Real-time reinforcement learning with online training for large-scale facilities

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

von der KIT-Fakultät für Elektrotechnik und Informationstechnik
des Karlsruher Instituts für Technologie (KIT)

angenommene

DISSERTATION

von

M.Sc. Luca Scomparin

geboren in Treviso (Italien)

Tag der mündlichen Prüfung:	28. March 2025
Hauptreferent:	Prof. Dr.-Ing. Dr. h. c. Jürgen Becker
Korreferent:	Prof. Dr. rer. nat. Frank Simon

Abstract

Modern advancements in machine learning (ML) offer solutions to complex problems, ranging from image classification, predicting the folding structure of proteins, providing agents with high-level conversational skills, or controllers capable of besting the current human champion in almost any conceivable game. Reinforcement learning (RL) is the perfect candidate for applying ML techniques to control problems, as it trains an agent to optimize outcomes through interaction with its environment, theoretically creating adaptable, turn-key controllers. The required large amounts of data for training, however, are impractical to gather in most real-world environments. Usually, the agent is thus trained on a simulated copy of the environment and then deployed to the real-world. This approach suffers from the so-called *sim2real* gap: a mismatch between simulation and reality can strongly reduce the performance of the agent. High-interaction-rate environments, like those in large-scale facilities, could quickly generate enough training data.

Online training on data gathered from a real-world environment entirely solves the issue of the *sim2real* gap. Adaptation of established RL algorithms to environments with real-time constraints requires extensive expertise, which is not commonly available to many experiments. In order to simplify the deployment of RL algorithms in these situations, the experience accumulator architecture was developed. This design pattern relies on a real-time agent, interacting with an environment, while its interactions are recorded. In this way the real-time constraints for the agent inference are automatically satisfied. These data is then transferred to another computation unit, where training can be performed with less stringent real-time constraints. A scheme was devised for computing the agent's reward at training time, in this way reducing the amount of components with real-time constraints and increasing the flexibility of the system by allowing exploration of different reward functions during an experiment.

The implementation of RL policies with real-time constraints demands specialized hardware, enabling these ML solutions to operate on the edge by processing data directly on the devices interacting with the environment to minimize overall latency. To implement this architecture and simplify control experiments, the KINGFISHER platform based on the AMD/Xilinx Versal family of devices was designed and implemented. The main underlying idea is to create a library of the interfaces to large-scale facilities and common components necessary when implementing experience accumulator systems. In this way, the data processing path responsible for the low-latency inference of the RL policies becomes more modular and facility-independent, reducing the deployment effort.

A synchrotron light source and accelerator test platform such as the Karlsruhe research accelerator (KARA) is an ideal place to test a technology such as KINGFISHER and experience accumulator systems. This large-scale facility provides several possible control problems that require actions within the microsecond timescale, thus being an interesting candidate for the application of an edge RL controller. The betatron oscillations offer a control problem that is well understood, for which a classical controller exists, allowing a proof-of-principle demonstration of the functionality of the system. On the other hand, a more challenging problem is needed to showcase the capabilities of these techniques. A good candidate is the microbunching instability, characterized by highly nonlinear self-interaction phenomena between an electron bunch and its own emitted coherent synchrotron radiation.

Furthermore, KARA offers cutting edge beam diagnostic infrastructure such as Karlsruhe pulse taking ultra-fast readout electronics (KAPTURE) and Karlsruhe linear array detector for MHz repetition rate spectroscopy (KALYPSO), providing continuous turn-by-turn information on the bunch position, synchrotron light emission, and charge density, that can be employed as the input of a controller. As part of this work, the KAPTURE system was integrated with the KINGFISHER platform, together with a transversal stripline kicker and the main radiofrequency (RF) system of the accelerator. These two systems are capable of acting with low-latency on the dynamics of interest for the control problem.

The system just described allowed performing experiments of the control of the horizontal betatron oscillations. KINGFISHER was first employed to test the conventional controller based on a finite impulse response filter feedback, as a further verification step that proved successful. Several RL-based controllers were then trained and tested. The system could reliably produce a functional controller every time. Often this achieved better performance than the conventional system. Furthermore, the amount of interaction time necessary to obtain enough training data by interacting with the machine is only 0.076 s, compared to the 17.6 s necessary on simulation. This result proves the effectiveness of the experience accumulator approach.

The more challenging problem of the control of the microbunching instability was then tackled. This kind of dynamics exhibits oscillations in two main frequency ranges: a bursting around a few tens of kHz, and a low-bursting around a few hundreds of Hz. These two phenomena represent very different control problems. Nonetheless, a classical controller developed in collaboration with the University of Lille, tackling the low-bursting was tested. This system was proven to be effective only in a specific range of beam currents. The RL controller also showed similar behavior, hinting to a fundamental characteristic of the instability. A few attempts were also carried out targeting the bursting oscillations.

In conclusion, the experience accumulator architecture with training time reward definition was conceived in order to deploy real-time RL agents. Albeit with a reduced flexibility compared to the libraries conventionally employed, the system was shown to be easily transferable to different control problems. Furthermore, the KINGFISHER system on the AMD/Xilinx Versal™ computing platform was designed and produced, leading to a first online learning edge system at a particle accelerator. Finally, two control problems at a large-scale facility were tackled with the system, showcasing its versatility and functionality.

Zusammenfassung

Moderne Fortschritte im Bereich des maschinellen Lernens (ML) bieten Lösungen für komplexe Probleme, wie z. B. die Bildklassifikation, die Vorhersage der Faltstruktur von Proteinen, die Ausstattung von Agenten mit fortschrittlichen Konversationsfähigkeiten oder die Entwicklung von Steuerungssystemen, die menschliche Champions in nahezu jedem erdenklichen Spiel übertreffen können. Verstärkungslernen (RL) ist ein idealer Ansatz, um ML-Techniken auf Steuerungsprobleme anzuwenden, da es Agenten durch Interaktion mit ihrer Umgebung dazu trainiert, Ergebnisse zu optimieren, und somit theoretisch anpassungsfähige und einsatzbereite Steuerungssysteme ermöglicht. Allerdings erfordert das Training große Mengen an Daten, die in den meisten realen Umgebungen schwer zu beschaffen sind. Üblicherweise wird der Agent daher in einer simulierten Umgebung trainiert und anschließend in der realen Welt eingesetzt. Dieser Ansatz leidet unter der sogenannten „Sim2Real“-Lücke: Eine Diskrepanz zwischen Simulation und Realität kann die Leistung des Agenten erheblich beeinträchtigen. Hochinteraktive Umgebungen, wie sie in großtechnischen Anlagen vorkommen, könnten jedoch schnell ausreichend Trainingsdaten generieren.

Das Online-Training mit Daten aus einer realen Umgebung löst das Problem der *Sim2Real*-Lücke vollständig. Die Anpassung etablierter RL-Algorithmen an Umgebungen mit Echtzeitbeschränkungen erfordert umfangreiches Fachwissen, das in vielen Experimenten nicht allgemein verfügbar ist. Um den Einsatz von RL-Algorithmen in solchen Situationen zu vereinfachen, wurde die Experience-Accumulator-Architektur entwickelt. Dieses Designmuster basiert auf einem Echtzeit-Agenten, der mit einer Umgebung interagiert, während seine Interaktionen aufgezeichnet werden. Auf diese Weise werden die Echtzeit-Anforderungen für die Agenten-Inferenz automatisch erfüllt. Diese Daten werden dann an eine andere Recheneinheit übertragen, auf der das Training mit weniger strengen Echtzeit-Anforderungen durchgeführt werden kann. Ein Schema zur Berechnung der Belohnung des Agenten während der Trainingszeit wurde entwickelt, wodurch die Anzahl der Komponenten mit Echtzeit-Anforderungen reduziert und die Flexibilität des Systems erhöht wird, indem die Erkundung verschiedener Belohnungsfunktionen während eines Experiments ermöglicht wird.

Die Implementierung von RL-Strategien mit Echtzeit-Computing erfordert spezialisierte Hardware, sodass diese ML-Lösungen direkt an der Quelle, also auf den Geräten, die mit der Umgebung interagieren, betrieben werden können, um die Gesamtlatenz zu minimieren. Zur Umsetzung dieser Architektur und zur Vereinfachung von Steuerungsexperimenten wurde die KINGFISHER-Plattform auf Basis der AMD/Xilinx Versal-Gerätefamilie entworfen und implementiert. Die zugrunde liegende Idee ist der Aufbau einer Bibliothek von Schnittstellen zu großtechnischen Anlagen sowie häufig verwendeten Komponenten, die für die Implementierung von Experience-Accumulator-Systemen erforderlich sind. Dadurch wird der

Datenverarbeitungspfad, der für die latenzarme Inferenz der RL-Strategien verantwortlich ist, modularer und anlagenunabhängiger, was den Implementierungsaufwand reduziert.

Eine Synchrotronstrahlungsquelle und Plattform für Beschleunigertests wie Karlsruhe research accelerator (KARA) bietet einen idealen Rahmen, um eine Technologie wie KINGFISHER und Experience-Accumulator-Systeme zu testen. Diese großtechnische Anlage stellt mehrere Steuerungsprobleme bereit, die Aktionen im Mikrosekundenbereich erfordern, und ist somit ein interessanter Kandidat für den Einsatz eines Edge-RL-Steuerungssystems. Die Betatron-Oszillationen bieten ein gut verstandenes Steuerungsproblem, für das ein klassischer Regler existiert, was eine Machbarkeitsstudie der Systemfunktionalität ermöglicht. Andererseits ist ein anspruchsvolleres Problem erforderlich, um die Leistungsfähigkeit dieser Techniken zu demonstrieren. Ein guter Kandidat ist die Mikrobunchinginstabilität, die durch hochgradig nichtlineare Selbstwechselwirkungen zwischen einem Elektronenbündel und seiner emittierten kohärenten Synchrotronstrahlung gekennzeichnet ist.

Darüber hinaus bietet KARA eine hochmoderne Strahldiagnose-Infrastruktur wie KAPTURE und KALYPSO, die kontinuierlich turn-by-turn-Informationen über die Bündelposition, die Synchrotronstrahlung und die Ladungsdichte liefern können, die als Eingabe für einen Regler verwendet werden können. Im Rahmen dieser Arbeit wurde das KAPTURE-System mit der KINGFISHER-Plattform integriert, zusammen mit einem transversalen Stripline-Kicker und dem Haupt-RF-System des Beschleunigers. Diese beiden Systeme können mit niedriger Latenz auf die für das Steuerungsproblem relevanten Dynamiken einwirken.

Das beschriebene System ermöglichte Experimente zur Steuerung der horizontalen Betatron-Oszillationen. KINGFISHER wurde zunächst eingesetzt, um den herkömmlichen Regler basierend auf einem FIR-Filter-Feedback zu testen, was als erfolgreicher Verifizierungsschritt diente. Anschließend wurden mehrere RL-basierte Regler trainiert und getestet. Das System konnte zuverlässig jedes Mal einen funktionierenden Regler erzeugen, der oft eine bessere Leistung als das herkömmliche System zeigte. Darüber hinaus betrug die für die Datensammlung durch Interaktion mit der Maschine benötigte Zeit nur 0.076 s, verglichen mit 17.6 s in der Simulation. Dieses Ergebnis belegt die Effektivität des Experience-Accumulator-Ansatzes.

Das anspruchsvollere Problem der Kontrolle der Mikrobunchinginstabilität wurde ebenfalls angegangen. Diese Dynamik zeigt Oszillationen in zwei Hauptfrequenzbereichen: ein Bursting im Bereich von einigen zehn kHz und einem Low-Bursting im Bereich von einigen hundert Hz. Diese beiden Phänomene stellen sehr unterschiedliche Steuerungsprobleme dar. Dennoch wurde ein in Zusammenarbeit mit der Universität Lille entwickelter klassischer Regler getestet, der das Low-Bursting adressierte. Dieses System erwies sich als effektiv nur in einem spezifischen Bereich von Strahlströmen. Der RL-Regler zeigte ein ähnliches Verhalten, was auf eine grundlegende Eigenschaft der Instabilität hindeutet. Einige Versuche wurden auch durchgeführt, um die Bursting-Oszillationen zu adressieren.

Zusammenfassend wurde die Experience-Accumulator-Architektur mit der Belohnungsdefinition während der Trainingszeit entwickelt, um Echtzeit-RL-Agenten einzusetzen. Obwohl das System im Vergleich zu herkömmlichen Bibliotheken eine reduzierte Flexibilität aufweist, konnte es leicht auf unterschiedliche Steuerungsprobleme übertragen werden. Darüber hinaus wurde das KINGFISHER-System auf der AMD/Xilinx Versal-Computing-Plattform entwickelt und implementiert, was zu einem ersten Online-Lernsystem an einem Teilchenbeschleuniger führte. Schließlich wurden zwei Steuerungsprobleme an einer großtechnischen Anlage mit dem System angegangen, was seine Vielseitigkeit und Funktionalität unter Beweis stellte.

Contents

1. Introduction

1.1. Motivation

Machine learning techniques have been proven successful in a wide variety of tasks that were before considered only approachable by a human. Driven by the availability of embarrassingly parallel computing platforms such as GPU, ever-larger models allowed to vastly extend the complexity of tasks that can be approached by these techniques. For example, unprecedented image classification was achieved in 2012 by AlexNet [1], paving the way to the usage of large convolutional neural network-based models. In 2022, a mere ten years afterwards, the large language model revolution occurred with OpenAI releasing ChatGPT. This large language model was the first widespread commercially available tool capable of carrying out conversations with a user through a chat-like interaction [2] while exhibiting human-level conversational skill. Furthermore, the 2024 Nobel Prize for Chemistry was shared by the AlphaFold [3] team. Their model was capable of predicting the folding structure of proteins, a previously unsolved problem in biochemistry. The difference in complexity that occurred in a span of only ten years is striking: AlexNet had 60 million parameters, while some versions of ChatGPT exceeded the 175 billion [4]. Recent versions of ChatGPT are estimated to have an even greater number of parameters. The possibility for ever greater computational power and machine learning models allows to approach problems that were previously impossible.

In the case of control problems, techniques such as reinforcement learning (RL) allow training an agent to optimize a reward signal by using information from interaction with an environment. These kind of techniques were employed in a wide variety of complex problems, such as playing Atari games surpassing expert human level [5]. The AlphaGo [6] model was capable of beating the Go world champion. Due to the sheer number of possible game states, it was considered much more complex than chess. For more scientific applications, RL algorithms were also capable of controlling the plasma in a fusion reactor [7].

One of the major benefits of RL techniques is their adaptability to varying conditions: the training procedure can be used, in case of drifts or variation of the control problem, to retune the agent. This capability is analogous to the one achievable with adaptive control techniques [8], were the parameters of a classical controller can be tuned based on the reconstructed state of the system under control. The key difference is the trade-off between problem complexity and stability guarantees. RL can solve extremely complex and non-linear problems, usually unattainable with the adaptive control approach. Its main drawback is that safety and stability guarantees are lacking, as safe-RL is still an open field of study [9], while the behavior of adaptive control techniques can be designed in a more deterministic manner.

Currently, the range of applicability of RL techniques to real-world problems is hindered by the large amount of data necessary for their training. Albeit some novel algorithms are mitigating this behavior [10], this usually makes training on a simulated version of the environment the only feasible option. Such a technique, although functional, encounters the so-called *sim2real* gap: subtle differences between the real-world environment and its simulation could lead to an under-performing, or non-functional agent.

The interaction rate with some real-world environments can be very high, in the order of several millions of interactions per second. This is often the case in large-scale facilities, which are specially designed installations equipped for specific purposes, requiring significant resources to operate, e.g. research, manufacturing, and commercial, such as experimental fusion reactors, particle accelerators, datacenters, etc.

In this case, obtaining enough data for training an agent would require only of a few seconds, opening the possibility of training an RL agent online by interaction with the real-world environment. The drawback of this approach is that these environments are challenging from a computational perspective, as they require an action to be chosen within a time frame dictated by the dynamics targeted by the control problem. This kind of requirement is known as a real-time constraint.

Performing computations and inference at such an high rate is challenging and requires specialized hardware and design techniques [11]. In order to minimize the inference latency of an agent, a common approach is for such systems to be deployed at-the-edge. Compared to cloud deployment, where computing resources are in a different physical location from where data is gathered, edge system process data were it is acquired. The reduction of data transfer times translates into a latency reduction. Furthermore, computing devices such as FPGAs allow to precisely define the timing of the data processing pipeline, making it possible to ensure that a given latency constraint is satisfied.

1.2. Research questions

This work uses a set of research questions to guide its investigation. These questions arise from the introduction, but their contextualization and answer will be carried out in the rest of this dissertation.

First of all, in order to approach these high repetition rate environments, one needs to investigate how RL algorithms can successfully be implemented in such a setting. The current implementations available cannot reach the required level of determinism and latency, and some new architectural ideas need to be conceived. Ideally, this new architecture should impact the flexibility of the system only marginally. Thus one could ask:

Question 1 *How can reinforcement learning algorithms be adapted so training on high-repetition rate environments can be performed?*

Similarly to the RL algorithm implementations, the computing platforms where they are usually deployed on, such as central processing units (CPUs) and graphics processing units (GPUs), are not intended to operate with real-time in mind. Employing the platforms such as the ones intended for edge computing could greatly improve the performance of the system. As such:

Question 2 *How can reinforcement learning be deployed to an edge computing platform?*

Such a RL platform should be verified to be functional, possibly by testing it at a large-scale facility or other high-repetition rate environments to ensure it operates as expected. Moreover, the fact that it is also capable of learning from the interaction with the environment and achieve control needs to be demonstrated. So:

Question 3 *Is the edge-computing platform capable of learning online from interaction with an environment?*

Finally, it is interesting to compare the performance of these self-learning RL algorithms with more traditional control approaches, in order to determine their range of implementation and adaptability. Specifically, the training capability of RL allows to obtain an adaptive controller, but adds a further layer of complexity to the system. Understanding the difference of this approach with respect to a more conventional one is thus fundamental to guide future applications of the results described in this work. Thus:

Question 4 *How does online edge reinforcement learning compare to more traditional control approaches?*

1.3. Objectives and contributions

In order to address question 1, a critical discussion of the timing constraints dictated by modern RL algorithms will be carried out. The main contribution of this work is a training scheme allowing to reuse current RL implementations as much as possible, so that the deployment and experimental effort is minimal, while trying to still maintain the microsecond latency performance needed during inference [12].

Additionally, the design of high-performance neural networks (NNs) for use as RL policies targeting the novel AMD/Xilinx Versal™ platform, together with the necessary logic required for gathering training data, will be discussed in order to answer question 2. The necessary interfaces to communicate with a large-scale facility, together with the training data logic, have been packaged into the KINGFISHER framework [13].

Finally, question 3 and question 4 will involve the application of the aforementioned platform to two real-world problems [12, 14]. This represents the first application of low-latency online-learning RL at a particle accelerator. The performance of the approach will then be compared with the current state of the art controller. The stability of the RL controller is beyond the scope of this work.

1.4. Outline

The work presented in this thesis is multidisciplinary by nature, as it requires an understanding of RL algorithms, how the electronics they will be deployed on operates, and what are the challenges encountered when working with particle accelerators. For the first deployment at a large-scale facility, before any intended hardware inter-operability standard is in place, one needs deep understanding of the functional aspect of the facility, together with the ones of the hardware trying to control it. As such, chapter 2 introduces in its three section these main background topics.

In section 2.1, an introduction to particle accelerators and the dynamics of particle beams will be carried out, together with a description of the Karlsruhe research accelerator (KARA) employed as a testing large-scale facility and its cutting edge diagnostic infrastructure.

To better understand the challenges of implementing RL algorithms, section 2.2 provides an introduction to machine learning, neural networks, and modern RL algorithms. Furthermore, the current state of the art implementations are described, together with some deployments to large-scale facilities that do not require strong real-time constraints.

Section 2.3 gives an introduction to the some available computing platforms, some of which will be employed in the later parts of this work. The challenging of real-time programming will be also discussed. Finally, the AMD/Xilinx Versal™ platform chosen as an edge platform will be described, together with its programming model.

After this introductory chapter, chapter 3 discusses the architecture of an RL edge system. Its implementation, together with a discussion of how neural network policies can be deployed to the AMD/Xilinx Versal™ platform, is carried out in chapter 4. Chapters 5 and 6 describe the application of the platform to two different kind of dynamics encountered at the Karlsruhe research accelerator. In chapter 5, the system is applied to the control of the horizontal betatron oscillations, for which a reliable classical controller is available, allowing to perform extensive comparisons, specifically regarding the adaptive nature of RL. Finally chapter 6 applies the system to the control of the microbunching instability, a problem for which a general solution has not yet been found.

2. Fundamentals and state of the art

The topic described in this work is multidisciplinary, combining elements from several different field. In order to fully contextualize the results, three subtopics will be introduced: particle accelerators in section 2.1, reinforcement learning in section 2.2, and computing platforms in section 2.3. Accelerators are a fundamental of this work, as their non-linear dynamics with timescales in the order of µs are the perfect candidate to apply the RL algorithms. This feat would be impossible without deep knowledge of computing platforms, specifically their timing characteristics.

2.1. Particle accelerators

As discussed in the Introduction, one example of a large-scale facility where extreme conditions might be encountered are particle accelerators, such as synchrotron light sources. In this chapter, a brief introduction to the physics governing synchrotron light sources is provided, with a special focus on the longitudinal and transversal dynamics. Furthermore, the effect of coherent and incoherent synchrotron radiation emissions will be discussed, with a particular focus on the microbunching instability. Finally, Karlsruhe research accelerator (KARA) at KIT is then described, together with some challenging phenomena that might benefit from high-speed controllers. Finally, the high-performance diagnostic system available at the facility will be described.

KARA will serve as the test facility for the control system presented in this work, as it is readily available and provides challenging conditions that are difficult to capture in a simulation.

2.1.1. Synchrotron motion

A charged particle can be guided along a circular path by means of an external magnetic field $\vec{B}$ thanks to Lorentz force

$$\vec{F} = q\vec{v} \times \vec{B}, \tag{2.1}$$

where $\vec{v}$ is the velocity of the particle and $\vec{F}$ is the force being applied to it[1]. In practice, the magnetic field is usually applied to a small volume called *beam pipe*, where the reference orbit

[1] The path is not actually circular due to synchrotron radiation emission, more fully described in sections 2.1.1 and 2.1.3

of a particle resides in. The pipe is kept under ultra-high vacuum conditions to avoid particle losses.

To balance the energy losses of the stored particles and to accelerate them, an alternating radiofrequency (RF) field is employed in order to produce a longitudinal electric field. In practice this is done by applying an high power RF signal to a *cavity*, where the standing wave mode produces the desired potential. This field usually oscillates at a frequency f_{RF} which is a multiple of the revolution frequency f_{rev}. Inside of the cavity, the RF potential encountered will have the form

$$V_{\mathrm{acc}} = V_0 \sin\left(2\pi f_{\mathrm{RF}} t\right), \tag{2.2}$$

where V_0 is the amplitude of the RF field, and V_{acc} the potential observed by a particle arriving at time t.

A particle accelerator can have several operation modes, defined by the combination of electromagnetic fields used for confinement and acceleration. Each of these modes has a *reference orbit* and energy defined during its design. A particle in such an orbit will remain on it if no perturbations are present. The revolution period along the machine is usually a multiple of the RF period. This allows to define the *harmonic number*, $h \overset{\mathrm{def}}{=} f_{\mathrm{RF}}/f_{\mathrm{rev}}$. If the path length and velocity are constant over the reference orbit, given the period matches a multiple of the RF period, its arrival phase on the RF sinusoid will be constant. This phase is called the *synchronous phase*, φ_0.

The set of magnetic fields of an accelerator, usually denoted as *lattice*, can be designed such that particles with an energy above the reference energy take a longer time to travel around the accelerator (figure 2.1). The opposite is true for particles with lower energy, as they will arrive earlier. For the correct slope of the RF voltage sinusoid, a lower energy particle, arriving late, encounters a stronger accelerating voltage, thus compensating its lower energy. Conversely, a higher energy particle will arrive late, obtaining a lower energy gain.

In the approximation of ultra-relativistic particles, for which $\beta \overset{\mathrm{def}}{=} v/c \approx 1$, it is possible to construct a toy model of the longitudinal dynamics of the accelerator as follows. The path-length difference of an off-energy particle is usually described with the longitudinal momentum compaction factor α_c, that is defined as[2]

$$\alpha_c \overset{\mathrm{def}}{=} \frac{dL/L}{dp/p_0} \approx \frac{\Delta t_{\mathrm{arr}}/dt}{dp/p_0}, \tag{2.3}$$

where p_0 is the reference momentum, L is the path length, and Δt_{arr} is the arrival time difference compared to the reference particle over a time interval dt. This allows to compute the phase difference of an off-momentum particle as

$$\Delta t_{\mathrm{arr}} = \alpha_c \frac{\delta}{p_0} dt. \tag{2.4}$$

[2] For non-ultrarelativistic particles, or in cases where α_c is in the same order of magnitude as $1/\gamma^2$, the so-called *phase-slip factor*, $\eta \overset{\mathrm{def}}{=} \alpha_c - 1/\gamma^2$, needs to be used, as it contains a part accounting for the mass of the particles.

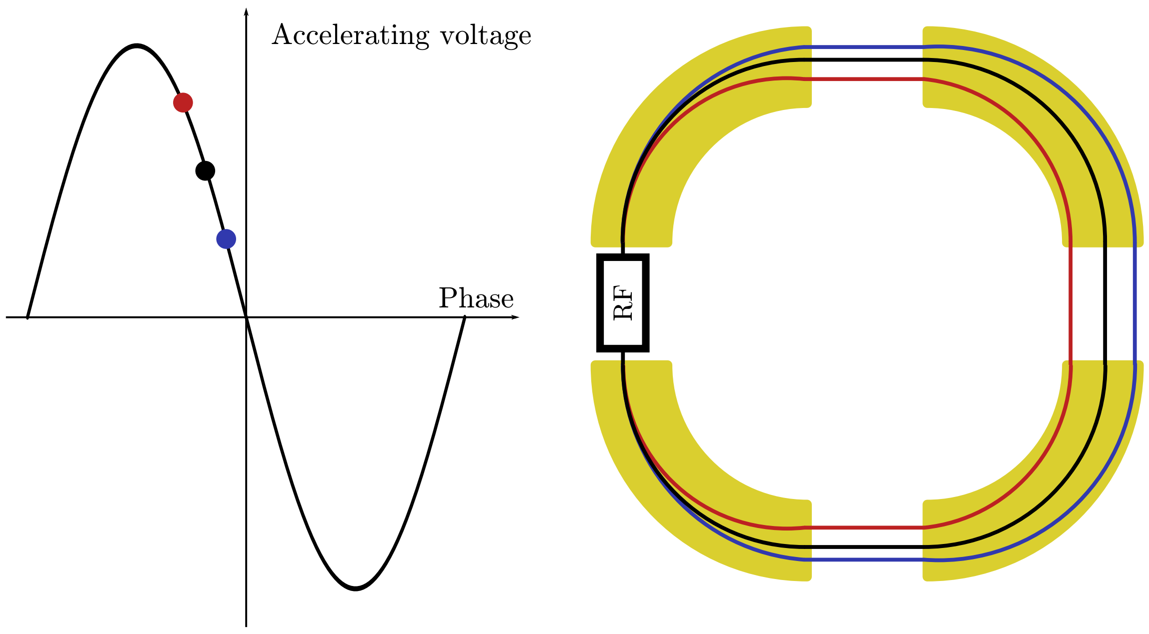

Figure 2.1.: Schematic of the working principle of phase focusing. Particles with higher energy (blue), will have a longer trajectory than the reference particle (black), arriving late and thus experiencing a smaller energy gain. The converse is true for the lower energy particles (red).

where $\delta \overset{\text{def}}{=} p - p_0$ is the momentum difference from the reference momentum.

By using the expression for the accelerating voltage of equation (2.2), one can linearize the problem and obtain the rate of energy gain for an off-energy particle with a given arrival time difference

$$d\frac{d\delta}{dt} \approx q\frac{dV_{\text{acc}}}{dt}\bigg|_{\varphi_0} \Delta t_{\text{arr}} f_{\text{rev}} = qV_0\cos(\varphi_0)2\pi f_{\text{RF}}\alpha_c\frac{\delta}{p_0}dt f_{\text{rev}}, \tag{2.5}$$

where the f_{rev} is used because the increase of energy from the acceleration potential happens at a rate of once per revolution.

By rearranging the terms, one obtains

$$\frac{d^2\delta}{dt^2} = \frac{2\pi qV_0\cos\varphi_0 {f_{\text{rev}}}^2 h\alpha_c}{p_0}\delta, \tag{2.6}$$

corresponding to the equation of an harmonic oscillator with frequency

$$f_s = f_{\text{rev}}\sqrt{\frac{qV_0\cos\varphi_0 h\alpha_c}{2\pi p_0}}, \tag{2.7}$$

known as the *synchrotron frequency*.

This acts as a reaction force leading to time-of-arrival, momentum, and longitudinal oscillations known as synchrotron oscillations. It is worth noticing how this has the effect of stabilizing the machine (figure 2.1): off-energy particles will not be lost, as they will start to oscillate around the reference particle. This phenomenon is called *phase-focusing* and is fundamental

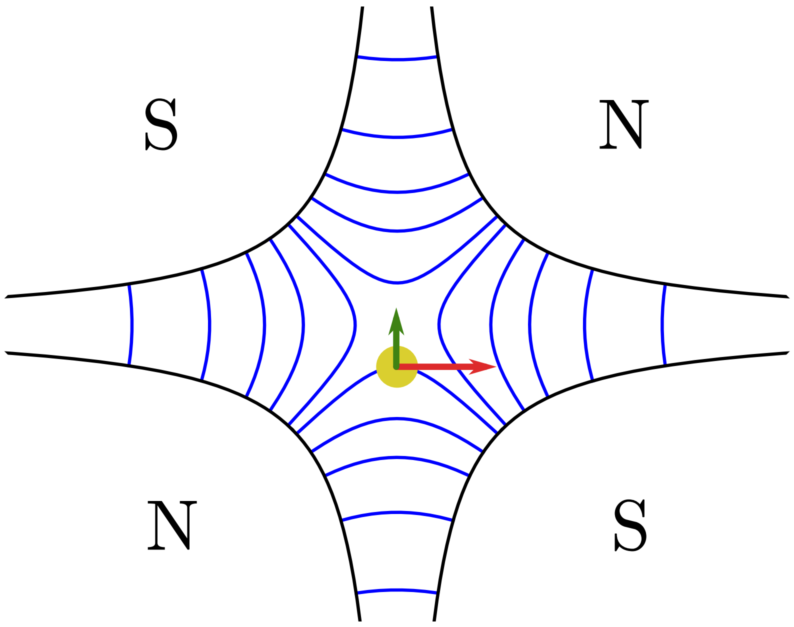

Figure 2.2.: Magnetic field lines of a quadropole magnet. An electron (yellow) traveling in the direction entering the page, experiences a force (green) due to the local magnetic field (red).

to the operation of these machines. Moreover, it gives insight on the name of the machine: to increase the energy of the stored beam it is sufficient to *ramp* the magnetic field. This will change the synchronous phase making the bunch oscillate around it. The RF voltage needs then to be varied *synchronously* with the magnetic field. A particle accelerator using these principles is called a *synchrotron*. The mechanism responsible for the damping of these oscillations is described in section 2.1.3.

2.1.2. Betatron motion

Similarly to how phase-focusing stabilizes the beam in a synchrotron along the longitudinal degree-of-freedom, it is possible to design the magnetic fields of the accelerator in a way where also horizontal and vertical stability is guaranteed. This is usually achieved with quadrupole magnets, albeit properly designed dipole magnets can nonetheless produce focusing.

As shown in figure 2.2, quadrupole magnets have four magnetic poles, arranged 90° from each other, with opposing polarities. A particle displaced by a distance d will experience a magnetic field $B = Gd$, where G is called the quadrupole gradient. Notably, $\vec{B}$ had direction perpendicular to d. As such, due to Lorentz law (equation (2.1)), a particle would experience a force proportional to the displacement and directed along it

$$F = qvGd. \tag{2.8}$$

Gauss's law for the magnetic field states that

$$\nabla \cdot \vec{B} = \frac{\partial B_x}{\partial x} + \frac{\partial B_y}{\partial y} + \frac{\partial B_z}{\partial z} = 0. \tag{2.9}$$

The partial derivative along z vanishes within the magnet, because it is symmetric along that direction. Along x the gradient is G by construction, so this fundamentally fixes the gradient along y to $-G$. Thus a quadrupole magnet is capable of focusing particles in one axis, while having a defocusing action in the other. This behavior can be fixed by combining several quadrupole magnets in doublets and triplets.

While traveling along the accelerator, a particle will experience a force according to equation (2.8), by substituting the expression of the momentum,

$$m\gamma \frac{d^2y}{dt^2} = qvG(t)y, \tag{2.10}$$

where $\gamma = \sqrt{1-\beta^2}^{-1}$. It is worth noticing that the gradient will vary along the accelerator, thus it has a time dependence. The direction y was chosen because for motions in the accelerator plane x, and additional component is present due to the circular motion of the particles around the ring. It is possible to rewrite equation equation (2.10) as

$$\frac{d^2y}{dt^2} + K_y(t)y = 0, \tag{2.11}$$

where K are called *focusing functions*. The equation for the x direction is identical, with the exception of containing an additional term within the K function.

An important characteristic of the function K is that it is periodic: every revolution the beam will return to a region with the same gradient. In the specific case where K is periodic, equation (2.11) is called Hill's differential equation. For accelerators an additional simplification can be performed: due to the fact that the gradient within a magnet is approximately constant, K is piecewise constant. As such it can be studied as the chaining of several harmonic oscillators that periodically change harmonic constant. The study of the stability of this function is beyond the scope of this work, but it is possible to design a lattice that has a stable solution. In that case, the solution to Hill's equation is

$$\begin{aligned} y(t) &= y_0\sqrt{\beta_y(t)}\cos(\psi_y(t) + \psi_{y,0}) \\ \psi_y(t) &= \int_0^t \frac{d\tau}{\beta_y(\tau)}, \end{aligned} \tag{2.12}$$

where β_y is the *beta-function*, a periodic function behaving like the local oscillation period.

An observer measuring the beam position at a fixed point in the accelerator will sample $y(t)$ at each revolution. Specifically,

$$y_r = y(rT_{\text{rev}}), \tag{2.13}$$

where r is now the revolution index, T_{rev} is the revolution period. The number of oscillation periods at each revolution is constant, due to the periodicity of the beta-function, as such, one can define the *betatron tune* as

$$Q_y \overset{\text{def}}{=} \frac{1}{2\pi}\int_0^{T_{\text{rev}}} \frac{d\tau}{\beta_y(\tau)}. \tag{2.14}$$

From y_r, only the measurement of the fractional part of Q can be performed, as we might be under-sampling the betatron oscillation.

It is worth noticing though, that this introduction is primarily meant as a way of understanding the control problem that will be at hand. The topic, in fact, is vast and there are effects like chromatic aberrations, where higher energy particles are less bent by magnetic fields, i.e. are less focused, or the fact that a properly designed dipole magnet could in theory serve as an additional horizontal focusing element, that are beyond the scope of this introduction.

2.1.3. Synchrotron radiation

Accelerated charged particles emit electromagnetic radiation. A particle on a circular orbit, even if its speed is constant, experiences a non-zero acceleration created by the change of direction of its velocity vector. The emitted power can be computed with Larmor's formula [15, 16]

$$P_{\parallel} = \frac{q^2}{6\pi\epsilon_0 c^3 m^2}\left(\frac{d\vec{p}}{dt}\right)^2 \tag{2.15}$$

$$P_{\perp} = \frac{q^2\gamma^2}{6\pi\epsilon_0 c^3 m^2}\left(\frac{d\vec{p}}{dt}\right)^2 \tag{2.16}$$

For a particle in a circular accelerator, the perpendicular component is dominant. By using the fact that, for a particle moving on a circle

$$|d\vec{p}| \approx p\,d\theta = \frac{\beta c dt}{R}p, \tag{2.17}$$

where $d\theta$ is an infinitesimal curvature angle, R is the curvature radius and $\beta = v/c$. If this expression is substituted into equation (2.16), one obtains the power emitted by synchrotron radiation, that in the ultrarelativistic regime when $\beta \approx 1$ becomes

$$P_s = \frac{q^2 c}{6\pi\epsilon_0}\frac{1}{(mc^2)^4}\frac{E^4}{R^2}. \tag{2.18}$$

Noticeably, $P_s \propto m^{-4}$ meaning lighter particles will radiate more power. This explains why this effect is extremely pronounced in electron accelerators. Additionally, the dependence $P_s \propto E^4$ makes it increase very sharply with the energy of the particle. The power spectrum of the synchrotron radiation, in this case for KARA that will be better described in section 2.1.5, is shown in figure 2.3.

For control problems, synchrotron radiation plays an extremely important role. In the case of synchrotron radiation, for instance, the emitted power is greater for higher energy particles, and lower for lower energy ones. This in turn damps the synchrotron oscillations described in section 2.1.1. In the case of the betatron oscillations, on the other hand, the recoil resulting from the emission of radiation provides a very similar damping phenomena. A rule of thumb is that the damping time is in the order of $\tau \approx E/P_s$, i.e. roughly the time it would take the synchronous particle to loose all its energy via synchrotron radiation.

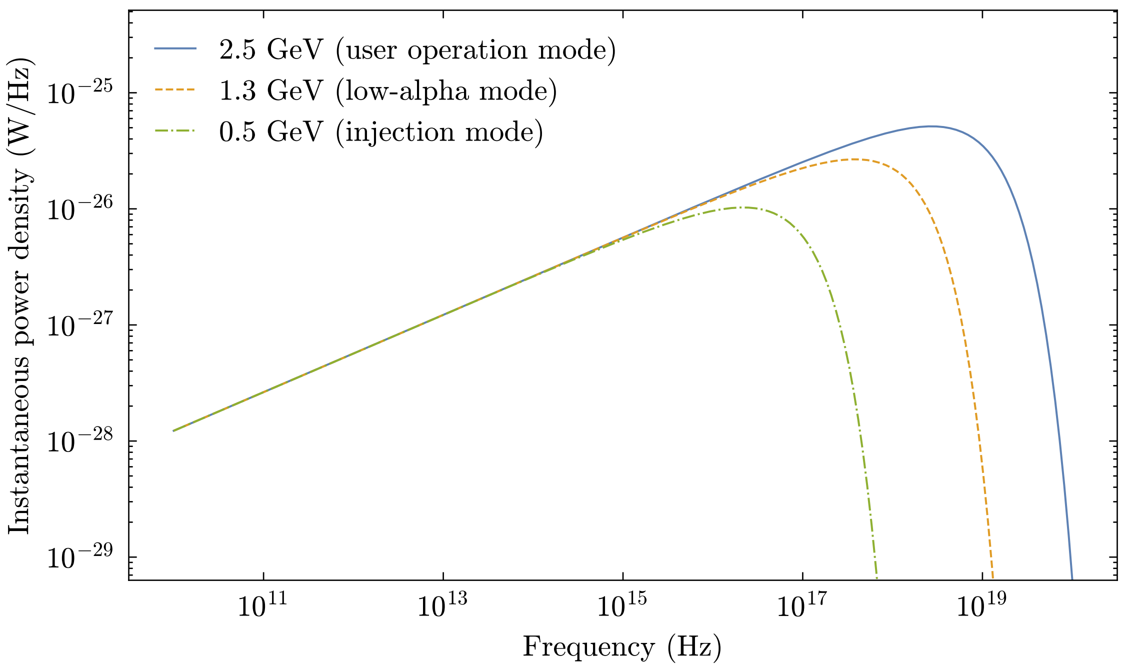

Figure 2.3.: Instantaneous radiated power per unit frequency for a single electron at three energies, which are typically set at KARA during operation.

Depending on the charge distribution of the bunch, synchrotron radiation can be emitted incoherently or coherently (left side of figure 2.4). Incoherent synchrotron radiation (ISR) originates when the wavelength of the emitted radiation is smaller than the size of the bunch. In this case, the phases of the electromagnetic field of each particle are not correlated, and thus the fields are summed randomly, leading to an intensity of radiation $I \propto N$, with N the number of particles. If the wavelength is greater or comparable to the size of the bunch, the radiation from the particles is summed coherently, in this case leading to coherent synchrotron radiation (CSR) with the characteristic behavior that $I \propto N^2$.

This strong enhancement in radiation emission is considerable intensity gain for accelerator-based light sources, allowing the production of light with unprecedented brilliance. In the next section, one of the mechanisms responsible for the emission of CSR at synchrotron light sources will be described.

2.1.4. Microbunching instability

The electromagnetic fields from the magnets and RF cavities are not the only one experienced by the particles in a synchrotron. A given bunch can interact with its own emitted electromagnetic fields, or with the field of the other bunches. This can be modelled as an additional potential, $V_{\text{wake}}(t)$ the wake potential, that is summed to the one of the accelerating structures. This can be computed by defining a wake function $W(t)$, the potential that would be produced by a

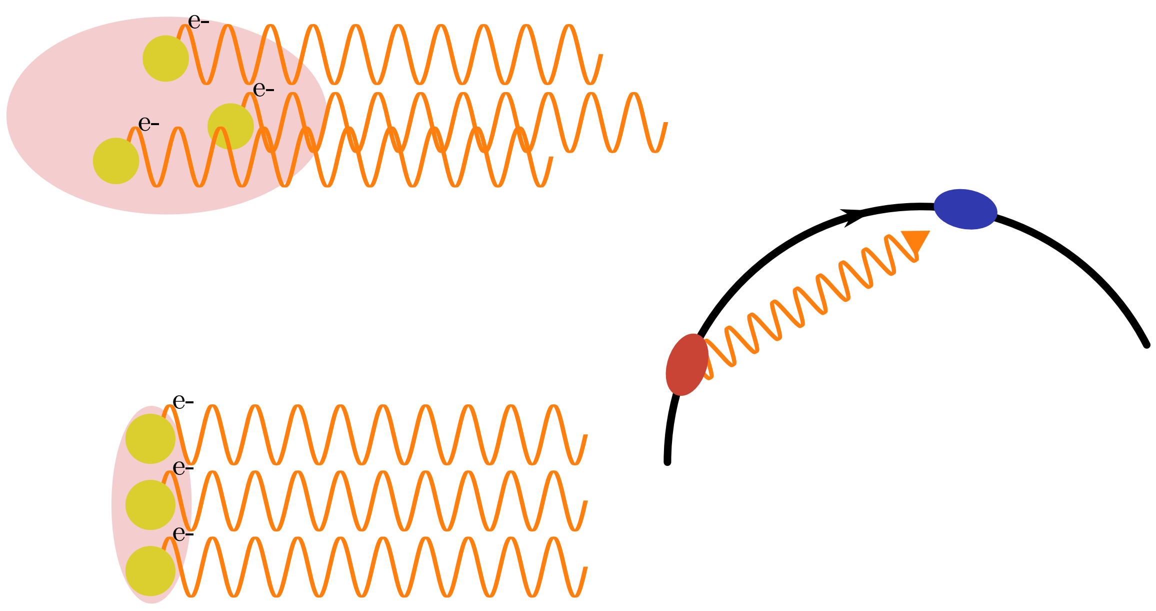

Figure 2.4.: On the left, this drawing compares incoherent (top-left) and coherent (bottom-left) synchrotron radiation emissions. The right side illustrates an exaggerated representation of the radiation emitted by the bunch (red) catching up with the bunch itself (blue).

single charged particle. The total wake potential can be computed as the convolution of the wake function with the charge density $\rho(t)$, i.e.

$$V_{\text{wake}}(t) = \int_{-\infty}^{+\infty} d\tau \; W(t-\tau)\,\rho(\tau). \tag{2.19}$$

In synchrotron light sources, when the charge density increases above a certain threshold, microstructures in the longitudinal phase space start forming. These structures have a characteristic size such that they usually emit coherently in the terahertz frequency range. This emitted CSR is moving on a straight path, while the beam is curved on a closed path inside a dipole magnet (figure 2.4). As such the CSR creates wakefields, that in turn interacts with the microstructures that formed it. The effect of this self-interaction is that the emitted CSR power fluctuates sharply, in a phenomenon known as microbunching instability (MBI).

Systematic studies of this instability have been performed, highlighting its strong dependence on several parameters as the longitudinal momentum compaction factor α_c, the accelerating voltage, the energy of the beam, the bunch current (or charge) and the longitudinal damping time [17].

In figure 2.5, the fast Fourier transform (FFT) of the CSR signal amplitude (shown in figure 2.6) is shown as a function of the bunch current, for one specific machine setting. The instability starts above a threshold, where a roughly sinusoidal oscillation starts. Due to the shape of this feature in the plot, this is usually referred as *finger* or bursting frequency, and usually originates from the rotation of the microstructures in the phase space. As such, it is usually a multiple of the synchrotron frequency. When the current increases more, a second threshold is reached. Here the interaction of the CSR with the bunch is strong enough that the microstructures are

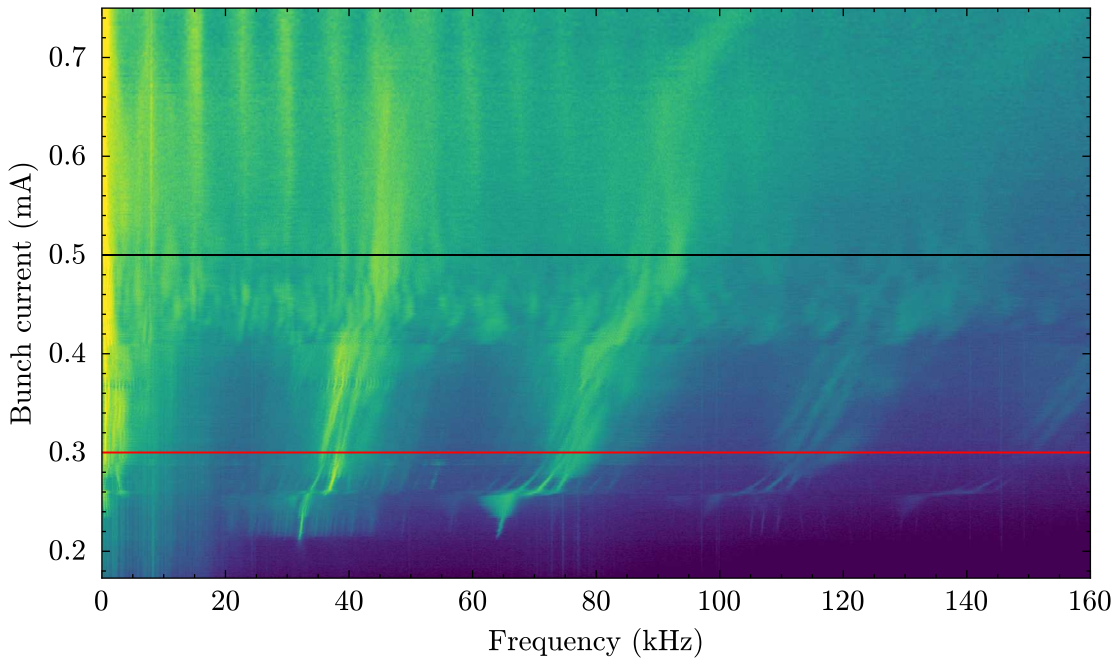

Figure 2.5.: Waterfall plot showing the FFT of the CSR power signal as a function of the current. The instability starts above 0.2 mA, with structures that strongly depend on the current. The horizontal lines indicate the current of the signal shown in figure 2.6. Data courtesy of Johannes L. Steinmann.

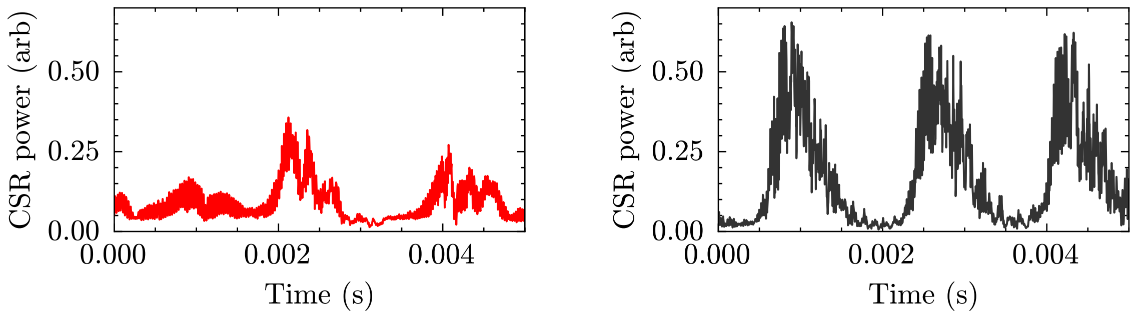

Figure 2.6.: Example CSR power signals corresponding to the currents of 0.3 mA (left) and 0.5 mA (right), also indicated by the horizontal lines in figure 2.5. Data courtesy of Johannes L. Steinmann.

washed out and the bunch length is usually increased. This stops the emission of CSR until the synchrotron radiation damping reduces the bunch size to a level where the microstructures can form and the process starts again. In this regime, the a slower, sawtooth shaped oscillation is superimposed to the bursting, and is called *low-bursting*.

