1 My Research

My research studies deep learning in the infinite-width regime, with a focus on incorporating symmetry. Training under gradient descent can be described in terms of the Neural Tangent Kernel (NTK). When the width of hidden layers tends to infinity, this kernel obtains certain properties that enable an analytic solution describing the training dynamics. This provides a clean mathematical description and connects neural networks to kernel methods. I have extended this framework to equivariant architectures, i.e. models where symmetries under e.g. (continuous) rotations and translations are satisfied by the model by construction. This makes it possible to analyze how symmetry constraints shape the dynamics and inductive biases of learning.

In addition, I also study possible ways to leave the idealized infinite-width picture. Real networks are always of finite size, and the NTK only describes the leading-order behaviour. Despite its past success, there are known effects that can not be captured in the strict limit. My current work therefore studies finite-width corrections. Here I use tools inspired by Quantum Field Theory, where the infinite-width limit plays the role of a free theory (non-interacting particles in physics) and corrections can be treated in a perturbative fashion. This approach provides a systematic way to capture the deviations from the infinite-width solution and to understand how wide but finite networks actually train.

Because my theoretical work relies on certain assumptions, it is a priori not clear how well those hold in practice. Therefore, testing the theoretical findings empirically is a major part of my research. For instance, I have implemented the infinite-width solution of rotationally equivariant neural networks and studied the convergence of finite-width equivalents to this limit.

2 Lecture Principles

So far, my research is fundamental in nature. Therefore, issues related to quality assurance, deployment etc. are not yet relevant for me. Nevertheless, there are some aspects of software engineering that I have found useful in my work.

2.1 Property-Based Testing

When implementing infinite-width solutions for rotation-symmetric neural networks, debugging turned out to be particularly difficult. The main challenge was not the mathematics itself, but ensuring that the code faithfully preserved the theoretical properties. What eventually worked was to build a test suite around the symmetries implied by the equations. By writing parametrized unit tests with randomly sampled inputs, I could automatically check that the expected invariances, such as rotation equivariance, were respected. This approach was essential for eliminating the first set of subtle implementation errors, and later it became just as important for detecting regressions when the code evolved.

In practice, this amounted to a form of property-based testing: rather than checking against fixed outputs, the tests enforced general laws that should always hold. This was not only more systematic, it was in fact the only practical way to make sense of the arrays of millions of values produced by the algorithm. Quantitative checks of symmetries was easily testable at several stages of the computation. Another approach, visualization, was not suitable because for one, only the very last output could be interpreted visually, and two, visual inspection relies on manual inspection.

2.2 Static Analysis

During the implementation of the infinite-width solution, I often ran into situations where the code executed without errors but still produced wrong results. The source of these issues was not always a simple programming mistake, but often something more subtle: an incorrect assumption about the mathematical properties, an oversight in the design of the algorithm, or a choice that led to numerical instability. In such cases, simply running tests was not enough to uncover the problem. What helped instead was to go through the code line by line together with collaborators. These walkthroughs were very similar in spirit to static analysis: the goal was to carefully examine the implementation without executing it, in order to reveal inconsistencies or overlooked details.

In some sessions this resembled "rubber duck debugging," where explaining the logic to someone else was enough to expose a mistake in my own reasoning. But there were also moments where the discussions revealed deeper issues, not in the coding but in the algorithm itself. For example, an assumption that looked harmless in the mathematical derivation could turn out to be problematic once translated into numerical routines. Having multiple perspectives in these reviews proved crucial, since what looked natural to one person could strike another as suspicious. In this way, collaborative code walkthroughs became not only a debugging tool, but also a method for validating the chosen numerical approximation.

3 Guest-Lecture Principles

3.1 Scalability and Performance Requirements

My research often requires implementing numerically intensive simulations. A significant challenge has been discovering late in development that an implementation, while algorithmically correct, scales poorly, severely limiting the scope of my experiments. This experience perfectly illustrates how a "defect" in a non-functional requirement like performance can lead to costly rework. The issue arose from picking a, from an implementation point of view, attractive method in the solution-space without fully considering the crucial but overlooked requirement of computational tractability for a target problem size.

To mitigate this, I adapted my workflow to incorporate these principles by defining performance and scalability even for proof of concepts before designing the actual algorithm. A minimum target problem size is now first defined, for which then a computation and memory estimate is then computed for a proposed method. Additionally, possible ways to partition the computation into batches are considered. By treating performance constraints not as an afterthought but as a core requirement, I can ensure that the chosen solution is both theoretically sound and feasible for interesting problem sizes, ultimately preventing the significant waste of development time and frustrating outcomes.

