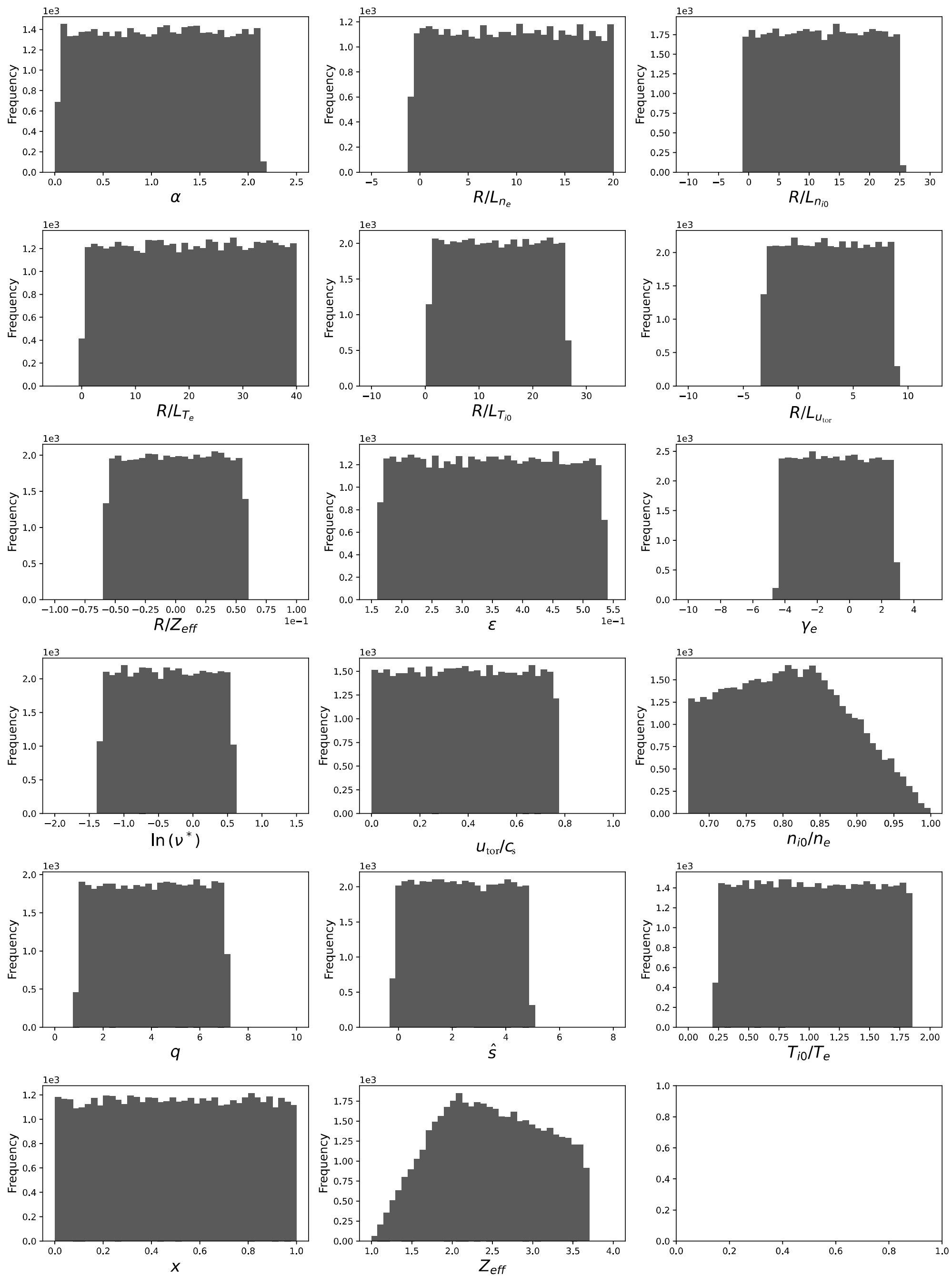


Figure 6.5: Uniform 1D distributed training seed dataset after being passed to the input filters. Intermediate step between figures 4.1 and 4.2. All changes can be attributed to the input ion density filter as the α filter rarely filters out any extra points.