2.1.5. The Karlsruhe research accelerator

The Karlsruhe research accelerator (KARA) is located at the Karlsruhe Institute of Technology, in Germany. It is a ramping electron storage ring, meaning that particles are injected at a lower energy, in this case 500 MeV, and then accelerated up to a maximum energy of

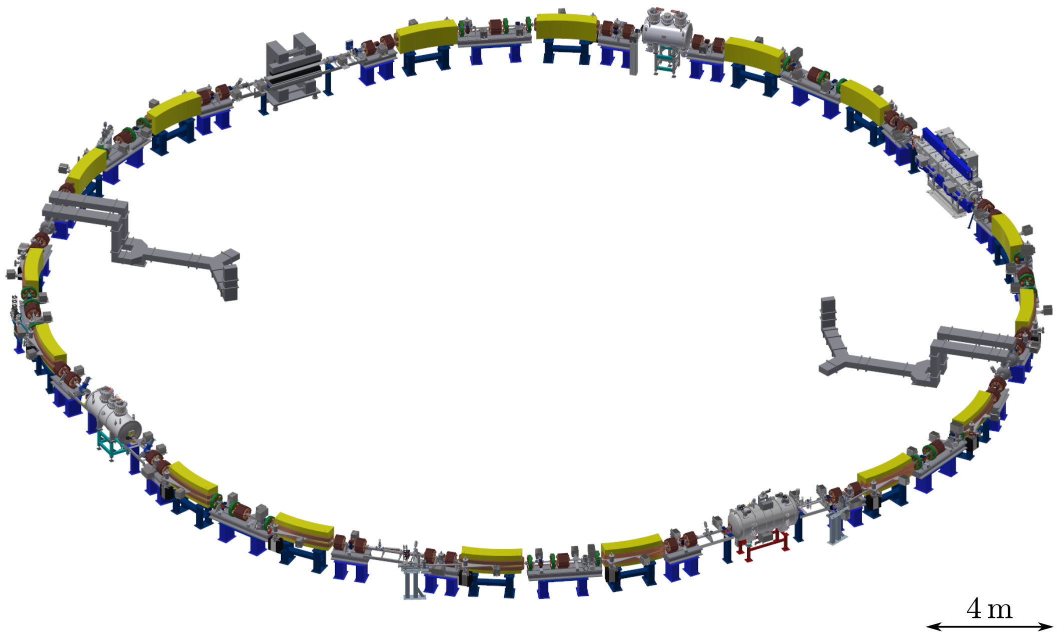

Figure 2.7.: Schematic of KARA. The beamlines are not shown. The dipole, quadropole and sextupole magnets are colored yellow, red, and green, respectively. Inside of the ring, the waveguides of the two RF stations are visible. Courtesy of Till Borkowski.

2.5 GeV. Its main RF system frequency is 500 MHz, with the accelerating voltage being applied in two station in opposing sectors of the machine. The ring is 110 m long, leading to a revolution frequency of 2.7 MHz and an harmonic number of 184. One of the peculiarities of this machine is the possibility of finely controlling the double-bend achromat lattice by tuning the five quadrupole and two sextupole families, in this way creating custom operation modes. Additionally, controlling the dipole, or *bend*, magnets, allows to control the energy of the specific operational mode [18].

The conventional operation mode for the synchrotron radiation users is at an energy of 2.5 GeV. Additionally, special modes with low, or negative, momentum compaction factor α_c are available at several different energies [19]. The low-alpha mode, used to study the microbunching instability described in the previous sections, is usually at an energy of 1.3 GeV, and allows the selection of a custom value of α_c from 10^{-2} (user mode) to 10^{-4}.

The beam is produced by an electron gun, at an energy of 90 keV [18]. It is then injected into a racetrack microtron that accelerates it to an energy of 53 MeV. Once a second, this system injects electron into the booster synchrotron that accelerates them to 500 MeV in roughly 500 ms. A system of three kickers and a septum merges the beam already being present in the KARA storage ring with the one coming from the booster.

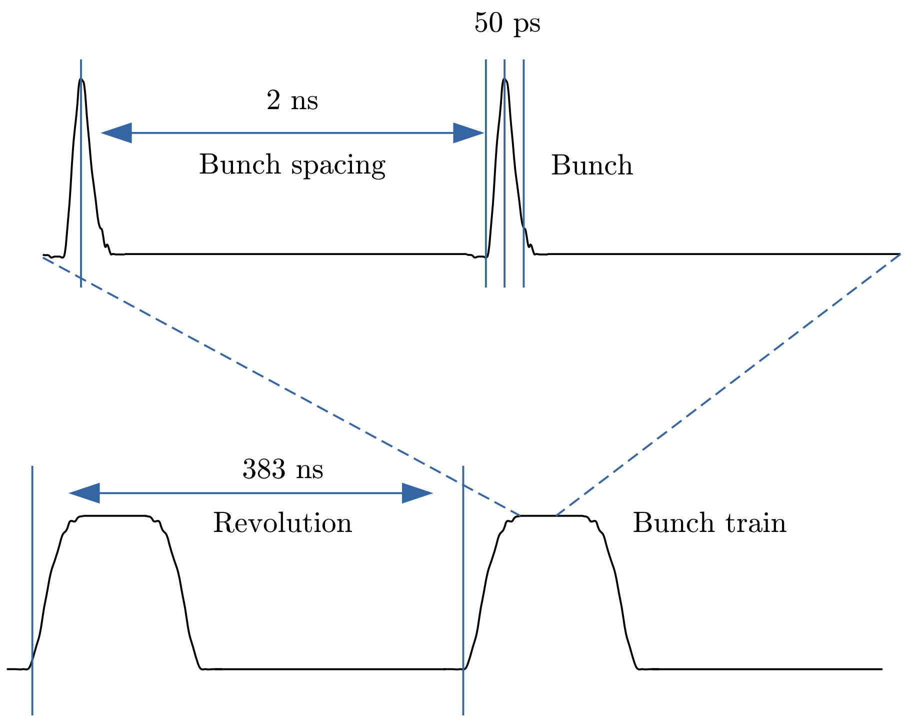

Figure 2.8.: Schematic representation of the signal produced by a synchrotron, highlighting the different timescales present. The pulse from a single bunch can have widths in the order of a few tens of picoseconds. Two consecutive bunches are separated by 2 ns, corresponding to the employed RF frequency of 500 MHz. The bunch filling pattern repeats at every revolution.

2.1.6. The KAPTURE system

The signals produced by the circulating electrons in a synchrotron, such as the ones from synchrotron radiation or a beam position monitor (BPM), have a peculiar time structure (figure 2.8). For example, an ideal generic signal from KARA would exhibit the following characteristics. At long timescales, in the order of a few tens to a few hundred of microseconds, the synchrotron oscillations will be apparent. At smaller timescales, in the order of the revolution time ($\approx$ 368 ns), the charge distribution of the bunches in the machine, also known as *filling pattern*, will repeat at the revolution frequency. Furthermore, bunches will pass in front of a fixed observer at the main RF frequency, leading to a repetition with a $\approx$ 2 ns period. The length of these bunches varies with operation mode, but is usually around a few tens of picoseconds. The response time of the detector used to produce the signal has a strong influence on the length and shape of the produced pulses.

In order to capture these fast peaked signals properly, one would need to sample the signal at very high rates in the order of hundreds of GS/s. Albeit technically possible, sampling at such high rates produces a challenging amount of data that needs to be saved and processed. This challenge is further compounded by the need to monitor the dynamics of these signals over several seconds, which significantly increases the amount of data generated and complicates both storage and analysis. This is unnecessary, though, given the pulse repeats at a much smaller rate compared to its width, most of the acquired signal will be empty.

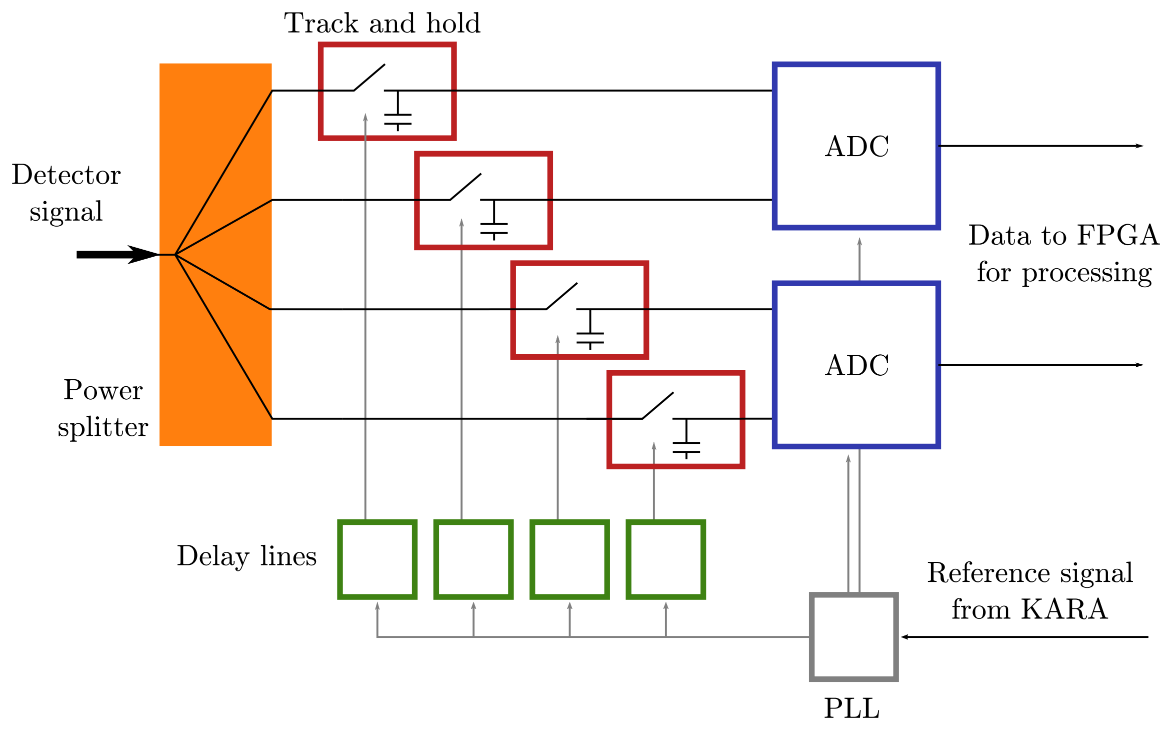

Figure 2.9.: Schematic of the working principle of the KAPTURE system. The signal to be acquired is split in four identical copies that are then sampled with a track and hold. The sampling time is finely tunable through delay lines. A PLL produces the reference clocks for the delay lines and the ADCs. The ADCs finally digitize the signal that is then processed by the FPGA.

Based on this observation, the Karlsruhe pulse taking ultra-fast readout electronics (KAPTURE) system [20] was designed and developed. As shown in figure 2.9, the signal to be sampled is split into identical copies, either by means of an active or passive power splitter. Each of these copies is then sampled by a high-bandwidth track-and-hold circuit. These devices have two modes, selected by an external signal: track and hold. In track mode, the input signal is repeated to the output, and is then held constant at the arrival of the hold signal. The hold signal arrival time is individually delayed for each channel by a finely controllable amount of 3 ps. In this way the sampling points can be distributed along the pulse. The held signal can then be digitized by a much slower, and less expensive, analog-to-digital converter (ADC). In a KAPTURE board four channels are digitized by two ADCs, each with two channels.

The delays lines and ADCs are all controlled by a field programmable gate array (FPGA). This device, that will be more thoroughly introduced in section 2.3, is fundamentally a lattice of logic gates interconnected by programmable switches. This allows the development of custom digital electronics. One HighFlex board can control two KAPTURE boards, for a data rate of 6.5 GB/s. This is usually transferred via a peripheral component interconnect express (PCIe) interface to the random access memory (RAM) of a host computer, where it can be later saved to persistent storage. It is worth noticing how with this system it is possible to sample the signal of a synchrotron with bunch-by-bunch and turn-by-turn repetition rate for several seconds.

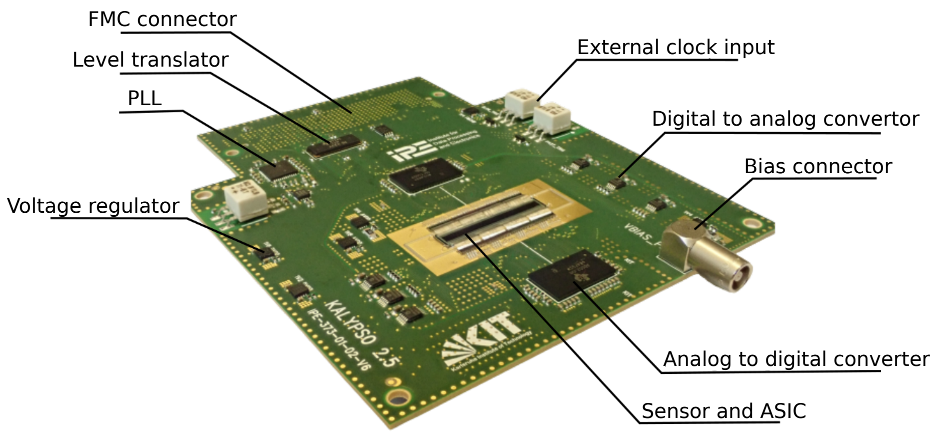

Figure 2.10.: Front-end PCB of a KALYPSO system, highlighting the main components. The external clock input and PLL allow to synchronize the sampling to the accelerator timing system. The sensor and ASIC are connected to ADCs that are then controlled by an FPGA that is connected to the FMC connector. Courtesy of M. M. Patil.

2.1.7. The KALYPSO system

Several beam diagnostic instrumentation used at KARA and other synchrotron light sources encode information into light. The KAPTURE system described in the previous section allows monitoring the amplitude and shape of pulsed light signal, provided they are converted into a suitable electrical signal first. Oftentimes, the spatial dependence of this amplitude provides a great deal detail. This definition is fundamentally describing a camera. In order to produce turn-by-turn information, the frame rate would need to be in the MHz range. Due to the absence of systems capable of achieving this rates for several seconds to hours, the Karlsruhe linear array detector for MHz repetition rate spectroscopy (KALYPSO) system was developed at KIT [21–23].

A KALYPSO system is composed of a front-end printed circuit board (PCB), shown in figure 2.10, mounting a sensor, an analog signal processing ASIC, and an ADC, and a HighFlex based back-end card. The latter uses the same principles described for the KAPTURE system. Several sensors can be employed, depending on the wavelengths of interest. Silicon microstrip sensors can be employed up to 1050 nm, while indium-gallium-arsenide sensors can be used in the range from 1050 nm to 1.7 μm. Each ASIC [24] has 128 input channels connected to the detector. Each channel has a charge-sensitive amplifier and shaping stage. A buffer holds the analog value until it is multiplexed to one of the sixteen output channels, where it is digitized by the ADC. The digital data is then transferred by an FPGA to the random access memory (RAM) of a computer, where it can be stored, visualized, and analyzed.

The KALYPSO system can be used to directly acquire the synchrotron light signal produced by an accelerator such as KARA. Furthermore, electro-optical (EO) systems can be employed to encode the temporal evolution of electric field strength into the polarization of the light signal [25]. More details on how this scheme can be employed for beam diagnostic are discussed in section 6.1.1.

2.1.8. Summary

A brief overview of the synchrotron and betatron dynamics was carried out, describing the fundamental phenomena that can be observed in an accelerator such as KARA. The synchrotron radiation emission characteristic of these machines was discussed, together with its impact on the beam stability. The betatron oscillations were shown to have the underlying dynamics of a harmonic oscillator. Its control could thus represent an interesting tool for testing novel control algorithms.

A more complex dynamics, the microbunching instability, appearing in special short bunch operation modes at synchrotron light source was described. Its usage for the production of strong terahertz radiation bursts makes it a useful tool for other scientific fields. Compared to the betatron oscillations, this dynamics exhibits strong non-linearity and threshold effects giving rise to abrupt changes in dynamics, leading to an extremely challenging problem for a controller.

Finally, the KARA synchrotron light source at KIT was described. This machine offers a wide variety of operation modes, allowing to experiment with the control of both the betatron oscillations and of the microbunching instability. Additionally, the cutting edge beam diagnostic infrastructure such as KAPTURE and KALYPSO offer powerful information that can be employed as the input of a controller.

2.2. Reinforcement learning

In this chapter the mathematical basis necessary for understanding both machine learning and reinforcement learning will be introduced. Special attention is given to neural networks, both to their inference and training. A short review of the modern algorithms and the underlying ideas used to perform efficient reinforcement learning training will follow, together with a discussion of the current state-of-the-art machine learning and reinforcement learning frameworks and a few of their applications to large-scale facilities.

2.2.1. Brief introduction to machine learning

Machine learning (ML) is comprised by a set of algorithms capable of learning from a dataset provided during a *training* phase, and then extrapolate this behavior to cases not covered in the training data. The following introduction on ML is based on reference [26]. A common use case for these algorithms are supervised categorization tasks. The problem can be stated as follows. A set of features $f \in F$, where the feature space F is usually $\mathbb{R}^n$, is the set of all possible objects. A label set L, is the set of possible labels that are applied to an element in F. A training set $T \subset F \times L$, is a labelled categorization example. Usually one can assume the correct set of labels are produced by a "correct" predictor $h^\star : F \longrightarrow L$. The goal of the algorithm is to learn a *predictor* $h : F \longrightarrow L$ that is capable of labeling elements. In order to measure how well a given predictor behaves, it is useful to define a *loss function*, that in the case of binary classification, i.e. $L = \{-1, 1\}$ is

$$\ell_F(h) = \underset{f \sim p(F)}{\mathbb{P}} \left[h(f) \neq h^\star(f) \right]. \tag{2.20}$$

Here $f \sim p(F)$ denotes that the samples f are sampled with a probability density function p over the set F. This expression denotes the probability that the tested predictor h is wrong on a given sample. It is worth noticing how such a figure of merit depends on the chosen probability distribution.

Another common use-case of ML is function approximation, i.e. regression. Specifically, the goal of the training algorithm could be to output a function $h : F \to \mathbb{R}$ approximating an unknown target function $h^\star : F \to \mathbb{R}$. Again, a training set $T \in F \times \mathbb{R}$ is provided. Several possible loss functions exist, for example

$$\ell_F(h) = \underset{f \sim p(F)}{\mathbb{E}} \left(h(f) - h^\star(f) \right)^2, \tag{2.21}$$

$$\ell_F(h) = \underset{f \sim p(F)}{\mathbb{E}} \left| h(f) - h^\star(f) \right|, \tag{2.22}$$

respectively called the mean-square error and mean absolute Z-score. The choice of function dictates how errors are weighted. For instance, equation (2.21) tends to penalize large errors compared to equation (2.22). The choice of loss function has a strong effect on the algorithm. Specific, problem-based tuning is sometimes necessary in order for the ML algorithms to work properly.

2.2.2. Neural networks

In ML, a common choice of predictor or of approximation function are neural networks (NNs). An artificial NN is constituted by a set of interconnected neurons, usually implemented with a *perceptron* (figure 2.11), defined as follows

$$y = \theta(\vec{w} \cdot \vec{x} + b). \tag{2.23}$$

Here $\vec{x} \in \mathbb{R}^n$ are the input features, while $\vec{w}$ and b are respectively called the weights and bias of the perceptron, and θ is a non-linear activation function. Common activation functions include the rectifying linear unit (ReLU), the sigmoid function σ and the hyperbolic tangent (tanh):

$$\mathrm{ReLU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}, \tag{2.24}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2.25}$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}. \tag{2.26}$$

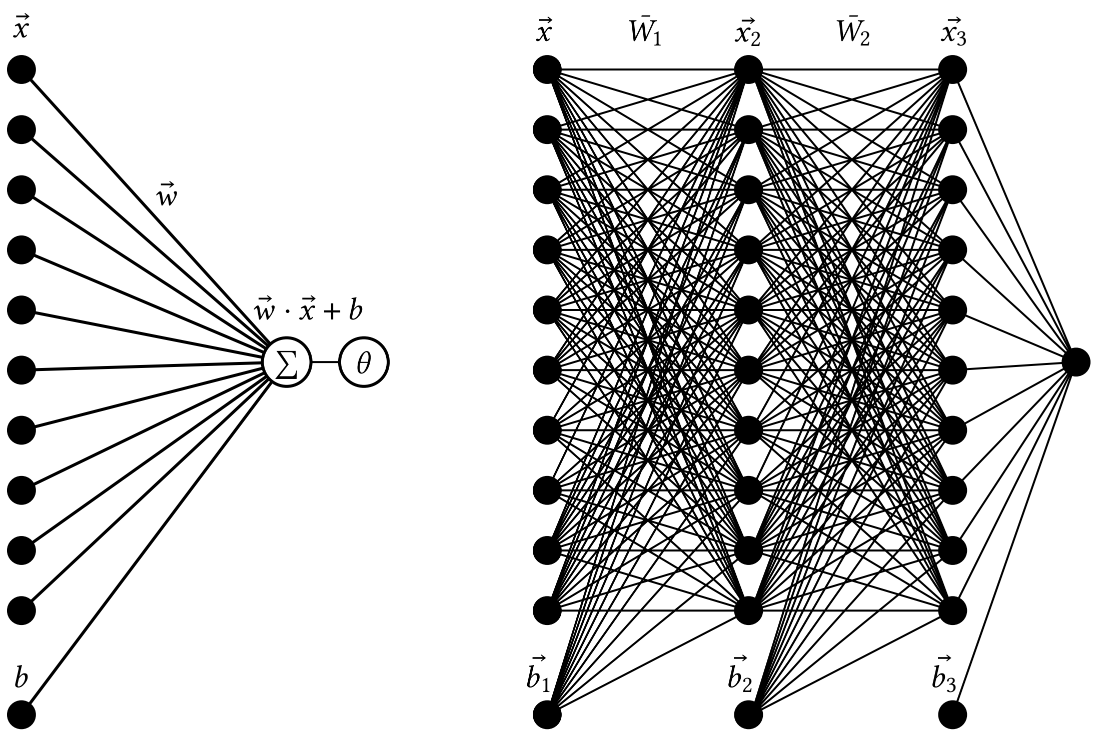

Figure 2.11.: On the left, drawing of a perceptron showing the sum used to perform the dot product, the activation function, and the weights and biases. On the right, a multilayer perceptron with two hidden layers and a single output neuron. The sum and activation functions have been coalesced into the neuron node for ease of representation.

Usually several perceptrons are grouped into a *layer*. The number of neurons in a layer is called its *width*. The perceptron definition of equation (2.23) can be expanded to a layer function $h : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ by promoting the weights and bias respectively to a matrix and a vector,

$$\vec{y} = h(\vec{x}) = \theta(\bar{W}\vec{x} + \vec{b}), \tag{2.27}$$

where θ is now performed element-wise, $\bar{W} \in \mathbb{R}^{m\times n}$ is the weight matrix and $\vec{b} \in \mathbb{R}^m$ is the bias vector, as shown in figure 2.11.

Several layers are composed together to form a NN known as multi-layer perceptron (MLP). Denoting the input layer $\vec{x}$, the output layer $\vec{y}$ is computed by chaining l hidden layers $\{h_i\}_{i=1...l}$, each one denoted by a specific set $\left\{\bar{W}_i, \vec{b}_i, \theta_i\right\}_{i=1...l}$. In this way, the MLP is

$$\vec{y} = H(\vec{x}) = h_l(h_{l-1}(...h_2(h_1(\vec{x}))...)) = (h_l \circ h_{l-1} \circ ... \circ h_2 \circ h_1)(\vec{x}) \tag{2.28}$$

In order to train a NN, one would need to find a set of weight and biases that minimize the loss function. If the L^2-loss of equation (2.21) is taken as an example, an estimator can be obtained by the substituting the expectation value with an average over the training set $T = \{\vec{x}_i, y_i\}_{i=1...N_t}$, thus giving

$$\ell_T(H) = \frac{1}{N_t}\sum_{i=1}^{N_t} \left(H(\vec{x}_i) - y_i\right)^2 . \tag{2.29}$$

If one considers that H is defined by the combination of all its weights and biases $\tilde{W}$, one could consider the loss as

$$\ell_T(\tilde{W}) \overset{\text{def}}{=} \ell_T(H_{\tilde{W}}). \tag{2.30}$$

One could attempt minimizing this function via gradient descent methods. The idea behind this class of methods is analogous to trying to reach the top of a mountain by following the increase of the slope. This approach is in some cases guaranteed to reach the global minimum, but strong requirements are necessary, for example the convexity of $\ell_T(\tilde{W})$. This kind of strong constraints are not generally true, nonetheless a local minima is usually reached. A basic gradient descent method iteratively computes the gradient of the loss, and updates the weights accordingly according to a rule as

$$\tilde{W} \leftarrow \tilde{W} - \eta \nabla \ell_T(\tilde{W}), \tag{2.31}$$

where η is a training parameter known as the *learning rate.*

In practice, directly minimizing the loss function is computationally intensive. This issue is usually solved by means of stochastic or mini-batch gradient descent methods, where the loss is computed respectively on single sample of the training set or on a mini-*batch* of samples $B = \{\vec{x}_i, y_i\}_{i=1...N_b}$, where N_b is called the batch size and $\{\vec{x}_i, y_i\} \in T$. This has the additional benefit of leading to a noisier update that can make the optimizer “jump-out” of local minimum, in this way increasing the chance of reaching the global minimum.

If one tries to calculate the gradient of the loss, a fundamental for gradient descent, leads to

$$\nabla_W l_B(\tilde{W}) = \frac{1}{N_b}\sum_{i=1}^{N_b} \nabla_W \left(H_{\tilde{W}}(\vec{x}_i) - y_i\right)^2 = \frac{1}{N_b}\sum_{i=1}^{N_b} 2\left(H_{\tilde{W}}(\vec{x}_i) - y_i\right) \nabla_W H_{\tilde{W}}(\vec{x}_i). \tag{2.32}$$

Efficiently computing the gradient of the NN, $\nabla_W H_{\tilde{W}}(\vec{x}_i)$, is thus fundamental in order to effectively perform training.

The chain rule for the jacobian matrix $\nabla_{\vec{x}}(g \circ f)(\vec{x})$ with $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ and $g : \mathbb{R}^m \longrightarrow \mathbb{R}^k$ is

$$\nabla_{\vec{x}}(g \circ f)(\vec{x}) = \nabla_f(g(f))\nabla_{\vec{x}}(f(\vec{x})) \tag{2.33}$$

In the case of MLP, the gradient has a peculiar property. Specifically, thanks to the chain rule of differentiation leads to

$$\nabla_{W_i}(h_l \circ h_{l-1} \circ ... \circ h_2 \circ h_1) = (\nabla_{\vec{x}_l} h_l(\vec{x}_l))\nabla_{W_i}(h_{l-1} \circ ... \circ h_2 \circ h_1). \tag{2.34}$$

This expression can be chained iteratively until the i-th layer is reached

$$\nabla_{W_i}(h_l \circ h_{l-1} \circ ... \circ h_2 \circ h_1) = (\nabla_{\vec{x}_l} h_l(\vec{x}_l))(\nabla_{\vec{x}_{l-1}} h_{l-1}(\vec{x}_{l-1}))...\nabla_{W_i} h_i(\vec{x}_i). \tag{2.35}$$

At this point, using equation (2.27), one can directly calculate the jacobian matrices as being

$$\nabla_{x_i} h_i(\vec{x}_i) = (\nabla_{\vec{m}}\theta(\vec{m})) \cdot W_i \tag{2.36}$$

$$\nabla_{W_i} h_i(\vec{x}_i) = (\nabla_{\vec{m}} \theta(\vec{m})) \cdot \vec{x}_i \tag{2.37}$$

One important property of equation (2.35) is how, except in the i-th jacobian, all previous derivatives are the same independently of the layer i we are differentiating. As such, based on equations (2.35) to (2.37) a procedure, known as *backpropagation* can be devised, were the gradient with respect to the weights and biases is computed iteratively from the output layer to the input.

Modern ML frameworks as PyTorch [27], Tensorflow [28], Keras [29] and Caffe [30] make heavy use of this technique to efficiently compute the gradients. Specifically, a key concept of these tools is *automatic differentiation* [31, 32], where the computation graph to obtain a given value is used to reconstruct the gradients in a way that is totally transparent to the user.

The MLP described in this section is widely employed and works well for some problems, but it is fundamentally not aware of the underlying structure of the input data. In the last decades several more advanced NN topologies have been developed, specifically to address this drawback. For instance, if the input to the network is an image, the position of the pixels and the scale is an important information that needs to be preserved. This is addressed with convolutional neural networks (CNNs) (figure 2.12), where a smaller convolution kernel is scanned over the image, creating a new representation called a feature map. This dramatically reduces the number of parameters required for the operation: if a 256×256 pixel image is provided, a simple MLP will require 65356 input neurons to process it. A CNN, on the other hand, can use a first convolutional layer with size of 8×8, for example, that would lead to a much smaller number of input neurons and thus of parameters. In addition, pooling layers are usually applied, that are capable of further reducing the intermediate representation combining clusters in the feature map. This method has the advantage of being able to perceive local features, thanks to the convolutional layers, while it is still capable of having a comprehensive large-scale perspective thanks to the summarizing effect of the pooling layers.

An additional kind of structured data is time-series data, for which recurren neural networks (RNNs) can be employed. This kind of networks can encode an history by storing an internal state that is updated during inference. Common implementations are the long-short term memory (LSTM) (figure 2.12) and gated recurrent unit (GRU).

2.2.3. Introduction to reinforcement learning

This introduction is based on [33, 34].

Reinforcement learning (RL) is a subset of ML algorithms that treat the learning of an agent from the interaction with an external environment (figure 2.13). Specifically, the agent receives a set of observation from the environment and tries to choose the best action to increase a cumulative reward signal.

In more mathematical terms, it is possible to model an environment as a Markov decision process (MDP), a tuple comprised of (S, A, P, R, γ), where S is the set of states, A is the set of

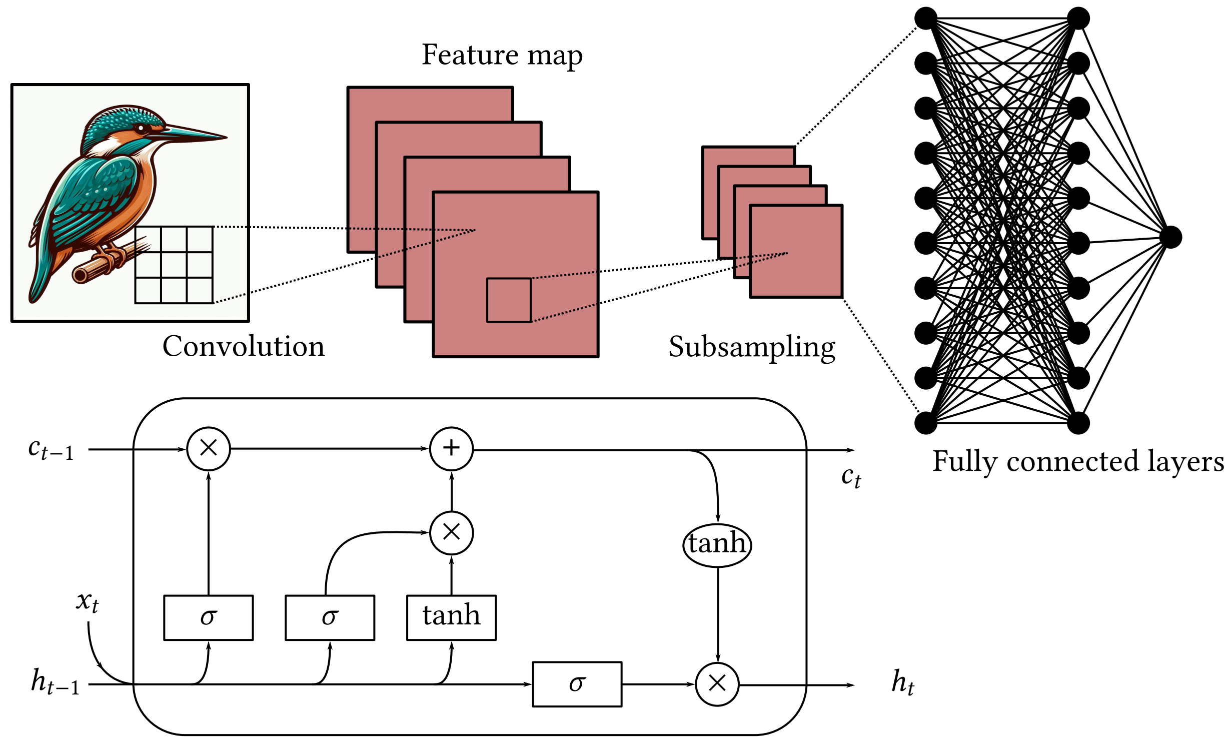

Figure 2.12.: A schematic representation of a CNN is shown on the top part of this figure. An LSTM is represented in the bottom part. Here x_t are the inputs, h_t and c_t are respectively the hidden and cell state vectors. In this notation, the rounded bubbles represent element-wise operations, while the square bubbles represent linear layers followed by the depicted activation function.

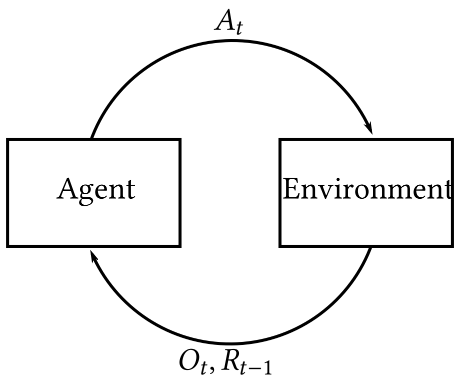

Figure 2.13.: Interaction of an agent with the environment. The agent receives an observation and provides an action to the environment. Together with the observation, the reward of the last step is provided.

all actions. P denotes the transition probability from a state s to a state s', provided the action a was taken in state s

$$P^a_{s,s'} = \mathbb{P}\left[S_{t+1} = s' | S_t = s, A_t = a\right]. \tag{2.38}$$

In the special case where the state space is discrete, P can be described as a transition matrix. The reward function R^a_s is defined as the expected reward obtained in the next time step given the agent is in step s and took action a, i.e.

$$R^a_s = \mathbb{E}\left[R_{t+1} | S_t = s, A_t = a\right]. \tag{2.39}$$

In order to consider long-term goals, the *cumulative reward* is defined as

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{i=0}^{\infty} \gamma^i R_{t+1+i}, \tag{2.40}$$

where $\gamma \in [0, 1]$ is the discount factor. This parameter is necessary to limit the divergence of the cumulative reward by reducing the weight of future rewards. Two extreme cases are possible. When $\gamma = 0$, the only reward considered is the one of the next step, i.e. the cumulative reward is evaluated in a short-sighted way. On the other hand, when $\gamma = 1$, the evaluation can watch arbitrarily in the future.

An important concept, that is embedded into equation (2.38), is the so-called *Markov property*. Specifically, the assumption is that the current state and action, s and a, contain all necessary information to define the transition probability into the next state. In this sense the history of the previous states will have no effect:

$$\mathbb{P}\left[S_{t+1} = s' | S_t, A_t, S_{t-1}, A_{t-1}, S_{t-2}, A_{t-2}, ...\right] = \mathbb{P}\left[S_{t+1} = s' | S_t, A_t\right]. \tag{2.41}$$

This is of course a very strong assumption. In real world environments it is uncommon to know the state of the system to such high-degree that we are capable of predicting transition probabilities. To circumvent this issue the partially observable Markov decision process (POMDP) is defined as the MDP with the addition of an observation state O and an observation function Z, i.e. $(S, A, P, R, \gamma, O, Z)$, where

$$Z^a_{s',o} = \mathbb{P}\left[O_{t+1} = o | S_{t+1} = s', A_t = a\right]. \tag{2.42}$$

Under this definition, one obtains observations that give some information of the underlying state of the system, without possessing full knowledge of the environment. For the sake of simplicity, in the following we only discuss the fully observable case $O \equiv S$.

An agent learns a policy π defining the probability distribution over actions in a given state (or observation)

$$\pi(a|s) = \mathbb{P}\left[A_t = a | S_t = s\right]. \tag{2.43}$$

In order to evaluate the performance of an agent it is useful to define the value function $v_\pi(s)$ as the expected cumulative reward if the agent is in state s and follows policy π,

$$v_\pi(s) = \mathbb{E}\left[G_t | S_t = s\right]. \tag{2.44}$$

Similarly, the action-value function $q_\pi(s, a)$, is the expected cumulative reward in case the agent picks state action a in state s, and then follows policy π,

$$q_\pi(s, a) = \mathbb{E}\left[G_t | S_t = s, A_t = a\right]. \tag{2.45}$$

One important property of these functions is the Bellman expectation equation:

$$v_\pi(s) = \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s\right], \tag{2.46}$$

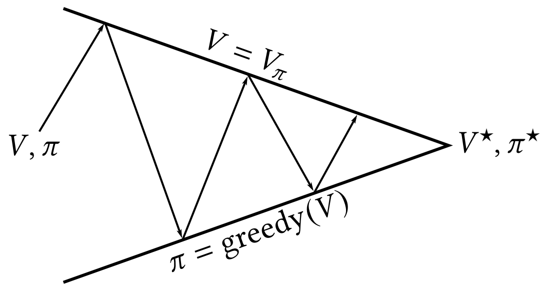

Figure 2.14.: Diagram showing the two stages of RL with dynamic programming: policy evaluation (top) and improvement (bottom). After several of these iterative steps the current policy and value function will be closer to the optimum.

$$q_\pi(s,a) = \mathbb{E}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a\right]. \tag{2.47}$$

A remarkable property of this expression is how it connects the reward function, something that can be evaluated by interacting with the environment, to the value functions, that are fundamental to plan long-term strategies. As such, this provides methods to evaluate value functions analytically, numerically and most importantly experimentally.

An early attempt at RL employed dynamic programming methods, which typically consist of two stages and assume complete knowledge of the MDP (figure 2.14). In the first stage, known as policy evaluation, the value functions are computed based on the transition probabilities P. The second stage, policy improvement, involves *greedily* improving the policy as

$$\pi_{\text{greedy}}(s) = \arg\max_{a \in A} Q(s,a). \tag{2.48}$$

This method can be proven to achieve the optimal policy $\pi^\star$ and value function $V^\star$. However, the requirement of full knowledge of the MDP significantly limits the applicability of these algorithms, as state transition probabilities are often unknown in general environments.

A following attempt at RL was with Monte Carlo methods in discrete state spaces. In this case the value function can be represented by a table, assigning to each state the corresponding expected return. Given a sequence of state-action pairs $(S_1, A_1), ..., (S_N, A_N)$, the value function can be empirically estimated by computing the expectation value of equation (2.45) as an average every time the agent visits a given state and takes a given action, likewise for the value function. In order to implement this method, one would need to count the number of times a state-action pair s, a is encountered $N(s,a)$ and the sum of the cumulative rewards after each visit $S(s,a)$. The action-value function can then be computed as

$$Q(s,a) = \frac{S(s,a)}{N(s,a)}. \tag{2.49}$$

This expression is impractical because it requires to store and update a great number of counters. It is possible to refine this method to update online, at the end of every episode. When the state-action pair (s,a) is visited, the action-value function is updated as follows

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[G_t - Q(s,a)\right], \tag{2.50}$$

where α is a parameter regulating the size of the update, usually in the range $[0, 1]$.

This estimate of the action-value function can be used to obtain a *greedy* policy as described in equation (2.48), a policy that at every step is trying to optimize the cumulative reward. It is worth noting that this kind of policy will have the tendency to stick to non-optimal behaviors, as it is trying to exploit the knowledge it already has. Dave Akin gives the following quote "You can't get to the moon by climbing successively taller trees."[3] This quote was originally applied to rocket development, but it also applies to RL agents: by exploiting a specific strategy, the agent might be missing a more refined technique that could potentially be discovered by additional exploration. This trade-off between exploration and exploitation is at the heart of RL.

One issue of Monte Carlo methods is they usually require a full sequence of action-state pairs before being able to perform an update. This issue is overcome by the so-called time difference (TD) methods. Here, instead of waiting for the rest of the state-action sequence to compute the cumulative reward, the return is *bootstrapped* by using an estimate of the value function. Specifically, the update rule now is

$$\begin{cases} Q(s,a) & \leftarrow Q(s,a) + \alpha \left[G_t - Q(s,a)\right] \\ & \leftarrow Q(s,a) + \alpha \left[(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})) - Q(s,a)\right] \end{cases} . \tag{2.51}$$

Here the quantity δ_t is conventionally known as the TD-error,

$$\delta_t \overset{\text{def}}{=} R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(s,a) \tag{2.52}$$

This update rule is usually called Sarsa, given the tuple it needs to be performed: the current state-action pairs, the reward give in the next step, and the next state-action pairs.

2.2.4. Modern reinforcement learning

The concepts described in the previous section can be extended in order to obtain more generally applicable RL algorithms. The first key idea concerns the representation of the value or action-value functions. Initial applications used discrete spaces that allow the usage of tables. This approach, though, is hardly applicable to the more interesting cases of continuous observation and action spaces [33].

A powerful technique, known as deep-RL, uses the capabilities of NN as general function approximators in order to learn a value function. This can be achieved by using a loss function that, for example, can be an mean-square error as in equation (2.29). One error that could be used is the TD-error described in the previous section.

[3] From `https://spacecraft.ssl.umd.edu/old_site/academics/akins_laws.html`

One of the first such algorithms was deep Q-learning (DQN) [5]. The main idea is to modify the update rule in equation (2.52) as

$$\delta_t^\star \overset{\text{def}}{=} R_{t+1} + \gamma \max_{a\in A} Q(S_{t+1}, a) - Q(s, a). \tag{2.53}$$

The addition of the maximum has the effect of learning the action-value function of the optimal policy, i.e. the policy that achieves the maximum expected cumulative reward. The policy can be obtained by by using the greedy policy defined in equation (2.48). Notably, this algorithm was employed to train a CNN to play Atari video games by processing the screen output.

An additional fundamental concept are policy gradients. Similarly to the action-value function, also the policy can be directly parameterized as a probability distribution depending on a set of parameters θ

$$\pi_\theta(s, a) = \mathbb{P}\left[a \mid s, \theta\right]. \tag{2.54}$$

This completely general definition allows the use of a NN as policy, the gradient of which can be calculated based on the policy gradient theorem [33, 34]

$$\nabla_\theta L_\pi = \mathop{\mathbb{E}}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a)\; Q(s, a)\right]. \tag{2.55}$$

Several algorithms have thus two NNs, one for value-function estimation and one for the policy. This class of algorithms are called actor-critic. The agent has a policy, the actor, and uses the insights provided by the value-function, the critic, to maximize the long term reward.

2.2.4.1. Deep deterministic policy gradient

DQN, albeit a fundamental step in the development of RL algorithms, had the clear shortcoming of not being able to treat continuous or high-dimensionality action spaces [35]. The deep deterministic policy gradient (DDPG) algorithm [35, 36] aims to solve this issue.

The learning of action-value function $Q_\phi(s, a)$ is performed by creating a loss function following the DQN loss (2.53) that at the same time enforces Bellman's equation (2.47) through a target value y:

$$\begin{aligned} L_\phi &= \mathop{\mathbb{E}}_{(s,a,r,s')\sim D}\left(Q_\phi(s, a) - y\right)^2 \\ y &= r + \gamma Q_\phi(s', \pi_\theta(s')) \end{aligned}. \tag{2.56}$$

Here D is the training dataset, composed of (s, a, r, s') tuples describing the transitions of the system.

DDPG is an *off-policy* algorithm, meaning it is capable of learning from experience that was gathered by a different policy than the current one. As such, a *replay buffer* of previous (s, a, r, s') is kept and a mini-batch for training is extracted for each stochastic gradient descent.

The actor tries to learn the greedy policy defined in equation (2.48). The maximum is computed by gradient ascent of

$$\mathop{\mathbb{E}}_{s\sim D} Q(s, \pi_\theta(s)). \tag{2.57}$$

In order to drive exploration, stochastic noise is added to the action. Several probability distributions are possible, as an example one could use a normal distribution, albeit a very common choice is the Ornstein-Uhlenbeck process [37]. One of the main advantages of this choice is that consecutive samples are correlated, leading to gradual changes between consecutive actions that could be desirable depending on the current environment.

The described update approach suffers from computational instability given the fact Q is used twice in equation (2.56): one to evaluate the error, and the other to give an estimate of the target value. Such an issue is addressed by having a copy of the actor and critic, called *target networks*, that update more slowly as

$$\begin{aligned} \phi_{\text{target}} &\leftarrow \tau\phi + (1-\tau)\phi_{\text{target}} \\ \theta_{\text{target}} &\leftarrow \tau\theta + (1-\tau)\theta_{\text{target}} \end{aligned}, \tag{2.58}$$

where $\tau \ll 1$ and ϕ_{target} and θ_{target} are the coefficient of the target networks. In this way, the target networks can be use to compute the target value equation (2.56).

2.2.4.2. Twin delayed DDPG

Depending on the problem at hand, DDPG can achieve high-performance, but it is frequently unstable with respect to the choice of its hyperparameters [38]. To overcome this the twin delayed DDPG (TD3) algorithm [39] is frequently employed.

Compared to DDPG, this algorithm adds two main ideas: double Q-learning and target policy smoothing. Specifically, two action-value functions, $\{Q_{i,\phi}\}_{i=1,2}$, are used, each one learning from the same target

$$y = r + \gamma \min_{i=1,2} Q_{i,\phi_{\text{target}}}(s', a'). \tag{2.59}$$

This technique reduces the chance of overestimation, by choosing the lower one of the two functions.

Moreover, the action used to compute the target y of equation (2.59), uses a smoothed, more stochastic version of the target policy, that is clipped to maintain it in the desired action domain. This is necessary because, if a peak forms in the action-value function, this will be exploited during the training of the policy, leading to inferior performance.

2.2.4.3. Proximal policy optimization

In the previous section, DDPG was described as an *off-policy* algorithm. *On-policy* algorithms, on the other hand, can only learn from data gathered with the current policy. An example of this class of algorithms is proximal policy optimization (PPO) [40], that was derived from trust-region policy optimization (TRPO) [41, 42]. A useful definition in this case is the one of *advantage function* defined as

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s). \tag{2.60}$$

This kind of function is useful for driving the training of policy gradient algorithms as it measures how a given action a taken in the current state s is beneficial compared to following policy π. In PPO, based on the documentation found in [38], the update rule is taken as follows[4]

$$L(s,a,\theta,\theta_k) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A_{\pi_{\theta_k}}(s,a), \operatorname{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon\right)A_{\pi_{\theta_k}}(s,a))\right). \tag{2.61}$$

In the formula above, θ and θ_k respectively represent the previous policy parameters and the new ones. This equation seems complex, but the idea is fairly simple: the clip function limits the size of the update steps. In this way, the difference between the current and new policy is limited, improving the stability of the algorithm by avoiding sudden and often detrimental changes to the policy.

The critic (or value function network), is updated based on the loss function

$$L_\phi = \sum_{t=0}^{T}\left(V_\phi(s_t) - \hat{R}_t\right)^2, \tag{2.62}$$

where $\hat{R}_t$ is the reward-to-go, i.e. the sum of the rewards from the current time step t up to the end of the episode.

PPO is currently a widespread algorithm thanks to its good convergence behavior and its stability with respect to variation of its hyper-parameters. Being an on-policy algorithm, though, leads to it being generally less *sample efficient*, meaning it usually requires more interactions with an environment in order to obtain a fully trained agent. Thus it represents a good choice for environments that are inexpensive to train on or that are fast to simulate.

2.2.4.4. Soft actor-critic

A different approach to balance the exploration-exploitation in RL is *entropy regularization.* This approach uses the definition of entropy for probability distributions

$$H(p) = \mathop{\mathbb{E}}_{x\sim p}\left[-\log p(x)\right], \tag{2.63}$$

than in layman terms can be considered a way to quantify the “randomness” of a given variable. In this case, the value function is modified to include a bonus proportional to the entropy of the policy, i.e.

$$V_\pi(s) = \mathop{\mathbb{E}}_{\pi}\left[\sum_{k=0}^{\infty}\left(R_{t+k+1} + \alpha H(\pi(\cdot|s_t))\right)\right], \tag{2.64}$$

[4] There are actually two implementation of PPO, PPO-clip and PPO-penalty. In this work PPO-clip is described, as it is more easily understandable.

where α is a parameter setting the trade-off between the entropy and the value function part. The action-value function is defined in a similar way.

An example of an algorithm using this training method is soft actor-critic (SAC) [43]. It includes several ideas shown in DDPG, like target networks and a replay buffer, while adding a double Q function as shown in TD3. The target values are updated to include the entropy part as in equation (2.64).

SAC is now widely adopted for its higher sample efficiency compared to on-policy algorithms like PPO, though it requires more computation time and careful hyperparameter tuning. This creates a trade-off between computation time and sample efficiency, a crucial consideration when applying RL techniques to real-world problems.

2.2.5. The current library landscape

A fundamental factor leading to the widespread adoption of ML techniques is the availability of high-quality user-friendly libraries that allow the deployment of standard algorithms in a handful of lines of code. In this section, a brief overview of these frameworks will performed with the focus of highlighting the most important features for this work. A similar effort has started for RL, originally intended to provide a baseline algorithms for reproducibility purposes, now proves as an invaluable resource in the adoption of the complex RL techniques in the real world.

2.2.5.1. Neural network libraries

Most of the key players in ML, both companies and research institutions, have developed a series of libraries to ease the implementation and training of NNs. As discussed in section 2.2.2, a non-comprehensive list of these frameworks is: PyTorch [27] developed by Facebook, Tensorflow [28] developed by Google, Keras [29], independent and now serving as frontend for other libraries[5], Caffe [30] developed by Berkley AI Research and JAX [44] developed by Google.

Most of these libraries are either native Python libraries, or offer a Python wrapper. For example, PyTorch is a wrapper of the C++ Torch library. One of the main reasons leading to this design choice is the expressiveness of the language, paired with its wide variety of libraries, allowing a seamless extension of functionality. Python is an interpreted language, and is thus penalized in some cases due to the performance hit of interpreting the source code. This issue is solved by its ability to include C or C++ compiled source code and integrate it as a library. In this way, the performance critical NN inference and training code can be developed in low-level languages with a high degree of hardware-aware optimizations, circumventing the performance hit due to the interpreter.

A key functionality of these libraries is GPU integration. GPUs of which a more comprehensive discussion will be performed in section 2.3.1, are devices that allow performing highly

[5] Later Keras was merged with Tensorflow. Currently PyTorch, Tensorflow and JAX backends are available.

parallelized arithmetic computations. Their debut in the ML field started with AlexNet [1], where the training of a large CNN was performed thanks to the high speed computational capability of these devices.