3.2 Stakeholders in my Research Code

While my research code is often primarily not intended for typical end-users, the concept of stakeholders becomes relevant when aiming to contribute to larger software libraries, such as *neural-tangents*, used by other researches in my field. In this context, the library itself, along with its developers, impose strict requirements that are less about the algorithm's scientific approach and more about its structure and interface. Adhering to the library's API, design patterns, and modularity principles becomes a set of constraints that are both important for the technical integration but also the acceptance by other maintainers. As such, despite the theoretical nature of my work, I still need to follow relatively strict constraints when implementing my numerical approach.

4 Data Scientists versus Software Engineers

4.1 Core Differences

Overall, I do agree with the differences that are outlined in the first chapters of the book. A data scientists work centers almost purely around the model and the data set; issues that arise in later steps of the development are usually not considered in their work. This is indeed the area of the software engineer. There are, however, some aspects that are different in my experience. Although one of the major tasks of software engineers is to consider all the constraints and requirements when developing a whole product, also the data scientists tend to consider certain kind of constraints. In particular, the scalability in terms of computation and memory of their chosen algorithm is often directly impacting the design. For instance, while transformers are still the de facto standard for most deep learning applications, their unfavorable quadratic scaling in the input length has led to a numerous alternative architectures that trade off some of the performance for better scalability.

4.2 Evolution of these Roles

Despite the mentioned "unicorns" mentioned in the book are obviously of immense value, I do not think that the amount of such people that have a deep understanding of both areas will grow significantly. In fact, I expect further specialization, up to the point that there will be a third role. This one could be focused on the pure engineering aspects of the machine learning pipeline such as testing, validation, versioning etc. They could adhere to strict requirements coming from further downstream applications that are taken care of by the software engineers.

5 Paper Analysis

5.1 "Investigating Issues That Lead to Code Technical Debt in Machine Learning Systems"

Core Ideas

In Ximenes *et al.* [1], the authors empirically study possible causes in machine learning that lead to technical debt. In particular, they investigate frequency and relevance of issues arising in data collection, data pre-processing model creation & training, as well as model evaluation. In particular, the pre-processing phase was found to be the most problematic one in terms of relevant issues. The authors identify a number of root causes and suggest possible strategies to mitigate them. For instance, inappropriate or wrong data pre-processing can be avoided by automated pipelines, quality checks and verification of completeness.

Relation to my Research

While I have only worked with well established benchmark data sets so far, I still have to perform some pre-processing steps for the particular adaptation at hand. For instance, when testing the rotation-equivariant networks on molecular data, the $3d$ coordinates and atom types had to be converted into signals on the sphere first. There is some freedom in how exactly this is done, so producing a full automated pipeline with transparent configuration of processing parameters is essential to have a reproducible and debuggable workflow.

Integration into a Larger AI-intensive Project

In a project on spherical image recognition for self-driving cars, omnidirectional cameras provide a 360° view of the environment. A central challenge arises already in the data collection phase, where differences in camera mounts and the actual car model introduce inconsistencies in particular at the poles of the spherical images. These variations disturb uniform coverage and create biases in the dataset. Further downstream tasks are then obviously affected and can increase the need for costly retraining. Robust, automated data pre-processing and homogenization can avoid accumulated technical debt from this stage.

Further problems appear in model choice, training, and evaluation. Treating spherical data with standard image methods, which assume a flat space, introduce distortions, especially at the boundaries (which do not exist in reality) and poles. Geometry-aware models can naturally resolve these artifacts. It is exactly those symmetry-aware models that I investigate in my research. Insights about their training dynamics and their connection to hyperparameters etc. can help to design robust algorithms.

Adaptation of my Research

A central issue in my workflow is feature design, particularly its representation on discretized spaces. The discretization scheme of the features has to be compatible with the discretization of the symmetry. In addition, the symmetries should be approximately satisfied even for very coarse-grained inputs. Otherwise, the advantage of the chosen architecture is lost due to the poor equivariance. In such a case, the main advantage compared to data augmentation is lost. Detecting such inadequate choices late forces costly redesigns and reruns. To avoid this form of technical debt, I want to introduce explicit checks for feature–space and model performance at the very beginning of the process, before scaling up experiments. This means systematically verifying that the chosen discretization is well-suited for the targeted symmetries and is also robust under the choice of input resolution. Another aspect that is relevant at this stage is also the scalability of the chosen approach. Unfortunately, one often has to find a trade-off between level of preserved symmetry and computational cost or memory footprint.