D: Classifier Performance

For completeness a few extra figures are included for each of the configurations considered. First in figure 6.6, an example of typical loss curves observed during training of the regressor neural networks is given. Following this, the performance of the classifier evaluated on the uniform test dataset is given in figure 6.7 and on the Gaussian test dataset in figure 6.8.

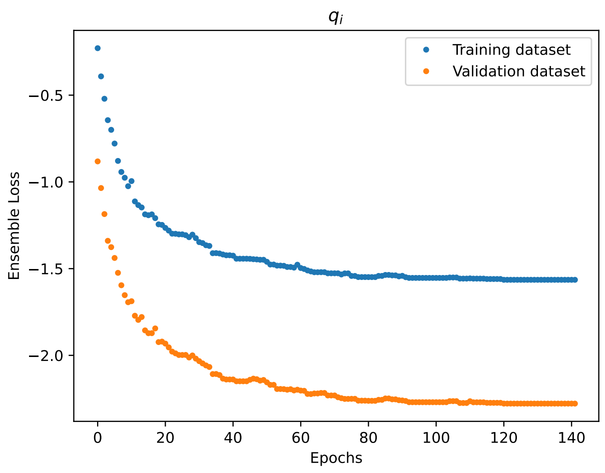
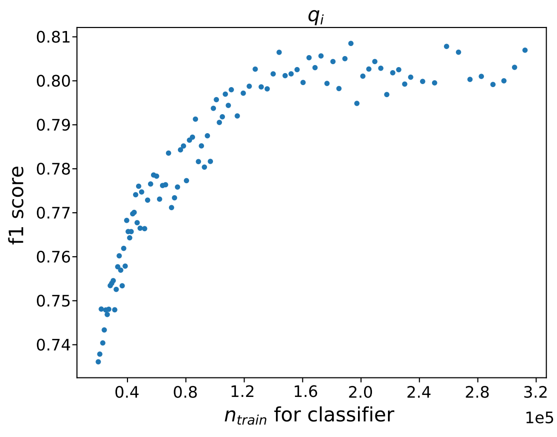
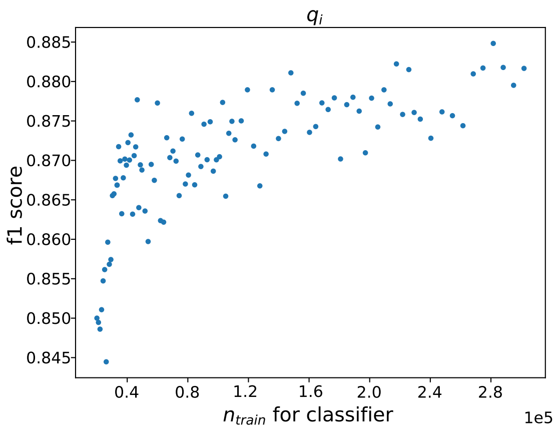


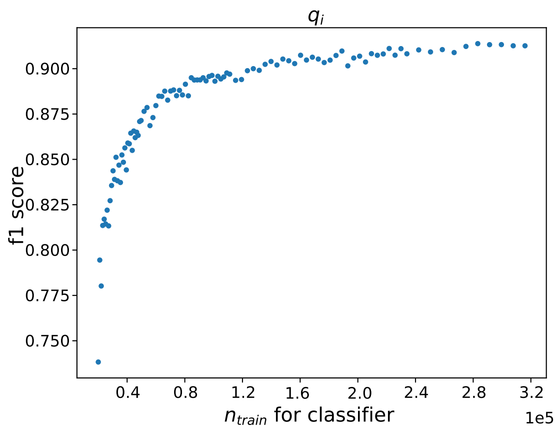
Figure 6.6: Loss curves of a regressor trained on the training dataset of the full Gaussian configuration at iteration 100. The model is evaluated on the whole validation and training datasets each training epoch.