Both PyTorch and JAX are currently offering just-in-time (JIT) compilation capability. This means that on the first call of a properly annotated function, this is converted into native implementation that is compiled, allowing a considerable speed-up compared to the interpreted version. In both libraries it is possible to JIT compile the backpropagation.

The libraries described in this subsection serve as the basis for a variety of other libraries, implementing, for example, RL algorithms. Additionally, their high-speed and easy to use implementation of a matrix-matrix multiplication is employed in fast simulations, that can be employed for training RL agents [45].

2.2.5.2. Reinforcement learning libraries

The algorithms that have been described in the previous section are complex. Their actual implementation is also nuanced and requires the careful attention of an expert versed in the algorithm.

In reference [46], the history of one of the standard implementations of the PPO algorithm is discussed. One striking take-aways of the article is how the reproduction of RL results is non-trivial, as the algorithm does not fully capture the complexity of how the training is performed. For example, there are several ways one could implement a stochastic policy for PPO. Specifically, the actor network is completely separate from the critic, and Gaussian noise is added to the output of the network. During implementation, one could have opted for the variance to be output by the neural network, i.e. to be state-dependent, as it is in fact often used for SAC. On the other hand, a state-independent variance was chosen. This choice did not lead to a particular performance variation, but other design choices, for example the way advantages are normalized, can have severe impact on the final effectiveness.

For this reason, a standardized implementation of RL algorithms is maintained in order to have a reproducible starting point, the Stable-baselines 3 library [47]. Incidentally, this library greatly benefits research that tries to apply RL to specific problems, removing the burden of self-implementing a very nuanced complex algorithm.

An additional common issue encountered when developing, testing or applying RL algorithms is the representation of the environment. As discussed in section 2.2.3, an environment accepts an action and provides an observation and a reward. The OpenAI Gym library [48] provides exactly this functionality, becoming the de-facto standard in RL for environment representation. Currently the project has been forked and is being maintained as the Gymnasium library [49]. In addition to providing some standard benchmarking environments, the library allows to easily apply wrappers, that can range from applying specific normalizations to the input and outputs of the environments, all the way to instantiating several environments in parallel to obtain training data more efficiently.

Current RL algorithm libraries are designed to accept a Gym or Gymnasium environment, in this way offering a standardized interface for the user. Albeit originally intended for simulation training, it is now also possible to wrap a real-world environment into a Gymnasium object, in this way allowing the application of baseline algorithms directly to the real-world [50].

2.2.6. Applications of reinforcement learning to large-scale facilities

RL has been successfully applied to a wide range of large-scale facilities, a portion of which are particle accelerators. The reinforcement learning for autonomous accelerators (RL4AA) collaboration [51] was created to share expertise between different centers in order to facilitate the application of these techniques. In [50], for instance, an agent was used to steer the beam in a linear accelerator in order to obtain the desired position and beam size. Notably, the agent was several times faster than a human operator. Another work focusing on particle accelerator control is [52, 53]: policy gradient techniques are successfully applied to the optimization of the FERMI free electron laser (FEL). Reference [54] optimizes the gradient magnet power supply of the Fermilab booster by training the agent on a surrogate model, mimicking the behavior of the real environment, and then deploying the trained agent into an FPGA in order to achieve low-latency inference. This methodology will be more thoroughly discussed in section 3.1. Beam current optimization based on a surrogate model of the LEIR accelerator at CERN was also carried out in [55].

In [7] an agent is trained to control the plasma configuration of a tokamak fusion reactor, showing unprecedented control capabilities. The agent is trained on a high-fidelity simulator and deployed in a real-time system. Similarly, despite using more advance techniques, [56] tunes the AWAKE accelerator at CERN.

Simulation studies for controlling the MBI described in section 2.1.4 at KARA was carried out in section 2.1.4. This work attempts to train an agent to measure the CSR power signal and stabilize it by acting on the RF or bunch-by-bunch (BBB) feedback system cavities. Testing these methods on the accelerator was unfortunately not possible at the time due to technical issues, namely that the time constraints on the agent action where to strict for the implementations described in section 2.2.5.1.

A common pattern in most, but not all, of the work discussed is the use of a simulation to perform training. In most cases, training on the large-scale facility would be prohibitively expensive, as a lot of algorithms are not very sample efficient. This issue is usually circumvented by training on a simulation that is usually quicker than the environment. Sometimes, a high-speed simulation can be a ML model trained on historical data gathered on the environment. One of the main issue with this approach is that the inevitable differences between simulation and real-world can adversely affect, and sometimes completely hinder, the performance of the trained agent. Several techniques have been proven effective to bridge this *sim2real* gap.

A special case where direct training on the environment is possible will be treated in chapter 3, while the technical difficulties that are encountered and the systems necessary to overcome them are analyzed in section 2.3.

2.2.7. Summary

The mathematical basis of ML was discussed, together with the structure of NNs and the backpropagation algorithm used to train them. These techniques can be applied to control problems through RL. The idea of this learning paradigm is based on an agent that learns a policy, in order to maximize a reward through interaction with an environment. The environment provides observations and accepts actions. The reward is chosen during the definition of the RL problem and it allows to define what kind of outcome should be achieved by the agent.

Modern RL algorithms have been described, together with their underlying structure, such as the actor-critic architecture, were an actor chooses the actions to be applied on an environment while the critic predicts the expected return.

A brief overview of current state of the art ML and RL libraries serves as a way to introduce the current methods employed in the field. Several applications to large-scale facilities already exist, but they all suffer from the *sim2real* gap. Given RL algorithms usually require large amounts of data in order to obtain a working policy, they are usually trained on a simulated copy of the real-world environment under study. This make experimentation easier but creates issues when transferring back the learned policy to the real world, as subtle differences in the behavior of the real and simulated environment can hinder, or completely destroy, the performance of the agent.

2.3. Computing devices

Almost any aspect of modern society is heavily reliant on the processing of information. It is thus of no surprise how several different devices address the variety of requirements of this process. In the current chapter, an overview of the most relevant computing devices is carried out, with special focus on system that allow low-latency and real-time processing.

2.3.1. Processing units

This section is based on [57, 58].

The main processing unit of a computer is commonly referred as central processing unit (CPU). These devices process a set of instructions that act on data. A CPU is comprised of a control unit, responsible for instruction decoding and keeping track of the program execution state and an arithmetic logic unit (ALU) that interacts with some registers used to store information, and a memory unit. This structure is known as the von Neumann architecture [59], meaning data and instructions are stored in the same memory unit, called random access memory (RAM). In order to be processed, information needs to be loaded in *registers*, a specialized memory structure within the CPU that allows quick access.

A fundamental step in the working of a processor is instruction decoding. Specifically, the control unit must direct the ALU on how to modify specific registers based on the stream of instructions corresponding to the program currently under execution. This program is usually stored in memory and accessed during execution. Some architecture, as the extremely widespread x86-64 bit architecture, require complex decoding operations. Two classes of architectures can be described based on complexity: reduced instruction set computer (RISC) and complex instruction set computer (CISC). The way of achieving higher performances are orthogonal: for the same operation, a RISC CPU might have to perform a set of small, simple, instructions that can be highly optimized. A CISC CPU, on the other hand, might be able to perform the operation with a far inferior number of instructions, but the final computation time could be comparable, as the more highly optimizable RISC instruction can be potentially executed faster.

Modern CPUs work in a pipeline, i.e. if instructions are not dependent on each other, as in the case where they are operating on different data, they can be scheduled before the previous one has completed execution. Furthermore, if two instructions are entirely independent, they can be executed in parallel. This is known as *instruction level parallelism*. An additional layer of parallelism stems from the fact that most modern CPUs have several *cores* capable of processing in parallel. Moreover, several computing architecture added the concept of vector processing, or single instruction multiple data (SIMD), where a single instruction is capable of operating on multiple data values in parallel. In summary, there is what one could call a parallelism hierarchy: a single instruction can operate on several data at the same time, several instructions can be executed in parallel, and there are several cores executing individual instructions.

Additionally, when there are branches in the execution, e.g. when the execution could potentially jump to a different part of the instruction memory, the decoding unit can speculate which branch is actually taken. If the ALU then finds out that was the incorrect branch, the state is rolled back and the correct branch is taken. This *speculative execution* scheme can be extremely performant when the branch is predicted correctly.

Graphics processing units (GPUs) were devices originally meant to facilitate the drawing of images on a screen. They comprised a set of simple instructions that could be executed in series to produce an output frame on the user screen, allowing the CPU to then perform other operations in the mean time. To this regard, they can be considered as a co-processor.

Modern GPUs are a fundamental component of computing systems. They are usually memory-mapped devices that are conventionally connected through the peripheral component interconnect express (PCIe) bus. These devices offer efficient in-silicon implementation of specific algorithms, for example video en/decoding or ray-tracing.

They are constituted of lattice of small processor capable of SIMD computations. The number of processors in modern architectures can reach several thousands. These cores share high bandwidth memory that is on device, and usually exhibit the cache level structure described in the previous section. The programs executed on these co-processors are usually referred as *compute kernels*. The computation capabilities of these devices has become similar to the one of CPUs, and thus obtained the name of general purpose GPU (GPGPU). Data can be loaded

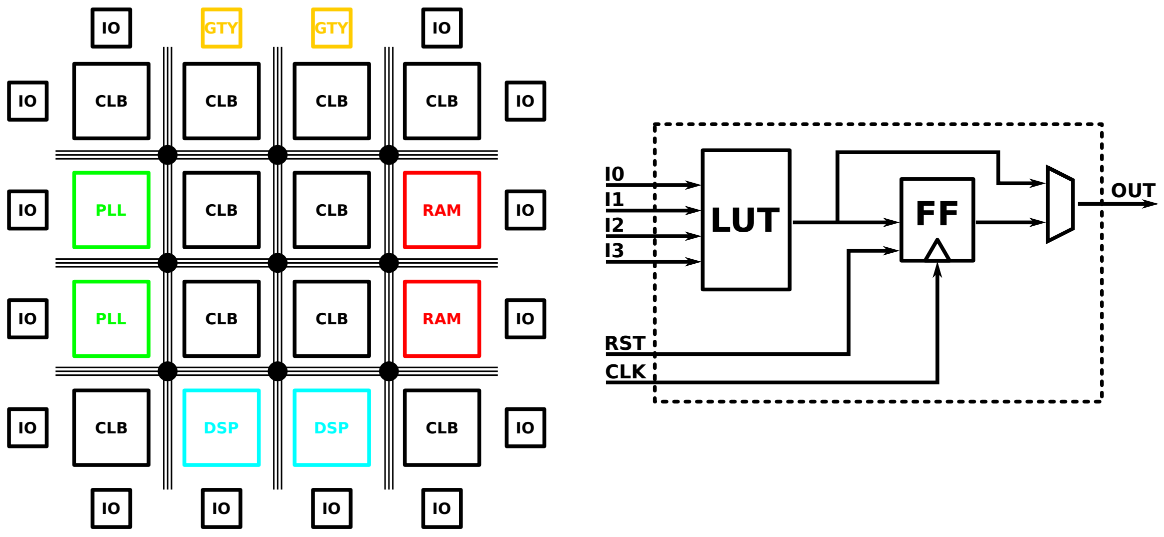

Figure 2.15.: Schematic representation of an FPGA lattice (left) and of a CLB (right).

from the CPU memory into GPU memory. It is worth noticing how GPUs are optimized for a specific kind of computation: large amount of data that is processed in a highly parallel way. Starting computing kernels can produce considerable overhead, so computation of small batches of information can incur in a considerable performance penalty. These tradeoffs will be more exhaustively discussed in section 2.3.3.

Several programming frameworks are available and allow the definition of kernels and data exchange between the main computer and the GPGPU. A general platform independent framework is OpenCL [60]. The two main producers of GPGPU also provide their own systems: Advanced Micro Devices, Inc. (AMD) offers ROCm [61], while Nvidia Corporation provides CUDA [62]. These frameworks greatly facilitated the adoption of GPUs in ML and are now a de-facto standard computing platform for the training and inference of large models.

The computing platforms just described are based on application specific integrated circuits (ASICs). This means that once the device is produced, no modification can be performed on the logic that has been placed in the chip. During prototyping this could be undesirable, as ASIC production is expensive and requires a considerable amount of time.

One alternative architecture is that of FPGAs. These devices are constituted by a great number of configurable logic blocks (CLBs) that are placed in a mesh and are interconnected by programmable switches (figure 2.15). In general, each of these cells contains at least a look-up table (LUT), a completely programmable n-bit logic function[6], and a flip-flop. A flip-flop is a memory element that allows the synchronization of digital signals to a reference *clock* signal. The production of clock signals at a desired frequency, is thus a key component of digital designs, not only of FPGAs. Additionally, it is frequently useful for the generated clock to have a well known phase and frequency relation with respect to a reference signal. This can

[6] Newer architectures allow a high number of inputs. The Versal platform that will be described in section 2.3.2.2 has 6-bit LUTs.

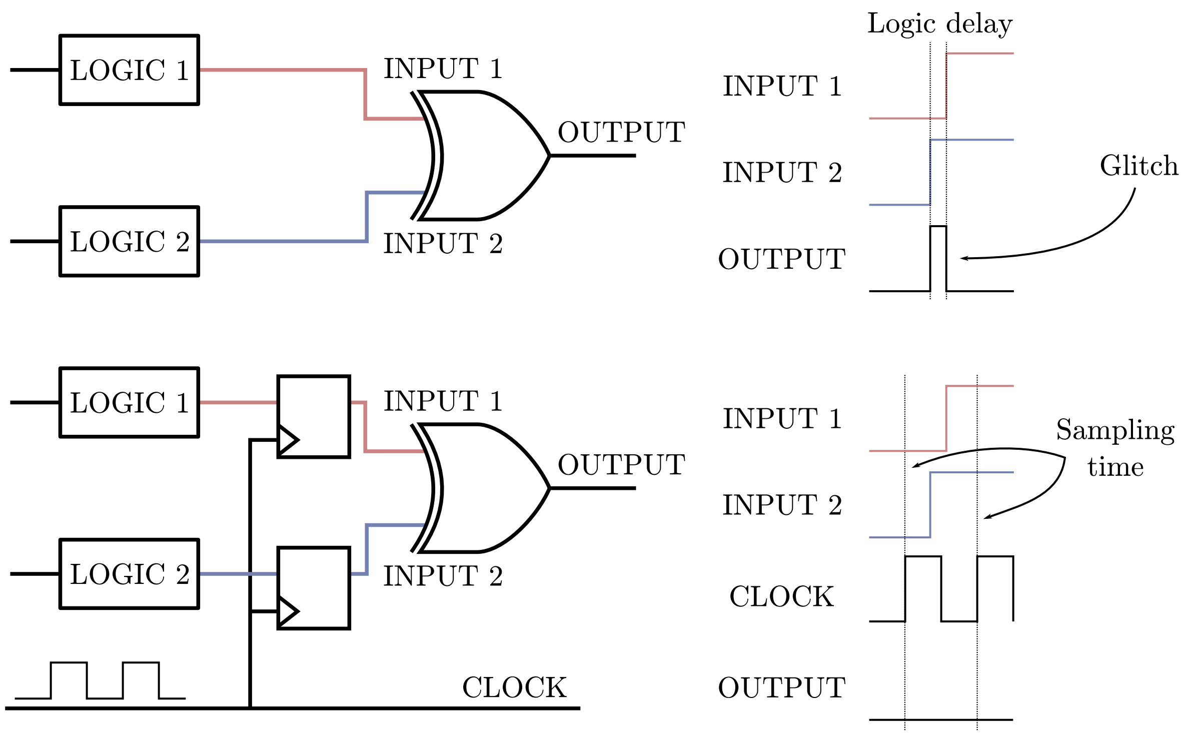

Figure 2.16.: In the top part, two logic blocks having different propagation delay produce a glitch at the output of the exclusive nor gate. In the bottom part, the introduction of flip-flops, sampling the signal aligned with the clock rising edge, removes the glitch.

be obtained via a phase-locked loop (PLL). A simple PLL would operate as follows. The device has an internal voltage controller oscillator (VCO)[7], whose output signal can then be compared with the reference, obtaining the relative phase between the two. Such phase signal is then filtered and applied as a control signal to the VCO. This creates a feedback loop with the following effect: if the output starts to lag with respect to the reference, the phase signal will start to differ from zero. This will increase the frequency of the VCO so that the output signal can "catch-up" with the reference and resynchronize. PLLs are a key component within an FPGA.

Combinatorial logic, digital circuits that are not synchronized to any clock, can easily exhibit *glitches* due to different time delays in the data paths, as shown in figure 2.16. If these phenomena are not properly considered, the system might not work. A clock signal allows signals to be sampled and to vary at precise times, in this way removing glitches.

For signal processing and ML applications, multiplications are key operations that need to be executed several times. Implementing multiplications with CLBs requires a lot of resources. FPGA designers overcome this issue by including special digital signal processing (DSP) cells that are optimized to perform multiplications. Modern designs can also operate on floating point numbers. Additionally, storing memory in CLBs is usually inefficient, as the memory element in the cell, the flip flop, only stores a single bit. Storing any meaningful amount

[7] The VCO usually operates at high frequency, so its output clock is usually divided before use.

of information would thus utilize a great number of resources. Special RAM cells are thus included, allowing efficient data storage.

In order to exchange data with external devices, FPGAs usually have a number of serial transceivers. These blocks implement a serializer-deserializer architecture, taking parallel data from the lattice and outputting the corresponding high-speed serial signal on one output pin. Similarly, high-speed serial signals can be converted into lower speed parallel signal, that are more easily processable. In some cases, the protocols required for external communication are complex to implement with the limited logic resources available. Several modern architecture supply special blocks developed on-silicon to offload part of the computations required. An example of these are the 100 gigabit Ethernet modules in the AMD-Xilinx Zynq Ultrascale+ family of devices [63].

In recent years, it has become evident that different devices approach computing in distinct ways. A CPU processes tasks in a linear and sequential manner, making it highly effective for sequential workloads. In contrast, GPUs operate in a parallel-sequential fashion, excelling at highly parallelizable tasks but offering relatively low performance for individual operations. Meanwhile, FPGAs enable spatially-parallel computation. To leverage the complementary strengths of these technologies, system-on-chips (SoCs) were developed, integrating an FPGA and a CPU, commonly referred to as programmable logic (PL) and processing system (PS), respectively. In this setup, the CPU manages and configures the PL, which can be accessed by the CPU as a memory-mapped device. In this way a single platform can achieve the best of both approaches, with processing units capable of addressing sequential and spatially parallel tasks.

This architecture can be naturally extended with several modules. In order to manage their communication a packet-based switching and routing system called network-on-chip (NoC), described in greater detail in section 2.3.2.3, has been included in modern devices. This allows the PS and PL to share double data rate (DDR) RAM memory of the system. Additionally, these devices have reached physical sizes where routing data through the lattice has become impractical, and fast streaming communication can be routed through the NoC.

FPGAs have very different architecture compared to a CPU. Naturally, the programming stacks also reflects the difference between the two devices. Hadware description languages (HDLs) are a special set of programming languages that, as their name suggests, are meant to precisely describe the functionality of a digital logic circuit. The two most commonly used HDLs are Verilog [64] and VHDL [65]. In order to implement a given design on the device, a *synthesis* stage, converting the described logic into gates, is required. After this, the synthesized design undergoes an *implementation* stage, where the logic gates are translated into the actual hardware resources available, i.e. in the LUTs of the CLBs and the switch matrices are set. Timing constraints are also applied and verified to ensure the functionality of the system. If this process is successful, the FPGA can be programmed with the generated *bitstream.* A logic block performing a specific kind of task is called an intellectual property (IP) core. This name originates from the fact that it is usually licensed, and as such they are the intellectual property of somebody.

HDLs require different programming techniques compared to conventional procedural programming languages, rendering access to FPGA computing more difficult for the average programmer. To overcome this high-level synthesis (HLS) toolchains [66] are provided by the biggest FPGA manufacturers, allowing the conversion of properly written C/C++ functions into fully functioning IPs.

2.3.2. Data transfer techniques

A fundamental aspect of computing is the way data is transferred to/from the processing unit. The most widespread form of data transfer is the one concerning memory.

As an example, figure 2.17 shows the memory structure of a CPU. Loading and storing data from RAM is an expensive operation, thus several cache layers are added in order to speed it up. A modern CPU has three layers of cache, L1, L2, and L3, where lower number means closer to memory. In some architectures there can be two separate L1 caches, one for data and the other for instructions. When a load operation is performed, and the requested information is not present in cache, an occurrence known as a *cache miss*, the data needs to be fetched from the underlying cache layer, with a performance penalty. If information is not present in any layer it will have to be retrieved from memory.

To operate on memory, the CPU uses *addresses* to reference different memory locations. Some devices, known as *memory-mapped* devices, can also be accessed by the CPU as memory locations at specific addresses within the physical address space. In earlier computing systems, memory was a highly valuable resource. When operating systems (OSs) began supporting the execution of multiple programs simultaneously, each program was allocated its own portion of memory. This memory remained allocated to the program even when it was not actively executing, limiting the efficient use of available resources. To address this issue, a mechanism for saving the memory allocated to a program to persistent storage (e.g., a hard drive) was introduced, allowing memory to be reclaimed and used by other programs. This innovation led to the concept of a *virtual address space*. When a program accesses a specific memory address, the OS translates this address into a physical memory location using a *page table*. A page, in this context, is the smallest unit of memory the OS can allocate. By modifying the entries in the page table, the OS can dynamically redirect a program's memory accesses to different physical locations. This flexibility is especially advantageous when a program's memory is saved to disk while the program is inactive. When the program needs to resume execution, its memory is reloaded from disk into a potentially different physical location, and the page table is updated to ensure that all virtual addresses continue to point to the correct data. This seamless mapping allows programs to function as though their memory was never moved. Modern system have access to extremely wide address spaces, with x86-64 architecture processors being able to address 256 TiB, with future extensions capable of reaching 16 EiB.

In order to avoid spending execution time for transferring data to and from memory, the concept of direct memory access (DMA) was introduced. A DMA unit takes a descriptor, given by the CPU, indicating how much data should be transferred and between which addresses or data streams, and copies data until completion without the need of intervention from the CPU.

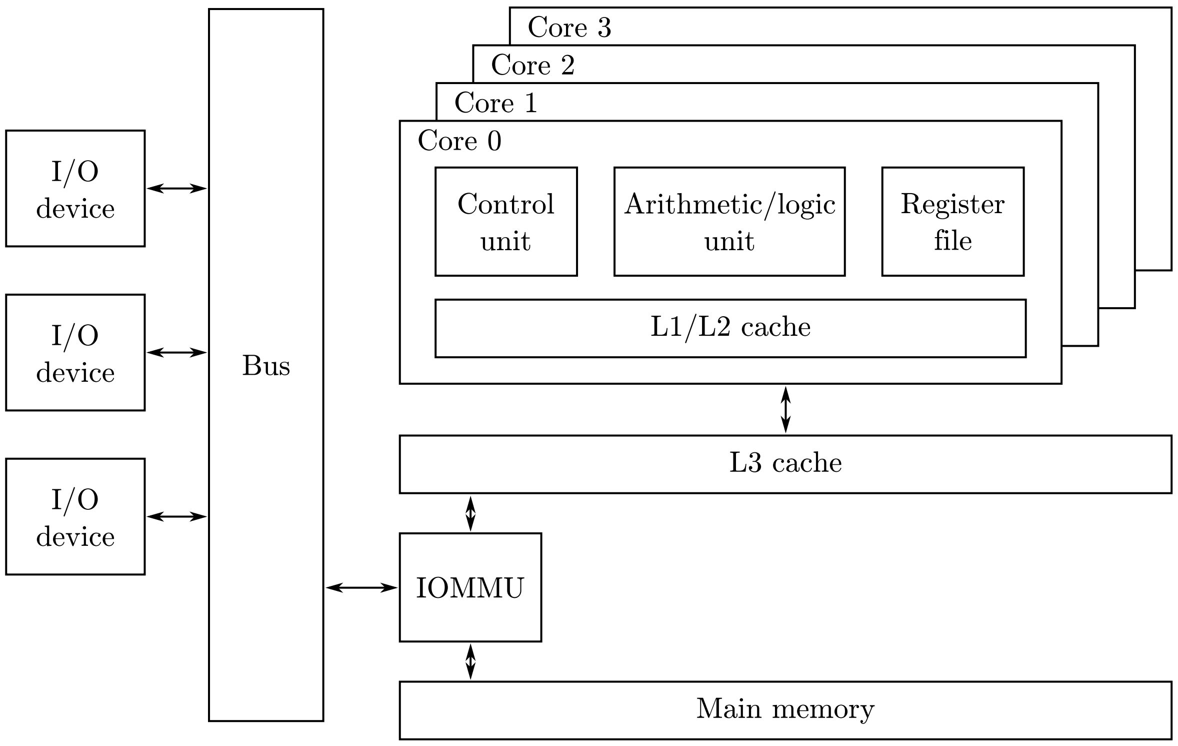

Figure 2.17.: Schematic example of a possible CPU architecture, showing how each core has their own internal compute units, registers, and cache. Different cores share the level 3 cache. Through the IOMMU they can access the main system memory and the various input/output devices.

DMAs are widespread within a computer architecture. They are usually included both in the CPU itself and in some peripherals. One example of such hardware is based on the PCIe bus, a widely available memory-mapped interface. A PCIe endpoint can copy data via DMA into the system memory and vice versa. This is commonly used in devices like network cards to fill packet buffers so that the CPU can process data in batches instead of having to continuously operate on a stream. In order to protect the system from malicious hardware, several systems implement an input/output memory management unit (IOMMU) that is able to provide virtual addresses to DMAs and verify that data is read and written from allowed memory regions. This unit is also responsible for memory paging.

2.3.2.1. AXI4

This section is based on [67, 68].

In order to reliably and modularly interconnect different components of a digital device, it is fundamental to define a communication protocol. For AMD-Xilinx programmable logic devices, the advanced extensible interface 4 (AXI4) protocol was chosen. Three kinds of interfaces are commonly used in these systems: AXI4-Stream, AXI4 and AXI4-Lite.

AXI4-Stream is a point-to-point protocol connecting a single master to a slave. Given its simplicity and the fact that several ideas are reused in the other two interfaces, it will be

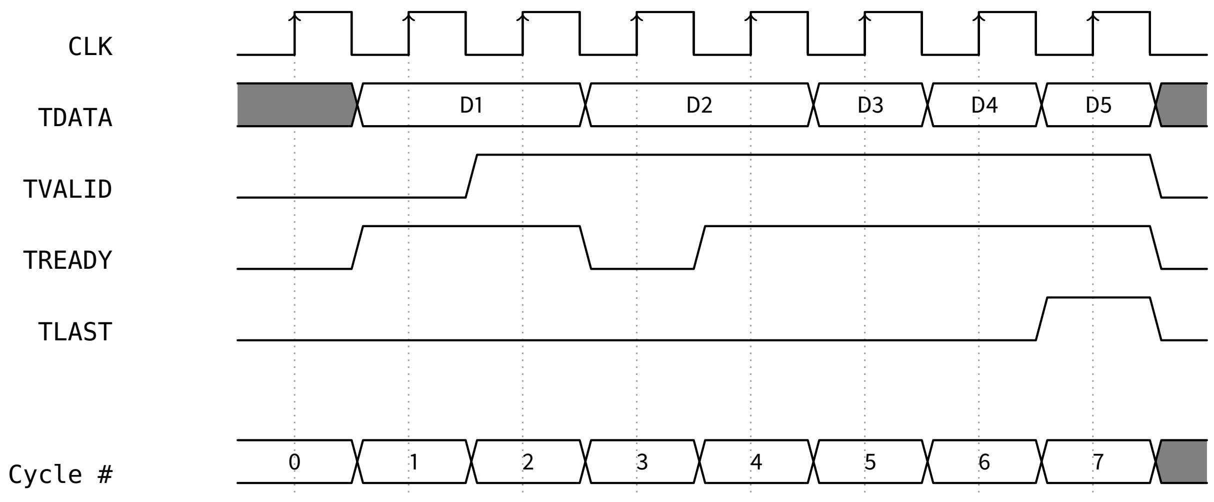

Figure 2.18.: Example signals of an AXI4-Stream bus. Between edge 0 and 1 the slave asserts TREADY, signaling it can accept data. Data is only transferred in edge 2, when TVALID is asserted. At that time, the master is ready to produce more data, but it has to wait for TREADY to be asserted again. At edge 7 the last transition of the packet is performed, with TLAST asserted.

described first. In a basic implementation, five signals are employed: clock, reset, TREADY, TVALID, and TDATA. The edge of the clock, conventionally the rising one, is used to register the values of the signals, as previously discussed in section 2.3.1. When reset is asserted the logic is brought into a known state. The standard specifies that the reset is active-low. TVALID is asserted by the master when the data on TDATA is valid, while TREADY is asserted by the slave to indicate when it is capable of receiving data. When both TVALID and TREADY are high, a handshake occurs indicating the data contained in TDATA has been transferred, as shown in figure 2.18. In order to avoid stalls, only the slave is allowed to wait for the master to set TVALID, while a master is not allowed to wait on a TREADY. The standard allows several other signal to bring side-channel information. For example, TLAST indicates the boundary of a packet. TKEEP is employed to indicate which bytes of TDATA remained valid. Several other facilities to transport user information (TUSER), to identify the stream (TID), to indicate a destination (TDEST) are described in the standard.

AXI4 is a protocol allowing memory-mapped communication from a master to several slave components. Compared to AXI4-Stream, that has a single transaction channel, AXI4 has five: read address (AR), read data (R), write address (AW), write data (W), and write response (B). The standard allows transferring data in “bursts”, meaning the master creates a transaction request in the address channel by specifying the start address and size. Several read or write data transfers can occur based on this request. A read transaction, shown in the left side of figure 2.19, works as follows: the master sets the address on the ARADDR signal, together with the number of transfers to read (ARLEN). The ARBURST signal describes the how the provided address is incremented in each transfer. FIXED mode continuously reads the same address, while INCR mode reads a sequential region starting with the provided addresses. Additionally, WRAP behaves as INCR but wraps back to the start address when a given limit is reached. An ARVALID and ARREADY handshake is used to register this data, exactly as in AXI4-Stream.

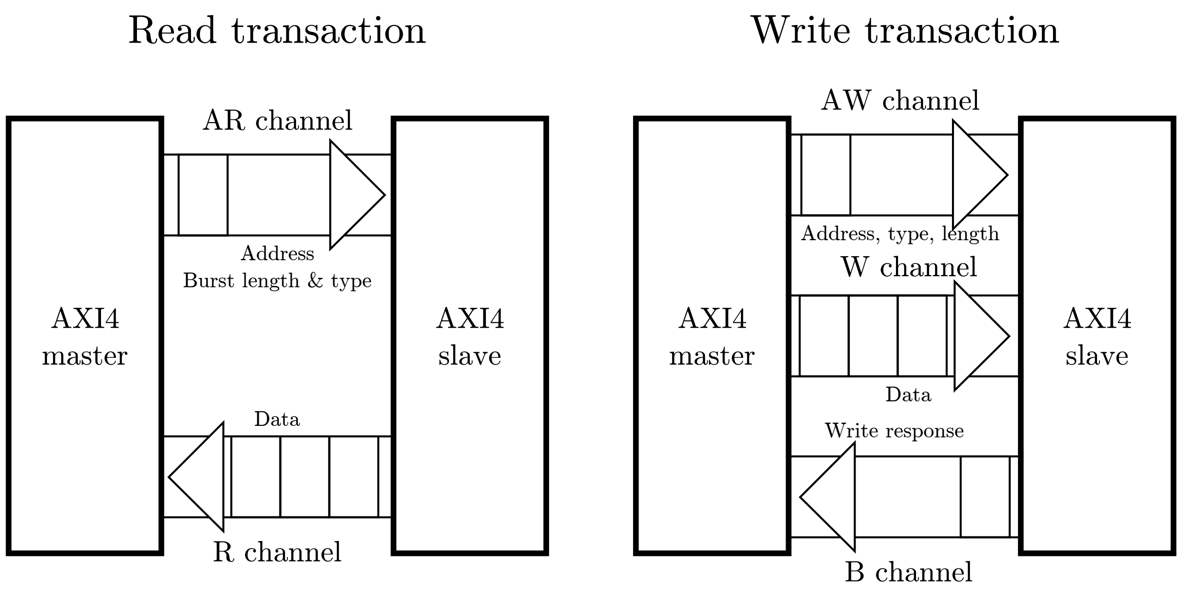

Figure 2.19.: Diagram describing the AXI4 read (left) and write (right) transactions. In a read transaction the master uses the AR channel to send the address to be read and the parameters as burst size and type. The slave responds by transferring the data on the R channel. Similarly, in a write transaction the request is produced by the master on the AW channel. The master then places the data to be written on the W channel. The slave responds with the outcome of the write operation on the B channel.

The read data will be produced by the slave on the R channel. A write transaction proceeds analogously (right side of figure 2.19). A descriptor is provided in the address channel AW and the data is then placed on the W channel. The main difference is the presence of the write response channel, where the slave signals the outcome of the operation. It is worth noticing how the protocol allows for a slave-dependent number of outstanding transactions, meaning several can be scheduled without the data being yet provided.

Some devices might benefit from memory-mapped addressing without the need of the high-performance burst transaction capability just described. The AXI4-Lite protocol targets this specific use case. It represents a subset of the AXI4 protocol where all burst lengths are 1, and all data accesses use the full width of the bus, i.e. 32 or 64 bit. This greatly simplifies the design and reduces the resources necessary for the implementation of a component. One of the main use cases of this protocol is for defining configuration registers of IP cores, so that they can be accessed by a CPU.

A fundamental aspect of AXI4 and AXI4-Lite are interconnects. These components allow the connection of multiple masters with several slaves. One of their fundamental tasks is performing the routing of transactions to specific slaves, depending on the design choices of the implementation.

2.3.2.2. The AMD-Xilinx Versal™ Heterogeneous Platform

As discussed earlier in this chapter, different computing platforms exhibit vast differences in their capabilities. As such, depending on the application, there is no "one size fits all"

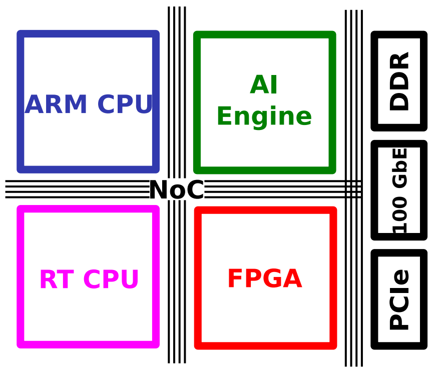

Figure 2.20.: Simplified drawing of the architecture of a Versal™ device, showing the NoC and how it interconnects the computation units (Arm processors and AIE array) and the interfaces.

solution. The AMD-Xilinx Versal™ advanced compute and acceleration platform (ACAP) [69] was originally introduced to target cloud and edge computing applications. This device is manufactured with the cutting-edge Taiwan Semiconductor Manufacturing Company 7 nm fin field-effect transistor process [70]. A simplified drawing of the architecture is shown in figure 2.20. It could be considered an extension of the SoC architecture described in section 2.3.1, with the addition of a programmable NoC and of an AI Engine (AIE) array[8]. Specifically a dual-core Arm Cortex-A72 processor is paired with a dual-core Arm Cortex-R5F real-time processor. These units access memory through the NoC, that can interface to DDR memory controllers. Both the FPGA and the AIE can also access memory through the NoC, allowing seamless DMA operation. In this way, all processing units can access memory in a fashion tailored to their requirements. This is achieved thanks to the capability of the NoC to set quality-of-service, latency and bandwidth constraints can be defined for different memory accesses.

Additionally, several high-performance interfaces are available. On some devices of the family, the high-speed transceivers allow reaching single-line transfer speeds of up to 32 Gbps. Moreover, hardened IP cores allow the implementation of protocols as 100 Gb Ethernet [71], together with DMA capable PCIe Gen4 transfers. Special defense-oriented versions also offer high-speed cryptographic cores and random number generation.

2.3.2.3. Network-on-chip

When developing designs targeting heterogeneous platform, the orchestration of data transfers between different units has profound impacts on performance. As discussed earlier, Versal™ has several processors and computation units that share the same memory controllers through a NoC [72], as shown in the left side of figure 2.21.

The main components of the NoC are the NoC master units (NMUs), NoC slave units (NSUs), and NoC packet switches (NPSs) (right side of figure 2.21). Each computation unit, e.g. the AIE

[8] AI in this case stands for adaptable engines, not artificial intelligence.

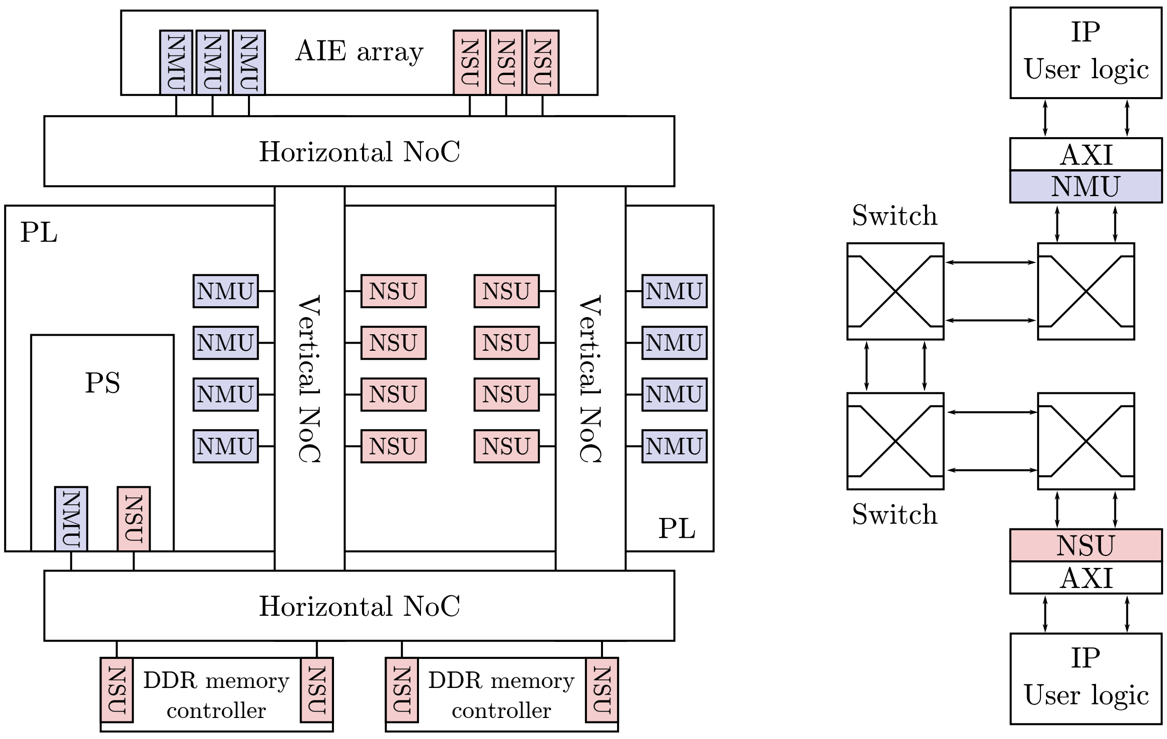

Figure 2.21.: Simplified schematic showing the structure and functioning principle of the NoC. On the left side, the Versal™ architecture is shown, highlighting the presence of vertical and horizontal NoCs, together with the NMUs and NSUs in the PL, PS, AIE array, and memory controllers. On the right side, an example connection between two AXI4 IP cores is shown. AXI4 requests are converted to NoC packets that are routed through several switches.

array, the PS, and the PL, all have NMUs and NSUs that they can use for transferring data through the NoC. In this way, the NoC serves as an advanced AXI4 network allowing high-speed data transfer with switching and quality-of-service to manage transaction priorities.

An AXI4 master is connected to a NMU, taking care of the clock domain crossing and packing the transaction into a NoC packet. This data is then routed by one or more NPS taking care of the quality-of-service and packet arbitration. Several different traffic classes are available targeting low-latency or best-effort use cases. Finally, a NSU converts the packet into AXI4 and transfers the reply of the slave. A special kind of NoC slave are the one connected to a memory controller. A command queue is present that allows sorting the commands sent to the memory based on their traffic class.

The Versal™ device NoC is partitioned in vertical and horizontal NoCs. The horizontal NoC is placed at the top and bottom of the die and connects the memory controllers, the processors and the AIE array. It is connected to the vertical NoC that interfaces to the PL.

2.3.3. Real time computing

This introduction is based on [73].

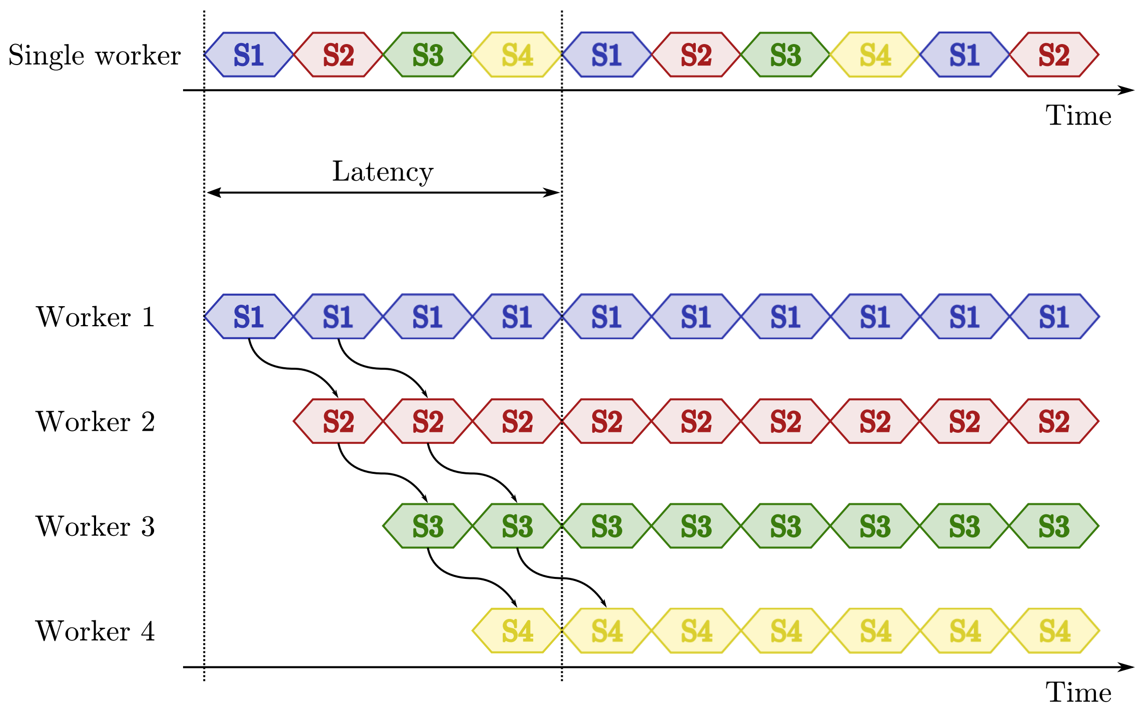

Figure 2.22.: Comparison of the single person (top) and pipelined work schedules (bottom). It is worth noticing how the latency from the input stage S1 to the output S4 in both cases is identical. The throughput, on the other end, four times higher in the multi-worker example after the initial "priming" stage.

When designing a data processing system, it is typically necessary to meet certain constraints dictated by the application. For instance, the processing unit must be capable of handling all incoming data within a specific time frame. The amount of data processed per unit time is termed *throughput*, and such a requirement is known as a throughput constraint. Similarly, some systems may demand that a particular computation be completed within a predefined duration. The interval between feeding input data into the processing unit and obtaining the output is called *latency*, leading to what is known as a latency constraint.

These two concepts are related, but can differ substantially. A classic example that can shed light on the difference is the one of a production plant, as shown in figure 2.22. A given appliance can be produced by a worker in about four hours. During an average eight hour working day, the worker will thus produce two appliances. We can thus consider the throughput to be two appliances per day and the latency to be four hours. If we consider the same plant, but the production of the appliance is divided into four steps of one hour each, four workers could be assigned to each step. In this case, as soon as worker 1 finishes working on the first product, they could immediately start the second. Noticeably, the latency of the process is unchanged, a complete production still takes four hours, but some tasks can now be executed in parallel, leading to a much higher throughput of an appliance per hour. It is worth noticing what happens to this assembly chain when it needs to start from scratch: the last workers need to wait for the first appliance to reach them. In this condition, the instantaneous throughput is affected, as the chain needs to be properly "fed" in order to achieve the maximum possible production rate. This structure in data processing is known as a *pipeline*.

If one examines the requirements for the implementation of a system interacting with the external environment, as a RL agent would, it is possible to notice how both throughput and latency constraints need to be taken into account. A task that has latency constraints, is usually referred as a *real-time* task. These kind of systems usually need to be able to respect deadlines for the time a given computation is completed. This has the striking effect of promoting the time available for computations to a resource, much like the available memory and processing power. Deadlines can be separated depending on the consequences of a missed timing deadline. If, for instance, consequences are catastrophic (e.g. loss of lives), the deadline is said to be *hard*. In the case where data coming after the deadline is not useful anymore, a deadline is *firm*. Lastly, if the usefulness of the late information decreases with time, a deadline is referred to as *soft*.

Real-time tasks can be divided in two classes: *periodic* and *aperiodic*, depending on their arrival time characteristics. Additionally tasks might be composed of several sub-tasks that depend on one another. As such, their scheduling is a complex problem that needs to be solved such that all deadlines are met. In order to do so, real-time systems are usually required to be predictable, meaning a given task is guaranteed to be handled in a given amount of time that is usually obtained through simulations. As it will be more thoroughly discussed in the next section, off-the-shelf consumer CPUs are not meant to perform real-time computations. Functionalities as caching and virtual memory paging, for example, can lead to spikes in latency that are difficult to predict. Special real-time oriented CPU architectures exist to increase the task predictability.

2.3.3.1. Moving data into a CPU

In order to evaluate the performance of a consumer CPU for real-time task, a simplified system was devised to measure the latency characteristics of PCIe communications between the CPU and an external FPGA. To do so, a desktop computer based on an 11th Gen Intel® Core™ i7-11700 CPU with 32 GiB of RAM was equipped with an HighFlex2 board [74] carrying an AMD-Xilinx Zynq Ultrascale+ XCZU11EG FPGA. A firmware based on the Ultrascale+ Integrated PCIe IP core [75] was developed, with the capability of monitoring a memory provided by the host region via DMA.

A drawing of the setup is shown in figure 2.23. The computer runs an Ubuntu Linux 22.04 OS running kernel version 5.15 with a real-time patch. A kernel module was implemented allowing the allocation of a single 4 KiB cache-coherent memory page that can be memory mapped from user-space. After allocation, the address of this page provided by the IOMMU is communicated to the FPGA. At this point the FPGA goes into polling mode, sending a read request every 256 clock cycles, i.e. $\approx$1 μs of the double word sized (32 bit) *reply request* memory region. When the value in this region changes, meaning the CPU sent a reply request, the FPGA increments an internal counter and writes the incremented value to a double word sized *reply send* region. After the reply is sent, the FPGA goes back to polling mode. In the meantime, the CPU is polling the reply send region waiting for the reply coming from the FPGA.

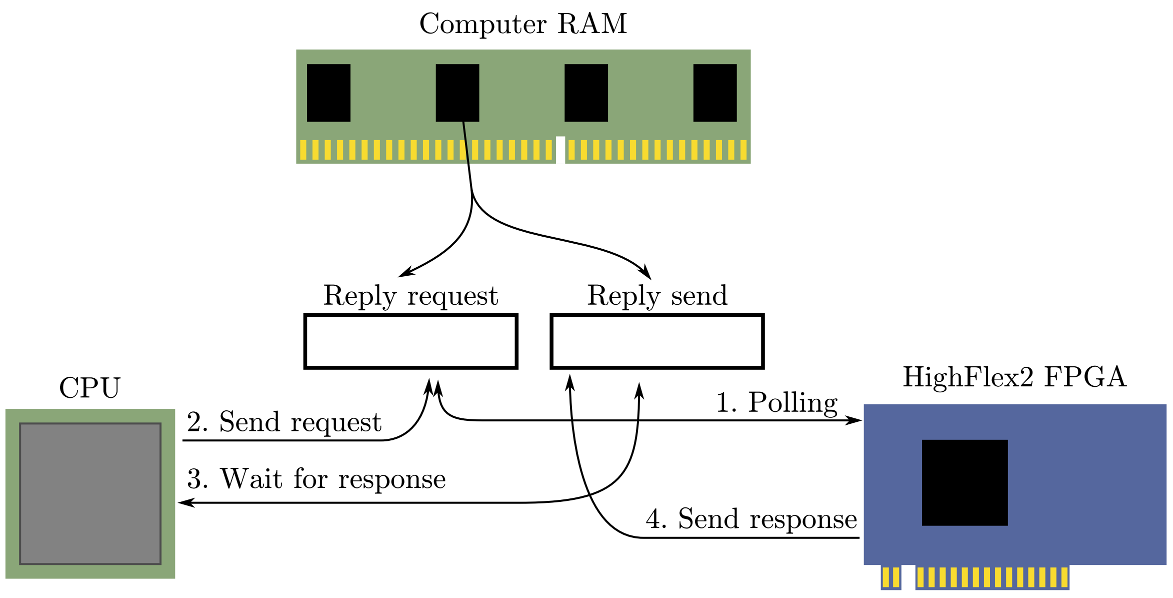

Figure 2.23.: Drawing of the system with the various read and write operations. The FPGA continuously polls the reply request region. As soon as a change is detected it sends a response by writing to the reply send region. The CPU starts a trial by writing a new value to reply request and then polling reply send until an answer is obtained from the FPGA.