Furthermore, when pre-processing the data and designing the input features, one often reduces the amount of information in the process. This does not need to be a problem per se, but it is important to notice the loss of possible crucial information early. In one of my projects, for instance, I have represented atomic environments by projecting the charge density around it on the unit sphere. This is obviously a strong reduction of the full information, but fortunately, the model was still able to learn the target properties in a satisfactory manner.

Another important point is the preparation for training, validating and testing the model. A strict dataset split into the three corresponding sets is essential to obtain trustworthy evaluation. So far, most of the data sets that I have used explicitly declared this split. In principle, one can deviate from that but then the comparison to other benchmark results is not justified anymore. Strictly separating these sets is important to avoid data leakage and reliability of my results.

In the future, I also plan to automate all parts of the model selection. For instance, instead of manually inspecting the training curves and selecting an epoch that has satisfying validation error, early stopping can easily be implemented instead. Otherwise, the risk is that training decisions depend on ad-hoc judgments, which both harms reproducibility and biases comparisons between models. In addition, it also leads to additional overhead in case of changing the setup and retraining the model.

Automatic hyperparameter tuning is also closely related to this. Relying on manual tuning introduces personal biases, since some models inevitably receive more careful adjustment than others. This leads to unfair comparisons and potentially wrong conclusions. Automated hyperparameter search would ensure a consistent exploration of the parameter space across different models, making comparative studies more meaningful. This is, however, not always as easy to fully automate as one might naively think. In my experience, when implementing a relatively little explored architecture, there are often several hyperparameters involved that are not yet well understood. Optimizing all of them simultaneously can easily lead to an intractably large exploration space. Still, one can e.g. first study the sensitivity of individual parameters and then only optimize the n most sensitive ones. While this does not guarantee an approximately optimal choice, it is at least intuitive, scriptable and hence reproducible.

I have found substantial value in using tools such as *Weights & Biases* to track experiments. Currently, I use it mostly to log different combinations of hyperparameters and their resulting performance. While this is already useful, this platform provides a broad range of additional features that address some of the previously mentioned issues. In particular, there exist hyperparameter sweeps and data exploration tools that can make these tasks more systematic. In the future, I plan to adapt my workflow to make more use of these features.

5.2 "Investigating the Impact of SOLID Design Principles on Machine Learning Code Understanding"

Core Ideas

The *SOLID* principles are a set of five well-known guidelines in software engineering aimed to improve code maintainability of object-oriented code. They stand for Single Responsibility (SR), Open-Closed (OC), Liskov Substitution (LS), Interface Segregation (IS), and Dependency Inversion (DI). While these principles are widely adopted in traditional software development, their application in machine learning codebases is less explored. In Cabral *et al.* [2], the authors empirically study the impact of following these principles on understandability of ML code. According to their findings, adhering to each of these guidelines leads to perceived improved understanding of the code. In addition, similar benefits as in traditional software engineering are observed, such as clear responsibilities, easier extensions and substitutions, minimal coupling and well-segregated interfaces.

Relation to my Research

A new project is often related to previous work, and hence, reusing and adapting existing code is a common practice. In addition, I often collaborate with other researchers, who then have to understand my code. Therefore, I do care about producing understandable code. There is obviously a trade-off to be made between spending time on making the code more maintainable and the actual research work. While I am familiar with the SOLID principles and have applied some of them in the past for traditional software, I have not yet systematically applied them to my research code. Interestingly, those easy to follow guidelines seem to also transfer well to ML code and thus .

Integration into a Larger AI-intensive Project

Imagine a mushroom-area-prediction project that aims to identify promising areas with high mushroom density by combining geospatial and environmental data with machine learning. Applying the SR Principle, for instance, the *DataPreProcessing* would not just read files but encapsulate the logic for handling heterogeneous inputs such as soil measurements, vegetation type, humidity recordings etc., ensuring that the rest of the pipeline receives a consistent dataset. On the model side, one could construct sub-models handling different kinds of input data with a final aggregation model that combines those before the final prediction. This follows the OC principle and would allow the inclusion of e.g. satellite imagery by adding a new sub-model and expanding the aggregation layer. Using this principle, only the new sub-model and the aggregation layer would need to be retrained. The other measures mentioned in the paper are general enough to be applicable in this context as well.

Since geospatial data is defined on a sphere, the geometric aspect of my project can become relevant. For instance, this spherical data is best treated with spherical CNNs, which are equivariant to rotations. This avoids artifacts resulting from projecting spherical data on a rectangular grid.