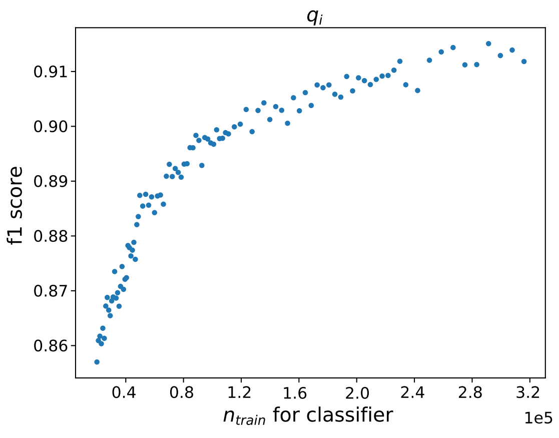
(a) Full Gaussian Configuration



(b) Uniform Seed Configuration

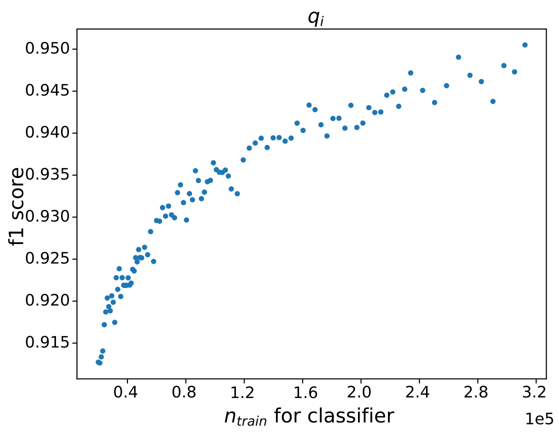


(c) Uniform Unlabeled Pool Configuration

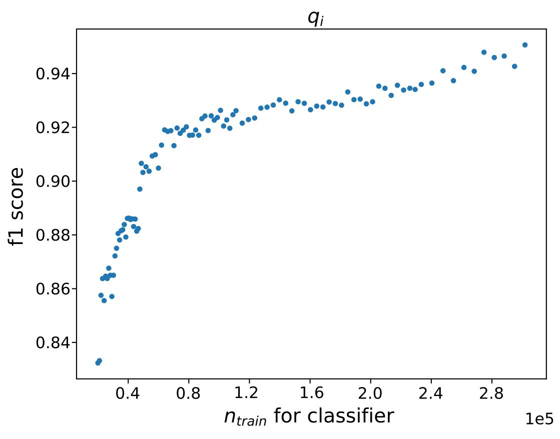


(d) Full Uniform Configuration

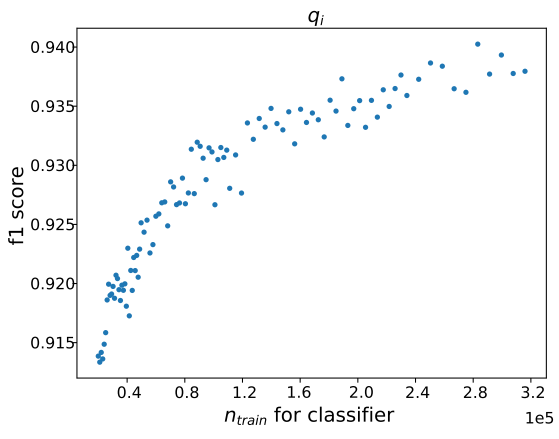
Figure 6.7: Comparison of $f1$ score of the classifier when evaluating on the uniform test dataset. New points are acquired through active learning, starting from an initial seed of 20000 points.



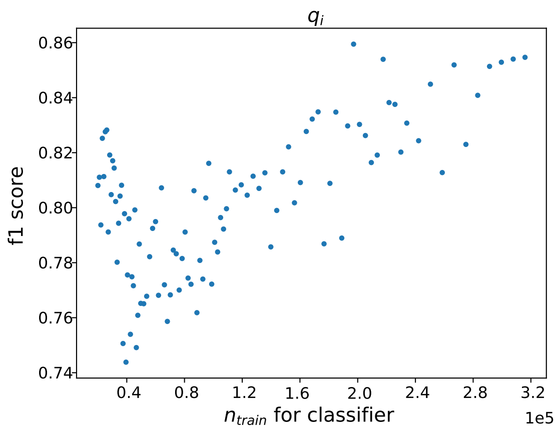
(a) Full Gaussian Configuration



(b) Uniform Seed Configuration



(c) Uniform Unlabeled Pool Configuration



(d) Full Uniform Configuration

Figure 6.8: Comparison of $f1$ score of the classifier when evaluating on the Gaussian test dataset. New points are acquired through active learning, starting from an initial seed of 20000 points, a new deep ensemble was initialized and trained after acquiring each set of points.