Listing 2.1: Kernel boot parameters used for latency measurement isolating core 4 and 5 in an 8 core machine.

```
ro quiet splash skew_tick=1 rcu_nocb_poll rcu_nocbs=4,5
kthread_cpus=0-3,6-7 nosmt=1 nohz=on nohz_full=4,5
isolcpus=domain,managed_irq,nohz,4,5 irqaffinity=0-3,6-7
intel_iommu=on iommu=pt intel_pstate=disable nosoftlockup
tsc=reliable vt.handoff=7
```

On the CPU a user space program measures the amount of time it takes to obtain an answer in the reply send memory region after it changed the value in the reply request region. The time measurement is preformed by using the RDTSC timestamp counter. In order to minimize the disturbances to this process, it is first of all executed on an isolated core, meaning that the core is not used by the OS scheduler. Additionally, the kernel is in tickless mode, minimizing the frequency of management interrupts obtained by the core to less than one per second. Furthermore, the core affinity for interrupts is disabled. The full Linux kernel boot parameters are shown in listing 2.1. Several trials are repeated consecutively, and a histogram of the latencies obtained is show in figure 2.24. Notably, the average latency is 1.11 µs, with a standard deviation of 0.27 µs and a median value of 1.02 µs. The minimum and maximum latencies observed are 0.88 µs and 8.92 µs respectively.

The great majority of trials have latencies of roughly a microsecond. This value nicely corresponds with the polling rate of the FPGA firmware. This rate cannot be increased as the IP core provided by AMD-Xilinx only allows posting a limited number of read-requests. The second peak structure around 2 µs in principle nicely corresponds to the trials where a second poll was necessary. As shown by the second small peak at 8 µs, comprising $< 0.05\%$ of the

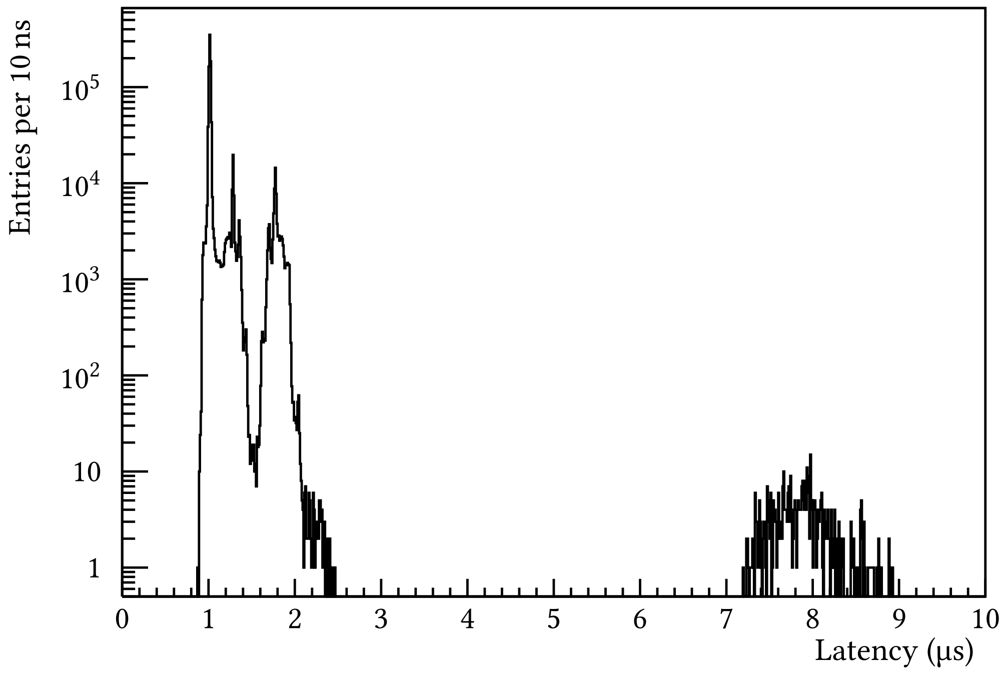

Figure 2.24.: Histogram of the round trip latency between a CPU and an FPGA via PCIe for 10^6 trials. In a few cases there are latency spikes around 8 μs that are unwanted but bound.

trials, some latency spikes are nonetheless possible. At the time of writing, the source of these spikes is unknown. Besides, it is worth noticing how a best case scenario would still lead to microsecond level round-trip latencies. This number, not comprising any form of computation, is high compared to the one achievable with more refined FPGA-based system. As such, for the rest of this work, the use of a conventional CPU or GPU for real-time tasks will not be considered, focusing more on devices like the ones described in section 2.3.2.2.

2.3.4. Specialized ML coprocessors

When ML models are deployed for day-to-day operation, their cost and sustainability are strongly dictated by the energy efficiency of their execution. As discussed in the previous sections, several different computing platforms exist, each with its own peculiar capabilities. Nonetheless, they all try to perform a general set of operations. An alternative approach is the use of computation accelerators, sometimes also referred to as artificial intelligence (AI) accelerators. This class of devices targets the most demanding task of a given workload and tries to compute it as efficiently as possible. The resulting electronics is not general purpose anymore, but can attain a high level of energy efficiency.

A computationally intensive task is matrix multiplication, that is necessary for the inference of a NN as described in section 2.2.2. Modern CPUs are already capable of performing SIMD instructions that facilitate NN inference. An example of these are the advanced vector extension

(AVX)-512 vector neural network instructions (VNNI) introduced by Intel in its latest family of processors [76]. Such an approach heavily relies on the preexisting structures of the CPU, and can thus achieve only a limited power consumption gain.

An alternative architecture, specially indicated for matrix multiplication, are systolic arrays [77]. These structures are composed of a large number of simple data processing units referred to as *nodes*, sometimes capable of performing only a multiply and accumulate instruction, that are interconnected with each other. At each clock cycle, each node loads data from its inputs and performs its computation, making the result available on its output. In this way, the computation is built at every "beat" of the clock, leading to the name of this architecture. Google named its hardware accelerators based on systolic arrays tensor processing unit (TPU) [78]. Low-power versions of these devices exist that allow ML inference on the same *edge* system that is gathering data. This has the additional impact of increasing privacy, as the data is processed locally where the user has more control.

A completely different approach is analog computing. Instead of relying on a digital representation of values, multiplications can in principle be performed by employing Ohm's law $V = RI$, where V is the voltage across a resistance R is traversed by a current I. If a device sets the resistance to a desired amount, for example a representation of the coefficient of a neuron, and a source sets the current to the input value of the neuron, simply measuring the voltage across the resistor gives the result of the multiplication. This kind of technique is leveraged by Mythic AI [79] to produce highly energy efficient edge devices.

Finally FPGAs can be employed as a computation accelerators for NN tasks that are not well suited for CPUs and GPUs. The AMD-Xilinx Versal platform described later in section 2.3.2.2 is, among other intended use cases, a promising ML accelerator. Coincidentally, RL training on FPGAs is one of the tasks that benefits greatly from hardware acceleration, as it will be discussed in section 3.1.

2.3.4.1. AI Engine array

This section is based on [80, 81].

As discussed earlier, one of the great advantages of the Versal™ family of computing devices is the availability of an AIE array allowing increased performance in multiplication-heavy workloads compared to an FPGA. This component resides in its own clocking domain, with a nominal frequency of 1 GHz, that can optionally be increased to 1.3 GHz. The array is constituted by a two-dimensional matrix of computational units referred as *tiles*. Each one of these tiles contains an interconnect module, a memory module, and an AIE (figure 2.25).

The AIE is comprised of a 32 bit RISC processor, two load units, one store unit, a vector processing unit, and an instruction fetch and decode unit. The latter supports very large instruction width (VLIW), meaning it allows issuing a separate instruction to each of the aforementioned units. This is clearly visible in the two lines of AIE assembly shown in listing 2.2: separated by the semicolon, an instruction is issued to the each unit. If a unit is not used, a NOP, i.e. no-operation instruction, is issued. The vector unit allows simultaneous

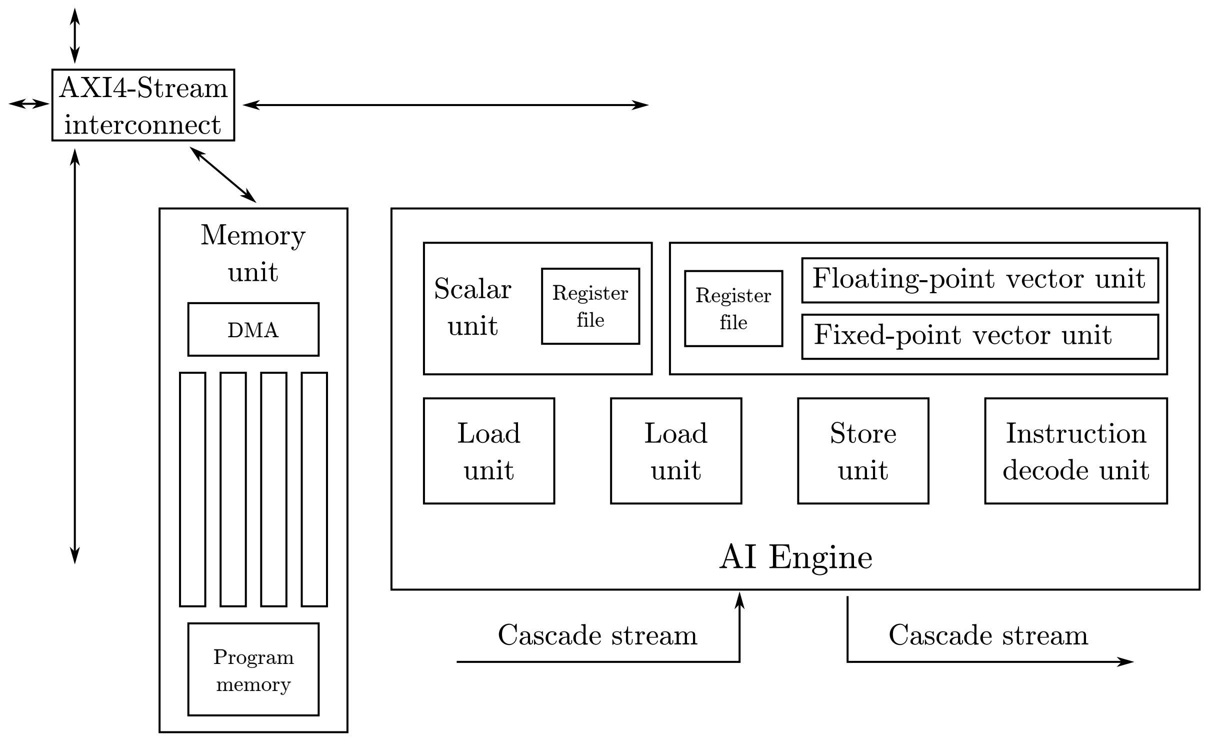

Figure 2.25.: Schematic highlighting the main components of the AIE tile architecture, showing the memory unit, AIE, and AXI4-Stream interconnect with a summary of their respective internal components.

arithmetic, logical, and permutation operations on vector lanes, on both integer and floating point numbers. Multiply and accumulate (MAC) operations can thus be performed efficiently and in parallel, as shown in table 2.1. A single tile, if programmed correctly, can reach perform 8 single precision floating point MACs per clock cycles, a level of computational power superior to current FPGA DSP blocks. It is important to notice that this kind of performance can only be achieved when the multiplication pipeline is completely full. Specifically, one MAC operation can be issued every clock cycle, but the output is produced with a latency of eight clock cycles. The operands of these instructions are stored in the vector register file. This component allows using 128, 256, 512 and 1024 bit vector registers. Additionally 384 and 768 bit accumulator register are employed in integer vector operations. Each tile integrates a 32 KiB memory block, divided into eight banks of 256 words × 128 bit, i.e. 4 KiB. Each tile can address the memory blocks of the tiles that in the array are located "north" and "south" of it, and either "west" or "east". As such, the total addressable memory is 128 KiB. The instruction memory is separate from the data memory and allows storing 1024 instructions, each 128 bits long, for a total of 16 KiB. In some cases shorter instruction sizes are possible: when the computational units are not completely used, more memory efficient encodings are supported. The two load units and one store unit can operate simultaneously, albeit with a five cycle latency, on three 256 bit words, provided they reside in different memory banks.

In order to efficiently execute loops, a special zero-overhead loop infrastructure is present. Specifically, loop start, end, and count registers are available in the processor. When the end pointer is reached, the processor automatically verifies if the number of necessary iterations of

```
VLDA wr1, [p0]; NOP           ; NOP               ; MOV.u20 ch0,  #274960;
NOP           ; MOV.s9 r11, #0; ST r11, [sp, #-24]; MOV.u20 ch1, #0     ;
```

Listing 2.2: Example of AIE assembly showing the VLIW structure of the instructions. Several NOP instructios are visible, showing units that are currently not processing data, together with vector loads (VLDA) and register moves (MOV).

Operand 1 (bits, type)	Operand 2 (bits, type)	Output (bits, type)	MACs per clock cycle
8 real	8 real	48 real	128
16 real	8 real	48 real	64
16 real	16 real	48 real	32
16 real	16 complex	48 complex	16
16 complex	16 real	48 complex	16
16 complex	16 complex	48 complex	8
16 real	32 real	48/80 real	16
16 real	32 complex	48/80 complex	8
16 complex	32 real	48/80 complex	8
16 complex	32 complex	48/80 complex	4
32 real	16 real	48/80 real	16
32 real	16 complex	48/80 complex	8
32 complex	16 real	48/80 complex	8
32 complex	16 complex	48/80 complex	4
32 real	32 real	80 real	8
32 real	32 complex	80 complex	4
32 complex	32 real	80 complex	4
32 complex	32 complex	80 complex	2
32 float	32 float	32 float	8

Table 2.1.: AIE MAC per clock cycle based on the operand and result types.

the loop has been performed, and sets the instruction pointer accordingly to jump to the start register or to continue the execution after the loop. This allows creating extremely efficient loops that require no compare or jump instructions.

As shown in figure 2.26, the tiles are distributed on a two dimensional array. Several communication schemes are available between tiles: streaming, cascade streaming, and window/buffers. Streaming communication is based on four 32-bit wide AXI4-Stream ports, two for outputs and two for inputs. Each stream has a first-in first-out (FIFO) buffer allowing storage of up to four words. A 32-bit word can be accessed every clock cycle, otherwise a 128-bit access every four clock cycles is possible. Several interconnects are available in the array, allowing deterministic routing to be defined during the design phase, together with optional FIFOs for buffering. Moreover a packet-switching mode is available, allowing to reach multiple destination ports from a single source port, with the drawback of making latency non-deterministic.

Two consecutive tiles in the same row are connected by a unidirectional cascade stream interface. This is intended for fast transfer of 384 bit accumulator register data per clock cycle. A two-deep FIFO is present on both input and output, allowing four values to be stored in-flight.

Finally, buffer access, sometimes referred in older documentation as window transfer, relies on the capability of neighboring tiles to share their memory banks. This is achieved via ping-pong buffers. Two memory regions with a size equal to the data transfer length are reserved. A lock memory region allows to coordinate access to the two buffers. When the source tile finishes writing data to memory, it releases the buffer to the destination tile. In the mean time the source tile can use the second buffer, if it was released by the destination tile. This scheme allows fast KiB sized transfers. Additionally, a DMA unit is present in the memory controller allowing transfer of data over AXI4-Stream between non-neighboring tiles. The DMA interface can also be employed to transfer AXI4-Stream data into memory while other computations are being performed. It is important to keep in mind that both direct buffer transfers and cascade stream are deeply linked to the placement of each program in the tile. This needs to be considered by the designer, as some topologies are not implementable.

In order to connect the AIE array with the rest of the computational units on Versal™, the bottom row is reserved for the interface tiles. Two different type of interface tile are available: NoC tiles and PL tiles. The former are intended to preform DMA over AXI4 memory-mapped, in this way accessing memory via the NoC, hence the name. The latter provides facilities to directly interface with AXI4-Stream ports within the FPGA. Both tiles have buffering capability and also take care of the clock domain crossing.

The NoC interfaces have the additional capability of exposing the internal data and instruction memory banks to external request. This allows programming, control, debugging, and profiling of the full array. Moreover, an application programming interface (API) is available allowing the setting of run-time parameters (RTPs) with a ping-pong buffer structure similar to the one used for tile-to-tile communication. This allows, for instance, a program running in the CPU to modify some parameters of the AIE programs at run-time. Additionally performance counters are available for run-time profiling.

The AIE array features a memory hierarchy similar to that outlined in section 2.3.2. In this hierarchy, the memory within each AIE tile serves as the L1 cache, offering low-latency access on the order of a few clock cycles. Data can also reside in the FPGA block RAM and be transferred to the array via the AXI4-Stream interface tiles. This introduces higher latency, making it comparable to an L2 cache. Finally, DDR memory is accessible through the NoC, incurring the highest latency for data access.

2.3.4.2. The VCK190 evaluation board

From the overview provided in the current section, the Versal™ family of devices has many interesting characteristic that can prove to be useful in the development of an low-latency real-time RL platform. The design of a custom PCB utilizing a Versal™ device is a complex engineering task. At the beginning of the work described in this thesis, only few commercially

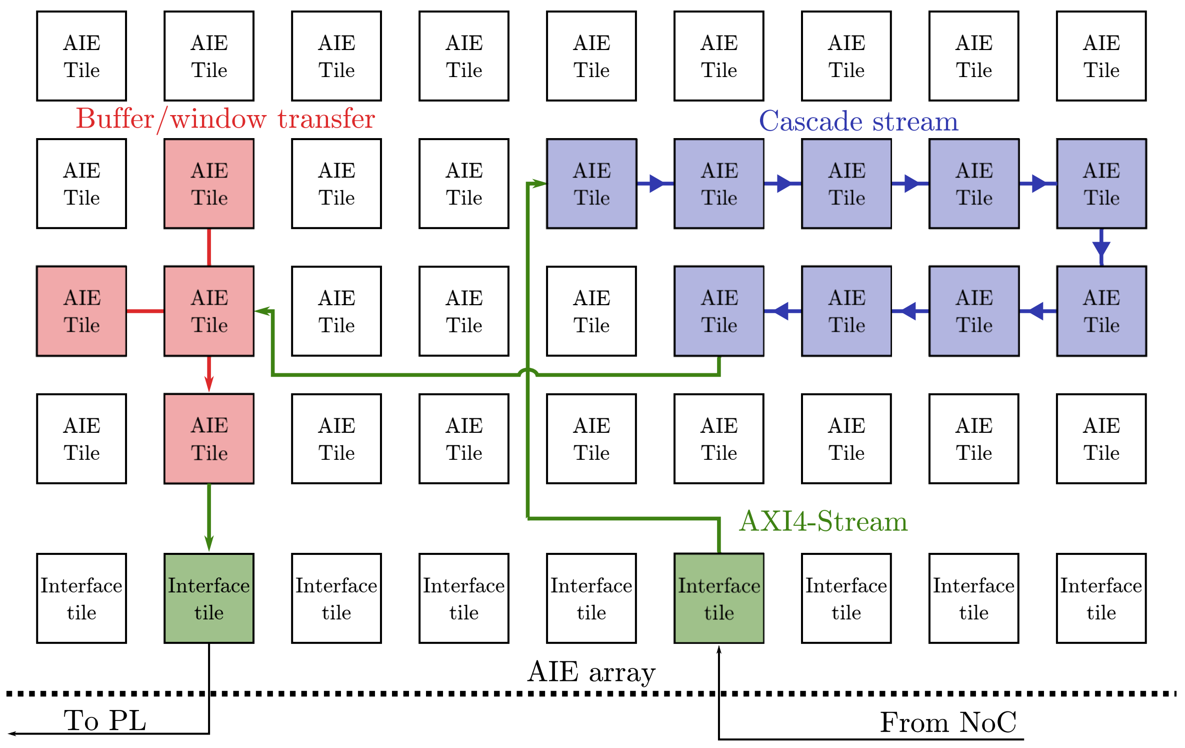

Figure 2.26.: Schematic showing the structure of the AIE array. Additionally, a potential graph is also shown, obtaining streaming data from the NoC, passing it between tiles with a cascade stream and window access, to then output it to the PL.

available Versal™ based system existed. Of these, only one, the VCK190 evaluation board [82], shown in figure 2.27, had the AIE array available. As such, this board was chosen for the initial evaluation of the architecture. Nonetheless, the high number of interfaces available allowed its usage for the development and testing of the RL systems described in the later chapters.

The VCK190 board is based on a AMD-Xilinx Versal™ XCVC1902 device. This device provides a dual-core Arm Cortex-A72 processor, paired with a dual-core Arm Cortex-R5F real-time processor, together with an AIE array with 400 tiles. The FPGA has 899840 LUTs and 1968 DSPs. The board features 8 GiB of DDR4 memory plus an additional 8 GiB of LPDDR4 memory connected to the memory controllers of the NoC. A USB type C connector allows to access the JTAG programming interface, together with three UART interfaces connected to the processors and PL. The high speed serial transceivers are connected to a PCIe Gen4 x8 interface, two HDMI, two SFP+, a QSFP, and the high-speed lanes of two FMC+ connectors. These last three connectors can be employed for long distance fiber-optic communication, that will be described in section 4.1.

Additionally, two Gigabit Ethernet interfaces are available and can be accessed from the PS. One expansion card is provided, allowing the connection of PMOD devices to the PL through one of the FMC+ connectors.

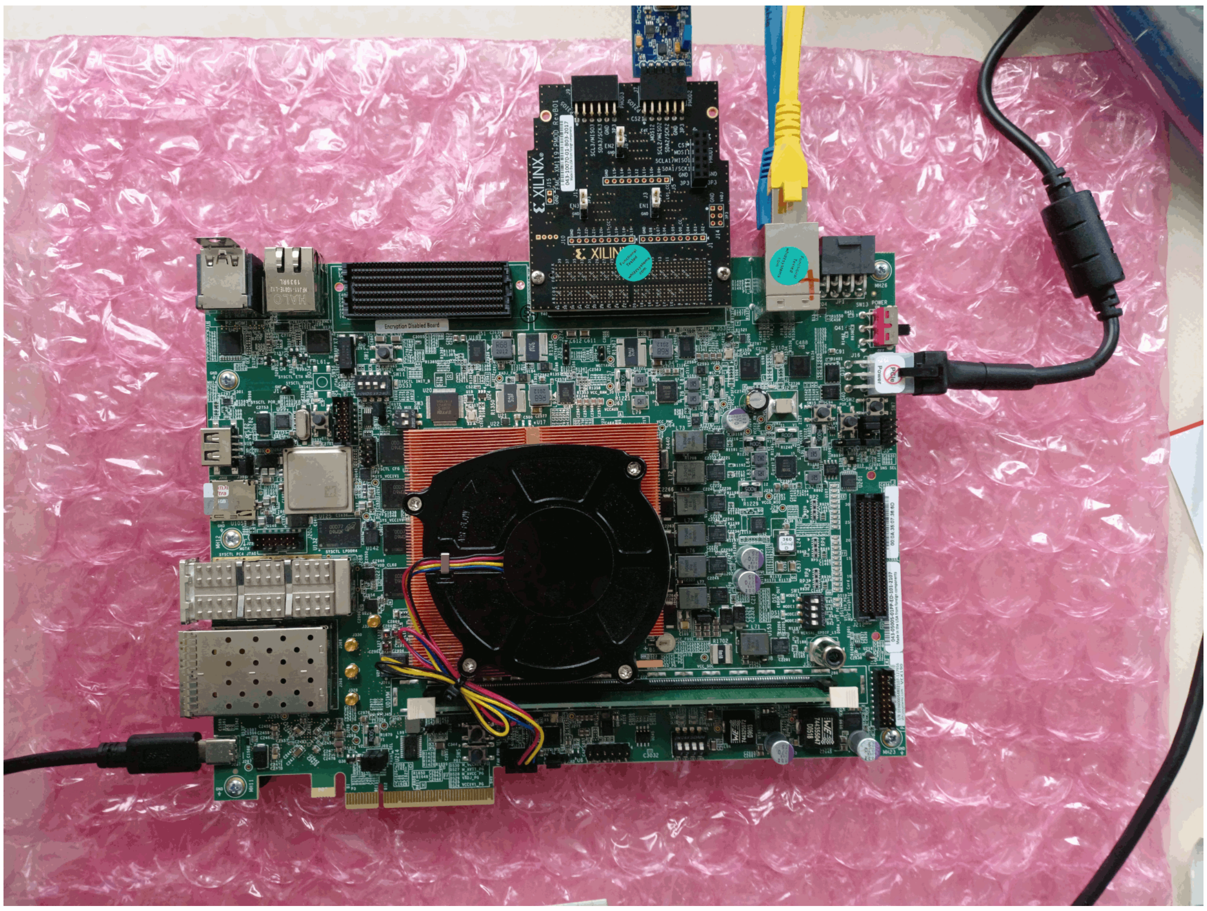

Figure 2.27.: Photograph of a VCK190 evaluation board. On the bottom left one can see the USB cable used to program and debug the device. On the top right two Ethernet and the power cables are visible. The black board that is plugged in the top side is the PMOD expansion card. On the left side, from bottom to top, the two SFP+ connectors and QSFP connector are present.

2.3.4.3. Programming Versal: Xilinx Extensible Platforms

To facilitate the development of solutions on FPGAs and heterogeneous platforms provided by AMD-Xilinx, the Vitis and Vivado software tools are offered. The development workflow is described in reference [83].

As shown in figure 2.28, the traditional flow consists of a fixed platform that is designed within Vivado. On top of this the embedded software, i.e. the one running on the processors, is developed with the Vitis IDE. Depending on the application, a customized Linux distribution could be produced with the Petalinux toolchain.

A more flexible design flow, the “Vitis Heterogeneous System Design Flow”, leverages the capability of HLS compilers to develop data processing IP cores in high-level languages. Specifically, the hardware platform can be defined in an extensible way, where all low-level input/output IP cores are instantiated and the memory-mapped interconnect structure is defined. These data producers/consumers are left disconnected, exposing their AXI4-Stream communication interface. This hardware design is then exported into a Xilinx support archive

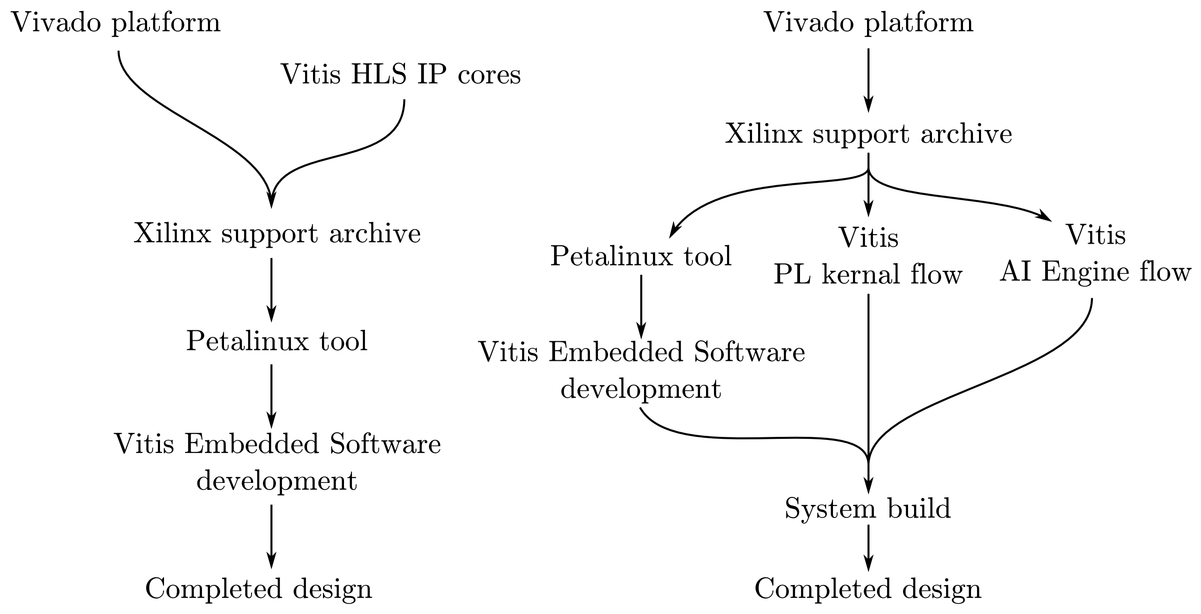

Figure 2.28.: Schematic highlighting the difference between the traditional design flow (left) and the heterogeneous system design flow (right).

(XSA) file. This file is then imported into Vitis. In this case, though, Vitis not only has the capability of developing the embedded software, but it can also produce data processing IP cores through HLS that are then connected to the ports defined in the XSA file. A custom Linux distribution can be created with the Petalinux tools.

In the specific case of Versal™, Vivado offers the ability of configuring the NoC topology, the memory controllers, the high-speed transceivers, and the connectivity of the PS. Moreover, the AIE array can be instantiated. The produced XSA file can then be imported in Vitis where the programming and validation of the processing cores can be performed. This way of partitioning the workflow has the benefit of abstracting the data processing part of the design from input/output components.

2.3.4.4. AIE programming

This section is based on [81, 84].

As discussed in section 2.3.4.1, the architecture of the AIE array is substantially different from the computing platforms described earlier in this chapter. This leads to a different programming philosophy. Specifically, the application architecture is based on an adaptive data flow (ADF) graph that is defined by the user as a C++ class using the `adf` library provided by AMD Xilinx. Each node of the graph represents a data processing *kernel*, whereas the edges directly map to a communication channel of an AIE tile. Additionally, facilities for packet switching, conditional execution, RTP definition, and HLS core integration are also provided. Each kernel can be seen as a separate program being executed on an AIE tile.

```cpp
class graph : public adf::graph {
public:
    // Graph inputs/outputs
    adf::port<adf::input>  graphInput;
    adf::port<adf::input>  rngStream;
    adf::port<adf::output> graphOutput;

    // Kernel definition
    adf::kernel circular_buffer;

    adf::kernel layer1_kernel;
    // Kernel parameters
    adf::port<adf::input> layer1_weights;
    adf::port<adf::input> layer1_biases;
    adf::port<adf::input> layer1_enable_ReLU;

    adf::kernel layer2_kernel;
    adf::port<adf::input> layer2_weights;
    adf::port<adf::input> layer2_biases;

    graph() {
        circular_buffer = adf::kernel::create(circular_buffer_cascade);
        adf::connect<adf::stream>(graphInput, circular_buffer.in[0]);   // Stream connection
        adf::source(circular_buffer) = "circular_buffer_cascade.cc";    // Kernel source code
        adf::runtime<adf::ratio>(circular_buffer) = 1.;

        layer1_kernel = adf::kernel::create(layer1);
        adf::connect<adf::cascade>(circular_buffer.out[0], layer1_kernel.in[0]);
        adf::source(layer1_kernel) = "layer1.cc";
        adf::runtime<adf::ratio>(layer1_kernel) = 1.;

        // Parameter connection
        adf::connect<adf::parameter>(layer1_weights    , adf::async(layer1_kernel.in[1]));
        adf::connect<adf::parameter>(layer1_biases     , adf::async(layer1_kernel.in[2]));
        adf::connect<adf::parameter>(layer1_enable_ReLU, adf::async(layer1_kernel.in[3]));

        layer2_kernel = adf::kernel::create(layer2);
        adf::connect<adf::cascade>(layer1_kernel.out[0], layer2_kernel.in[0]);
        adf::source(layer2_kernel) = "layer2.cc";
        adf::runtime<adf::ratio>(layer2_kernel) = 1.;
        adf::connect<adf::parameter>(layer2_weights    , adf::async(layer2_kernel.in[1]));
        adf::connect<adf::parameter>(layer2_biases     , adf::async(layer2_kernel.in[2]));

        adf::connect<adf::stream>(layer2_kernel.out[0], graphOutput);
    }
};
```

Listing 2.3: Example of an ADF graph with four kernels. Each kernel has several asynchronous parameters and are connected with either a stream or a cascade stream.

```
 1 void layer1(
 2      input_stream <accfloat>* restrict  inputStream,
 3      output_stream<accfloat>* restrict outputStream,
 4      const float (&weights)[16*8],
 5      const float (&biases) [16  ],
 6      float enable_ReLU
 7 ) {
 8 ...
 9 v8float output_buffer[2];
10 ...
11 if (enable_ReLU > 0.) {
12     output_buffer[0] = fpmax(
13         output_buffer[0],
14         null_v8float()
15     );
16     output_buffer[1] = fpmax(
17         output_buffer[1],
18         null_v8float()
19     );
20 }
21 ...
22 }
```

Listing 2.4: Example of an AIE kernel. The "…" mean some lines have been removed for readability.

An example of an ADF graph is shown in listing 2.3. This design implements a basic RL NN agent by chaining a circular buffer (lines 22 to 25) and two NN layers (lines 27 to 44). On lines 3 to 6, the definition of input and output ports of the graph are visible. Lines 11 to 15 show the declaration of a kernel with its parameter inputs. In the constructor of the class, the definition of a kernel is shown (lines 22 to 25). The main parts of this definition are referencing where the kernel source code is (line 24), how much execution time of a given tile this kernel is going to occupy (line 25), and the connectivity of the kernel (lines 23 and 32 to 35). It is worth noticing how the connect instance allows specifying which kind of connection is employed. In this case `adf::cascade` means a cascade stream is used, `adf::stream` means an AXI4-Stream is employed, while `adf::parameter` means the connection is a run-time parameter (RTP). For a parameter connection `adf::async` means the parameter can be defined asynchronously with the execution of a kernel. If this directive was not provided, the value of the parameter needs to be provided before each execution.

Listing 2.4 shows an example kernel, corresponding to one of the layers in listing 2.3. Each kernel is defined as a function. The parameters of the function directly map to the connections of the AIE kernel. At compile time, each parameter is interpreted as either an input or an output. They are then put in the `in` or `out` vectors in order of appearance, so that they can be referenced from the ADF graph definition. Lines 9, 12, and 16 show another key aspect of AIE kernel programming. Specifically, line 9 defines a `v8float` variable, that usually is directly mapped to a 256 bit vector register. Moreover, operations on vector quantities are performed with *intrinsics*: special functions that directly map to hardware functionality or assembly instruction. In lines 12 and 16, `fpmax` is one of these intrinsics, it performs the element-by-element comparison of two vectors, `output_buffer[ ]` and the all-zero vector `null_v8float()`, and returns the element-by-element maximum value. A more detailed description of AIE kernel programming concepts is carried out in section 4.3.

In order to verify that a given AIE design is working properly, and to guide the optimization of kernel source code, AMD Xilinx provides a simulation/emulation suite. Two kinds of simulators are available: the x86 simulator and the AIE emulator. The former compiles kernel code into x86 executable programs. Each kernel runs on a separate thread and communicates with other kernels through inter-process communication. This simulation is fast and can be used for a first test on the functional behavior of a given design. Moreover, it allows to detect stalls and possible deadlocks of the graph. AIE emulation, on the other hand, is a cycle accurate SystemC signal simulation that can capture hardware effects, as for example the presence of FIFOs in communication channels, memory access, and DMA capabilities. This emulation system allows timing information to be extracted, that is precious in the evaluation of latency and throughput of a design. Its main drawback is its computational intensity, requiring far more resources and time than the x86 simulation.

During development, the designer would first create a scalar version of the program, without the use of intrinsics. The logic functionality can then be tested with a x86 simulation, followed by an AIE simulation to verify that no hardware drawbacks are present. The program can then be vectorized and pipelined and its functionality is verified through x86 simulations, while its performance is benchmarked with AIE emulation. Once this process is completed the design can be tested in hardware, using the AIE tracing functionality to monitor performance.

2.3.5. Summary

Several systems allow performant and efficient processing of information. Among the most widespread, CPUs excel at processing a sequential set of instructions. For ML workloads, the highly parallel computing architecture of GPUs, and the even more specialized one of AI accelerators, offers a huge performance improvement, while restricting the generality of the operations that can be carried out. At the opposite end of the spectrum are FPGAs, that allow the definition of fully customizable digital logic, in this way defining custom data processing systems.

When one wants to address a control problem, and needs to interact with an environment within a finite amount of time, a real-time constraint is produced. Conventional computing systems do not usually allow for real-time programming. This was shown experimentally by transferring data to a CPU via PCIe.

The cutting-edge AMD/Xilinx Versal™ computing platform, combining CPUs, an FPGA, and AIE was described, together with its high-speed internal data transferring bus, the NoC. The structure of the AIE array, allowing for the acceleration of NN inference was carried out, together with a description of its programming workflow.

Due to its innate computational and timing capabilities, the AMD/Xilinx Versal™ family of devices was chosen as the computing platform for the rest of this work.

3. Platform development for real-time RL

The RL techniques described in section 2.2 are a powerful tool that can greatly aid the operation and commissioning of large-scale facilities. In some cases, though, the *sim2real* gap might be too wide to cover. Some environments, characterized by high-interaction rate, could in theory be viable for online learning, thus completely circumventing the issue of transfer from a simulation. These high-data rates, however, create real-time constraints that the controller needs to satisfy. This is a complex technological task that requires the development of new hardware platforms.

3.1. Real-time reinforcement learning

Part of this section have been published in [12].

As previously discussed in section 2.2.3, a defining aspect of RL is the interaction of a learning agent with an environment. Two classes of environments can be defined: *interruptible* and *non-interruptible.* The internal dynamics of an interruptible system allows its evolution to be stopped in between interactions with the agent. An example of these kind of systems is a simulation: the evolution of the environment can be delayed in order for the evaluation of a policy or for the training of an agent to complete. Non-interruptible environments, on the other hand, will inevitably undergo a change in their internal state independently of whether an action is applied to them or not. Most real-world environments fall in this second category. If an agent is applied to this category of environments, it will be constrained to produce an action within a fixed amount of time after it is provided the observations.

Based on the definitions of section 2.3.3, this constraint can be identified as a real time constraint. In general, a deep-RL actor-critic algorithm (section 2.2.4) will be organized as follows. The actor will interact with the environment for a number of steps. In some cases several actors in parallel are used to increase the interaction rate. These data are then used to train the actor itself, by using the value function estimations from the critic. Finally, the critic is also trained by using the rewards. The proper scheduling of these task, considering their relation, is fundamental for the deployment of these algorithms in non-interruptible environments.

Conventional devices as CPUs and GPUs struggle to reliably satisfy the timing constraints set by these kind of systems, as previously discussed in section 2.3.3.1. A possible solution consists in the employment of edge computing platforms such as FPGAs and AI accelerators. The deployment of RL algorithms on FPGAs has been already investigated in the literature. A brief outline of the currently available implementations of RL on FPGA is described below. A more comprehensive review can be found in reference [85].

There are two main classes of implementations, depending on the nature of the value function and policy. Several works store the action-value function in a tabular form, and based on this choose the action with the maximum expected cumulative reward for each state [86–90]. The main issue of this kind of system is that its resource usage grows exponentially with the size of the observation and action space, while being not directly applicable to environments with continuous action and observation spaces, leading to impractical hardware requirements for particle accelerator problems.

A different approach is deep-RL, where the value function and policies are approximated with a NN [91–97]. This approach scales better with increasing size of the observation and action vector, but tends to have a more complex training routine. These implementations lack the flexibility of choosing different training algorithms and changing the NN topology during deployment. Distinct algorithms and their respective hyperparameter selections might exhibit different performances based on the problem at hand. Therefore, the capability of flexibly choosing these components greatly reduces the deployment effort and expedites the development process. Moreover, the policy NN implementations shown in the works discussed in this section do not have real-time applications in mind, and as such the application to real-time particle accelerator controls would be limited. For these reasons, a general, turn-key deep-RL system requires the capability of performing such dynamic reconfiguration and naturally fulfilling real-time constraints.

A common technique used to deploy agents in real-time environments is the one presented in [98]. Here, a high-fidelity, high-throughput surrogate model is trained by using ML techniques. This is then used to train an agent offline on the simulation. The agent can then be deployed to an FPGA by means of libraries like hls4ml [99, 100]. These NN FPGA-based implementations have the great benefit of having a well defined latency, together with high data processing throughput. The downside of this scheme is that the *sim2real* gap is not addressed, and one needs to produce a surrogate model with sufficient fidelity to train on. For dynamical environments that depend on several external parameters, this is often not feasible.

A possible alternative is the one discussed in [74, 101]. This work describes a fully hardware accelerated scheme where an agent deployed in an FPGA gathers data in real-time with an environment. This information is then employed to train the agent, again with FPGA acceleration. This technique, albeit powerful, suffers from lack of flexibility, as changing the agent NN structure would require modifying completely the training logic. Additionally, this system requires rewriting RL algorithms into HDL or HLS programming languages, a cumbersome and bug prone procedure.

These drawbacks are addressed in [102]. Here a simulation-to-experiment toolchain has been developed and applied to the control of electric motors. The key idea is to deploy policies to a real-time capable device, in this case a real-time x86 processor, and gathering data so that training can be performed online. Albeit powerful, this system is limited in its applicability due to the chosen computing platform. The latency of the system is not suitable to be applied to the dynamics described in section 2.1.

This exposes the issue of the choice of computing platform when transferring agents to a real-time environment. Specifically, the devices described previously all suffer from their own drawbacks. CPUs, as shown in section 2.3.3.1, suffer from latency spikes that can hinder their real-time usability. Additionally, more real-time focused system tend to have reduced performance due to the need of determinism. Moreover, the total latency necessary before being able to process data is high compared to systems like FPGAs. A possible alternative would be the employment of GPUs and AI accelerators, albeit their real-time performance is still an open issue in the field. FPGAs, on the other hand, can operate in a way where the processing time is fully deterministic, but workloads requiring a high number of multiplications tends to be particularly challenging for these kind of devices. Tools like the aforementioned hls4ml mitigate this issue with two main techniques: quantization and pruning. The former consists in the representation of floating-point values with fixed-precision integers. The multiplications with these kind of number representation tend to be less resource intensive that full single-precision floating-point computations. The latter, pruning, consists in the removal of the neurons that do not contribute sufficiently to the output. Online training on quantized and pruned model is a challenging mathematical problem, that usually hinders the capability of directly retraining the model.

In conclusion, different possible computing devices can be used to deploy real-time RL algorithms. Most of these platforms, though, suffer from drawbacks that require careful planning before the actual application of the system.

3.2. Experience accumulator architecture

As discussed in the previous section, the application of RL algorithms to non-interruptible environments requires careful consideration of real-time constraints. The ex-novo implementation of these algorithms so that they can target hardware with better timing performance is a challenging task. As mentioned in section 2.2.5.2, the validation and debugging of these algorithms is a time consuming endeavor that requires vast experience in RL.

A fundamental observation one can do about RL algorithms using a NN-based policies is that training tends to require more complex computations than the policy inference. Additionally, an actor with stale parameters, if well designed, can still satisfy the timing constraints dictated by the environment. This means that the real-time requirements for training are less stringent than the one for inference.

This observation is employed by an *experience accumulator architecture*, of which a schematic is shown in figure 3.1. The key idea is to decouple the policy and training algorithm implementations. A real-time policy, π_θ^{RT} where θ are a set of parameters, is implemented in such a way that it is guaranteed to satisfy the required timing constraints. This policy is then let to interact with the environment, thus *accumulating* the observation-action-reward tuples (O, A, R) provided by the environment into an interaction buffer, representing the *experience* of the agent. The training of the real-time policy is periodically carried out by asynchronously transferring the buffer outside of the real-time inference domain and applying the training

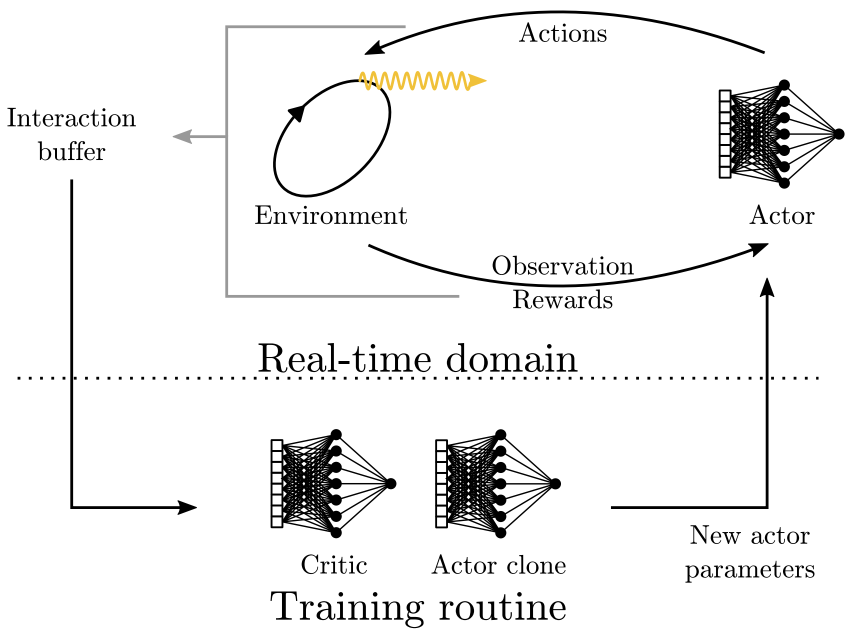

Figure 3.1.: Schematic of the experience accumulator architecture. The interactions with an environment of a real-time low-latency RL actor are stored in a buffer. Periodically, this data is transferred and employed to train a new version of the agent. The new agent coefficients are then transferred in the real-time domain. The procedure is repeated until an agent with satisfactory performance is obtained.

procedure dictated by the algorithm of choice on a non-real-time copy of the policy. A similar system has been implemented in [102] and applied to the real-time RL control of electric motors.

This kind of RL algorithm deployment scheme has several advantages. First of all the standard RL libraries can be used with only minimal modifications, with the possibility of easily changing the algorithm employed. Furthermore, a trained policy can be directly applied to the environment, with none of the issues that commonly arise from the *sim2real* gap. Additionally, the only part of the system with strong real-time constraints is the policy. An important detail in the development of policies for an experience accumulator is that simply satisfying a latency constraint is not sufficient for this kind of system to perform as intended. For a non-interruptible system, a variation in the time an action is produced might lead to slightly different evolution time of the environment. As such, when possible, it is advisable that the timing characteristics of the policy are completely independent on its parameters.

The necessity of training on a real world environment can be also considered a drawback, as in some cases the testing might be expensive and there is no guarantee that it will lead to a functional controller. Nonetheless, if the environment has high interaction rates, the data required to train an agent can be produced in a handful of seconds. In the case where a simulation is available, initial training can be performed in a simulated environment. The pre-trained agent can then be transferred into the real-time policy and a final training procedure can be carried out.

In the case of on-policy algorithms this deployment architecture exhibits an efficiency disadvantage. When the system is being trained, a real-time policy can still gather data. After

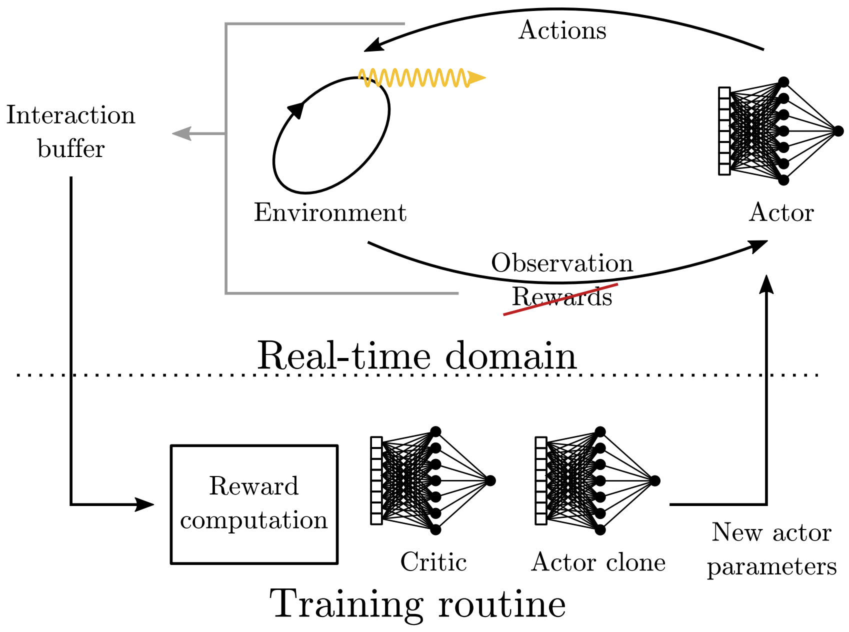

Figure 3.2.: The experience accumulator architecture shown in figure 3.1 is modified such that the computation of the reward is not carried out in the real-time domain anymore, but it is computed based on the observations at training time.

the training procedure is completed, though, these data cannot be used anymore as the policy has changed. In most cases, this does not lead to issues as a the data production rate of the environment is high enough that inefficiencies can be tolerated. A possible solution would be to employ off-policy algorithms and include the data gathered during training into the dataset that is employed in the next iteration of the algorithm.

3.3. Training-time reward definition

As previously discussed in section 2.2.3, the MDP structure used to define a RL problem requires a reward to be given to the agent after each interaction with the environment. In general, this would require the reward to be computed in real-time. The process of finding the correct reward to obtain the intended effect on the environment, called *reward engineering*, is considered a fundamental part of deploying RL algorithms. The reward signal is in fact the main way to communicate with the agent whether an emergent behavior is desired or not. Thus, having the reward function as a real-time subsystem would lead to a reduction of flexibility, as modifying it would require careful verification that the timing constraints are not violated.

When an experience accumulator is employed, a crucial simplification can be performed. All of the modern deep-RL algorithms described in section 2.2.4 do not generally use the reward function during inference, albeit in some cases it can be provided as part of the observation vector. Very often, it is only required during training. In the case where the reward at step t can be computed as

$$r_t = R(\{\vec{o}_i\}, \{\vec{a}_i\}, t), \tag{3.1}$$

where $\{\vec{o}_i\}$ and $\{\vec{a}_i\}$ are the sets of observation and action vectors obtained in the current episode, the reward can be computed at training time [12]. Specifically, the evaluation of R only requires data that is already being stored in the interaction buffer. If the RL problem is defined in such a way that the rewards used are as equation (3.1), then the computation of the reward can be entirely performed just before training.

This approximation allows the *training-time reward definition*, in this way simplifying the design by removing one of the real-time components in an experience accumulator system. In the KINGFISHER system, that will be described in the next chapter, the training procedure is carried out by a Python script that employs the Stable Baselines3 library. As such, the reward function can be written in a high-level language such as Python. The reward engineering can thus be carried out by a user without specific real-time programming training.