Adaptation of my Research

In my own ML research code, the ease of applicability of the SOLID principles depends on the chosen framework. *PyTorch* already encourages a design that follows some of these principles through its ecosystem of classes: `nn.Module` for models, `Dataset` and `DataLoader` for structured access to training data, `torchvision.transforms` for preprocessing, and additional abstractions such as `Optimizer` or `Scheduler` for training control. This separation of concerns aligns well with the SR principle, while the modularity of `nn.Module` and `torchvision.transforms` naturally supports the OC principle. It is straightforward in PyTorch to swap one model component for another, or to extend the data pipeline without rewriting the entire workflow.

By contrast, my research code is mostly based on *JAX*, which is purely functional. While JAX provides great advantages in serial speed and large-scale parallelism, its functional style makes it far less natural to incorporate object-oriented designs in the spirit of SOLID. In JAX, responsibilities often remain spread across pure functions, i.e. stateless routines, and explicit data transformations, rather than being encapsulated in reusable classes. This means that adopting SOLID principles requires some adaptation in the structure. Since JAX is a relatively low-level framework, there exist other libraries that provide higher-level abstractions like PyTorch does. One example is *Flax*, and indeed it provides a class-based interface for models and layers, which can facilitate adherence to SOLID principles. However, as it is visible in there documentation, functions like conducting a training step or evaluating the model are still implemented as pure functions. There is a good reason for this: those are expensive steps that one would like to make as efficient as possible. A straightforward way to do this in JAX is to use its `jit` compilation. While powerful, it is restricted to pure functions.

Despite these challenges, I still see value in applying SOLID principles to my JAX-based research code. For instance, I can still encapsulate data loading and preprocessing logic in dedicated classes or modules, even if the training loop itself remains a pure function. Some SOLID principles like the DI are also less specific to object-oriented programming and can be applied in a functional context as well. For instance, a function that is more likely to change should take arguments that are considered more stable, while the other way would tie both implementations together. An adaptation of the LS principle would be to design my functions in a duck typing manner, allowing for flexible substitution of passed argument types.

Overall, the main takeaway in my opinion is not the strict adherence to all SOLID principles, but rather the underlying mindset of designing code that is modular, extensible, loosely coupled, and easy to understand by having independent meaningful chunks of code. This mindset is valuable regardless of the programming paradigm and certainly leads to more maintainable research code. Despite sounding almost trivial, the usual creation process of research code is often quite the opposite: quickly throwing together a prototype to test an idea, which then evolves into a more complex codebase that is hard to understand and modify. The fact that this is also often done in *Jupyter* notebooks, which are not ideal for structuring code, makes it even more important to consciously apply such principles. While my projects already feature a reasonable level of modularity and separation of concerns, I think that I can certainly improve on decoupling concepts from the actual implementation. In particular, I will now focus more on defining abstract interfaces and separate them from concrete implementations. This will make it easier to swap out components, experiment with different algorithms, and reuse code across projects.

6 Research Ethics & Synthesis Reflection

The screening process for finding relevant papers was as follows: I first started with the most recent iteration of the CAIN conference and looked at the titles of the papers. This narrowed down the selection to around five papers. After reading the abstracts, only one seemed reasonably relevant to my research. I then continued to do the same with the accepted papers of 2024's CAIN conference, where the first paper that featured a relevant sounding title turned out to be a suitable choice for this assignment. I did not encounter misleading titles, but some of them where not clear to me at first which is why I needed to read the abstract to make a decision. I ensured originality of this assignment by writing everything in my own words and not copying any text from the papers or bluntly copying output generated by LLMs.

References

1. Ximenes, R., Alves, A. P. S., Escovedo, T., Spinola, R. & Kalinowski, M. *Investigating Issues That Lead to Code Technical Debt in Machine Learning Systems* in *2025 IEEE/ACM 4th International Conference on AI Engineering – Software Engineering for AI (CAIN)* 2025 IEEE/ACM 4th International Conference on AI Engineering – Software Engineering for AI (CAIN) (Apr. 2025), 173–183. `https://ieeexplore.ieee.org/abstract/document/11030012` (2025).
2. Cabral, R. *et al. Investigating the Impact of SOLID Design Principles on Machine Learning Code Understanding* in *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI* (Association for Computing Machinery, New York, NY, USA, June 11, 2024), 7–17. ISBN: 979-8-4007-0591-5. `https://dl.acm.org/doi/10.1145/3644815.3644957` (2025).