3.4. High-level policy structure

As discussed in the previous section, the design of a policy with deterministic timing is a fundamental aspect of an experience accumulator system. In order to employ one of the standard RL libraries such as Stable-baselines 3, one needs to match the agent implemented in the real-time domain, with the one employed in the training domain. In the default PPO policy implementation of the library, a feed-forward NN directly transforms the observation vector into an action vector. In order to drive the exploration of the algorithm, gaussian noise is added to each element of the vector. The variance of the distribution applied to each element is different, and is updated by the training procedure.

If one wants to implement such a policy on a hardware platform such as the AMD-Xilinx Versal™, one needs to implement:

- a feed-forward NN, requiring:
  - matrix-vector multiplications (section 4.3.2),
  - non-linear activation functions (sections 3.4.2 and 4.3.3),
- a random number generator stage to produce Gaussian-distributed samples (section 4.3.6).

Furthermore, the system should be optimized to process time series data, so that it can operate with arbitrary inputs. The construction of a generalized observation vector will be discussed in the next section.

3.4.1. Observation vector

As discussed in section 2.2.3, the agent in a RL problem is usually provided the state of the environment it needs to control, or some observation vector linked to it. The system under development should be designed to be as general as possible. The environment under control can be often considered to produce time-series data. In such case, it is important to preserve this information in the structure of the observation vector provided to the agent. For a system that generates samples $x_t \in \mathbb{R}$ at each time t with a sample period of τ_s, a method for producing an observation vector is to use a shift register that stores the most recent n samples. Thus, an observation vector can be defined as

$$\vec{o}_t \overset{\text{def}}{=} \begin{pmatrix} x_t \\ x_{t-1} \\ x_{t-2} \\ \vdots \\ x_{t-n+1} \end{pmatrix} \in \mathbb{R}^n. \tag{3.2}$$

If this vector is fed into a feed-forward NN agent, it will contain information only for a time corresponding to the previous n samples that are provided, limiting the awareness time of the NN to $t_a = n\tau_s$. In the case where a KARA bunch is considered, τ_s is dictated by the revolution frequency. This corresponds to $\tau_s \approx 370\,\text{ns}$, that for $n = 64$ leads to $t_a \approx 23.7\,\mu\text{s}$. This means that, if the sampling rate and NN input layer size are fixed by hardware constraints, the timescale the agent is aware of is also fixed. One possibility of circumventing this phenomenon is to reduce the sampling rate of the input signal in a process known as decimation, creating a signal $\tilde{x}_t$ with sample period $\tau_d = m\tau_s$, when m is the decimation factor. From the Nyquist-Shannon sampling theorem [103, 104], in order to completely determine a signal with no higher frequencies than f_{max} it is necessary to have a sampling frequency of $f_s = 2f_{\text{max}}$. The maximum frequency uniquely represented at a given sampling rate as Nyquist frequency, $f_{\text{nyq}} = f_s/2$. Thus, in order for the decimated signal to be unique, a filtering stage is necessary to limit the frequency components that would fall above the Nyquist frequency. Specifically, the new Nyquist frequency after downsampling is $f_{\text{nyq}} = f_s/(2m)$. In this way, the new awareness time is extended to $t_a = nm\tau_s$, at the price of losing higher frequency information.

In section chapter 6, it will be further discussed how, in cases where a precise action sampling rate is required by the environment, an additional interpolation stage is necessary to match the data production rate of an agent to the environment. Interpolation can be regarded as the opposite of decimation, the sampling period is reduced by a factor $\tau_i = \tau_s/h$, where h is the interpolation factor. Its working principle is identical to the decimation, but works the other way around: every sample from the input signal is padded with $h-1$ zeroes. In order to remove spurious high-frequency component a filtering stage is applied with a cutoff frequency of $f_{\text{nyq}} = f_i/(2h)$. The shift register design works identically, only that it always needs to evaluate the finite impulse response (FIR) filter and sometimes it requires zeroes to be fed in.

More complex RL problems will likely require different NN topologies, as the GRU described in section 2.2.2 and implemented in [105]. Additionally, the feed-forward NN approach could still hold, but with higher level observables as described in [106]. The implementation challenges of some of these observables, specifically a low-latency FFT are described in appendix A.

3.4.2. Activation functions

Several activation functions are widely used in the development of NN models. tanh and sigmoid, defined in equations (2.25) and (2.26), are not simple to implement in FPGA or AIE based systems. In order to compute them based on their definitions, both functions require efficient and accurate computation of natural exponentials e^x, that is usually resource expensive. In essence, a novel implementation needs to be devised in order to target the AIE. This was carried out in collaboration with Michail Sapkas and Chenran Xu, and is described in his master thesis [105].

The approach relies on the nature of the activation functions. Their exact numerical values are likely not fundamental for the successful training of a model. As such, one could devise non-linear functions that are similar to the hyperbolic tangent and sigmoid, but that are far easier to implement. If the plot of the hyperbolic tangent is observed in figure 3.3, one can make three observations:

1. the function is odd, meaning that $\tanh(-x) = -\tanh(x)$;
2. the function reaches a plateau at approximately $x = \pm 2$;
3. the plateau value is ± 1.

As a first approximation attempt, a polynomial of the form

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \tag{3.3}$$

can be utilized. Constraint 1 forces all the coefficients corresponding to an even power to be zero. Additionally, constraint 2 leads to the condition that

$$\left.\frac{dp}{dx}\right|_{x=\pm 2} = 0, \tag{3.4}$$

while constraint 3 forces

$$p(\pm 2) = \pm 1. \tag{3.5}$$

If the degree of the polynomial is selected, equations (3.3) to (3.5) can be solved to obtain the coefficients. In order to reduce the computational cost to the minimum, a third degree polynomial was chosen. This is the lowest degree polynomial that can be selected that has non-linear properties in $[-2, 2]$ and satisfies condition 1. The resulting approximated hyperbolic tangent is

$$\tanh_{\text{app}}(x) = \begin{cases} -1, & \text{if } x < -2 \\ -\frac{1}{16}x^3 + \frac{12}{16}x, & \text{if } -2 \leq x \leq 2 \ . \\ 1, & \text{if } x > 2 \end{cases} \tag{3.6}$$

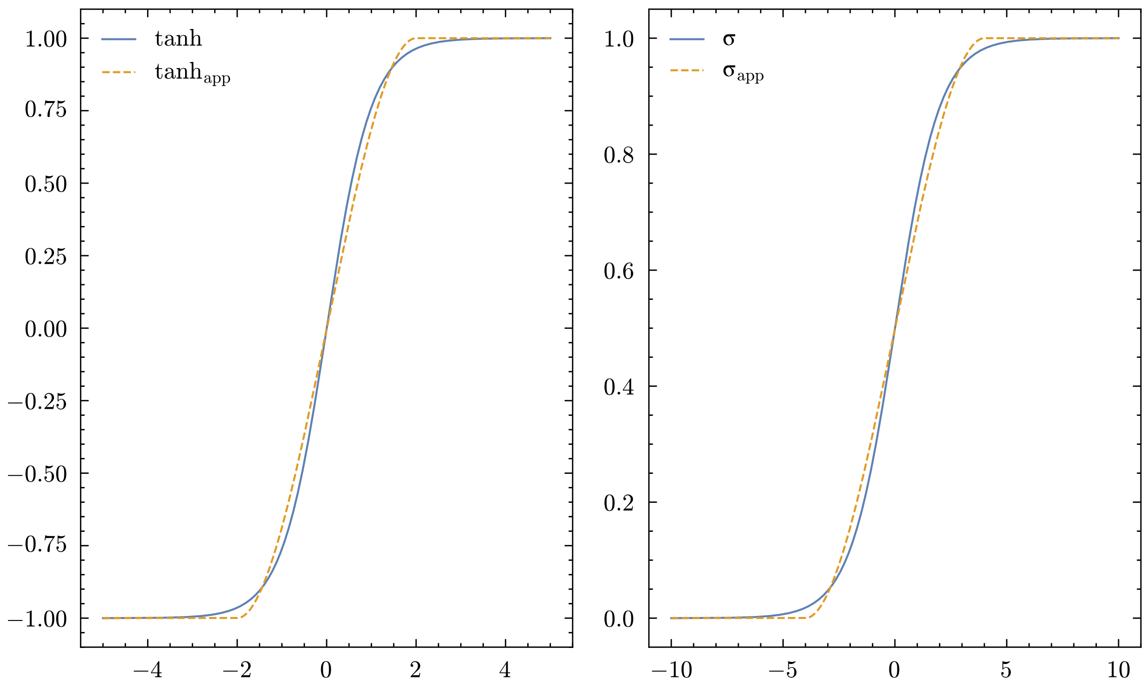

Figure 3.3.: Comparison of approximated and non-approximated activation functions for tanh (left-side) and sigmoid σ (right-side).

An identical procedure can be applied to the sigmoid resulting in

$$\sigma_{\text{app}}(x) = \begin{cases} 0, & \text{if } x < -4 \\ -\frac{1}{256}x^3 + \frac{3}{16}x + \frac{1}{2}, & \text{if } -4 \leq x \leq 4 \,. \\ 1, & \text{if } x > 4 \end{cases} \tag{3.7}$$

The activation functions and their approximations can be compared in figure 3.3, showing clear qualitative agreement. In order for training to be possible with these functions a Pytorch [27] `torch.autograd.Function` was defined. The expressions described in equations (3.6) and (3.7) was implemented in the `forward` method, while the analytical derivative was implemented in the `backward` method. This class was then wrapped into a `torch.nn.Module`, allowing its use when training a model. Finally, training of a feed-forward NN model utilizing these functions was performed and compared to the one with the non-approximated functions. The model was trained to classify the hand-written digits of the MNIST dataset [107]. The training loss is shown in figure 3.4. In both cases, the approximated and non-approximated activation functions lead to almost identical performance.

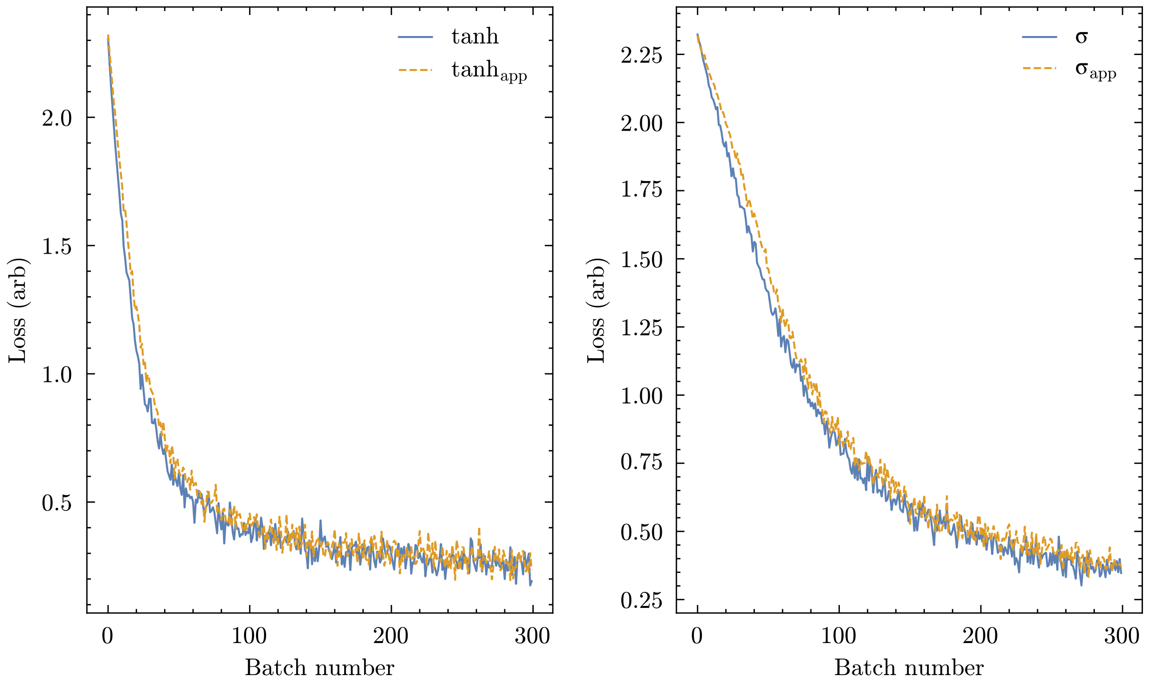

Figure 3.4.: Comparison of training loss for the MNIST hand-written digit classification of a feed-forward with activation functions $\tanh_{\text{app}}$ and tanh, and σ_{app} and σ.

3.5. Summary

This chapter described how some environments are non-interruptible, meaning the policy needs to be evaluated and provide an action within a specific time frame. The development of RL algorithms targeting these conditions is a challenging engineering problem.

The design can be greatly simplified by using the experience accumulator architecture. In this kind of system, the only real-time component are the policies, thus allowing the use of the standard algorithm implementation for training. The training-rime reward definition allows doing reward engineering online during the experiment, by computing the reward function from the observations, in this way removing the requirement of a real-time reward implementation.

A scheme to construct an observation vector that can be applied to a wide variety of time-series data was devised, together with functioning approximations to non-linear activation functions that are more easily implementable in edge computing devices such as the AIE tiles of Versal™.

4. The KINGFISHER hardware platform

As discussed in the previous section, the experience accumulator architecture is a good candidate for deploying real-time capable RL algorithms in the real world. Given the generality of the environment definition, and the capability of performing the training time reward definition, a correctly implemented system can be applied to several different environments with only minor modifications.

In section 2.3.4.3 the “Vitis Heterogeneous System Design Flow” was described. One of its crucial aspects is the decoupling of the system design (comprising I/O, memory topology, etc…) from the data processing part. Additionally, the integration of HLS methodologies allows the definition of performant data management IP cores in high-level languages.

The KINGFISHER platform, shown in figure 4.1, was conceived [13][1] with the goal of offering robust IP cores for interfacing with a large-scale facilities such as KARA, together with the required instrumentation and monitoring to produce training data that is fundamental in order to reliably implement an experience accumulator. Finally, a Petalinux-based OS can be employed for the remote control of the platform. Through the use of a virtualization platform such as Docker, it is possible to deliver software packaged as containers. This allows the validation of training algorithms on a conventional computer, reducing the requirements for an algorithms design platform.

KINGFISHER is packaged as an XSA targeting the VCK190 Versal evaluation board. This board was chosen due to the availability of an AIE array providing greater performance in the development of NNs. A comparison with previous generation devices will be carried out in section 4.4.1. It is important to stress how the platform can not only be used for RL experiments, but the controller instrumentation and connectivity allows also experiments with more classical controllers (sections 5.3 and 6.3).

The fundamental goal of this kind of framework is to create a system that is as general as possible, allowing easier integration in several facilities, while reducing the integration effort as much as possible.

The KINGFISHER platform is comprised of several input/output IP cores that expose external functionality through AXI4 and AXI4-Stream interfaces. Additionally, the software stack allows integration within the KARA control system and the deployment of RL algorithms. A detailed description of these components will be carried out in the following sections. First of all the integration with the KAPTURE diagnostic system described in section 2.1.6 will

[1] I was asked many times what the KINGFISHER acronym stands for. The answer is simple. It is not an acronym. I was planning to find one because I really liked the name, but I never did.

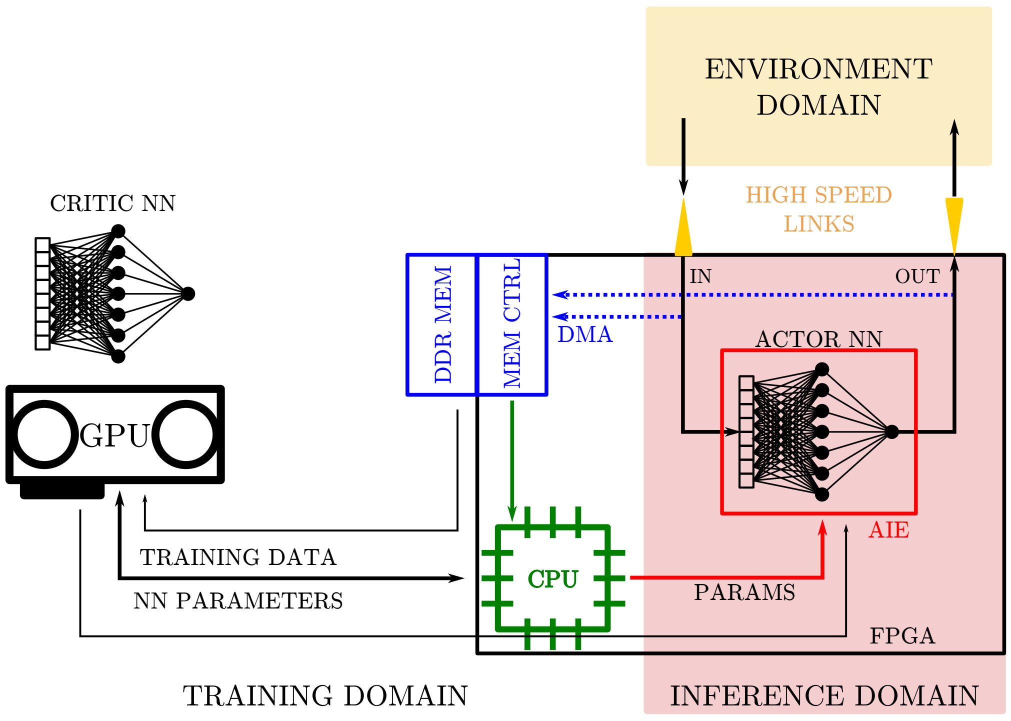

Figure 4.1.: The KINGFISHER platform provides the infrastructure to deploy RL algorithms with the experience accumulator architecture. It allows monitoring the RL actor in the real-time domain and transfer this data to memory. The CPU can then pass this data to an external platform for training, and apply the new parameters to the NN.

be discussed in section 4.1. The instrumentation of agents and the way data is managed within the platform will be discussed in section 4.2. NNs design on the AIE array will be examined in section 4.3. The slow-control of these building blocks and the integration with the KARA control system will be explored in section 4.5. Finally, the training algorithms and their adaptation to the experience accumulator architecture will be described in section 4.6.

This chapter will not examine the interfaces to apply actions to KARA, as they are strongly linked to the problem that is being targeted. The implementation of a digital-to-analog converter (DAC) interface in order to control the horizontal betatron oscillations (HBO) will be discussed in section 5.2. Interfacing with the KARA low-level radiofrequency (LLRF) system will be explored in section 6.1.3.

4.1. Integration of KAPTURE

As discussed in section 2.1.6, KARA is instrumented with KAPTURE systems that allow bunch-by-bunch and turn-by-turn acquisition of the peaked signals produced by beam diagnostic detectors at electron synchrotrons. The system utilizes a PCIe-based DMA implemented on an FPGA to transfer data into the memory of a computer. Given its versatility and the possibility of connecting it to a variety of detectors, it represents and ideal candidate as the input for the KINGFISHER system.

There are three main requirements for the integration of this kind of device: low-latency, possibly at the sub-microsecond level, long distance connectivity, in the order of $\sim$ 100 m, and capability of continuous streaming. The latter is fundamental for long-term data acquisitions and to ensure bounded latency in the data transfer. Several possible mechanisms can be devised to interface KAPTURE with the VCK190:

1. connection of a KAPTURE board to the FMC+ connector of the VCK190;
2. connection of a KAPTURE system to the VCK190 via PCIe;
3. streaming data via Ethernet using the computer of a KAPTURE system;
4. streaming data through the FPGA of a KAPTURE system.

Option 1 is not viable: the FMC+ standard provides several differential lines, denoted as HA and HB, that are optional. KAPTURE requires these lines to be fully populated in order to function properly. On the VCK190, none of the two FMC+ connectors is fully populated, as such the direct connection of a KAPTURE board would lead to some mandatory signals to not be properly connected.

Option 2 is potentially viable, but cumbersome to implement. Direct PCIe communication among peripherals connected to the same bus has been demonstrated [108]. Furthermore, KAPTURE requires a software stack to control and configure the system, that is usually executed on a conventional computer that is connected to KAPTURE via PCIe. This software stack would need to be re-implemented and validated. Additionally, the latency of this kind of communication scheme is greater than one microsecond [108], strongly impacting the latency budget. Finally, the range of a PCIe connection is limited to a few tens of centimeters.

Option 3 solves the issue of porting the software stack, as the control computer will still be used. Data would need to be continuously transferred by the CPU from memory to an external interface as Ethernet. Such a scheme can potentially allow continuous streaming of data over long distances, but suffers from latency variations that span orders of magnitudes, as already discussed in section 2.3.3.1.

Option 4 solves all the aforementioned problems. The control computer can still be employed for configuration and monitoring. The KAPTURE FPGA firmware would need to be modified in order for an additional interface to be added. This modification can be implemented as a data path that is parallel to the one used for PCIe operation, in this way allowing contemporary streaming and PCIe DMA transfers.

In conclusion, option 4 was considered the safest option, with higher expected performance, while still fulfilling the distance and continuous streaming requirements. In order for it to be carried out, the conventional HighFlex board employed in the KAPTURE system was exchanged with the newer version, HighFlex2 [74]. This board is based on an AMD-Xilinx Zynq Ultrascale+ XCZU11EG device and provides a Samtec Firefly interface [109], allowing high-speed copper or fiber optic based interfaces. Two separate approaches will be described: a 1 Gbps Ethernet interface through the PS of the Zynq and another based on a 40 Gbps Aurora interface.

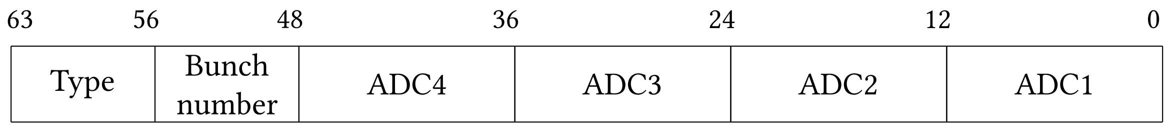

Figure 4.2.: Structure of the 64-bit long KAPTURE quanta.

The KAPTURE data stream was structured into 64-bit quanta as shown in figure 4.2. The top byte of a quantum is used to denote the type of the data, specifically 1 denotes KAPTURE data. The byte before the top denotes the bunch number the data that follow correspond to. Finally, the last six bytes are the four 12-bit long ADC samples for that bunch. For this data structure and an RF frequency of $\approx$ 500 MHz, this corresponds to a data rate of 32 Gbps.

4.1.1. Ethernet based connection

Ethernet is a protocol that allows full-duplex transport of data among computers in a network. Today, it serves as the underlying layer of several other protocols and is widely adopted in most networking infrastructures. Several different mediums are supported as a physical layer, comprising twisted pair, fiber optic links, and wireless. Each computer has a unique 48 bit identifier called media access control (MAC)[2] address.

The stream of data is divided in frames, or packets, ranging from 64 to 1522 bytes long. A frame is composed by a header, a payload, and a sequence used for error correction. The header contains the source and destination MAC addresses, and two byte-long field, encoding either the size of the payload (values $\leq$ 1500) or a protocol identifier (values $\geq$ 1536). The payload ranges from a minimum of 42 bytes up to a maximum of 1500 bytes.

Albeit being originally intended as a shared-medium protocol, where an interface is constantly listening for data transfer addressed to it, modern network infrastructure is commonly based on point-to-point connection to a switch. The switch has several ports, each one connected to a single device, and routes packets only to the port connected with a device having the corresponding MAC address. In large-scale systems, this has the benefit of reducing traffic congestion, albeit with the possibility that some packets are either dropped or need to wait in a buffer before being routed.

The ARM processor in the PS of an AMD-Xilinx Zynq Ultrascale+ or Versal device has a gigabit Ethernet MAC (GEM), a device capable of interfacing with the physical layer of the protocol [110, 111], also denoted as PHY, via the reduced gigabit media-independent interface (RGMII) protocol. On the VCK190 the physical layer is an RJ45 connector for twisted pair cables with a maximum length of 100 m. The MAC disposes of a DMA, allowing it to load/store packets to/from memory. The processor can then process/produce packets by interacting with memory. Configuration registers allow automatic filtering of packets based on their address. The MAC automatically adds the preamble, checksum, and postamble to each packet.

[2] In this subsection MAC stands for media access control and not multiply and accumulate.

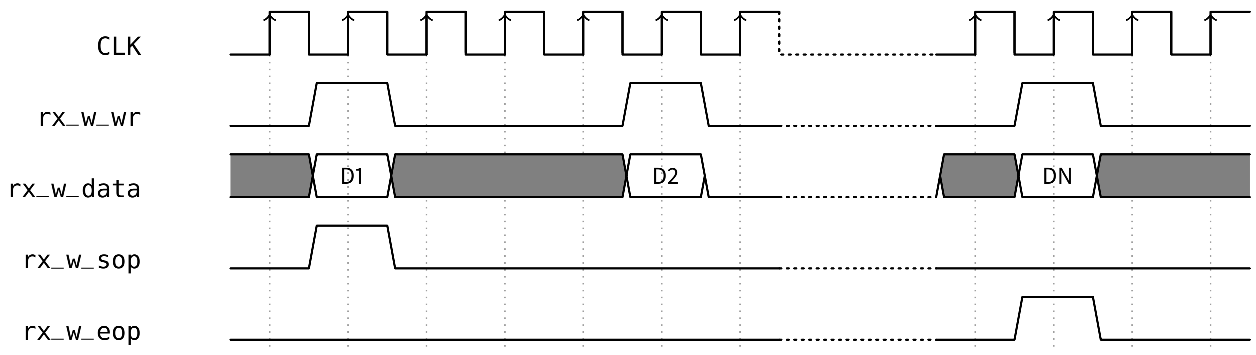

Figure 4.3.: Timing diagram of the receive GEM FIFO interface to the PL on Zynq Ultrascale+ and Versal devices.

A custom data path is available allowing low-latency high-throughput communication by routing the raw packets to/from the FPGA. The interface is identical in the AMD-Xilinx Zynq Ultrascale+ and Versal families. An IP core was developed in Verilog in order to convert the FIFO-like interface used by the GEM into an AXI4-Stream interface that can be easily connected to the data processing logic in the "Vitis Heterogeneous System Design Flow".

The receive channel works as shown in figure 4.3. The 8-bit wide `rw_w_data` contains a valid packet byte when `rx_w_wr` is asserted. The first and last bytes of the packet are signaled by the assertion of `rx_w_sop` and `rx_w_eop`, respectively. This interface can be remapped to AXI4-Stream, by treating `rx_w_wr` as TVALID. The TLAST signal is used to signal the end of a packet. Further IP blocks were implemented to perform filtering based on the address or to remove the header.

The design of the transmit channel controller, needs more care. The IP core can indicate to the GEM that data is available by asserting the `tx_r_data_rdy` signal, as shown in figure 4.4. The GEM will then assert the `tx_r_rd` signal to start a transmission. Starting from the following clock cycle, 8-bit of data can be placed on the `tx_r_data` lines and validated with the `tx_r_valid`. This one cycle synchronization was achieved with a first-word fall-through FIFO. As for the receive channel, `tx_r_sop` and `tx_r_eop` denote the boundary of the packet. The IP core accepts parameter signals setting up the source and destination MAC addresses and the size of the payload. It additionally takes care of inserting an header when needed. The core can be configured to work in a single-data mode, zero-padding the payload to reach the minimum permitted size for an Ethernet packed in case the latency due to packing several data together becomes too high.

In order for this scheme to work, the Linux kernel driver for the specific GEM needs to be disabled, so that the configuration registers can be properly set up for this mode. Given only an handful of registers needs to be set up, a simple bash script was used instead of writing a custom kernel driver.

An additional benefit of this system is its ease of debugging. Most of modern computers have their own gigabit Ethernet interface. As such, they can directly interact with the hardware by utilizing commonly available software. For instance, the data sent over the Ethernet interface

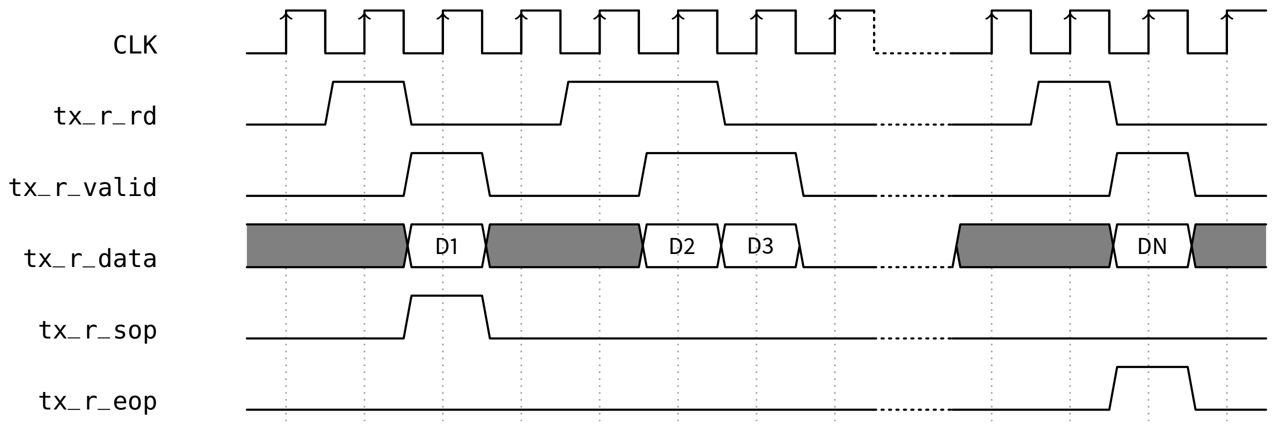

Figure 4.4.: Timing diagram of the transmit GEM FIFO interface to the PL on Zynq Ultrascale+ and Versal devices.

can be easily visualized with a packet sniffer. Moreover, a custom software using raw sockets was developed, allowing a computer to emulate the data produced by a KAPTURE system for testing purposes.

Based on the data rate described in the previous section, a gigabit interface is not sufficient to continuously stream the entirety of data produced by KAPTURE. A filtering stage was thus added that would drop unwanted bunches in order to reduce the data rate. The interface can support continuous streaming of up to five bunches [13].

4.1.2. Aurora based connection

Aurora is a protocol allowing point-to-point transfer of data. In this work the version using the 64b/66b encoding was employed. The data stream can be moved over one or more high-speed serial links [112]. The protocol allows simplex and duplex connection capabilities. The channel is used both for data and for frame specifiers, if used.

A serial link is a way of transferring data one bit at a time. If the data contains enough level transitions the clock can be recovered from the signal. For an arbitrary set of data bits, potentially long sequences of ones or zeros can be present, creating periods were the clock is not recoverable. The data is thus encoded, by adding bits to the transmission, to include a minimum number of level transitions that would allow for the clock to be recovered. The Aurora IP core used by KINGFISHER employs the 64b/66b encoding. This encoding adds a two bits header to each 64-bit blocks. A `00` or `11` header denotes an error, while `01` and `10` denote respectively a data and control block. In this way a at least one transition every 64-bits is guaranteed.

High-speed lanes are currently based on differential lines that are AC-coupled to the transmitter and receivers. The lines thus behave as a low-pass filters. In case the signal on average stays more in one state rather than the other, the baseline of the signal can wander, leading to decoding errors. This issue is solved by the addition of a scrambler that makes the data resemble a series of random bits. Before commencing transmission of data, each high-speed link is synchronized by sending specific control blocks. After all links and their clocks are synchronized, the skew of different lanes is adjusted and the channels are bonded together.

An IP core implementing this protocol is provided by AMD-Xilinx [113] targeting both the Zynq Ultrascale+ and Versal architectures. The core exposes AXI4-Stream transmit and receive interfaces and automatically performs the necessary lane and channel initialization procedures. Two modes are available: framing and streaming. The former adds a TLAST signal to the receive and transmit interface, allowing to set the boundary of the frame. It is worth noticing that frame boundary messages needs to be transmitted on the same link used for data, and as such can lead to a performance degradation. The latter, on the other hand, does not transfer any framing information. Given the quanta-based KAPTURE encoding described in the previous section, no framing is necessary, thus streaming mode was selected.

In order for the IP core to start up properly a specific reset procedure needs to be performed. This was carried out with a custom IP core developed in Verilog. The `reset_pb` signal is held high for 256 clock cycles, then `pma_init` is asserted for at least one second, so the remote interface detects that the system was disconnected. Then `reset_pb` and `pma_init` can be deasserted. After this the core provides `user_clk`, a clock signal derived from the transmit interface that is synchronous to the AXI4-Stream interface.

Point-to-point 100 Gbps data transfer using four 25 Gbps links over copper cables were tested and proved reliable. Their main limiting factor is that the connection distance was limited to $\approx$ 20 cm. This could be overcome by using optical transceivers and using fiber optic as a medium (figure 4.5). The use of Firefly connectors on HighFlex2 limits the single lane transfer speed to 14 Gbps. On the Versal side a QSFP+ 850 nm transceiver was employed, allowing a four full-duplex link operating at 10 Gbps, leading to an aggregate speed of 40 Gbps. The system latency was tested by connecting the VCK190 device to a loopback with a 1 m long OM3 fiber optic cable and monitoring the delay of a counter. The resulting latency is 370 ns.

4.1.3. Comparison

As discussed, both the gigabit Ethernet and the Aurora based approaches are capable of continuos data streaming with low-latency. The Ethernet approach has the advantage of straightforward integration with conventional computers for testing and validation. With the current implementation, though, there is no path for scaling the data rate in case more throughput is required. Aurora, on the other hand, allows easy scalability to higher throughput platforms. Additionally, it employs the same hardware that is required for the implementation of higher speed fiber-based Ethernet standards, as for example 40 Gb and 100 Gb Ethernet. As such, if the switching capabilities of the Ethernet protocol are required, or communication with high-performance computing clusters is necessary, higher throughput Ethernet standards can be implemented in the FPGA.

In conclusion, Aurora was chosen as the communication standard between the diagnostic equipment and KINGFISHER, with the possibility to utilize 40 Gb or 100 Gb Ethernet on the same hardware if this becomes necessary.

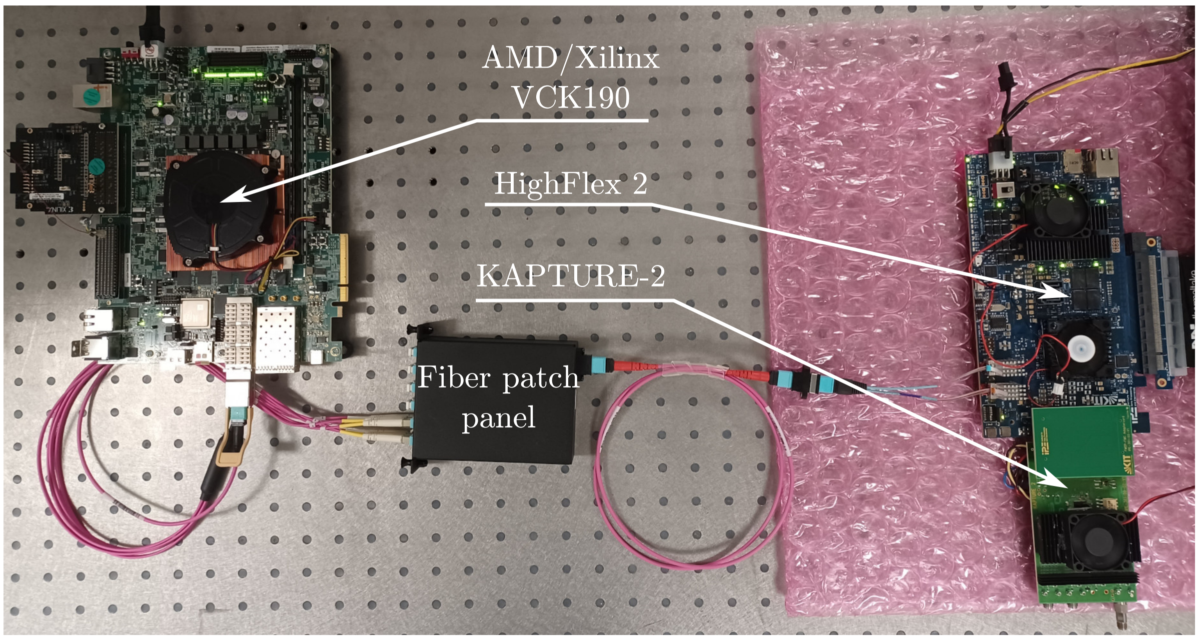

Figure 4.5.: Photograph of KAPTURE and Versal connected via optical link. A patch panel was necessary to connect each link to the corresponding fiber.

4.2. Data management system

As previously discussed, in order for an experience accumulator system to work, a way of monitoring the interactions with the environment is necessary. This can be achieved with a DMA capable of monitoring AXI4-Stream interfaces. AMD-Xilinx provides an AXI DMA with this exact capability [114]. The effect of this IP core on the real-time data paths should be considered with care. Directly connecting the DMA to the data processing path could lead to stalls when the memory writes stall, a common occurrence if the memory controllers are busy from some other master. This issue can be mitigated by proper design of the NoC topology, but it can never be totally eliminated. As such, a fully isolated AXI4-Stream monitor that cannot apply back-pressure to the data-processing chain IP, called *spy feed-through*, was designed. The core can be remotely controlled via an AXI4-Lite interface, that exposes registers for setting the number of bytes to read from the interface and to start the transfer. The core then monitors TREADY and TVALID and transfers TDATA to a FIFO when a handshake is detected (section 2.3.2.1). This data is then transferred to the stream to memory mapped (S2MM) channel of the AXI DMA IP mentioned above.

A subtle implementation detail needs to be considered. In order for the AXI DMA IP to properly complete the memory transfer, a TLAST signal needs to be provided on the last AXI4-Stream transfer. Additionally, if TLAST is asserted before the number of bytes requested is transferred by the DMA, the transfer will be terminated too. The spy feed-through IP needs to consider this aspect in its design, otherwise part of the data is not flushed to memory and remains into the AXI DMA.

```
/ {
    reserved-memory {
        #address-cells = <2>;
        #size-cells = <2>;
        ranges;

        reserved: buffer@0 {
            no-map;
            reg = <0x9 0x00000000 0x0 0x80000000>;
        };
    };

    reserved-driver@0 {
        compatible = "xlnx,reserved-memory";
        memory-region = <&reserved>;
    };
};
```

Listing 4.1: Device tree options necessary to reserve a memory region. Specifically, 0x80000000 bytes (2 GiB) are reserved starting from physical address 0x900000000.

The DMA and spy feed-through are controlled by user space applications running on Petalinux. The DMA requires the physical address of a memory region where data can be written. A 2 GiB buffer is reserved with an option in the Linux device-tree (listing 4.1), preventing the OS from using it for memory allocation. This region can then be accessed from user space to save the gathered data.

4.3. Policy design

The experience accumulator architecture by construction relies on a performant design of the policy so that inference can be performed with strong real-time latency constraints. It should thus not be surprising how the implementation of such a policy requires a great deal of care. For this work, the policy is implemented using a NN, requiring the operations described in section 3.4. This allows to employ the RL libraries described in section 2.2.5.2 with only minor modifications.

AMD-Xilinx provides the Vitis AI Software [115], allowing simple deployment of NN models to FPGAs and heterogeneous platforms as Versal™. A special firmware is loaded on the board that implements a deep processing unit (DPU), specifically optimized to target NN workloads. The main capability of this implementation is its ability to employ the different computing elements of a system. For example [116], the matrix multiplications are performed on the AIE array, the convolutional layers are implemented on the FPGA, while the non-linear activation functions are performed in the ARM processor. The issue with this kind of implementation is twofold. First of all, all data is placed in memory before being processed, increasing latency and reducing its determinism. Secondly, data is processed in batches, further impacting latency. The reason for these drawbacks is that the currently available DPUs are mainly targeting high-throughput ML acceleration, where latency is not the main metric. Low-latency versions are currently planned but not available. As such, in the current work a manual implementation approach was chosen for the deployment of NNs on the AIE array.

A fundamental aspect dictating the implementation strategy is the choice of the numerical representation used during the computation. The AIE tiles allow both computations with single precision floating point numbers, and 8-bit, 16-bit, and 32-bit integer numbers. As shown in table 2.1, using integer precision allows for an higher number of multiplication to be performed in parallel. Floating point multiplications, on the other hand, are better indicated when a wider range of numbers needs to be accurately represented. Training of RL policies is usually performed with floating-point precision. The addition of the quantization stages necessary for training NNs with integer precisions would need to be included into the RL algorithms. Otherwise, this would lead to the unwanted effect that the real-time policy is only an approximation of the policy used for training, leading to potential biases in the training procedure.

In conclusion, floating point precision NNs have been developed targeting the AIE tiles and used as policies for the RL experiments that will later be discussed in this work. The implementation of the necessary building blocks, as described in section 3.4, namely matrix-vector multiplication, activation function, and random-number generation, will be carried out in the following subsections after the introduction of the AIE MAC intrinsics.

4.3.1. Intrinsics

This section is based on [81, 117].

As discussed in section 2.3.4.4, in order to fully exploit the vector processing capabilities of the AIE tiles, the use of intrinsic functions is required in order to indicate to the compiler that a vector operation needs to be performed. The same operation, be it a comparison, multiplication, addition, etc..., is applied to each element of a vector, called a *lane*. For floating point operations, eight lanes are available. The AIE vector processing unit has the capability of shuffling the elements of an input vector into a the desired lanes (figure 4.6). The shuffling scheme of floating point vectors is based on two parameters: `offsets` and `start`. The `offsets` parameter is composed of a 32-bit integer value that contains the eight 4-bit indices of the element to be placed in each lane. As shown in figure 4.6, `offsets = 0x01237654` would mean placing the element of the vector with index 4 into the first lane, the first element of the vector in the last lane, and so on. The `start` simply adds a constant offset to the index of each lane. Depending on the operation that is used, the `start` parameter of one of the two offsets might have to be a run-time constant, while generally `offsets` can vary at run-time. Albeit a `fpshuffle` intrinsics is available to rearrange the elements of a vector, this operation can be usually performed in combination with other intrinsics.

A MAC operation works as follows. An accumulator value a is incremented with the result of the multiplication of two operands, x and z,

$$a \leftarrow a + x \times z. \tag{4.1}$$

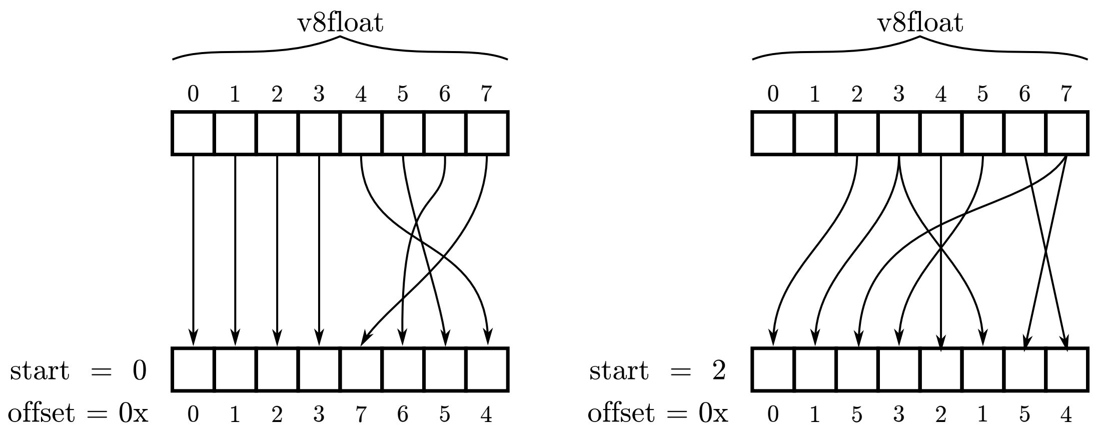

Figure 4.6.: Behavior of the lane permutation shuffling scheme. The left side shows the case with no starting offset, while the right side has an offset of 2.

This operation is executed through the `fpmac` intrinsic shown below.

```
v8float fpmac(
    v8float      acc,
    v8float      xbuf,
    int          xstart,
    unsigned int xoffs,
    v8float      zbuf,
    int          zstart,
    unsigned int zoffs
)
```

Here `v8float` represents a vector of eight single precision floating point elements. The first argument, `acc`, represents the vector with the accumulator values. The `buf`, `start`, and `offs` contains a multiplication operand and its shuffling parameters. Two multiplications operands, `x` and `z`, are provided. The function returns the per-lane MAC. It is important to notice no shuffling is allowed for the accumulator and return values. If the offsets are considered an array of indices and the return value is indicated as `ret`, the following pseudocode describes the functionality of this block

```
for (int i = 0; i < 8; i++) {
    ret[i] = acc[i] + xbuf[(xstart + xoffs[i]) % 8] * zbuf[(zstart + zoffs[i]) % 8];
}
```

The MAC unit is capable of a wide variety of operations. In general the signature of the intrinsic allowing maximum flexibility is the following

```
    v8float fpmac_conf(
        v8float       acc,
        v8float       xbuf,
        int           xstart,
        unsigned int  xoffs,
        v8float       zbuf,
        int           zstart,
        unsigned int  zoffs,
        bool          ones,
        bool          abs,
        unsigned int  addmode,
        unsigned int  addmask,
        unsigned int  cmpmode,
        unsigned int& cmp
    )
```

The pseudocode can be found below in listing 4.2. Additionally to the permutation of the input vectors that was just described, by using the addmode and addmask parameters it is possible to selectively flip the sign of each lane. Furthermore, the ones parameter allows to exchanged the shuffled zbuf with a vector of ones. By setting abs to true, the absolute value of the result of the multiplication can be added to the accumulator. Finally, cmpmode can be used to select the per-lane minimum (fpcmp_lt) of the accumulator and multiplication result, the maximum (fpcmp_ge), or to just forward the result (fpcmp_nrm). A bit in cmp is set based on whether the accumulator was chosen in fpcmp_lt and fpcmp_ge modes, while in fpcmp_nrm mode it sets for the negative lanes.

```
if (addmode == fpadd_add   ) neg = addmask ^ 0x00;
if (addmode == fpadd_sub   ) neg = addmask ^ 0xFF;
if (addmode == fpadd_mixadd) neg = addmask ^ 0xAA;
if (addmode == fpadd_mixsub) neg = addmask ^ 0x55;
for (i = 0; i < 8; i++) {
    operand_1[i] = xbuf[(xstart + xoffs[i]) % 8];
    operand_2[i] = zbuf[(zstart + zoffs[i]) % 8];

    if (ones)
        operand_2[i] = 1.0;

    mult_result[i] = operand_1[i] * operand_2[i];

    if (abs)
        mult_result[i] = abs(mult_result[i]);

    if (neg[i])
        mult_result[i] *= -1.0;


    mac[i] = acc[i] + n[i];

    if (cmpmode == fpcmp_nrm) {
        cmp[i] = signbit(mac[i]);
        ret[i] = mac[i];
    }
    else if (cmpmode == fpcmp_lt) {
        cmp[i] = signbit(mac[i]);
        ret[i] = cmp[i] ? -mult_result[i] : acc[i];
    }
    else if (cmpmode == fpcmp_ge) {
        cmp[i] = ~signbit(mac[i]);
        ret[i] = cmp[i] ? -mult_result[i] : acc[i];
    }
}
```

Listing 4.2: Pseudocode of the general MAC that can be executed by the vector unit of an AIE tile.

With this additional options it becomes possible to perform a wide variety of operations. For instance, if the `ones` option is enabled, it is possible to perform a sum operation between a shuffled version of `xbuf` and unshuffled values in `acc`. By properly setting the `addmode` and `addmask`, one can also perform subtractions. Similarly, if `ones` is enabled, and `cmpmode` and negations are properly set, one can select the per-lane maximum or minimum between two vectors. All these operations have a shorthand intrinsic, for example `fpadd`, `fpsub`, `fpmin`, and `fpmax`.

Additionally, the vector unit can operate on complex numbers. A `v8float` can be divided in pairs of floating point numbers. Each pair represent a complex number composed of a real and imaginary part. For clarity, this vector is then called a `v4cfloat`, signaling it contains four complex floating point numbers. The underlying register and vector processing units are the same. By shuffling and negating the appropriate intermediate result, two `fmac_conf` operations can be combined to perform complex number MAC operations. Additionally, manipulation of the negation settings allows to complex-conjugate one of the elements of a vector.

When high-performance is targeted one needs to consider the pipeline structure of the vector unit, shown in figure 4.7. The first cycle is required to decode the instruction. Then the registers are acquired and the shuffling is performed. Afterwards the eventual substitution with ones can take place and the multiplication can start. The multiplication operation is performed in parallel on the eight lanes, has a three cycle latency and can accept new data every clock cycle. In the following clock cycle negation is applied. The accumulator register is loaded during the last part of the multiplication operation. The two-clock cycle latency accumulator is then applied. A fundamental aspect of this unit is that it is not possible to add lanes of the same vector. In total every single MAC operation has a latency of eight clock cycles with a maximum throughput of one operation per clock cycle.

This fact has deep implications on the development of algorithms. Specifically, summing all elements of a `v8float`, also known as reduction by addition, incurs in a latency penalty. This operation can be performed with three consecutive `fpadds`, creating a serial dependency. After the first addition is scheduled, the following ones need to wait eight clock cycles for the result to be produced by the pipeline. This means a reduction by addition would result in a minimum latency of 24 clock cycles. Similarly, the computation of powers, e.g. taking x and computing x^5, requires a serial dependency chain.

Proper scheduling of the intrinsics is thus fundamental for ensuring low-latency and high-performance. In the following two sections the impact of these considerations on the development of low-latency NN will be considered.

4.3.2. Matrix-vector product

As shown in equation (2.27) on page 20, a fundamental part of computing a NN are matrix-vector multiplications. The goal of this section is to write a performant matrix-vector product

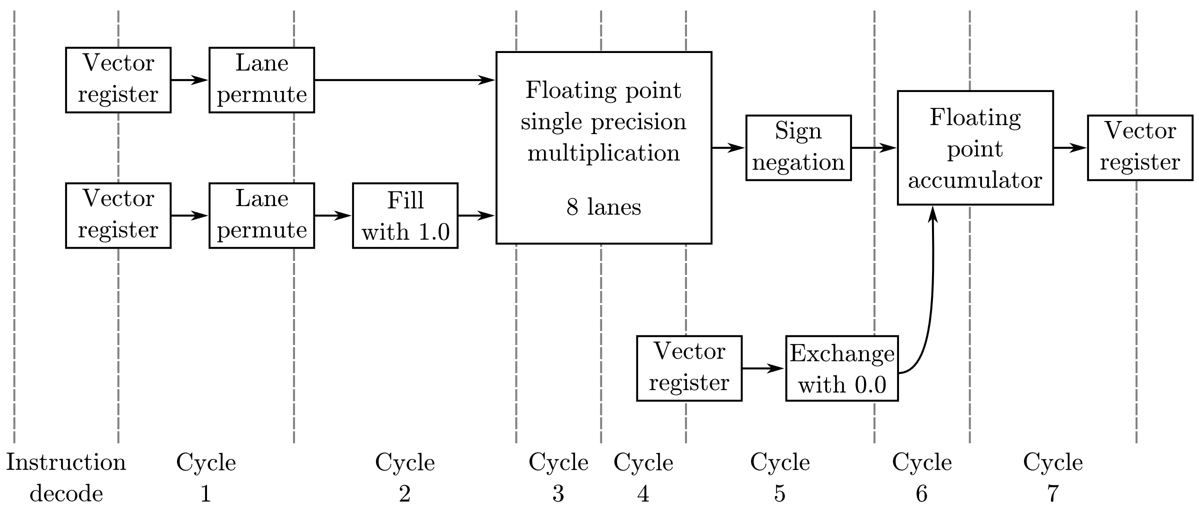

Figure 4.7.: Schematic view of the data pipeline for the floating point MAC operation in the AIE vector processing unit. Eight clock cycles are required. The first one is used for instruction decoding, three are used for the multiplication, and two for the result accumulation.

implementation targeting the Versal™ AIE and describing the required design choices. For a matrix $\bar{W} \in \mathbb{R}^{m \times n}$ and a vector $\vec{x} \in \mathbb{R}^{n}$ the product can be written as follows

$$(\bar{W} \cdot \vec{x})_{i=1,\dots,m} = \sum_{j=1}^{n} W_{i,j} x_j. \tag{4.2}$$

The C code equivalent to this equation is

```c
void matrix_vector_product(float W[][], float x[], float result[], int n, int m) {
    for (int i = 0; i < m; i++) {
        result[i] = 0.;

        for (int j = 0; j < n; j++) {
            result[i] += W[i][j] * x[j];
        }
    }
}
```

In order to fully understand the implication of the above code, one first needs to understand how arrays are handled in C (figure 4.8). Writing `float x[]` is completely equivalent to writing `float *x`, meaning that the variable `x` contains a pointer to a `float` value, i.e. it contains the address of a `float` value. By writing `*x`, a pointer is *dereferenced*, meaning it is substituted with the pointed value. Additionally, it is possible to perform arithmetic on pointers: `x+4` is still a pointer, but it points to the address that is after the one contained in `x` by the size of four `float`. In order to access the fourth element of an array, one would need to first increment the pointer and the dereference it, i.e. `*(x+4)`. This notation is fairly cumbersome and makes the code hard to read. To overcome this issue, the `x[i]` construct was introduced. This is a shorthand version of `*(x+i)`. A fundamental aspect that needs to be kept in mind is that C performs no bound check on an array. If a region outside of the array is accessed, undefined

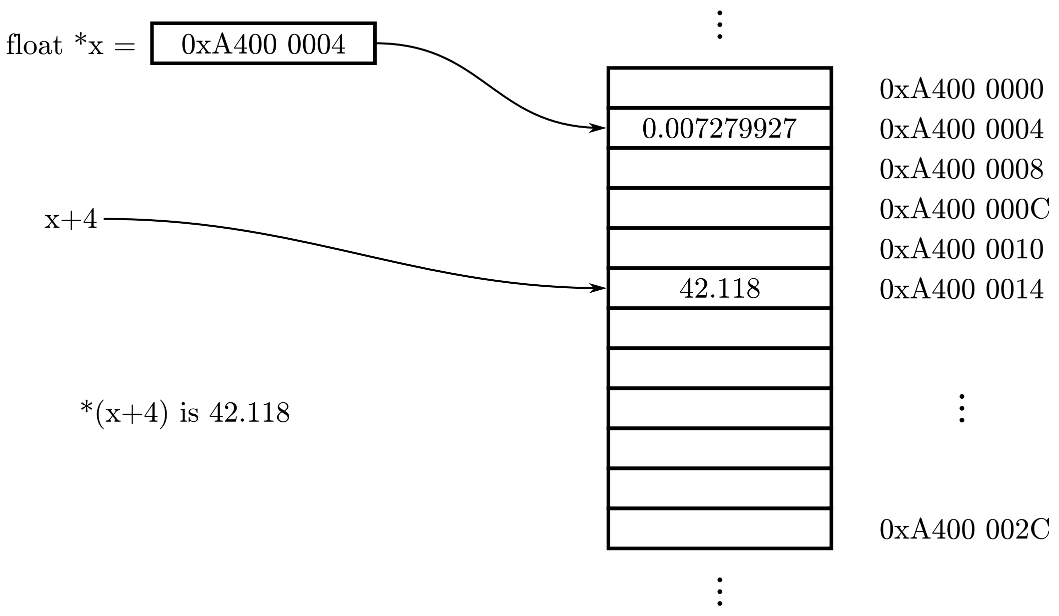

Figure 4.8.: Schematic showing the memory behavior of C pointers.

behavior can occur. In fact, when calling the above code one needs to ensure the arrays have exactly the correct size.

Furthermore, `float W[][]` is directly translated to an array of pointers, meaning that `W[i][j]` expands to obtaining the `i`-th pointer after `W`, and obtaining the `j`-th floating point value after that. As such, in order to obtain a single value of the matrix, two consecutive loads need to be performed. Such an issue can be overcome by treating `W` as an array of $n \times m$ elements. One can store the rows on $\bar{W}$ sequentially into the array. In this representation, element $\bar{W}_{i,j}$ corresponds to `W[j + i*n]`. The above code can thus be rewritten as

```c
1 void matrix_vector_product(float *W, float *x, float *result, int n, int m) {
2     for (int i = 0; i < m; i++) {
3         result[i] = 0.;
4 
5         for (int j = 0; j < n; j++) {
6             result[i] += W[j + i*n] * x[j];
7         }
8     }
9 }
```

Listing 4.3: C function performing matrix-vector multiplication with a flattened matrix representation.

This version of the matrix-vector product can in principle be compiled for the AIE with no modification. The compiler, though, will by default use the scalar processing unit, that cannot operate on floating point values natively and need to emulate the operation as integer and bit manipulation. In order to efficiently perform floating point operations, one needs to employ the intrinsics discussed in the previous section. In order to do this, all variables need to be converted into `v8float` type. Line 6 of the above code is a MAC operation, and can thus be directly replaced with an intrinsic. This leads to

```
void matrix_vector_product(v8float *W_v8, v8float *x_v8, float *result, int n, int m) {
    for (int i = 0; i < m; i++) {
        v8float res_v8 = null_v8float();

        for (int j = 0; j < n/8; j++) {
            res_v8 = fpmac(res_v8, W_v8[j+i*n/8], x_v8[j])
        }

        result[i] = reduce_add(res_v8);
    }
}
```

The `n/8` factors need to be added because each iteration of the inner loop now processes eight elements. The inner loop uses the `res_v8` to store eight partial sums that are then combined by the `reduce_add` function. Additionally, only vectors with a number of elements that are an integer multiple of eight can be multiplied. This implementation, albeit functional, suffers from a fundamental drawback. The inner loop cannot be efficiently scheduled due to the linear dependency chain caused by `res_v8`. Each `fpmac` instruction depends on the result of the previous one. As such, it needs to wait for the previous `fpmac` to finish before starting. This is exactly the same issue encountered when trying to efficiently implement the `reduce_add` function, as shown below for the `buf` variable

```
inline
float reduce_add(v8float val){
    // Sum the first four values with the last four
    v8float buf = fpadd(val, // sum val to itself
                        val,
                        0, 0x7654);

    // Of these last values sum a pair of two
    buf = fpadd(buf, buf, 0, 0x23);

    // Sum the last element
    buf = fpadd(buf, buf, 0, 0x1);

    // Extract the first element into a float
    return ext_elem(buf, 0);
}
```

The assembly code of this function is shown below. The `VFPMAC` instructions are separated by six `NOP` instructions in order to wait for the data to come out of the pipeline. It is also interesting to notice the large amount of registers [80] fed to the `VFPMAC` instruction, from vector registers (`wd1`, `ya`, `wc0`), to configuration registers (`cl1` to `cl6`), and general purpose registers (`r9`).

```
NOP;    NOP;    VFPMAC wd1, r1, wd1, ya, r9, cl5, wc0, #0, cl6, #0, cl1
NOP
NOP
NOP
NOP
```

```
NOP
NOP;    VMOV wr0, wd1
NOP;    NOP;    VFPMAC wd1, r1, wd1, ya, r9, cl4, wc0, #0, cl6, #0, cl1
NOP
NOP
NOP
NOP
NOP
NOP;    VMOV wr0, wd1
NOP;    NOP;    VFPMAC wd1, r2, wd1, ya, r9, cl3, wc0, #0, cl6, #0, cl1
NOP
NOP
NOP
NOP
NOP
NOP;    NOP;    NOP;    NOP;    NOP;    VEXT.32 r3, vdl1[0];    NOP
```

An important observation to further optimize this code can be done by observing that listing 4.3 can be rewritten in a way that exchanges the order of the two `for` loops. In order to do this, the initialization of the `result` vector needs to be performed separately. When implementing equation (2.27), this vector can be prefilled with the bias $\vec{b}$, that would otherwise need to be added later.

```
void matrix_vector_product(float *W, float *x, float *result, int n, int m) {
    for (int i = 0; i < m; i++) result[i] = 0.;

    for (int j = 0; j < n; j++) {
        for (int i = 0; i < m; i++) {
            result[i] += W[i + j*m] * x[j];
        }
    }
}
```

In this way, two consecutive executions of the inner loop are operating on different elements of the `result` vector. In this way the linear dependency chain can be broken. In order to optimize the memory access when reading the matrix, the representation of `W` needs to be changed in column-wise, with respect to the row-wise used before. In this way, consecutive MAC operations user consecutive values of `W`, that can then be loaded linearly. The vectorized version of this code is shown below, together with the generated assembly.

```
void matrix_vector_product(v8float *W_v8, v8float *x_v8, v8float *result_v8, int n, int m) {
    for (int i = 0; i < m/8; i++) result_v8[i] = null_v8float();

    // Cycle over input vectors
    for (int j_vec = 0; j_vec < n/8; j_vec++)
        // Cycle over input vector elements
        for (int j_elem = 0; j_elem < 8; j_elem++)
            // Cycle over output buffer
            for (int i = 0; i < m/8; i++)
                result_v8[i] = fpmac(
                    result_v8[i],
                    x_v8[j_vec],
                    j_elem,
                    0x0,
                    W_v8[i+j_elem*m/8+j_vec*m],
                    0x0,
                    0x76543210
                );
}
```

```
VLDA wc0, [p7], m0;                                    VFPMAC wd0, r7, wd0, ya, r15, cl1, wc0
NOP
VLDA wc1, [p0], m0;    VST .SPIL wd0, [sp, #-256];    VFPMAC wr2, r7, wr2, ya, r15, cl1, wc1
VLDB wc1, [p5], m0;    VLDA.SPIL wd0, [sp, #-192];    VFPMAC wd1, r6, wd1, ya, r15, cl1, wc1
VLDA wc0, [p6], m0;    NOP;    NOP
NOP
NOP;    VST.SPIL wd0, [sp, #-224];    NOP
VLDA wd0, [sp, #-288];    VMOV wd0, wr1;               VFPMAC wd0, r5, wd0, ya, r15, cl1, wc0
VLDA wc0, [p4], m0;       VMOV wr1, wr2
VMOV wr2, wd0;                                         VFPMAC wr3, r0, wr3, ya, r15, cl1, wc1
                                                       VFPMAC wr2, r2, wr2, ya, r15, cl1, wc1
```

In order to improve the readability of the above listing, some NOPs within one line were removed, while maintaining the number of lines unaltered. Additionally some argument of the VFPMAC that were identical were removed. This implementation achieved close to one VFPMAC operation per cycle. A further improvement can be obtained by decorating the function pointers with the restrict keyword. This keyword allows to specify that the pointers are guaranteed to not overlap, in this way allowing more aggressive optimization.

Further performance improvement can be achieved by performing the address computation through the cyclic_add intrinsic. This maps directly to a component of the memory unit allowing load and post-modification of the pointer in the same instruction.

A final remark concerns the memory usage of this kind of matrix-vector multiplications when used for NN inference. In this specific case the matrix $\bar{W}$ corresponds to the weights of the NN and needs to be modifiable at run-time but not during inference. It will thus need to be stored into the AIE tile memory as a RTP. In case this parameter is transferred synchronously, the absolute maximum memory available would be 128 KiB, corresponding to a 256×128 matrix of floating point values. In the asynchronous case, the available size for RTP is halved due to the necessity to store two copies of the parameter in order to implement a ping-pong buffer, leading to a maximum theoretical matrix size of 128×128. It is fundamental to keep in mind that this sizes are likely not achievable in real implementations, as some memory is still required for the storage of the stack and heap.

More complex schemes could be devised in order to expand the usable matrix size. A matrix could be partitioned into blocks that are computed in separate AIE tiles. Alternatively, some tiles could be used purely for storage and transfer data through the stream interfaces. Finally, one could steam the parameters from the PL or NoC using the interface tiles previously described. These more complex architectures will not be required for the remainder of this work, as the RL problems at hand require small NNs.

4.3.3. Activation functions

The final missing element to implement a layer of a NN, as described in section 2.2.2 on page 19, is the application of element-wise non-linear functions to the elements of the result of the matrix-vector multiplication, known as activation functions θ. Specifically, for $\vec{x} \in \mathbb{R}^n$, this means

$$y_i = \theta(x_i). \tag{4.3}$$

Of the several activation functions that are commonly employed in ML, one of the simples to implement is ReLU (equation (2.24) on page 19). This function behaves like the identity if the input value is greater or equal to zero, while being equal to zero otherwise. A possible C implementation would be

```c
void relu(float *x, float *y, int n) {
    for (int i = 0; i < n; i++) {
        if (x[i] >= 0) {
            y[i] = x[i];
        }
        else {
            y[i] = 0.;
        }
    }
}
```

Such an implementation could be directly compiled for the AIE, but it would rely on the emulation of the floating point operation in the scalar unit. Moreover, a conditional jump is potentially added. In order to obtain a higher performance implementation, the ReLU function can be rewritten as follows

$$\text{ReLU}(x) = \max\{0, x\}. \tag{4.4}$$

Every time $x \geq 0$ the maximum will be x. In the case $x < 0$, the maximum will be 0, thus implementing a ReLU. This definition allows to employ the the `fpmax` intrinsic, performing the element-wise selection of a maximum, effectively allowing performing eight ReLUs per instruction as

```c
void relu(v8float *x_v8, v8float *y_v8, int n) {
    for (int i = 0; i < n/8; i++) {
        y_v8[i] = fpmax(
            x_v8[i],
            null_v8float()
        );
    }
}
```

As discussed in the intrinsics section, the `fpmax` operation relies on the vector processing unit `VFPMAC` instruction, meaning it has eight clock cycles latency and one instruction per clock cycle can be issued.

Activation functions such as tanh and sigmoid can be implemented with the techniques described in section 3.4.2. Such an approximation is required given function approximation on hardware platforms is a well known problem, exhibiting a tradeoff between memory and computation. In principle, one could utilize a LUT to store every possible value of the function and load it instantaneously. Assuming a 32-bit floating point number, this would mean storing 4 bytes for every permutation possible 32-bit permutation, requiring

$$2^{32} \times 4\,\text{bytes} = 16\,\text{GiB}. \tag{4.5}$$

The size of LUT makes it impractical for an AIE deployment. On the other end of the spectrum, one could use iterative methods, as Newton's method, to achieve the desired level of accuracy. This approach is in principle viable for an AIE implementation, but could lead to issues in a real-time system. Specifically, if the number of iterations is allowed to vary, it is possible that the system latency will fluctuate. Furthermore, an iterative method specifically creates linear dependency chains that can negatively impact the performance of the system if not designed properly. A more refined, hybrid, method that is currently used in GPU is described in [118]. This technique relies on a LUT containing interpolation coefficients. Part of the digits of the input number is used to extract the coefficients from the table, that are then employed to compute a polynomial with the remaining digits. In this way the implementation tries to balance memory usage and computation requirements.

A possible implementation of the approximated activation functions is the one shown below.

```c
void activation_appox(float *x, float *result, float glue_low, float plateau_low, float
    glue_high, float plateau_high, int n) {
    for (int i = 0; i < n; i++) {
        if (x[i] < glue_low) {
            result[i] = plateau_low;
        }
        else if (x[i] > glue_high) {
            result[i] = plateau_high;
        }
        else {
            result[i] = polynomial(x[i]);
        }
    }
}
```

The main drawback of this implementation is that the chain of `if` cannot be directly implemented with intrinsic. It is important to notice that the polynomials constructed above have a special property: they are continuous in the *glueing* points between the polynomial and the plateau. This means that, if the polynomial is evaluated on the glueing points, it will produce the value of the corresponding plateau. Thus, one can filter the input array of all the values falling in a plateau, and substituting the corresponding glueing point. This implementation is shown below. The left panel uses standard C notation, while on the right panel intrinsics are used.

```c
void activation_appox(float *x, float * result, float glue_low, float glue_high , int n) {
    for (int i = 0; i < n; i++) {
        if (x[i] < glue_low) {
            x[i] = glue_low;
        }
        else if (x[i] > glue_high) {
            x[i] = glue_high;
        }
    }

    for (int i = 0; i < n; i++) {
        result[i] = polynomial(x[i]);
    }
}
```

```c
void activation_appox(float *x, float * result, v8float glue_low, v8float glue_high, int n) {
    for (int i = 0; i < n/8; i++) {
        x[i] = fpmax(x[i], glue_low);
    }

    for (int i = 0; i < n/8; i++) {
        x[i] = fpmin(x[i], glue_high);
    }

    for (int i = 0; i < n; i++) {
        result[i] = polynomial(x[i]);
    }
}
```

Using this detail, it is now possible to fully implement the activation function. In order to efficiently pipeline the `polynomial` function, the computation of x^3 requires care in order to minimize the dependency chains. An example implementation is shown below

```c
void third_power(v8float *x, v8float *xcubed, int n) {
    for (int i = 0; i < n/8; i++) {
        xcubed[i] = fpmul(x[i], x[i]);
    }

    for (int i = 0; i < n/8; i++) {
        xcubed[i] = fpmul(xcubed[i], x[i]);
    }
}
```

It is worth noticing how for a single `v8float`, before `xcubed` is multiplied a second time, one needs to wait for the result to be produced by the MAC pipeline. Thus, the above code will inevitably have to add `NOP` instructions in order to wait for the first `fpmul` to complete. The addition of further vectors does not have such a strong impact, as they can be used to keep the pipeline filled.

The polynomial can then be evaluated by combining its terms through MAC operations. In conclusion, this section describes the successful implementation of low-latency non-linear activation functions targeting the AIE tiles, that are directly usable for training with Pytorch.

4.3.4. Observation vector

To implement the observation vector defined in section 3.4.1, two stages are required: a filtering and decimation stage, followed by a shift register stage. The former can be carried out by means of a N-th order FIR filter

$$y_t = \sum_{i=0}^{N} c_i x_{t-i} \tag{4.6}$$

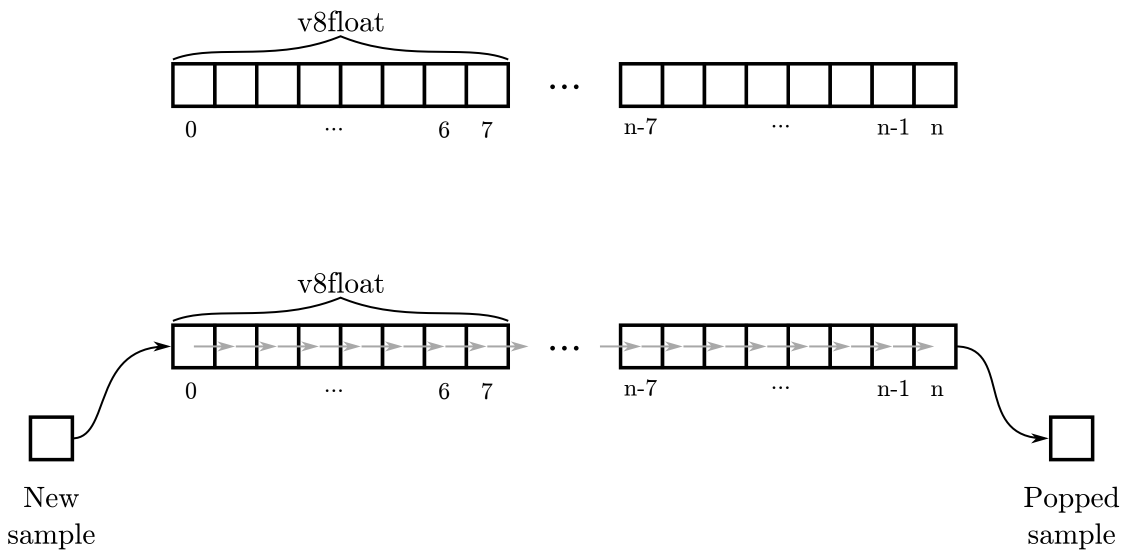

Figure 4.9.: Shifting scheme used to implement a shift register spanning several vectors.

where c_i are the coefficients of the filter. After filtering, one every m samples can be kept, in this way producing the decimated stream $\tilde{x}_t$.

The first part of the implementation of this algorithm for the AIE tiles is identical with the generation of an observation vector, and consists of a shift register. The `shft_elem` intrinsic allows to shift all elements in a `v8float` by one element, appending a provided value to the first element and popping the last value out of the vector. In order to perform a multi-vector shift, one needs to handle the boundary case. As shown in figure 4.9, one can start with the last vector and shift the $(n+1)$-th sample out, while shifting the last sample of the next vector in. This can be performed then in a chain for all vectors except the first, where the shifted-in value is the latest sample from the stream.

For the observation vector, then this buffer can be directly transferred to a matrix-vector multiplication stage to start the first layer of a NN. In order to compute the FIR filter value for decimation, one can then perform MAC operations between the coefficients and the shift buffer, as shown below

```
float fir_filter(v8float *buffer, v8float *coeffs, float x, int n) {
    // Shift x into the buffer
    // ....

    v8float result = null_v8float();

    for (int i = 0; i < n/8; i++) {
        result = fpmac(
            result,
            coeffs[i],
            buffer[i]
        )
    }

    return reduce_add(result);
}
```

Here a `result_add` is necessary: if several FIR filters were to be computed in parallel, then one could potentially be computed in each lane. In this specific case, though, the only way to leverage the MAC capability of performing parallel multiplication is to keep this lane reduction part in the end. An additional aspect to be kept in mind is that when decimating, some samples after filtering will be dropped. These samples are not used, so the design can avoid computing them. This corresponds to shifting values into the buffer m times, but only evaluating the FIR once.

4.3.5. Reconfiguration control

As discussed in section 2.3.4.4, the computing kernel being executed on an AIE tile behaves as a C/C++ function. The parameters of the functions represent the various interfacing options: some indicate a data stream or a buffer, while other represent RTP. When an experience accumulator architecture is implemented special care needs to be taken in order to allow seamless run-time modification of parameters. The ADF library allows to modify RTP from the PS in two main ways: synchronous and asynchronous. An asynchronous parameter can be modified at any time, thanks to the underlying ping-pong buffer structure. The new set of RTPs will be utilized from the following execution of the processing kernel. In the synchronous case, on the other hand, the kernels wait for a new set of RTPs before each kernel execution. This leads to two main practical differences. An asynchronous RTP consumes double the memory of a synchronous one, as it needs to store the parameter twice in order to implement the ping-pong buffer. On the other hand, a synchronous buffer requires more control from the PS as it needs new parameters to be explicitly provided in each execution.

For an experience accumulator, the possibility of flexibly defining new parameters is of the utmost importance, while timing dependencies on the PS before commencing operation should be avoided. Based on this, the asynchronous RTP scheme was selected.

Fundamentally, though, one needs a way to terminate a kernel execution, so that the new RTP values are utilized. This was achieved by adding control signaling to the data streams between kernel. In the case of AXI4-Stream communication, the TLAST signal can be employed as an execution termination signal. If it is received in the input data stream the execution of the kernel stops and the command is propagated to the following kernels. Similarly, for cascade stream and buffer transfers an extra data value is transmitted, encoding a stop signal. In the case of the cascade stream, for example, an extra `v8float` is sent, containing a non-zero value in cases the stream should be stopped.

An AIE control IP core was then developed. This Verilog module takes an input data stream and instruments it with the capability of sending the TLAST-containing AXI4-Stream signal based on a control register. This will then stop the following AIEs tiles. Additionally, the core can stop the stream to the AIE entirely, preventing the stream FIFOs from filling. For the experiments in the next two chapters, the block was extended so that it could optionally start forwarding its input stream only when an external trigger signal was received, or by letting only a fixed number of samples through.

This computation scheme adds an additional step in the verification process of the AIE kernels. Specifically, it is necessary to ensure that no deadlocks are formed between kernels, and that all units receive a proper stop signal. These kind of condition become increasingly complicated when dealing with RNN, and they have been more thoroughly studied in collaboration with Michail Sapkas in [105]. In these cases, the x86 simulation capability to examine stalls and locks in data interfaces has proven invaluable.

4.3.6. Random number generation

As discussed in section 2.2.3 most RL algorithms require a random number generator in order to explore their environment. Some version of Versal™ provide a true random number generator, but they require special authorization due to their possible military applications, and are as such difficult to obtain.

Some algorithms, known as pseudorandom number generators, allow the creation of seemingly random data streams from a set of deterministic operation as the ones that can be performed by a computer. A class of these algorithms, that are implementable on FPGAs, are linear-feedback shift registers. They consist of a shift register whose input bit is computed through a linear function of its bits. An example of these class of algorithms is Xorshift [119].

Another class of algorithms are linear congruential generators. The main idea is that the new state of the generator, s_{t+1}, is computed from the current state as

$$s_{t+1} = (as_t + b) \bmod c, \tag{4.7}$$

for some parameters a, b, and c, where mod represents the integer modulus operation. In computers the modulus operation is oftentimes efficiently computed by using the overflow of a given numerical representation, meaning that for a 32-bit number $c = 2^{32}$. If a and b are then selected properly, a pseudorandom sequence can be generated.

In [120] it is discussed how the randomness characteristics of this implementation are not optimal, specifically two consecutive samples are strongly correlated. The algorithm, can be extended to produce a more random data stream. This is achieved by using part of the state to scramble the remainder part of it. In this way, a 64-bit state can be used to produce 32-bit random values by using the remaining bit for scrambling. This algorithm is known as permuted congruential generator (PCG).

Given the high quality of the produced values and the availability of DSP units on Versal™, PCG was chosen as the KINGFISHER random number generator. The XOR shift random rotation variant with a 64-bit hidden state was developed. This implementation produces a 32-bit random unsigned integer every two clock cycles and has a maximum clock frequency of 250 MHz. A floating point conversion module was implemented to convert this value into a uniformly distributed single precision floating point value between zero and one.

Oftentimes RL algorithms require non-uniform distributions. One example are PPO and SAC, in the Stable Baselines3 implementation they require a Gaussian distributed sample. The central limit theorem states that for a random variable X_i with mean μ and variance σ^2, $\sqrt{n}(\bar{X} - \mu)$ approaches a normal distribution with zero mean and variance σ^2 for $n \rightarrow \infty$, where $\bar{X}$ is the average of n samples extracted from X_i. In the case of the uniform distribution between zero and one, $\mu = 1/2$ and $\sigma^2 = 1/12$, meaning that

$$\frac{1}{\sqrt{n}} \sum_{i=1}^{n} X_i - \sqrt{n}\mu, \tag{4.8}$$

is distributed as a gaussian with $\mu = 0, \sigma = 1$.

This equation can be efficiently computed in an AIE tile. The factors $1/\sqrt{n}$ and $\sqrt{n}\mu$ are provided as RTP, while the summation can be performed by accumulating n samples from the PCG stream. Setting the two RTP properly allows the selection of μ and σ. For KINGFISHER, $n = 16$ is used as it provides a good compromise between computation speed and accuracy.

4.4. Benchmarks

The policy building blocks discussed in the previous section need to be tested in order to demonstrate that they satisfy the required timing constraints. Two benchmarking approaches are considered: one is simulation based, while the other is hardware based. The former will be used to decide which communication scheme to employ between consecutive layers, while the latter is employed to highlight the effect of buffering on the latency of the data stream.

4.4.1. Feed-forward neural network benchmarks

The components described in the sections above can be combined into a feed-forward NN. Specifically, an AIE tile can implement a matrix-vector multiplication and an activation function, in this way creating a NN layer. Several of these tiles/layers can be chained together by using the communication schemes described in section 2.3.4.1, to form a feed-forward NN. In order to evaluate the latency performance of the XCVC1902 Versal™ device on the VCK190 evaluation board, the same NN described in [101], composed of four 64-neuron wide hidden layer with ReLU activation functions and two output neurons, was implemented. Data was transferred between layers by means of window/buffer transfers, while it was fed into the AIE array by DMAs written with HLS that were instrumented with the integrated logic analyzer IP core provided by AMD-Xilinx [121]. The computation was carried out with single precision floating point numbers and the NN was fully configurable at run time. This resulted in a latency of 4.5 µs. Compared to reference [101], where the same NN was implemented with 8-bit integer precision, obtaining a latency of 17 µs, this represents a more than a factor of three improvement. This fundamental result shows the capabilities of this platform and confirms its choice as a basis for KINGFISHER.

Interface type	AXI4-Stream	Cascade	Buffer
Latency	3.07 μs	4.67 μs	2.54 μs

Table 4.1.: Latency comparison of a feed-forward NN using different interfaces to transfer data between layers.

The computation of a NN could be in principle performed on a single AIE tile. Such an implementation, though, would fundamentally limit the amount of parameters that can be stored to the memory of a single tile. A feed-forward NN can be naturally pipelined by computing each layer in separate tiles. This scheme increases the latency of the implementation by the time necessary to transfer the output of each layer between tiles. On the other hand, it has the benefit of increasing the size of NNs that can be implemented. Furthermore, given the layers are pipelined, the throughput of the system increases.

The latency performance of the NNs will depend on the interface used to transfer data between layers. A systematic evaluation of each interface was carried out in collaboration with Ziyu Xie. In order replicate the conditions that are as similar as possible to the ones of a real-life deployment, a NN with an input size of 64 elements, two hidden layers each 64-neuron wide, and an output size of 2 elements were chosen. Additionally, the input of the NN was created by means of the shift register structure described in section 4.3.4. Three implementations, differing only in the interface used between layers, were tested in the AIE hardware simulation environment. In order to avoid FIFO-filling effects on latency (section 4.4.2), the sample rate was set to 0.25 MHz. The results are shown in table 4.1.

As expected, the buffer transfer results in the lowest latency. The asynchronous reading/writing capabilities of the ping-pong buffer allow data transfers with minimal stalls between master and slave tiles. Cascade stream, on the other hand, shows poorer latency performance compared to the simple AXI4-Stream transfers, despite the higher throughput. This phenomenon could be explained by the amount of memory buffers that are present in the communication channels. Cascade streams can only store a total of four eight float-wide outstanding transaction before being busy. This means that in order to transfer a complete layer both tiles need to be in a state where they can transfer data. AXI4-Stream, in this case, was implemented with much deeper data transfer FIFOs, in this way allowing the data transfer to not stall and thus latency to have a better behavior.

4.4.2. Policy benchmarking platform

The characterization of a given policy is fundamental to ensure its proper functionality when deployed in a real-world environment. The fulfilling of latency and throughput constraints, together with the proper functionality of the kernel execution control infrastructure, need to be verified, possibly on the same hardware they will be deployed to.

In order to perform more automated and reliable testing, a hardware benchmarking platform was developed together with Michail Sapkas [105]. This system instruments an AMD-Xilinx DMA with a time-stamping unit, allowing to record when a given data was produced/accepted.

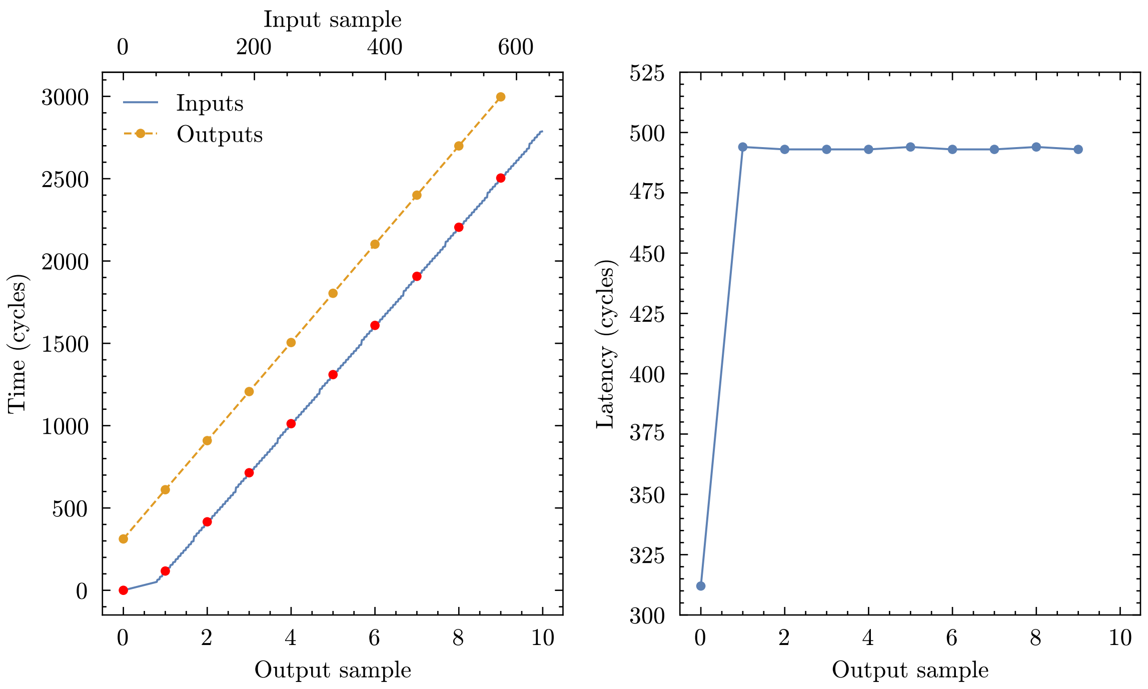

Figure 4.10.: Timing measurements on hardware for a dot product kernel. On the left side the sample transfer time is shown. On the right side the latency of each produced value is reported. The y-axis was offset in order to highlight latency variations. The anomalies at the start of the test showcase the effects of the input FIFO.

Furthermore, an IP core allows the rate of the data stream to match the one of the non-interruptible environment under study, allowing a user-selectable number of data points to be produced with a controllable rate. A Python control suite allowing data generation, DMA control, parameter setting, and data and metadata storage was developed.

In order to show the information that this system can provide, a single kernel computing a dot product was examined. A vector of 64 single precision floating point values is transferred via AXI4-Stream. These values are then multiplied with a 64 element vector provided as RTP. The result of the dot product is then streamed back to the PL. The timing measurement resulting from a benchmark of this kernel are shown in figure 4.10, for a clock period of 4 ns.

In the right side of figure 4.10, one can see that the latency of the first sample is much lower than the following ones. By examining the timing information in the left side of the plot, one can see that data are initially accepted at high rate. Just before the first output sample is produced, the input data rate changes. This phenomenon is due to the presence of a FIFO in the input AXI4-Stream interfaces of the AIE. The first output sample has low latency because data is processed right away. For all following samples the FIFO accepts data before they are ready to be processed, thus increasing latency.

If one fixes the input data rate to be below the maximum throughput of the kernel, most of the data will be computed right away. An example is shown in figure 4.11. The reduced input rate allows for all data to be processed as soon as they are provided. In this case the latency is much lower than the plateau one of the previous example.

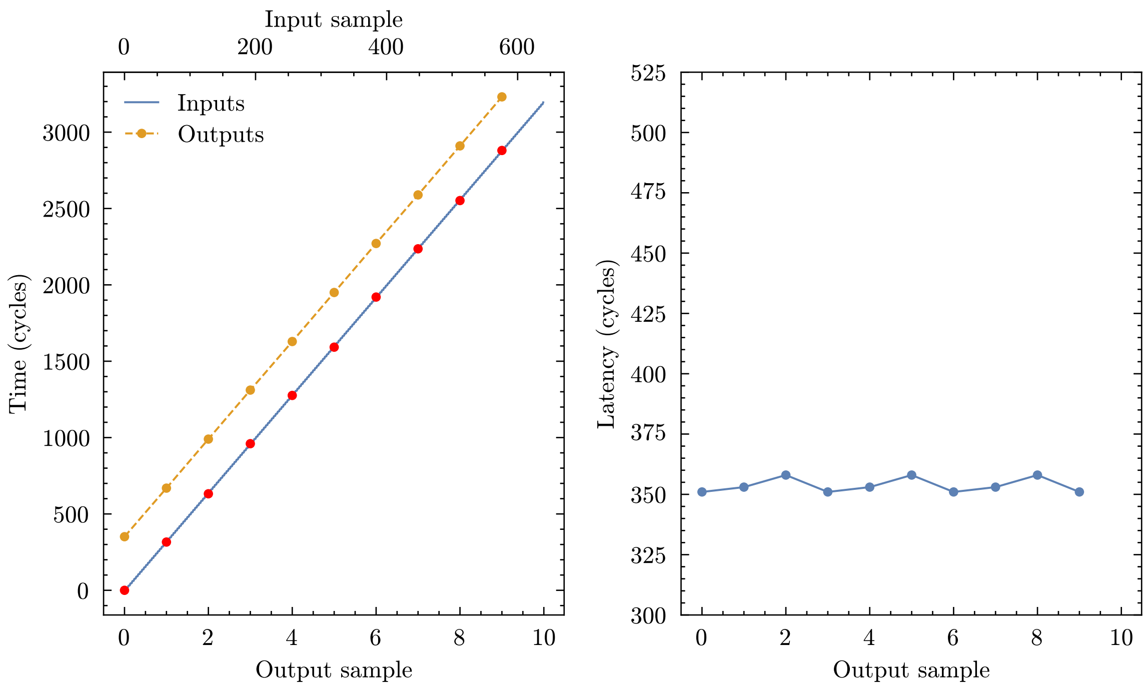

Figure 4.11.: Timing measurements on hardware for a dot product kernel while limiting the input data throughput. The latency y-axis is the same as in figure 4.10. No FIFO anomalies appear at the start of the test.

This FIFO-filling latency effects, creating a dependency between throughput and latency, need to be taken into account during the validation of a given policy. If a non-interruptible environment is targeted, one should match the data production rate during testing to the one expected from the environment. In this way the latency of the system can be estimated correctly.

4.5. Control system integration

In the previous sections, the main building blocks of the KINGFISHER framework were described. Each one of these blocks has several RTPs that need to be tuned during deployment. Within large-scale facilities, the control signals are usually merged into a control room. This happens for operational reasons, as hand-modifying parameters of instrumentation that is scattered over hundreds of meters is extremely inefficient. Additionally, safety concerns as the protection of operators from radiation exposure can in principle drive the need for remote control schemes.

Several protocols currently exist that allow to digitally communicate with devices in order to remotely control and monitor them. This kind of infrastructure is known as a supervisory control and data acquisition (SCADA) system. Among the most commonly used of these control systems one can find the experimental physics and industrial control system (EPICS) [122], gRPC [123], DOOCS [124], and TANGO [125]. At KARA, that will be used as testing facility

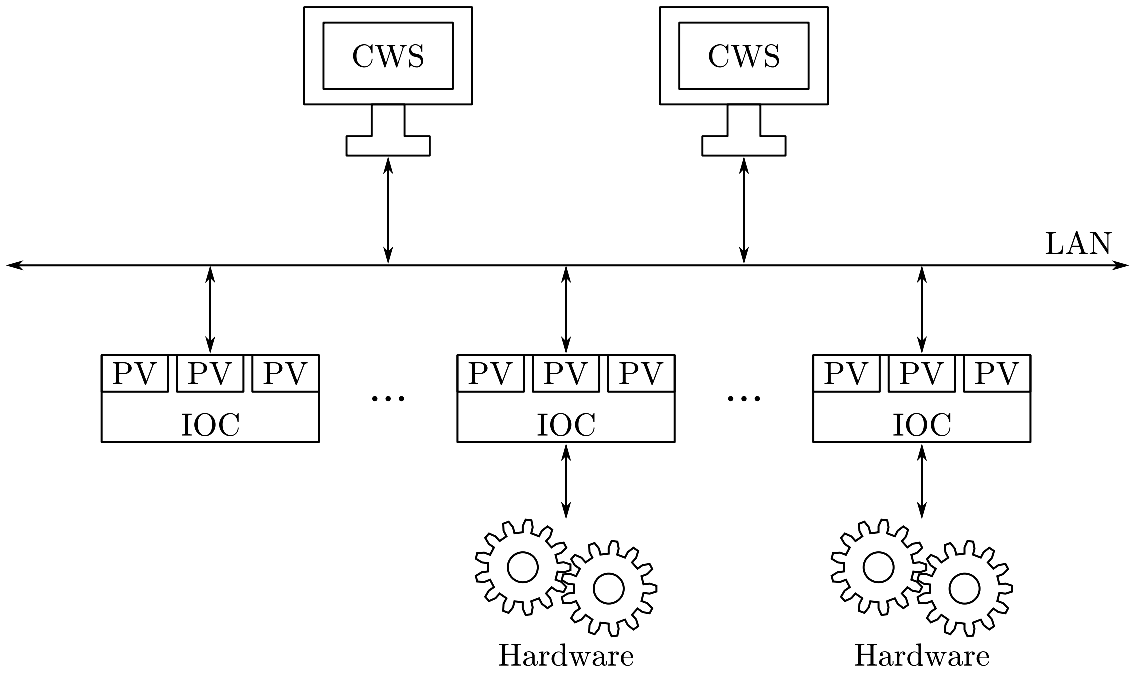

Figure 4.12.: In EPICS, IOC servers expose variables, known as process variables (PVs), through the local network. Each IOC can be used to control some underlying hardware, or to perform data processing. The computers, in this model known as CWS, can access the PVs from the IOCs.

for KINGFISHER, EPICS is employed. Given its versatility and the possibility of converting it to other protocols, this was chosen as the control system of KINGFISHER.

An EPICS control system usually employs a local Ethernet-based network for communication between its components (figure 4.12). Each device exposes a number of attributes known as process variables (PVs) through an input/output controller (IOC) server. The IOC does not necessarily require to be a physical input/output device, but it can be used for data processing systems. A client workstation (CWS), usually a computer in the control room, can read and write PVs. EPICS is designed to be high performance and scalable, with systems with millions of PVs being currently used. Additionally, it is event-driven: a CWS is directly notified of the change of a PV it is monitoring, with the possibility of setting alarms. IOCs for the control of a variety of devices at particle accelerators and other large-scale facilities are routinely provided by the EPICS collaboration.

The KINGFISHER integration with EPICS is structured as follows. A user-space application running on a Petalinux OS in the PS allows reading/writing physical memory addresses corresponding to the registers of AXI4-Lite peripherals in the PL. Additionally, an application based on the Xilinx run time (XRT) library allows modifying the AIE RTP. This application based approach adds an abstraction layer that facilities system debugging and the migration to other control systems.

The KINGFISHER IOC is implemented through the caproto library [126], a Python implementation of part of the EPICS protocol. PVs can be defined as members of an IOC class. The asynchronous input/output Python framework allows for event driven task execution.

```python
from caproto.server import PVGroup, pvproperty

class ExampleIOC(PVGroup):
    my_pv = pvproperty(value=0, doc="My PV")

    @my_pv.putter
    async def my_pv_put(self, instance, value):
        # Modify register to value
        # ...

    @my_pv.scan(period=0.1)
    async def my_pv_scan(self, instance, async_lib):
        # Read register value from hardware
        # Set the PV to that value
        instance.value = value_from_register
```

Listing 4.4: Example of an IOC written using the caproto library.

When a CWS modifies a PV, a `put` method is executed, calling the corresponding user-space application and modifying the corresponding register. For monitoring, a periodic `scan` task can be executed that monitors the underlying value and applies it to the PV. Start-up and shutdown tasks are also possible. An example of IOC is shown in listing 4.4.

4.6. Online training and reconfiguration

As discussed in section 2.2.5.2, libraries as Stable Baselines3 allow users to employ state of the art and verified RL algorithms. The library is written in Python and implements several commonly used algorithms, for example PPO, DDPG, SAC, etc. The currently available release is mainly meant to gather data from interaction with a Gymnasium environment. The inference of the policy is performed in the `collect_rollouts` method that is called during training. This precludes the creation of special Gymnasium environment that replays data gathered by an experience accumulator, as there is no guarantee that the pseudorandom numbers that are produced during the model inference will correspond to the one of the AIE produced as in section 4.3.6. A possible solution would be to re-implement the PCG-based Gaussian random number generator in Python and include it in Stable Baselines3. This would require careful tracking of the internal generator state during inference and training process.

An alternative solution is to store both the output of the NN and the random numbers produced by the KINGFISHER generator. The `collect_rollouts` can then be modified to load this data from file and use it for training. Special care is needed in order to verify that the NN deployed on KINGFISHER and the one used for training match. The difference is verified to be in the order of machine precision, otherwise a warning is produced.

In order to carry out reproducible experiments metadata needs to be gathered in order to monitor the state of the accelerator. This can be performed by saving the state of a list of PVs that are relevant for the current application. Additionally, the agent obtained after each

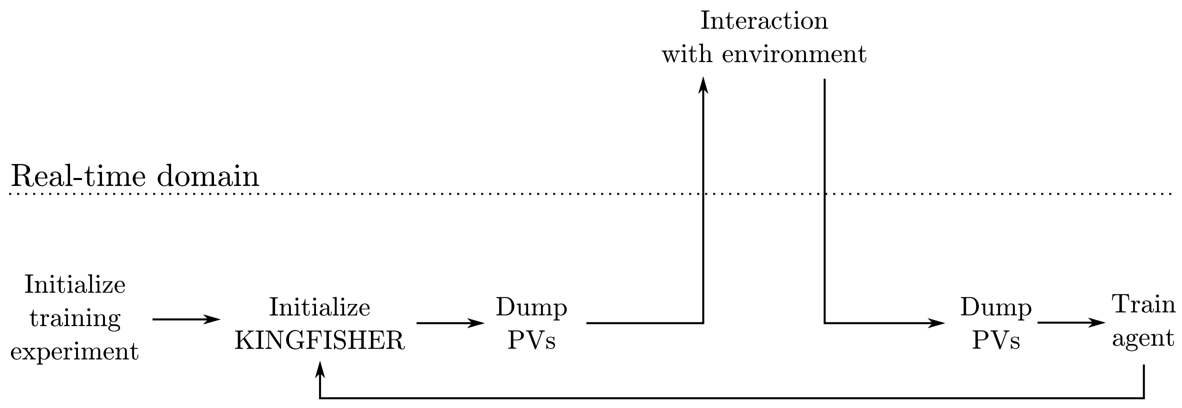

Figure 4.13.: Chart showing the steps carried out during training. KINGFISHER is initialized, with the parameters describing the RL agent. Then the state of the facility is captured by saving the PVs. The real-time agent is then let to interact with the environment, producing training data. The PVs are stored again, to track the variation of the parameters that occurred during the interactions. Finally, training is carried out and the new coefficients are applied.

training iteration can be stored. An example of the flow-chart for the system is shown in figure 4.13. The KINGFISHER agent is initialized and the PVs of interest are saved. The agent interacts with the environment for a predetermined number of steps, while its observation, action, and random number vectors are stored in memory and dumped into a file. This file can then be accessed from the PS, or remotely from a control room computer. Moreover, the state of the PVs is recorded again. This is useful to monitor possible beam losses induced by the agent. A training iteration is carried out on the data that were just gathered. The new set of coefficients is loaded into Versal™ and another training cycle starts.

During testing it is sometimes useful to vary the size of the NN used for inference. In order to not affect the latency and throughput characteristics of the real-time NN this should be implemented by switching-off the non-utilized neurons, instead of reshaping the matrices in the AIE. This ensures the same number of operations are carried out in both cases, mitigating the system dependency on latency.

4.7. Summary

The KINGFISHER platform has been developed to aid the execution of experiments in large-scale facilities. The system is based on a AMD/Xilinx Versal™ device and implements real-time high-performance NNs in the AIE array. FPGA IP cores allow communication with KARA and the real-time monitoring of the policy to gather training data. The CPU has the task of managing the system, by loading the new NN coefficients and by forwarding training data to an external platform.

An in-depth discussion of the integration within KARA and the design of the system is carried out. Special care was used in describing the development of AIE NN-based policies employing the high performance MAC vector unit. Specifically, the required intrinsic function used to employ these units was thoroughly described, together with the pitfalls and linear-dependency mitigation strategies necessary to harness the full potential of an AIE tile. Benchmarking of these policies to extract the latency characteristics was also carried out, producing latency figures in the microsecond range. Additionally latency fluctuations effects produced by FIFO buffers in the data stream were shown to be relevant in some cases.

Finally, integration of the system in the KARA control system is discussed, together with the scheme utilized to carry out RL experiments with online training of policies.

5. Reinforcement learning control of the horizontal betatron oscillations

The results of this chapter have been published in [12, 127].

In order to test the system described in chapter 3, an environment with high interaction rate and well understood dynamics is required. The betatron oscillations described in section 2.1.2 represent a feasible candidate. First of all, their dynamics are easy to simulate. Secondly, at KARA, their oscillation frequency is in the order of 700 kHz, allowing turn-by-turn interactions with the accelerator and thus producing data at rates where an experience accumulator is applicable. These undesired oscillations are usually managed in accelerators with transverse dampers, and bunch-by-bunch closed loop feedback systems [128]. A commercial system is in operation at KARA, which served as benchmark in these studies. The system is vital during operation in order to avoid beam losses induced by the transversal instabilities.

5.1. Problem description

In the presence of HBOs, an observer fixed in a given position of KARA will only observe the fractional part of the tune Q_x, corresponding to an oscillation frequency in the order of 700 kHz, depending both on the operation mode, the beam current and more generally the state of the machine. This oscillation frequency sets the timescale at which the RL agent must be able to act, in this case in the order of a few microseconds.

A commercial BBB feedback system [129] is installed at the machine. This setup is based on an FIR filter that takes the input position signal and applies a π rad phase at the frequency of the instability. In this way, a linear kick is produced with an opposite sign compared to the displacement, damping the oscillations. The output of this kind of filter can be computed as

$$y(t) = \sum_{i=0}^{N} c_i x(t-i) \tag{5.1}$$

where c_i are the coefficients of the filter, and N is its order. The tuning of the filter coefficients directly impacts the performance of the controller, as it defines its behavior with respect to external noise and the bandwidth over which a suitable phase offset is produced. So far this is usually hand-tuned. A review on the topic is provided in [128].

In order for this system to work, a BPM is already installed, together with a stripline kicker, based on the design from [130], and the necessary driving amplifiers. This kicker capable of affecting the HBOs by applying a bunch-by-bunch and turn-by-turn horizontal force to the beam based on the signal provided into an analog input.

The injection in the KARA storage ring makes use of three strong kicker and one septum magnet in order to properly merge the beam already in the machine with the one that is being injected. For the injection process, the kickers are pulsed at a rate of 1 Hz, creating a kick field for a few revolutions, in this way moving the beam on a displaced orbit in the horizontal plane. When the kicker switches back off, the displaced bunches will start performing betatron oscillations around the reference orbit. The goal of the RL agent is to damp this oscillation as quickly as possible. Notably, the strength of the injection kickers is orders of magnitude stronger than the stripline kickers used for feedback, thus an agent cannot damp an oscillation in a single kick, and turn-by-turn control is required.

5.2. Closing the feedback loop

A BPM is, in the case of KARA, a structure composed of four *buttons* that are embedded in the beam pipe. Each one of these is an electrode behaving as an antenna, producing a signal when an electron bunch passes by. The amplitude of the four signals can be used to reconstruct the transversal position of the beam together with its charge. The time-of-arrival of the signals allows to determine the longitudinal position of the bunch. An analog Dimtel BPMH-20-2G hybrid [131] combines the button signals into horizontal and vertical positions signal, plus a sum signal that can be used for time of arrival measurements. Each one of these signals is a fast bipolar peak that can be sampled with the KAPTURE. The KAPTURE data is then streamed to KINGFISHER by employing the connection scheme described in section 4.1 with the 40 Gbps Aurora 64b/66b link [112] configuration.

The action data-stream needs to be converted into an analog signal that is amplified with a Rohde & Schwarz BBA150-A125 wideband power amplifier with a frequency range from 9 kHz to 250 MHz and an output power of 125 W. The signal was produced through a Pmod DA3 module based on the AD5541A DAC, with a large signal settling time of 1 μs. This rate allows turn-by-turn control of a single bunch, but it does not allow multi-bunch control. In order to ensure that the new DAC value is synchronized with the arrival of the bunch, special care needs to be used when developing the control IP core. An example data transaction is shown in figure 5.1.

In order to provide a new sample to the DAC, sixteen clock transitions are needed to transfer the set-point, plus an additional one for the data to be shifted from the input register to the conversion stage. If $\overline{\text{LDAC}}$ is tied low, data is converted as soon as it is available. Thus, a total of at least 17 clock cycles are required per revolution in order to provide a sample to the DAC. The maximum sustainable SCLK frequency is 50 MHz. The KARA main RF clock frequency is not constant, but it is adjusted during operation to stabilize the orbit. To ensure that a single DAC sample is produce at each revolution, corresponding to a KAPTURE sample for a single bunch, the DAC clock has to be locked to the KARA clock. For simplicity, in the following

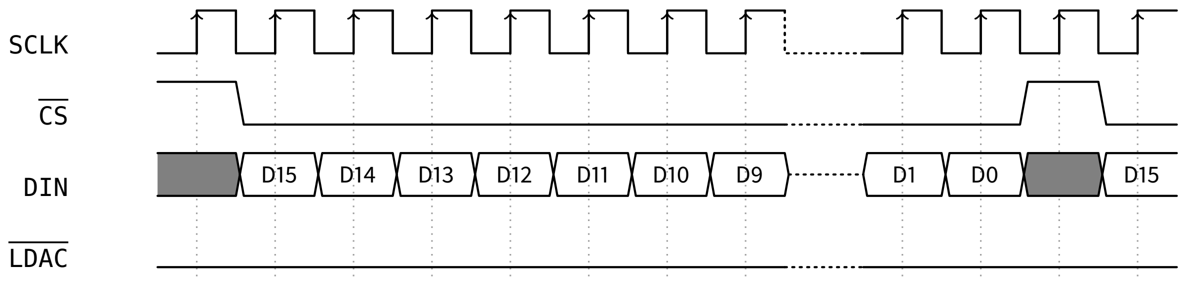

Figure 5.1.: Control signals for the AD5541A DAC. Data is transferred after $\overline{\text{CS}}$ goes from high to low. In order to start the data conversion $\overline{\text{LDAC}}$ needs to be set to low. In this case it is tied to low, forcing conversion as soon as possible.

time analysis $f_{\text{KARA}} = 500\,\text{MHz}$[1]. To achieve this, the KARA clock was divided by a factor of 16 within of the timing distribution units of the facility, creating a $f_{\text{KF}} = 31.250\,\text{MHz}$ reference that was fed into KINGFISHER. There SCLK needs to be generated with a frequency of

$$f_{\text{DAC}} = f_{\text{rev}} \times 17 = f_{\text{KARA}} \times \frac{17}{184} = f_{\text{KF}} \times \frac{34}{23}, \tag{5.2}$$

where the factor 184 is the harmonic number and f_{rev} the revolution frequency. The 34/23 factor can be used to program one of the internal PLLs of Versal™ to obtain the correct clock. Due to the constraints of the operation frequency of the VCO and the multiplier/divider values, an equivalent factor of 102/69, creating the necessary frequency.

Additionally, a trigger input was employed to start the agent action synchronously with the injection kickers. A photograph of the setup is shown in figure 5.2.

5.2.1. Effects of coupled bunch instabilities

A first KAPTURE readout test was carried out at KARA with a single bunch train composed of approximately thirty bunches, in the machine conditions summarized in table 5.1. The signal produced with no external excitations is visible in figure 5.4. Strong noise was observed, making it impossible to observe the HBO produced by the injection kickers, thus no control experiments could be performed in those conditions.

Energy	E	0.5 GeV
Total RF voltage	V_{RF}	357 kV
RF frequency	f_{RF}	499.754 MHz
Synchrotron frequency	f_{sync}	35.7 kHz
Momentum compaction factor	α_{c}	8×10^{-3}

Table 5.1.: Machine parameters employed during the HBO control experiments.

[1] The KARA main RF frequency depends on several parameters, such as energy, external temperature, current, etc. At the time of writing its value during injection is 499.7540 MHz.

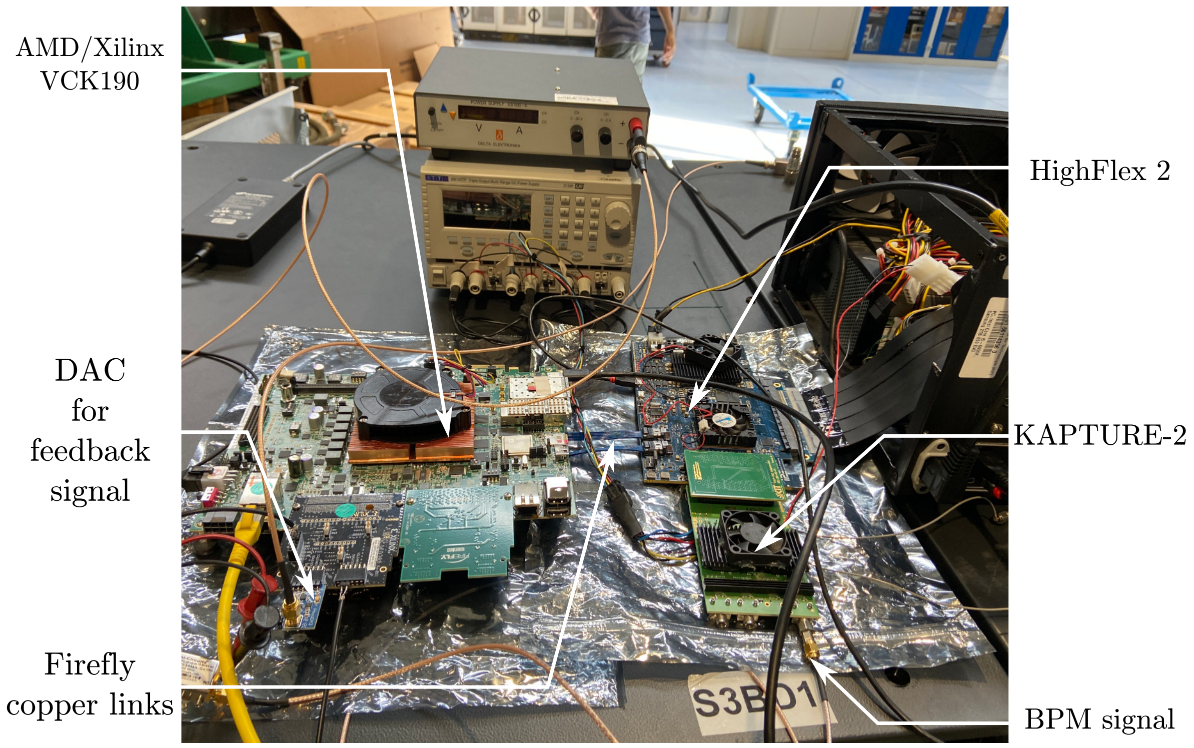

Figure 5.2.: Photograph of the setup after the measurement campaign at KARA. The BPM signal enters from the bottom right side, is sampled by KAPTURE-2 and sent over an Aurora link by an HighFlex 2 board to the AMD/Xilinx VCK190. Here the data stream is processed by a NN or FIR filter. The resulting feedback signal is brought to the analog domain with a DAC, and is fed into the power amplifier before being applied to the stripline kicker (off picture).

This behavior was attributed to jitter in the system, either affecting the time of arrival of the BPM pulses, or the KAPTURE sampling clock. The BPM signal was thus measured with an oscilloscope in order to find its timing characteristics. The pulses were bipolar signals with width smaller than 20 ps, mainly limited by the bandwidth of the oscilloscope. The jitter of the output clock coming from KAPTURE was measured to be 2.3 ps. Albeit sizeable, this was not considered to be the main contributor to the observed jitter. Nonetheless, KAPTURE was configured to use the clock from the KARA master oscillator, reducing the jitter to 0.72 ps.

If the FFT of the noise signal is examined (figure 5.5), a strong peak appears at roughly 60 kHz. This corresponds to exactly twice the synchrotron frequency at the time of operation, leading to believe that longitudinal oscillations were being excited through the coupled bunch instabilities [18]. Such a phenomena originates from the interaction of each bunch with the electric field produced by the other bunches.

A trivial way of mitigating this effect is to utilize a single-bunch filling-pattern. Usually, the BBB feedback system is employed to excited betatron oscillations in the unwanted bunches, leading to their loss, while stabilizing the ones that are intended to be kept. In this case, though, this operation is not trivial, as the amplifiers and stripling kickers of the BBB system were being utilized for the KINGFISHER feedback. A separate BBB is installed in the booster, allowing bunch cleaning in the injector.

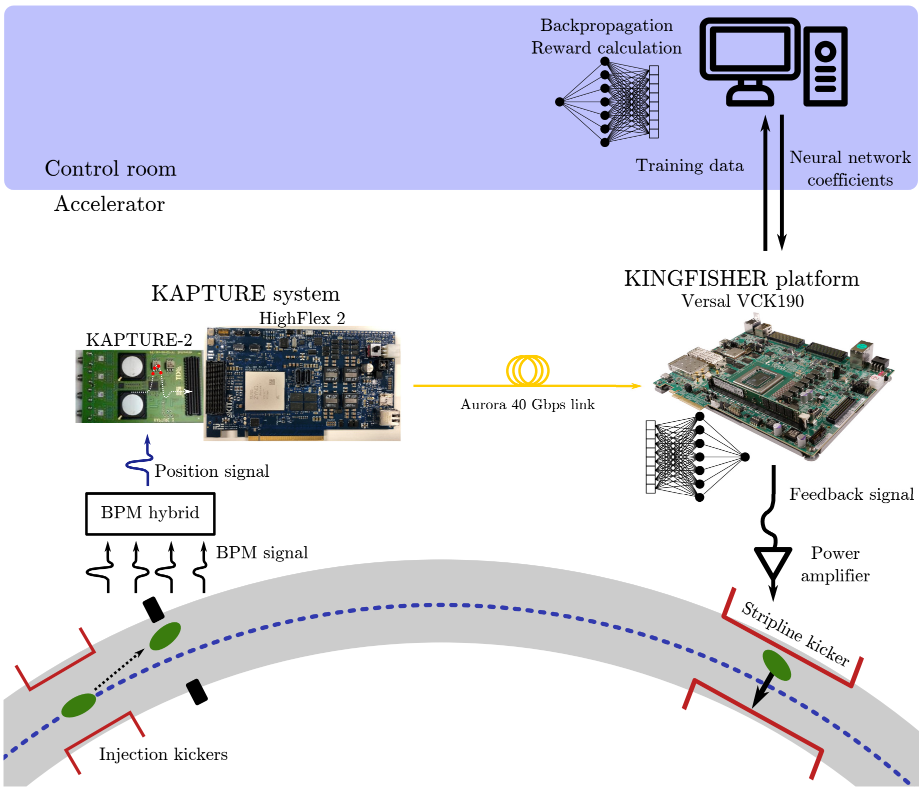

Figure 5.3.: Drawing of the hardware infrastructure employed in this work. The bunch position in the beam pipe is measured by processing a BPM signal with an analog hybrid. The produced analog signal is sampled with KAPTURE and then forwarded by an HighFlex 2 board through high-speed links to a Versal device. The KINGFISHER system programmed on this Versal then connects this data stream into the RL controller, while applying the output action to a stripline kicker. Every episode, comprised of 2048 interaction steps with the accelerator, data is sent back to the control room for training.

With a single bunch, the produced signal as shown in figures 5.4 and 5.5 do no exhibit the disturbing noise anymore, confirming that the effect was mainly due to time-of-arrival differences induced by the coupled bunch instabilities.

5.3. FIR-based control

In order to test the functionality of the complete feedback system, an FIR filter controller with 32 coefficients was deployed to KINGFISHER. This can also serve as a conventional controller benchmark to measure the performance of the RL agent. The implementation was executed on a single AIE tile. The system was tested with a signal generator and an oscilloscope by setting its coefficients to the identity, allowing to measure its latency of 2.5 μs [127].

A commonly employed set of coefficients c_k are

$$c_k = \sin\left(2\pi f \frac{k}{N} + \phi\right), \tag{5.3}$$

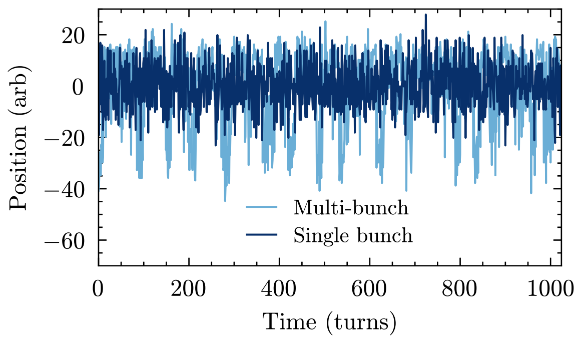

Figure 5.4.: Comparison of acquired BPM signal with KAPTURE for single and multi-bunch filling patterns. The multi-bunch case shows noise spikes that are not present in the single bunch case.

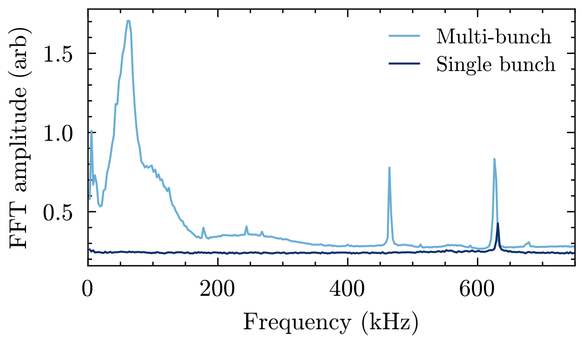

Figure 5.5.: Comparison of the FFT of BPM signals acquired with KAPTURE for single and multi-bunch filling patterns. The multi-bunch case shows a broad peak at roughly 60 kHz, corresponding to twice the synchrotron frequency. The peak around 620 kHz is due to the betatron oscillations.

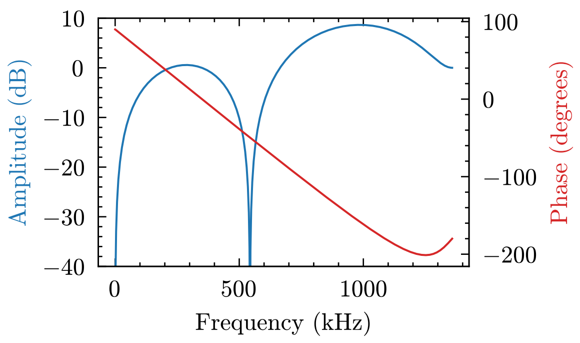

Figure 5.6.: Response function of an FIR filter based on the coefficients from equation (5.3) for $N = 5$, $f = 2$, and $\phi = 90°$.

where N is the number of non-zero coefficients, while f and ϕ can be used to tune the filter response. Assuming $\omega_0 = 2\pi f/N$, the frequency response can be computed analytically using

$$F = \frac{1}{2i}\left(e^{i\phi}\frac{e^{i(\omega_0-\omega)N}-1}{e^{i(\omega_0-\omega)}-1} - e^{-i\phi}\frac{e^{-i(\omega_0+\omega)N}-1}{e^{-i(\omega_0+\omega)}-1}\right). \tag{5.4}$$

To a first order of approximation, N can be used to control the filter bandwidth, while varying f and ϕ one can control the position and phase of the maximum of the response function. The main idea of this kind of controller is to shift the HBO with a 180° phase, in this way the feedback will have opposite sign compared to the displacement and thus will damp the oscillation.

In order to not saturate the DAC and the power amplifiers, the output of the FIR filter is multiplied by a gain factor, allowing to control its scale. Additionally, this also allows to study the effect of the feedback intensity on the beam. For a first test the parameters were chosen as $N = 5$, $f = 2$, and $\phi = 90°$. This leads to the response function shown in figure 5.6.

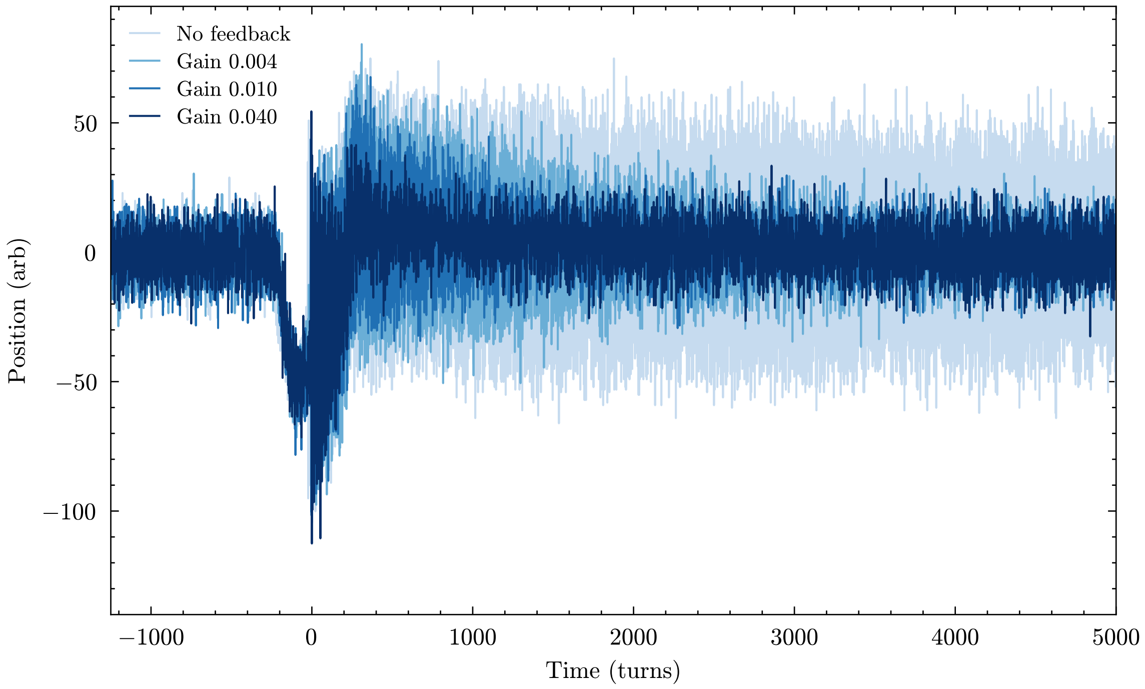

Figure 5.7.: Position signal with an FIR controller as a function of gain. Higher gains correspond to a stronger effect on the beam, leading to faster damping.

These filters were used to try damping the oscillations generated by the injection kickers, the result of which is shown in figure 5.7. The injection septum is switched-on at $t \approx 250$ turns, producing a shift in the baseline position. The kickers act at $t \approx 0$ turns, starting the oscillation. For increasing gains, the rate of damping increases. Such an effect is shown in greater detail in figure 5.8, were the envelope of the signal was computed and the exponential decay rate was obtained through a fit.

An interesting effect that can be employed to verify the functionality of this feedback system is the appearance of a notch in the BPM spectrum [132, 133]. This phenomenon is produced when noise is sampled and is fed in the FIR filter. This gives a 180° phase to the noise signal around the tune frequency. The frequency response of the beam has a resonance at this frequency making it behave as an amplifier. The noise signal is then sampled again by the electronics front-end. In this case, though, it has a 180° phase, thus it is subtracted from the original noise signal, leading to a noise reduction. This produces a notch in the FFT that goes below the noise floor of the front-end electronics. Fundamentally, though, this notch appears only in the acquired signal used for feedback. An additional monitoring channel would observe an increase in the noise floor, due to the noise injected from the FIR filter.

The KINGFISHER-based FIR controller exhibits this phenomenon, as shown in figure 5.9. If the gain is zero, i.e. the feedback is off, a peak is visible corresponding to the self-excitation of the HBO. When the gain is increased, the depth of the notch increases due to the stronger action of the feedback system.

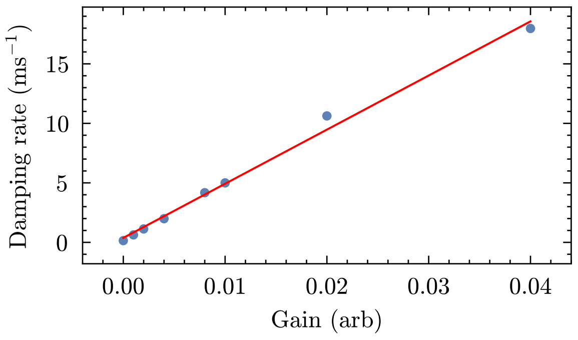

Figure 5.8.: The damping rate is obtained from a fit of the envelope. A higher gain applies a stronger action on the beam, leading to a higher damping rate.

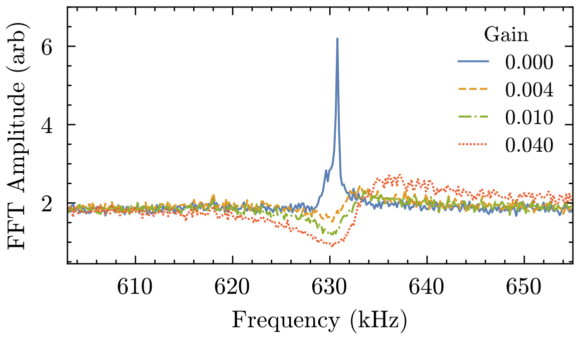

Figure 5.9.: FFT of the feedback-BPM signal. When the gain is zero (feedback off), a peak is visible at the frequency of the HBO. When the feedback is switched on a notch appears.

In conclusion, the system designed so far is capable of controlling the HBO, and can thus be used to deploy RL agents.

5.4. Formulation as an RL task

For the current problem of controlling the HBO, the RL environment was modeled as follows. Given the HBO dynamics at a fixed position in the storage ring can be approximated by an harmonic oscillator, the position x and its derivative $\dot{x}$ are sufficient to have full knowledge of the state of the system. In a discrete-time setting, the derivative can be computed from the time difference of two consecutive samples.

$$\dot{x}(t) = \frac{x(t) - x(t-1)}{\Delta t} \tag{5.5}$$

This in turn means that the two latest position samples, $x(t)$ and $x(t-1)$, are also a full representation of the system's state. In practice, though, only having two values is subject to measurement noise. Thus, the observation vector was defined as the last eight positions $\vec{o}_t = (x(t), x(t-1), ..., x(t-7))^T$. The signal is sampled at the revolution frequency, i. e. ca. 2.7 MHz, thus eight samples span roughly two periods of the betatron oscillation.

The action is a force that is applied to the bunch through a stripline kicker. In the harmonic oscillator model, this corresponds to a driving force. Under this definition, the system is a MDP.

Several different reward definitions were chosen, and the respective performance of the final agents are compared in section 5.6.1. All rewards studied in this paper penalize the agent when the x position differs from zero, corresponding to the reference orbit.

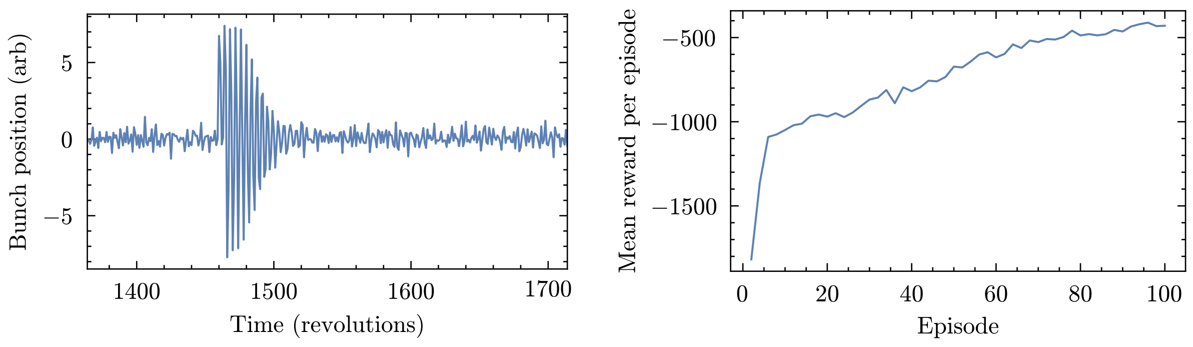

Figure 5.10.: Simulated training episode (left) and reward plot (right) showing that a PPO agent can control a simulated version of the HBO.

5.4.1. Simulation study

In order to study the interaction of an agent with the HBO, an environment based on the Gymnasium library [49] was developed. The dynamics was modeled as a damped harmonic oscillator with user selectable undamped angular frequency ω_0 and damping ratio Γ. The environment stores the actions a_i performed on the system and convolves this vector with the Green's function $B(t, t')$ of the damped harmonic oscillator

$$B(t, t') = \Theta(t - t') \frac{e^{-\Gamma(t-t')}}{\omega} \sin \omega(t - t'), \tag{5.6}$$

with $\omega = \sqrt{\omega_0^2 - \Gamma^2}$. As such, the position $x(t)$ is computed as

$$x(t) = \sum_i B(t, (i + \Delta\tau)T_{\text{rev}})\, a_i, \tag{5.7}$$

where $\Theta(t)$ is the Heaviside step function and T_{rev} is the revolution period of the accelerator. An additional user selectable delay $\Delta\tau$ is added to the argument of the function to study the effect of latency. Gaussian noise is added to the samples, reflecting the behavior of real-life data. A kick of intensity one order of magnitude higher than what the agent can perform is applied at a random time to simulate the external kicker.

In order to guide the selection of an RL algorithm and agent structure, the environment was used to test the training performance with the algorithms available in the Stable-baselines3 library [47]. This is necessary in order to disentangle a hardware platform failure from an issue with the RL problem formulation, in the case control could not be achieved during the tests at KARA. PPO and the observation vector definition of section 5.4 were thus validated in simulation, as shown in figure 5.10, before testing the complete system on the accelerator. The other algorithms discussed in section 2.2.4, i.e. SAC, DDPG, and TD3, did not train a functional agent.

5.5. PPO policy implementation on KINGFISHER

The algorithm used for this experiment is PPO [40]. This choice was dictated by its stability with respect to the change of hyperparameters. The reduced sample efficiency compared to off-policy algorithms like SAC does not affect the current application, as it is counterbalanced by the high experience collection rate. Other RL algorithms are nonetheless easy to integrate thanks to the experience accumulator architecture.

The actor network is implemented in the AIE. The employed PPO algorithm implementation uses a NN to select the mean value of a Gaussian distribution, from which the action applied to the environment is chosen. The standard deviation of the probability distribution is a trainable parameter, that is updated together with the NN coefficients.

A schematic of the internal data processing within the actor and the KINGFISHER platform is shown in figure 5.11. The first AIE tile implements a circular buffer and streams the latest eight samples to the following kernel using the cascade stream interface. These eight samples represent the observation vector, the choice of which will be described more thoroughly in section 5.4. The next kernel implements the linear layer of a NN computing the values of the sixteen hidden neurons. A ReLU activation function is applied to the outputs. Such an activation function was selected because of its simple implementation. This function can also be turned off while the system is still running in order to implement FIR filters as in section 5.3. The output of the network is then computed with a final linear layer and passed to the last output kernel. All these kernels keep forwarding the eight input values. The last kernel takes 16 values from uniformly-distributed random number data stream and sums them, producing an approximately Gaussian-distributed sample due to the central limit theorem. This is used to add a Gaussian noise with a standard deviation selectable at run-time to the output of the network. A final data vector containing the eight input values, the random value, and the output of the network previous to the noise addition is given to the experience accumulator logic.

A computer in the control room then fetches the data and uses a modified version of the Stable-baselines3 library [47] PPO [40] implementation to train the policy using the hyper-parameters in table 5.2. The new parameters are then loaded onto the AIE kernel at run-time and new data for training is gathered. The whole inference loop has a latency of 2.8 µs. For this work, a number of 2048 action steps was chosen. This number has been manually selected in such a way that the agent would not have enough time to cause beam loss and disrupt operation.

The critic network was chosen to be identical to the actor network. The remaining hyper-parameters are available in table 5.2.

5.6. Reinforcement learning based control

The system discussed in this study was allowed to interact with the accelerator for 2048 revolutions (corresponding to 784 μs). During this period an external kicker excited the

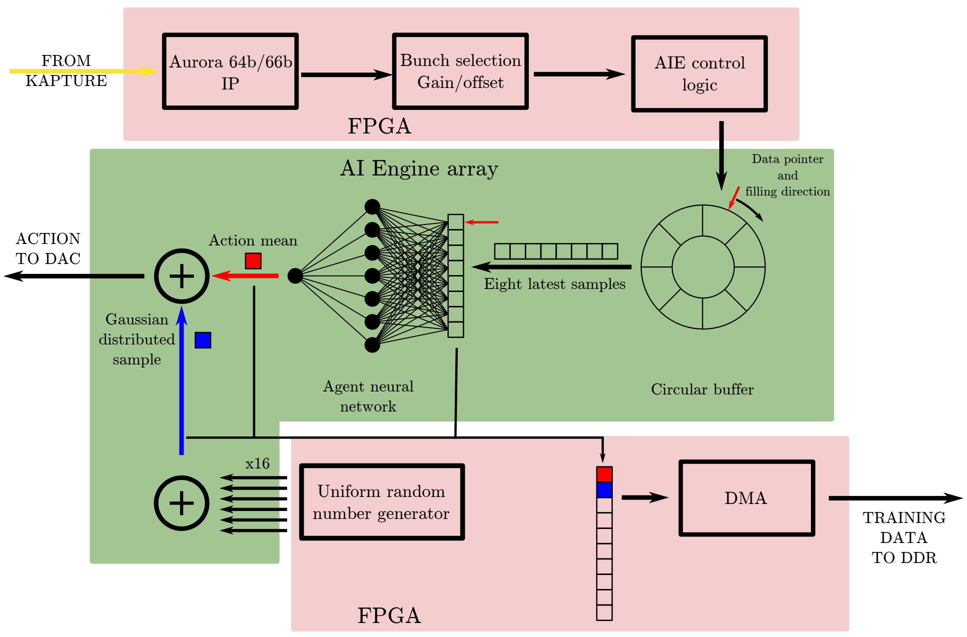

Figure 5.11.: Schematic representation of the data path within the Versal VCK190 firmware. In the top part, the KAPTURE data stream coming from HighFlex 2 is decoded by an Aurora IP from the protocol used for fiber-optic communication. The bunch of interest for the control experiment is selected and gain/offset correction are applied. FPGA logic takes care of forwarding data and gracefully stopping the AIE. In the middle, a buffer stores the latest eight samples and feeds them to the NN. The exploration noise is added and the action is then output to a DAC, for control of the kicker. At the bottom, an FPGA block takes the latest data from the inference and stores them in memory through a DMA for later training.

Hyper-parameter	Value
Learning rate η	0.0012
Discount rate γ	0.99
Number of steps	2048
Batch size	64
Number of epochs	1

Table 5.2.: Hyper-parameters used for the PPO algorithm.

oscillations as shown in figure 5.12. After each of these episodes, a training step was performed, updating the coefficients of the NN. The new set of weights and biases were uploaded to the agent and the operation was repeated.

In order to study the evolution of the oscillation amplitude, the amplitude of the oscillation was obtained as the absolute value of the Hilbert transform of the raw oscillation signal $x(t)$. As shown in figure 5.12 an increase in the damping rate of the oscillations is an indication that the agent in question is achieving control of the environment. Moreover, it is possible to examine the trend of the cumulative reward obtained during each training episode as shown in figure 5.13. A clear increase in the obtained cumulative reward is visible as more episodes are used for training. This clearly shows that the agent improves with experience, as it is expected.

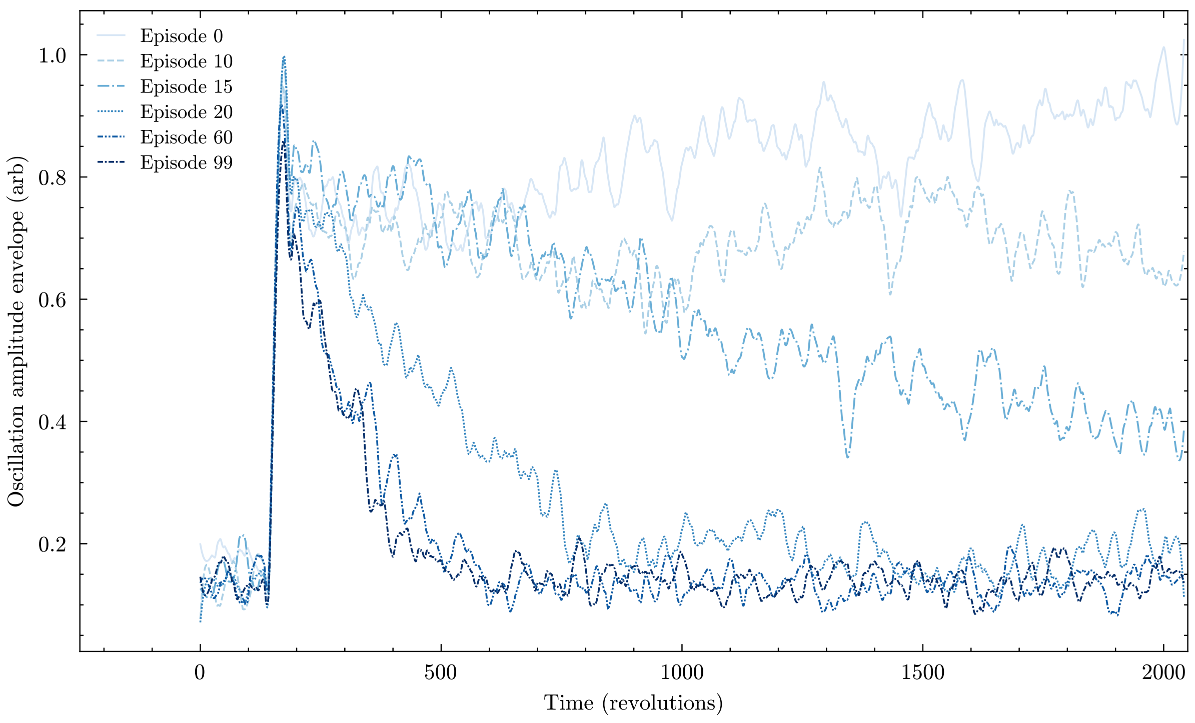

Figure 5.12.: Smoothed envelope of oscillations measured by a BPM. The sharp increase around $t = 150$ revolutions is due to the injection kicker emulating an instantaneous external excitation. Notice how at step 0 the randomly selected agent is destabilizing the beam, leading to an increase of the oscillation amplitude. Moreover, the rate of damping increases with the number of training steps, i.e. with the agent experience.

Several different training configurations were tested, each one with a different reward definition and the number of neurons in the hidden layer of the actor. This led to agents with different performances. An example of training configuration is L2, 12 N, meaning the L2 norm defined in table 5.3 is used, together with an actor having 12 neurons in the hidden layer.

5.6.1. Training-time reward definition

Figure 5.14 compares the performance of different reward functions employed during training. To do so, the oscillation amplitude was fitted with an exponential function

$$f(t; A, \lambda) = Ae^{-t\lambda} \tag{5.8}$$

and the damping rate λ was employed as a reward independent metric. Provided x is the position obtained from the BPM, the reward functions definitions are listed in table 5.3.

The agents trained with all of these three reward choices reached a final performance better than the FIR controller and the baseline with the untrained agents.

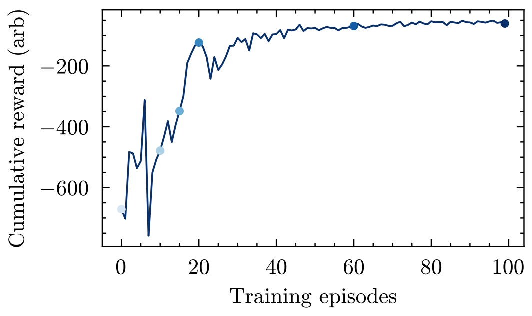

Figure 5.13.: Cumulative reward obtained by an agent as a function of the number of training episodes. It can be seen how experience is gained (i.e. more episodes are used for training) and eventually plateaus. The colored points correspond to the episodes depicted in figure 5.12.

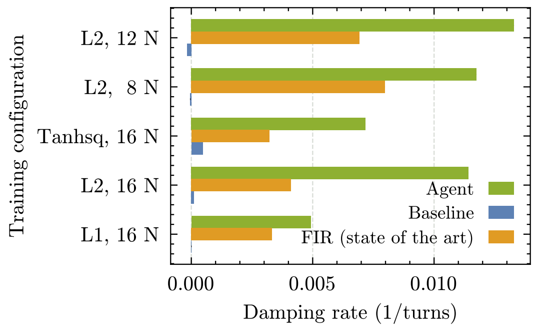

Figure 5.14.: Damping rates of an instantaneous external excitation that is achieved for different kinds of training parameters. The y-axis denotes the reward function used for the specific training according to table 5.3 and the number of neurons in the hidden layer. The trained and untrained (baseline) RL agents are compared against an FIR controller. Note that the negative baseline values are caused by the agents with random coefficients actually exciting the instability. All agents outperform the state-of-the-art FIR controller showing higher damping rates. The FIR performance varies from case to case because the feedback signal linearly depends on the bunch current, so at lower currents a weaker action is applied.

Reward name	Definition
L1	$-\lvert x\rvert$
L2	$-x^2$
Tanhsq	$-\tanh\left(x^2\right)$

Table 5.3.: Definition of the different reward functions used experimentally, where x denotes the transverse horizontal position of the beam read by the BPM.

5.6.2. Online neural network reconfiguration

In order to further increase the level of flexibility of the system, the possibility of dynamically modifying the NN structure without the need of re-implementing or re-packaging the firmware was implemented. This was achieved by embedding a smaller network into one with a greater number of neurons and layers by appropriately switching off different weights. Additionally, maintaining the number of computations constant allows to have identical latency between different training trials, thus removing this variability when comparing different agents.

Agents with several different layer sizes have been trained and their performance is shown in figure 5.14. The performance of all agents increased with training, outperforming the traditional FIR controller. The best performing agent, with 12 neurons in the hidden layer and trained with an L2-norm reward (in short notation: L2, 12 N; cf. also table 5.3), is used throughout the rest of this work for comparison with classical control techniques.

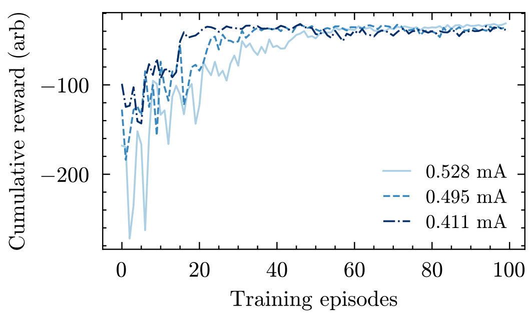

Figure 5.15.: Comparison of different training episodes with different beam current conditions, where the final agents achieve almost identical performances.

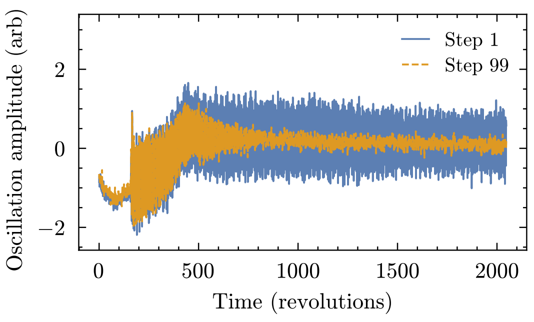

Figure 5.16.: Evolution of the horizontal position of the beam after an instantaneous external excitation at time $t \approx 175$ revolutions, influenced by an RL agent before and after training. The septum magnet was active, inducing a baseline shift superimposed on the usual exponential decay. Nonetheless, the trained agent (orange) is capable of damping the oscillation compared to the untrained one (blue).

5.6.3. Training stability and robustness

The training procedure was repeated several times, with the same setting but a different beam current, to study the stability of the agents produced. One would expect an effect for two reasons. First of all the BPM signal is not normalized, so the amplitude will vary with current. Second of all the HBO tune is current dependent [18]. Despite a 20% reduction in beam current due to the natural decay of the beam, all resulting agents achieve a very similar final reward, as shown in figure 5.15. This shows the robustness of the RL agent against variations of current.

One of the main components of the KARA storage ring injection line is the injection septum magnet. Its impulse activation is necessary to guarantee the injection of the electron bunch coming from the booster into the main ring. The leaking magnetic field, though, also affects the beam that is already in the storage ring. This effect is visible in figure 5.16, which corresponds to a shift in the position of the beam. Such an effect was not present in the simulation, but it was nonetheless possible to train an agent capable of correctly handling this new phenomenon. This is an example of the versatility and adaptability of RL algorithms, that are sometimes able to autonomously learn from situations they are not originally designed for.

5.6.4. Improvement during cumulative reward plateau

As can be seen in figure 5.15, the cumulative reward reaches a plateau around step number 50. Nonetheless, if one studies the trend of the damping rate measured at a BPM in a different part of the ring, it is still possible to observe an increase of the damping rate even around step 100 as shown in figure 5.17. This is due to the fact that noise in the input data adds an offset to the cumulative reward that hides small improvements. Such a phenomenon needs to be

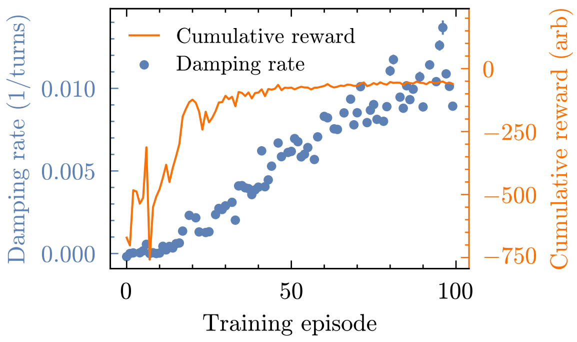

Figure 5.17.: Damping rate (blue) measured with a different BPM system as a function of the training step, compared with the reward function (orange). Notice the absence of a plateau in the damping rate curve.

considered in future experiments as it could potentially hinder further improvement of the agent.

5.7. Performance of the controller

Several working agents were trained and their performance can be compared and evaluated.

As can be seen in figure 5.14, the performance of the FIR controller is not constant. This behavior is mainly due to variations in the beam current that, due to its linear nature, affects the action signal amplitude. This is not the case for the RL agent, as it is able to automatically adapt to the variation in beam current. Thus, the trained agent outperformed the FIR controller in all of our evaluations. Additionally, the untrained RL agent is shown as the baseline, with clearly inferior performances when compared to the trained agent and the FIR controller.

Compared to the FIR controllers conventionally used, NN agents are capable of exhibiting non-linear output response. This allows the implementation of more complex policies. Particularly shallow NNs with ReLU activation functions have been shown to behave linearly in some cases. When pure sinusoidal input is fed into a black-box, the non-linear behavior produces harmonics of the fundamental sinusoidal input. The total harmonic distortion plus noise (THD+N), is defined as

$$\text{THD+N} = \frac{\sqrt{A_N^2 + \sum_{i=2}^{\infty} A_i^2}}{A_1}, \tag{5.9}$$

where A_i is the amplitude of the i-th harmonic, where $i = 1$ is the fundamental, and A_N is the noise amplitude. This metric expresses the amount of non-linear components in the output of a given device. For a linear controller, like a FIR filter, in the case where noise is negligible, THD+N is approximately zero. The THD+N was computed for different amplitudes of a sinusoidal input at the betatron oscillation frequency. The behavior for the L2, 12 N agent is shown in figure 5.18. The amount of non-linear harmonic content is consistent, with a steep increase at a level compatible with the noise floor of the signal provided to the agent. This might indicate that the agent is learning to apply more complex actions in the case of high-amplitude, and thus highly penalized, observations.

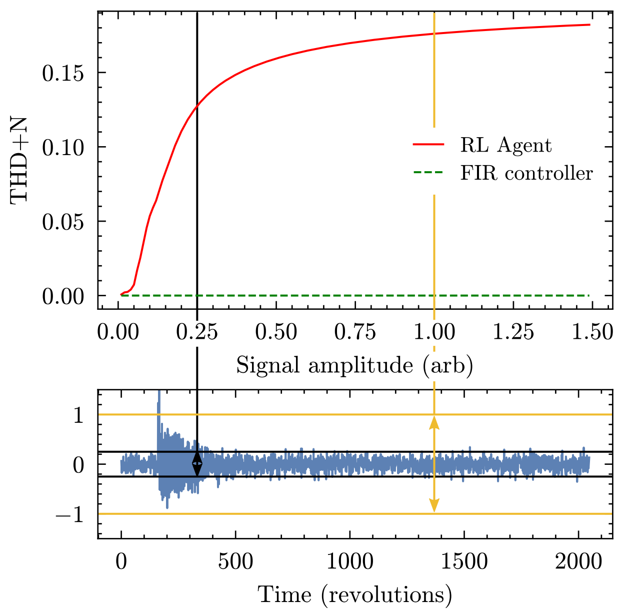

Figure 5.18.: Total harmonic distortion plus noise (THD+N) as a function of the input signal amplitude for an (L2-norm, 12 neurons) agent. Note that the FIR filter with negligible noise will have a (THD+N) close to 0. In the bottom plot, a training signal is shown, allowing to determine which part of the training episode employs a higher non-linear behavior. The noise level is indicated with the black lines, while a higher amplitude is shown with the yellow lines. Signals above the noise level tend to produce more non-linear actions.

The trade-off between training through interaction with a simulated or real-world environment is an important aspect of the application of RL to large-scale facilities. A simulation-driven approach is sometimes necessary to ensure safety, both of the facility and its personnel, while in other cases, it is dictated by the time necessary to obtain the training dataset. One advantage of real-world training is that trained agents are directly transferable into operation. This is not the case for agents trained on simulation, as their transfer to the real world is potentially hindered by non-modeled phenomena and, more in general, differences between the training and real-world environments. This work presents the opportunity to compare these two approaches. To do so, the time necessary to train an agent through interaction with the accelerator and simulation is reported in table 5.4. It is important to consider that the online case comprised waiting for a 1 Hz trigger, controlling the kickers. These numbers comprise the overhead of transferring training data, the time necessary for NN back-propagation and the interaction time with the environment. Given both approaches used the same model, the back-propagation time can be assumed to be equal. The interaction time, on the other hand, is 17.6 s for the simulation, while the trial on the accelerator takes 0.076 s or 76 ms. It is relevant to notice how the simulation being employed, and discussed in greater detail in the method section, is lightweight while still performing two orders of magnitudes slower than gathering data on the machine. As such, approximately 50% of the real-world training time is consumed by data-access overhead. This could be reduced in future implementations of the system with several approaches: using higher speed network links paired with lower overhead network protocols, or by accelerating the training directly on the FPGA and AIE array sharing memory with the experience accumulator hardware. In conclusion, for systems with dynamics that is computationally intensive to simulate, the techniques described in this

Training platform	Interaction time	Training time
Simulation CPU	17.6 s	137 s
Simulation GPU	17.6 s	227 s
Online CPU	0.076 s	260 s

Table 5.4.: Comparison of the time necessary to perform 100 training steps for different training platforms.

article will greatly improve the time necessary for training. An environment-driven training procedure, i.e. training directly on the real-world task, becomes thus not only possible but also more flexible than a simulation study as the total deployment time would be reduced.

In certain scenarios characterized by rapid dynamics and computationally intensive simulations demanding high-performance computing clusters, it may be conceivable that training directly on the accelerator consumes less energy than utilizing simulations. Such a possibility could significantly influence the sustainability of ML methodologies.

5.8. Summary

The conventional control techniques for the HBO based on BBB feedback systems employ a FIR filters producing a phased version of the oscillation signal. The idea is to produce a kick opposite to the direction of oscillation, thus producing a stabilizing effect.

In order to perform RL control experiments, a DAC output was included in the KINGFISHER system in order to drive the stripling kicker usually employed by the commercial BBB feedback system employed at KARA. An FIR filter deployed on KINGFISHER was shown to be effective in the control of the HBO.

RL agents were then trained on the machine, testing different activation functions and comparing their performance. The result was shown to be robust and to reliably produce a working controller through the sole interaction with KARA. The time required for training and obtaining data was also compared, showing that an interaction time of 0.076 ms was sufficient for the training of a functioning policy. The simulation of the same dynamics on a CPU required 17.6 s, a two-orders of magnitude gain.

6. Reinforcement learning control of the microbunching instability

In the previous chapter, the KINGFISHER system was shown to be effective in the RL-based control of the HBO at KARA, consistently reaching and sometimes surpassing the performance of a commercial system. As discussed in section 2.1.4, when KARA is operated in short-bunch mode, microstructures can form in the longitudinal phase space leading to the emission of CSR. The self-interaction between a bunch and its own emitted CSR leads to strong fluctuations in the radiation power output with frequency components spanning from a $O(100\,\mathrm{Hz})$ to $O(100\,\mathrm{kHz})$. In [106], RL-based methods were shown to successfully control this instability in simulation. Due to the lack of a suitable low-latency hardware platform fully integrated with KARA, this approach could not be tested on a real accelerator. The KINGFISHER-based RL system that has been developed as part of this work is a perfect candidate to approach this problem, as it addresses both the latency and KARA integration issues.

The microbunching instability control problem serves not only as an ideal candidate to investigate the capabilities of this system, but also as a case study for real-time RL applications to highly non-linear control problems at particle accelerator. In turn, this opens the way to a new class of general turn-key controllers that can be applied to large-scale facilities.

6.1. Task description

The microbunching instability has a rich phenomenology. A variety of techniques have been employed for its characterization, ranging from investigating the properties of the emitted CSR by using direct methods [17, 134], or by using electro-optical means to study the time-structure of each CSR pulse [135]. Additionally, investigation of the longitudinal charge density have also been performed [22, 23].

The dynamics of the instability has been shown to depend on several machine parameters, as for example the shape of the beam pipe, such as the presence of precisely engineered corrugated structures [136]. Furthermore, the dynamics are also affected by the configuration of the magnetic lattice [19], the main RF voltage, and the longitudinal damping time [17]. A simulation suite called Inovesa was developed at the Karlsruhe Institute of Technology in order to study the microbunching instability [137, 138].

The goal is to stabilize the emitted CSR. This can be achieved by employing information on the longitudinal beam dynamics provided by one or several of the aforementioned diagnostic methods to apply feedback to a parameter that affects the instability. The two following subsections will deal with the selection of possible observables and action parameters that could be employed in a low-latency real-time RL feedback.

6.1.1. Possible observables

At KARA, there are three main properties that can be measured to obtain information on the microbunching instability: the emitted CSR, the emitted ISR, and the charge density of the bunch. Each one of these can be measured with a variety of different techniques. In order to perform continuous feedback, the selected observables must be able to be measured continuously. At KARA, the KAPTURE and KALYPSO systems described in sections 2.1.6 and 2.1.7 have this capability. Additionally, they also allow turn-by-turn diagnostic.

The CSR pulses emitted by a bunch in each turn can be detected by means of a Schottky diode, producing a signal of amplitude proportional to the energy in the pulse. These signals can then be sampled with KAPTURE [139], providing important information on the underlying dynamics. A more powerful technique, known as far-field EO, described in [135], uses the EO-sampling [22] and photonic time-stretch in order to directly sample the electric field of the THz CSR pulses. The terahertz readout sampling (THERESA) system [140] will allow continuous streaming of this data.

The ISR provides information on both the longitudinal beam size and energy spread. The former is measured using the streak camera setup currently available at KARA [141]. However, this device does not support continuous data streaming and therefore cannot be used for feedback. The energy spread, on the other hand, can be determined by measuring the horizontal beam size in a dispersive section, which can be achieved with a KALYPSO system. These observables, which are related to longitudinal beam dynamics, provide insights into microbunching instability. However, neither system currently has sufficient resolution to image microstructures [23].

KARA has the unique capability of having an EO sampling crystal in the beam pipe that can be brought in close proximity to the beam [22, 142]. This method, known as near-field EO, provides excellent information on the microstructures in the longitudinal charge distribution, and can even be employed to reconstruct the longitudinal phase space [142]. Actions on the beam, though, can lead to displacements of the physical position of a bunch that can lead it to an impact with the crystal, leading to beam loss and potential damage.

In conclusion, the diagnostic method capable of giving continuous information on the microbunching instability, and that can be readily and safely realized at KARA at the time of this thesis, is the sampling of the CSR pulses with a Schottky diode and KAPTURE.

6.1.2. Selection of action

Several systems can be used to affect the microbunching instability. The lattice of the accelerator can be modified, leading to different dynamics. As an example, using a negative-α_c

Modulation type	Modulation amplitude	Modulation frequency (f_{sync})
BBB	0.69 kV	(2 or 3)±0.2
LLRF V_{mod}	15 kV, 30 kV, or 45 kV	(2 or 3)±0.2
LLRF ϕ_{mod}	1°, 2°, or 2.5°	(2 or 3)±0.2

Table 6.1.: Different types of modulations employed during the measurements. Courtesy of A. Santamaria Garcia [143].

operation mode leads to completely different dynamics than positive-α_c [19], with low-bursting that is much more regular. Furthermore, insertion devices such as wiggler and undulators can be employed to modify the longitudinal damping time [17], affecting the low-bursting rate. At KARA, these systems do not currently allow feedback at timescales comparable to the dynamics of the microbunching instability. Modifying the lattice at those rates would require strong modulation of the magnets, a technically challenging tasks due to the high currents involved and the current stability that needs to be guaranteed. Insertion devices in some cases require moving mechanical parts, or ramping superconducting magnets to high-currents, again tasks that are challenging to perform with the required speed of tens of kHz in the current installation.

In reference [136], a systematic simulation study showed that specially designed corrugated structure affect the bursting frequency and threshold. The presence of the structures adds an impedance, and thus a wakefield, that can strongly affect the dynamics. Again, albeit promising, using this system in a low-latency feedback loop is technically challenging.

At KARA, only two systems allow acting on the longitudinal degree of freedom of a bunch at a rate higher than a few tens of kHz. The first is the longitudinal BBB feedback system is intended to mitigate coupled bunch instabilities, providing a cavity that can kick the beam longitudinally. Its effect on the microbunching instability needs to be examined. The second relies on the LLRF system, by applying modulations to the RF voltage within the accelerating cavities. The cavity voltage can be expressed as

$$V_{\mathrm{RF}}(t) = (V_{\mathrm{mod}}(t) + V_0)\sin\left(2\pi f_{\mathrm{RF}}t + \phi_0 + \phi_{\mathrm{mod}}(t)\right), \tag{6.1}$$

where $V_{\mathrm{mod}}(t)$ and $\phi_{\mathrm{mod}}(t)$ are amplitude and phase modulations. Their effect is shown in figure 6.1. Phase modulations of the RF voltage was shown to strongly affect the microbunching instability [134].

In order to select the most suitable among these approaches, a systematic study of their effect on the microbunching instability was carried out in [143]. A more detailed overview of the working principle of the LLRF system used to perform these kind of modulations will be discussed in section 6.1.3. Sinusoidal kicks were applied with the BBB feedback system with variable amplitude and frequency. Similarly, amplitude and phase modulations were applied with the LLRF. A summary of the employed frequencies and levels is shown in table 6.1.

Data was acquired with the KINGFISHER platform. An example of modulation experiment is shown in figure 6.2. The BBB has no perceivable effect, compared to the non-modulated

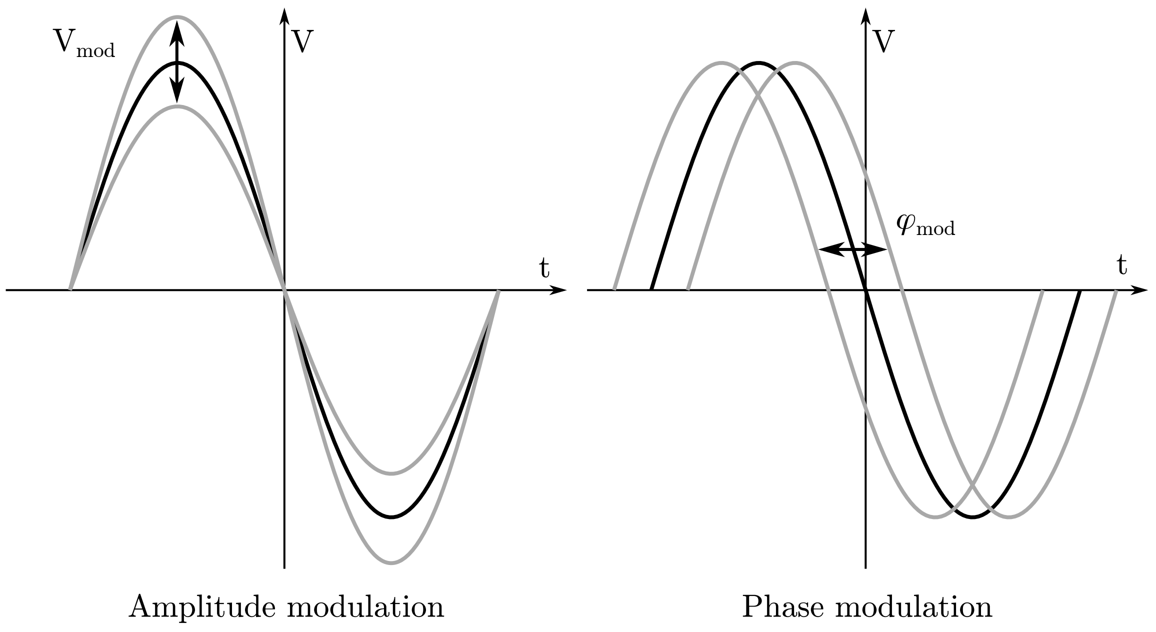

Figure 6.1.: Graph comparing amplitude and phase modulations.

case. As discussed in [106], the current setup can apply a maximum BBB longitudinal cavity voltage of $\approx$ 500 V, assuming an input power of 200 W and a shunt resistance of $\approx$ 1000 kΩ, several times lower than the LLRF. The LLRF modulations, on the other hand, disturb the microbunching instability, going as far as almost stopping the emission of CSR.

In order to investigate the systematic effect of the modulation, two parameters were extracted for each measurement: the area or integral of the FFT, and the frequency of the low-bursting instability. As shown in figure 6.3, all LLRF modulations have strong effect on the integral of the FFT, indicating a variation of the emitted CSR power with respect to the non-modulated case. The BBB system does not exhibit the same behavior, with only minor variations discernible in the frequency of the instability.

The usage of the longitudinal BBB feedback for control experiments would require less development effort for its KINGFISHER integration, as the same system described in section 5.2 can be employed [13]. Using the LLRF system as a KINGFISHER output requires more development effort, as the system was originally not intended to receive a low-latency modulation signal. Additionally, the system is safety-critical and thus special care is necessary when implementing modifications. Nonetheless, LLRF modulations have far stronger effect on the microbunching instability compared to the BBB system, that has basically none. As such, LLRF-based modulations are selected as the main feedback for control experiments.

6.1.3. Low-level radiofrequency system integration

In order to ensure proper operation of an accelerator, the RF amplitude and phase in the accelerating cavities need to be properly controlled. Several effects, such as beam loading and drifts in the system, can move the RF field away from the desired set-point parameters.

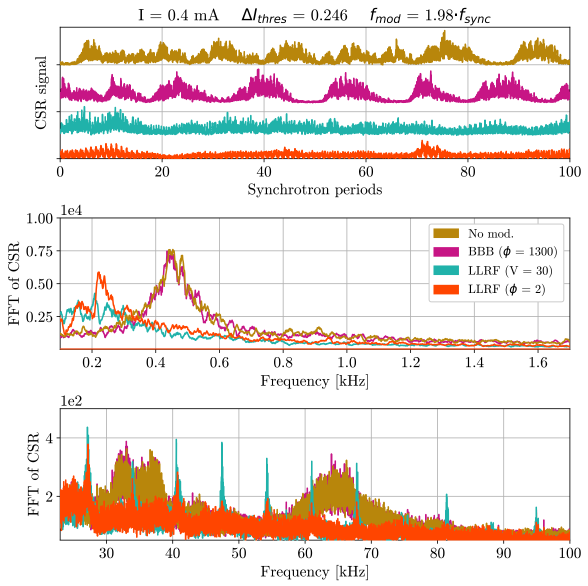

Figure 6.2.: Effect of different modulations on the microbunching instability. In the top plot the CSR time signal acquired with KAPTURE is shown. In the middle and bottom the FFT of the above signal is shown. The middle shows the low-bursting region, while the bottom shows the bursting region. Courtesy of A. Santamaria Garcia [143].

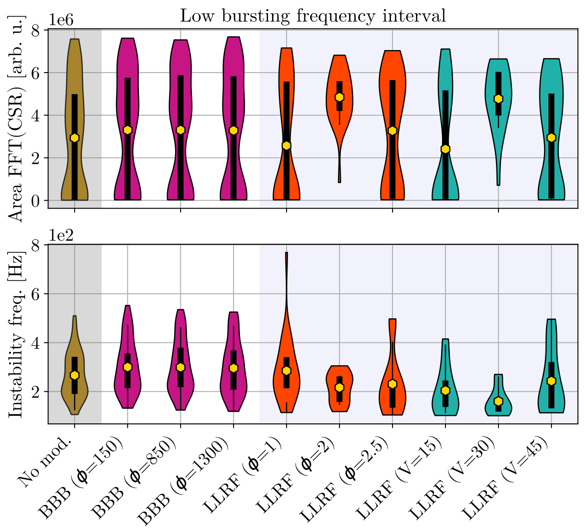

Figure 6.3.: Comparison of the distribution of the FFT integral and low-bursting frequency depending on modulation. All measurements were included in the top plot, and the third harmonic modulations were excluded from the bottom plot. The yellow marker indicates the mean of the distribution, the black box marker the interquartile range, the whiskers the 95 % confidence interval, and the vertical shape indicates the probability density Courtesy of A. Santamaria Garcia [143].

A low-level radiofrequency system implements a feedback loop that allow to stabilize the RF amplitude and phase to specific set-point values [144]. In general LLRFs deal with the low-power signals that are used to drive the high-power RF electronics. A simplified schematic of a LLRF system is shown in figure 6.4. A pick-up samples the field within the cavity. This signal is then compared with a reference, giving information on the current amplitude and phase of the field. These values are then compared with the desired set-point, producing an error signal that can be used for feedback. The amplitude and phase of the output signal are then adjusted. Such a signal is then used to drive the power-amplifiers that drive the cavities.

Two main categories of LLRF system exist: digital and analog, depending on how the signal processing is carried out. Thanks to the advancements in DSP and FPGA, low-latency digital RF controls are now usually desired thanks to their flexibility and re-configurability.

Furthermore, LLRF systems are usually a key safety component in accelerator machines. Monitoring channels allow to verify that the various elements of the high-power RF chain are not operated outside of their safe parameters. Additionally, interlock signals allow for fast inhibition of the accelerating potential in case of anomalies in the radiation safety system.

At KARA, two Dimtel LLRF9 [145] units are installed, one per RF station. A schematic of the system is presented in figure 6.5. Each station has one klystron as power amplifier, capable

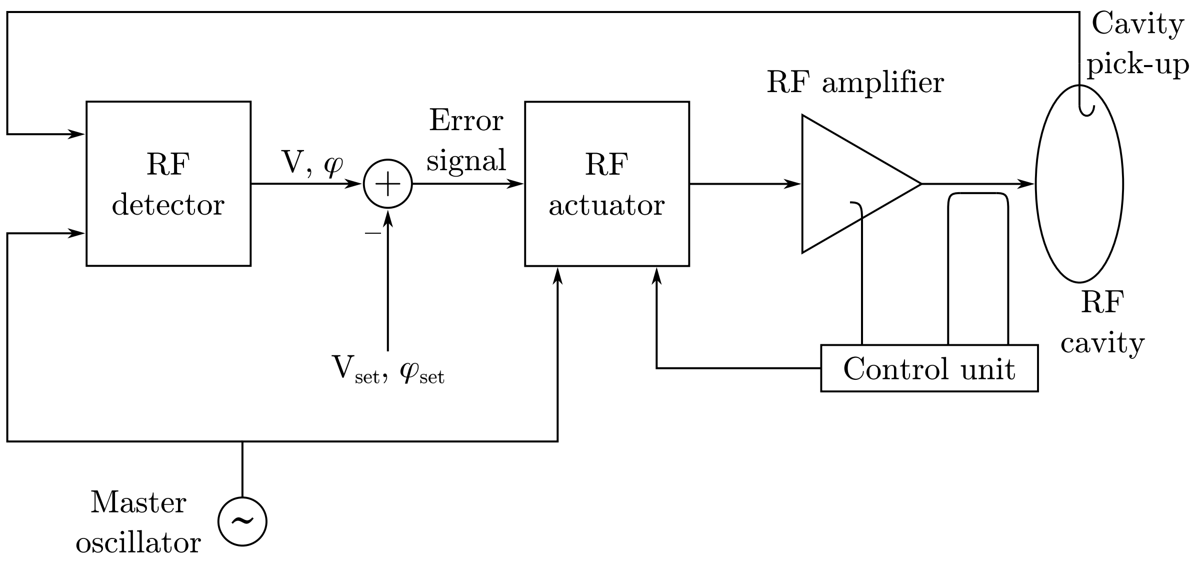

Figure 6.4.: Simplified schematic showing the working principle of a LLRF unit. The field inside of the cavity is sampled with a pick-up, and is then compared with a reference and set-point to obtain an error signal. Based on this, an actuator produces a suitable signal for the RF amplifier. A control unit monitors the reflected and forward power to the cavity, as well as the amplifier. If anomalies are detected, the RF signal is switched off.

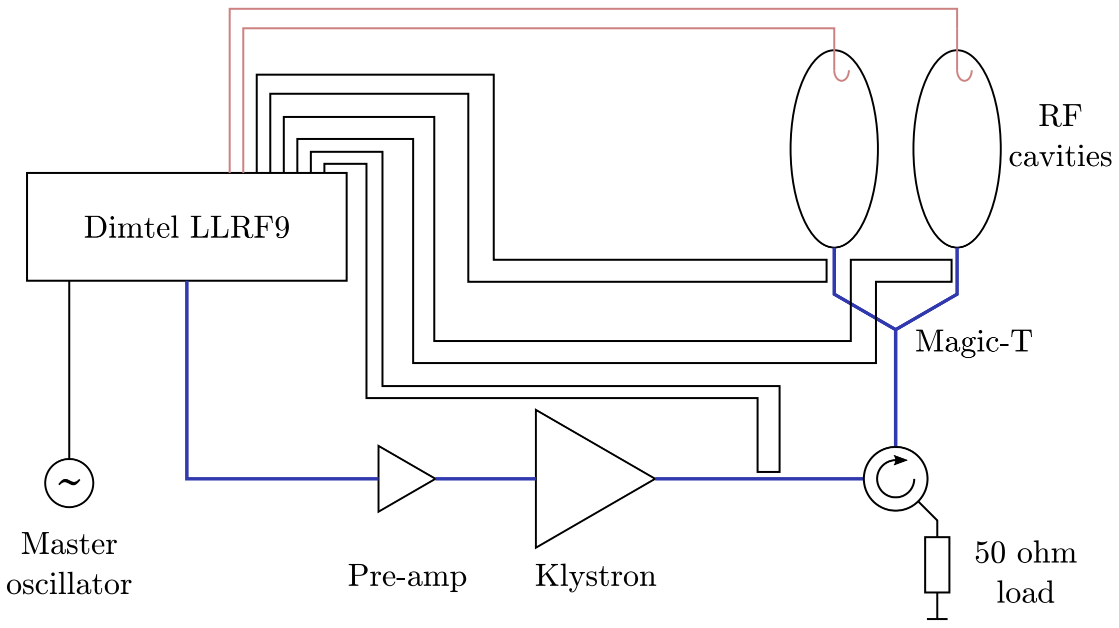

Figure 6.5.: Schematic of the KARA LLRF system. The blue lines indicate the main RF path, while the black double lines indicate the directional couplers used to measure the forward and reflected power. The pick-ups for monitoring the RL amplitude and phase in the cavities are indicated in red.

of delivering approximately 150 kW, that feeds two cavities through a circulator. This device ensures that the reflected RF power from the cavities does not damage the klystron. A "magic-T" splitter divides the power evenly among the two cavities.

The standard Dimtel LLRF9 system allows storing set-points as ramp profiles. After a trigger is registered, the system will automatically switch to the next set-point step in the ramp profile

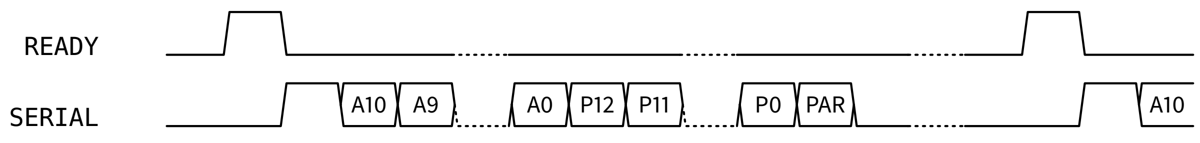

Figure 6.6.: Logic timing diagram of Dimtel LLRF9 modulation interface. A10 to A0 denote the eleven amplitude modulation bits, P12 to P0 the thirteen phase modulation bits, and PAR the parity bit.

at predetermined time intervals. Up to 512 steps can be defined. This construct allows to ramp the RF voltage, as it is usually done during acceleration, or to apply sinusoidal modulations as described in the previous section. The ramping table can be modified via EPICS, with update rates in the range of a few tens of Hz. Such a rate is not sufficient to perform feedback at the timescale of the microbunching instability.

In order to circumvent this issue, a custom digital modulation interface was implemented by Dr. Dmitry Teytelman from Dimtel, the manufacturer of the LLRF9 used at KARA. The following requirements were set, based on the experience of the previous section:

1. modulation sample rate higher than 100 kHz;
2. both amplitude and phase modulations should be possible;
3. the amplitude swing should be at least 100 kV;
4. the phase swing should be at least 2°;
5. the interface should be digital, allowing simple integration with an FPGA;
6. this system customization should have no effect on the safety critical components of the LLRF.

Items 3 and 4 where chosen based on the results of section 6.1.2 and [143].

The interface allows to add offsets to the set-points with minimal latency. As shown in figure 6.6, the protocol is based on two signals. The LLRF operates as a receiver, toggling the READY signal, implemented with the LVCMOS 3.3 V standard, signaling that to the transmitter that a modulation sample should be produced. On a LVDS line, a 26-bit word is then transferred, composed of one start bit, 24 data bits, and one parity bit. The 24 data bits are are composed of two two-complement values: a 11-bit amplitude offset, and a 13-bit phase offset. At KARA this allows a 200 kV amplitude swing and a 11.25° swing. The interface has a sample rate of $f_{\mathrm{samp}} = f_{\mathrm{RF}}/1104$, corresponding to a sample every six revolutions around KARA, or $\approx$ 453 kHz. The modulation can be enabled/disabled through EPICS. Furthermore, the interface was devised in such way that it would not interact with the safety systems of the LLRF. Due to the signaling standards that were chosen, the maximum cable length is roughly 1 m.

The ratio $f_{\mathrm{rev}}/f_{\mathrm{samp}} = 6$ is a deliberate design choice. In the originally proposed interface, this ratio was $1008/253 \approx 3.984$. Since KAPTURE generates data for each bunch at every revolution, this ratio determines how frequently an action must be applied to the LLRF in terms of KAPTURE samples. If the ratio is not an integer, a fractional decimation scheme

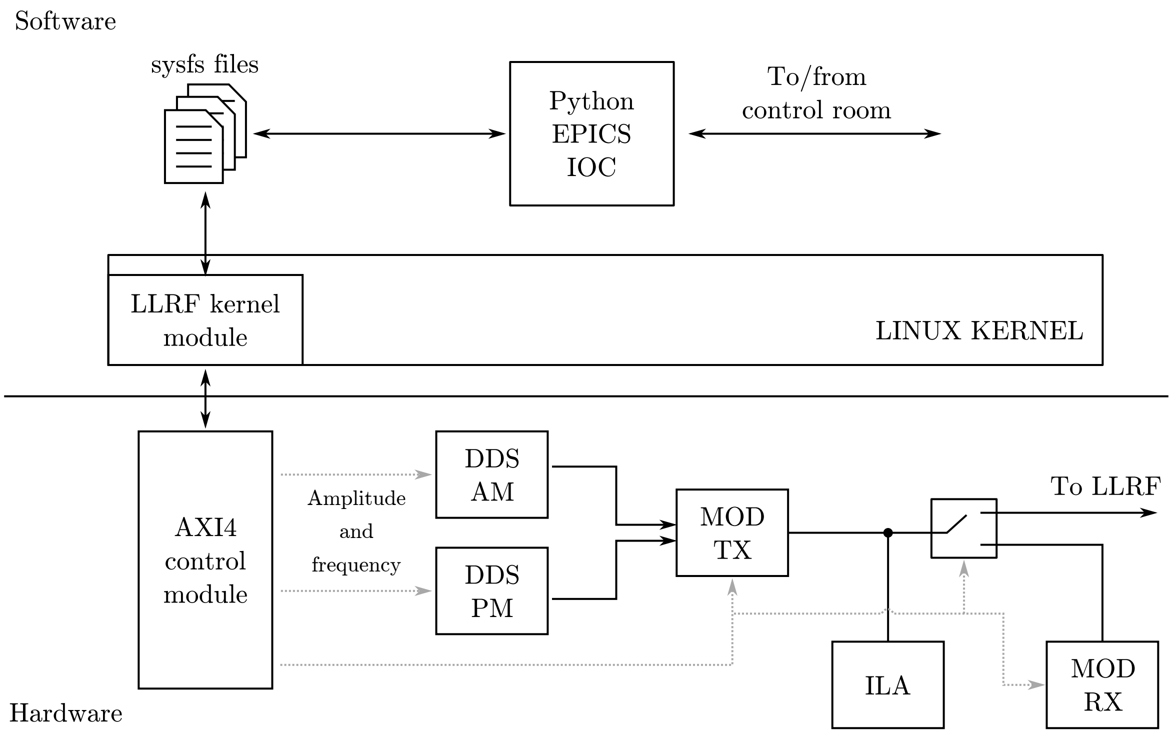

Figure 6.7.: System schematic of the LLRF modulation tester. An AXI4 based control module sets the amplitude and phases of the DDS-based sinusoidal generators, that are then fed into a transmitter IP of the LLRF interface (MOD TX).

would be needed to align the two sampling rates. Otherwise, leaving the rates unmatched would introduce latency fluctuations, as each sample would have to wait for the fractional portion of $f_{\text{rev}}/f_{\text{samp}}$ to elapse before being applied to the machine.

In order to verify the functionality of the system, a LLRF modulator controller system was designed, based on the AMD/Xilinx Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit [146]. The system allowed to apply sinusoidal amplitude and phase modulations with selectable amplitude and frequency. This allowed to test the functionality of the interface and to verify that the modulations were affecting the beam. A system schematic is shown in figure 6.7. A direct digital synthesizer (DDS) IP core was instantiated in the FPGA, allowing the production of sinusoidal samples of selectable frequency. A gain stage could be employed to set the amplitude of the modulating signal. All these parameters well controlled through an EPICS IOC that interfaced with a Linux kernel module, allowing the control through the access of files from user space. An integrated logic analyzer (ILA) IP core allows to monitor the interface for debug purposes. A receiver IP as the one in the LLRF firmware can be employed in order to verify the functionality of the interface.

The system was thus validated, showing promising results and allowing the remainder of the system integration at KARA to be carried out.

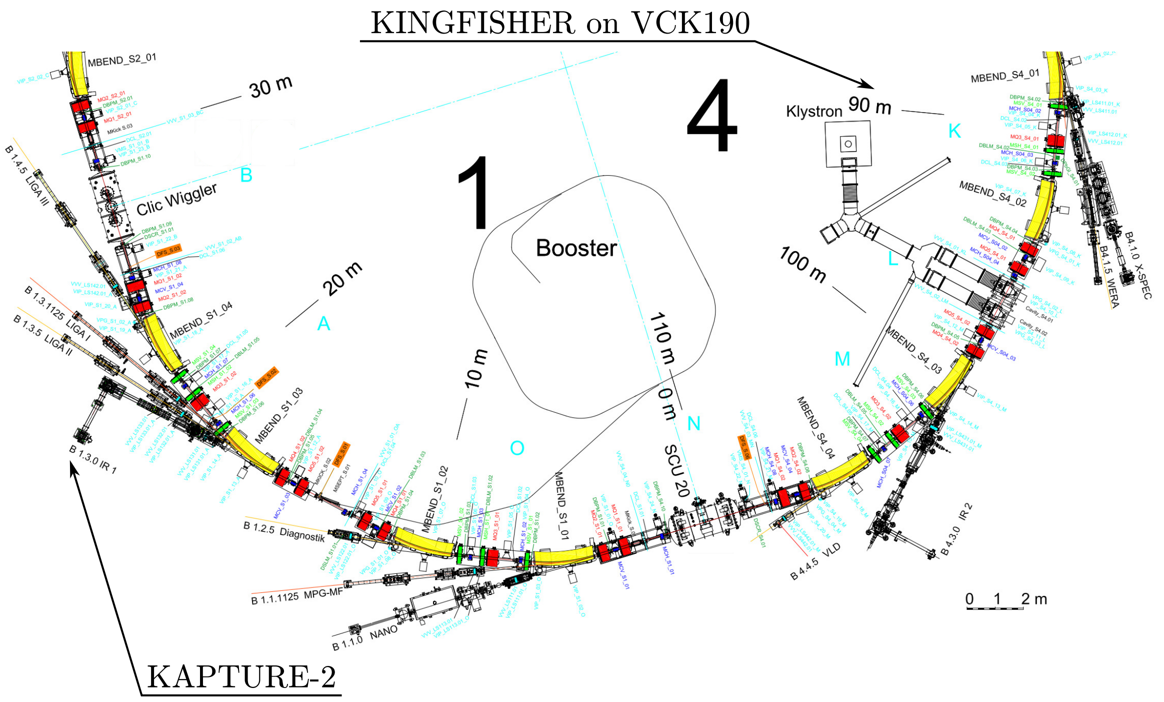

Figure 6.8.: This schematic shows the position of the two main components of the RL system installation in the KARA hall. The KARA drawing is courtesy of Ursula Herberger and Till Borkowski.

6.1.4. Final KINGFISHER installation

As discussed in section 5.2, the system was validated at KARA with a well known control problem. The link between the two devices was carried out through copper cables, with a distance of roughly 30 cm. This was possible due to the vicinity of the required equipment: the BPM signal and the power amplifiers were all in the same rack.

In the case of a LLRF feedback loop for the microbunching instability, the physical layout is more complicated. The LLRF units are located inside of the ring, thus forcing KINGFISHER to be located in close proximity to it. KAPTURE, on the other hand, needs to be in close proximity with the THz detector used to measure the CSR power. At KARA, this is usually in two infrared beamlines, denoted as IR1 and IR2. A schematic of the KARA hall is shown in figure figure 6.8. The two regions are separated by a radiation shielding wall. The straight-line distance between the beamlines and the LLRF units from $\approx$ 15 m up to $\approx$ 30 m. The medium used for the Aurora link between KAPTURE and KINGFISHER was thus changed to OM4 optical fiber, allowing this distance to be covered with minimal interference from the other system present in the hall. The LLRF unit in sector 4 was chosen for feedback, as it is in close proximity to a rack that offer optical fiber connectivity to the beamlines. The IR1 beamline was chosen due to its high time availability. Eight fibers were laid in order to have four uplink and four downlink channels, each operating at 10 Gbps. The KINGFISHER transceivers, based on the QSFP standard, were selected in the 850 nm band to match the Firefly used on KAPTURE.

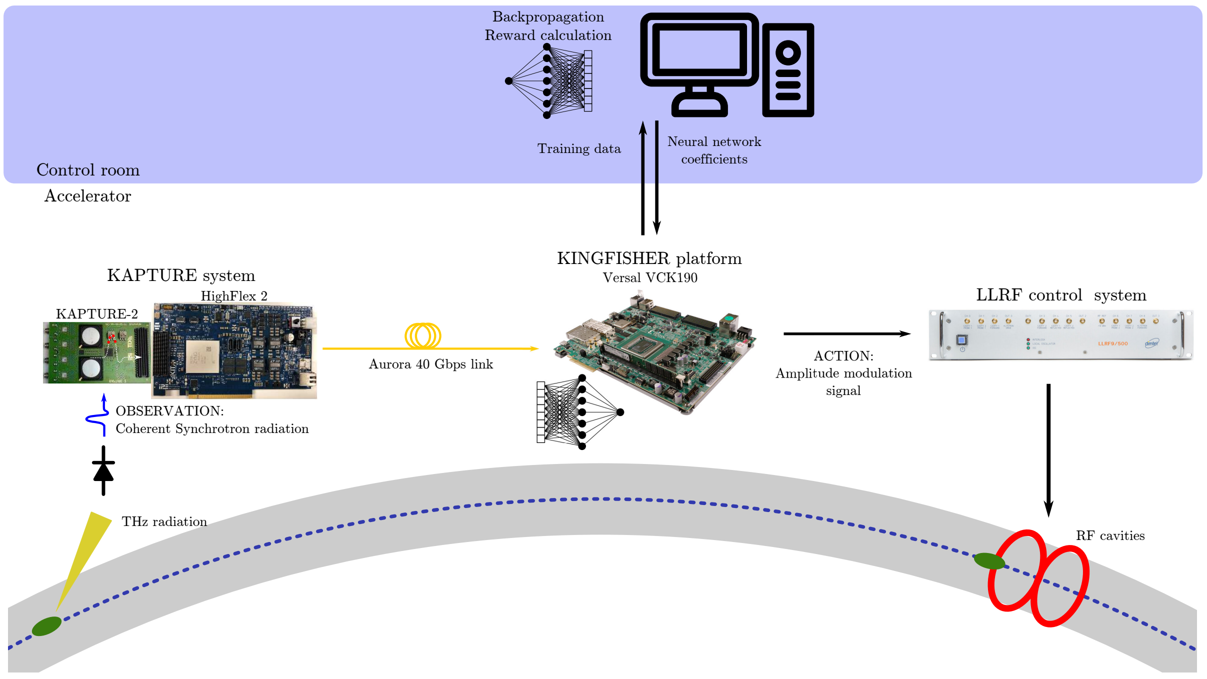

Figure 6.9.: Drawing of the hardware infrastructure employed in this work. The power of the CSR emitted by the beam is measured with a Schottky diode. The produced analog signal is sampled with KAPTURE and then forwarded by an HighFlex 2 board through high-speed links to a Versal device. The KINGFISHER system programmed on this Versal then connects this data stream into the RL controller, while applying the output action to the LLRF system, in this way modifying the RF field in the accelerating cavities. Every episode, comprised of 2048 interaction steps with the accelerator, data is sent back to the control room for training.

A loopback test was carried out in order to estimate the losses on the link and the fiber length. A latency of 1.19 μs was measured, corresponding to a fiber length of 85 m. In the 30 min time elapsed for the test, no error was observed, putting an upper bound on the bit-error rate of $< 5 \times 10^{-14}$. This would mean that, in the worst case, for a 1 s long control experiment, the probability of an error is < 0.05 %. Such a value can thus be considered satisfactory, given it would involve a single bit that is potentially for bunches that are not currently employed for feedback.

A schematic of the final setup is shown in figure 6.9. The main difference with figure 5.3 is the usage of a Schottky diode for measuring the CSR, instead of using the signal of a BPM. Additionally, instead of employing a DAC, the feedback signal was provided to the LLRF through the serial link described figure 6.6. This new feedback output allows for a stronger effect on the beam. While on one hand this is useful for experiments, on the other this increases the possibility that a controller could inadvertently kick-out the beam. Furthermore, a mis-shaped signal can lead to spikes in reflected power from the cavity, leading to the LLRF switching off the RF output to avoid damages to the hardware. This occurrence is know as a *trip* of the RF.

An additional effect that needs to be kept in mind is the strong dependency of the microbunching instability with respect to beam current. Particularly in short-bunch operation mode, the beam lifetime is degraded due to the higher particle density, enhancing losses due to intra-bunch scattering. This can lead to lifetimes in the order of 40 min, corresponding to current reduction of 5% in two minutes. These variations of bunch charge can induce to considerable changes in the dynamics of the microbunching instability in a span of minutes. Some accelerators have a full-energy injector, allowing periodic top-up of the beam that was lost. This system is not available at KARA. One alternative to this system can be obtained by leveraging the multi-bunch capability of KAPTURE, by employing a triangular filling-pattern, and selecting the bunch with current closer to the one intended for the experiment. This technique can be considered a *virtual top-up*, as it allows to select a reproducible bunch current for control experiments. Consequently, one can train agents with less variable conditions, increasing the amount of training episodes that can be performed before the dynamics changes, and in turn improving the chance that an actor will learn a successful policy.

A filling-pattern monitoring IP was designed and deployed to KINGFISHER in order to continuously monitor the CSR signal produced by KAPTURE. KARA has a filling pattern monitor based on single photon counting of the ISR emitted in the visible range by each bunch [17]. This device allows to obtain a bunch-by-bunch current measurement, that is strongly correlated with the one obtained from the KAPTURE monitor IP. Given an unknown bunch number offset is present between the two systems, one of the bunches in the train is removed through the BBB feedback system. This creates a notch in the filling pattern that can be used for aligning the two systems. In this way, the optical monitor can be employed to select the bunch with the desired current, that can the be utilized for feedback.

6.2. Feedback-based control

So far, several works have attempted to control the microbunching instability through RF modulations. In [147], a controller was devised that was capable of controlling the low-bursting at the French national synchrotron facility SOLEIL. The terahertz power signal, $P_{\mathrm{THz}}(t)$, was low-pass filtered in order to remove the oscillations due to the bursting, the resulting filtered signal is denoted as $X(T)$ and can be obtained through

$$\frac{dX(t)}{dt} = \frac{1}{\tau_{\mathrm{LP}}}\left(P_{\mathrm{THz}}(t) - X(t)\right), \tag{6.2}$$

where τ_{LP} is the time constant of the filter. The control signal

$$\Delta V(t) = G\left(X(t) - X(t-\tau)\right), \tag{6.3}$$

depending on the two parameters, gain G and delay τ, was applied as an amplitude modulation to one of the RF stations. The phase of the modulating station was selected in such a way that the bunch would arrive at the zero-crossing, instead of the synchronous phase. This ensures that the feedback is less likely to induce longitudinal oscillations, while still affecting the slope of the RF field, and consequently the bunch length. Other RF stations are kept in a nominal state, without modulation and additional phase adjustments, so that they keep providing the energy required by the beam to maintain its orbit.

A parameter scan of G and τ needs to be carried out, as a set of functioning parameters is not known a priori and they depend on the machine operating conditions. For some specific values, the low-bursting was completely eliminated, while leaving constant THz emission with periodic bursting oscillations. The current dependence of this controller was not characterized, as SOLEIL can maintain a constant current through beam top-up. Furthermore, only a single set machine parameters was tested.

An additional study was carried out at SOLEIL based on a gain-switching method [148]. The beam starts below the bursting threshold. The RF voltage is then ramped up to $V_{\max}$ such that the beam is above threshold for some amount of time $T_{\max}$. For some special combinations of $V_{\max}$ and $T_{\max}$, a strong pulse of CSR is produced when the RF voltage is reset to the previous value. This process is extremely reproducible, without the need of feedback, when the proper set of parameters is found. This phenomenon originates due to the bunch squeezing producing high charge densities during the high-voltage period. When the high-voltage is switched off a strong enhancement of the microstructures is produced.

A different approach was employed in [106]. The goal of this work was to employ an RL-based approach in simulation with the aim of stabilizing the periodic bursting at currents just above the bursting threshold. Similarly to [147], the CSR power signal was employed as an observation and a modulation signal was produced. The selected actions were the amplitude and frequency of a sinusoidal amplitude modulation. A set of eight high-level observables were extracted from the CSR signal. The first two were the mean and standard deviations of the CSR signal normalized to the uncontrolled conditions,

$$x_1(t) = \frac{\mu_{\mathrm{CSR}}(t) - \mu_{\mathrm{CSR}}^{\mathrm{init}}}{\mu_{\mathrm{CSR}}^{\mathrm{init}}}, \text{ and } x_2(t) = \frac{\sigma_{\mathrm{CSR}}(t) - \sigma_{\mathrm{CSR}}^{\mathrm{init}}}{\sigma_{\mathrm{CSR}}^{\mathrm{init}}} \tag{6.4}$$

with

$$\mu_{\mathrm{CSR}}(t) \overset{\mathrm{def}}{=} \frac{1}{n}\sum_{i=1}^{n} P_{\mathrm{CSR}}(t_i), \tag{6.5}$$

$$\sigma_{\mathrm{CSR}}(t) \overset{\mathrm{def}}{=} \frac{1}{n}\sum_{i=1}^{n} \left(P_{\mathrm{CSR}}(t_i) - \mu_{\mathrm{CSR}}(t)\right)^2 . \tag{6.6}$$

The trend of the emitted power was captured by comparing the average CSR at two different times

$$x_3(t) = \frac{2}{\pi}\arctan\left(\frac{\mu_{\mathrm{CSR,end}}(t) - \mu_{\mathrm{CSR,start}}(t)}{n}\right). \tag{6.7}$$

The next three features, x_4, x_5, and x_6, were used to capture the frequency components of the system, representing the amount of power in the maximum bin of the FFT, its normalized frequency and phase. In order to include information of the applied action, x_7 was chosen as the phase difference between the CSR signal and the action signal. Finally, the last feature is reserved for a termination condition, used to interrupt the inference of an agent if a predetermined return performance is not observed.

This approach was only tested in simulations, but could not be transferred to KARA due to the complexity of obtaining a real-time implementation of the employed features. An attempt was carried out in [74, 101], but it only achieved a simulation stage.

The three approaches discussed earlier employ different techniques. A key point to emphasize is that each study focuses on a specific aspect of the microbunching instability dynamics, resulting in three distinct control problems, each addressed with a different method. For instance, the condition selected by [106] for RL-based control, characterized by the presence of bursting oscillations but the absence of low-bursting oscillations, would be considered stable under the controller definition tested at SOLEIL [147], due to the lack of low-bursting. Conversely, the scenario deemed successfully controlled in [147] would be classified as unstable according to the definition in [106].

In conclusion, the approaches available in the literature are not generalized to the wide range of currents and conditions that are present in synchrotron light sources. Both studies, in fact, employed a single target current for their control experiments. How general the obtained controller is with respect to changes of beam current and other machine parameters remains to be determined. Thus, a broader experimental work is fundamental in order to investigate the conditions were stable control of the low-bursting, the bursting, or both, can be achieved. Furthermore, an approach capable of attacking both control problems could theoretically help bridging the gap between the two approaches. This work aims to address these aspects, as detailed in the following sections.

6.3. Study with proportional controller

The application of the controller discussed in [147] to KARA would provide insights in the generality of the approach, which is why this is explored in this section. The decay of the beam current, due to the absence of a top-up mechanism, makes the application of the technique particularly challenging. The resulting continuous variation of the experimental condition during the parameter scan needed to determine G and τ of equation (6.3) makes it different to disentagle whether a specific effect was due to the controller or resulted from the current dependence of the dynamics.

Clément Evain from the Université de Lille showed evidence that substituting equation (6.3) with a simple proportional controller of the form

$$\Delta V(t) = GX(t), \tag{6.8}$$

can in some cases still lead to successful low-bursting mitigation. Given this expression depends on a single parameter G, the parameter scan can be far quicker, reducing the effect of the current decay. An additional high-pass filter with time constant τ_{HP} was added. Given an offset in voltage has a strong effect on the dynamics, this would prevent signal offsets from entering the data processing chain and affecting the performance of the controller.

In a joint experiment in collaboration with the group of Clément Evain, the controller was deployed as a HLS IP core on KINGFISHER. A schematic of the system is shown in figure 6.10. The core managed both the storage of data via AXI4, but also the extraction of important variables such as the variance of the controlled signal. A data acquisition system was build in Python that allowed to set up the controller parameters and scans through a YAML file.

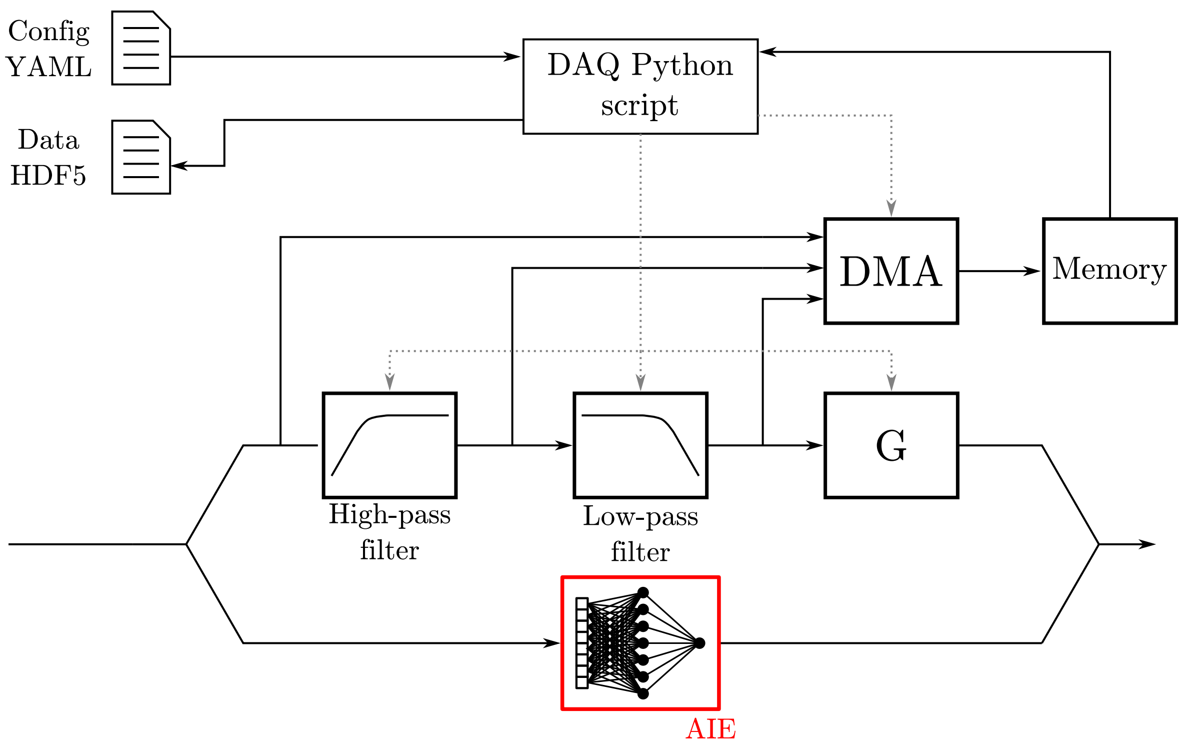

Figure 6.10.: Schematic of the data acquisition system used to implement and test the controller described by equation (6.8). The low-latency data path is parallel to the one employed for NN inference in RL experiments.

The data was stored in HDF5 file format, comprising also the state of a list of EPICS PVs as metadata. The system allowed also the online viewing of the controlled signal variance during a parameter scan.

Once deployed, the controller was frequently triggering the LLRF interlock due to the reflected power from the cavities being too high. Such an effect was attributed to the fast changes in amplitude produced by the controller. In order to mitigate this phenomenon, a slew-rate limiting stage was added to the IP core, in the form

$$\Delta V_{\mathrm{lim}}(t) = \Delta V_{\mathrm{lim}}(t-1) + \mathrm{clip}\left(\Delta V(t) - \Delta V_{\mathrm{lim}}(t-1), \pm\Delta V_{\mathrm{max}}\right), \qquad (6.9)$$

were ΔV_{max} is the maximum accepted increase in voltage allowed at every sample ($f_{\mathrm{rev}}/6$). During the experiments, 30 % of the amplitude swing was sufficient to ensure no trip, corresponding to a slew rate of $30\,\mathrm{kV}/2.2\,\mu\mathrm{s} \approx 14\,\mathrm{GV/s}$.

To bring the modulating cavity to the zero crossing, a sinusoidal amplitude modulation at twice the synchrotron frequency was applied. This frequency was selected because it induces strong oscillations in the bunch position, with low risk of beam loss. The oscillation amplitude of the bunch was measured as a function of the phase set-point. At the zero crossing, a pure amplitude modulation has no effect on the voltage experienced at the center of the bunch, as the sinusoidal component in equation (6.1) is zero. Therefore, at the zero crossing, the oscillation amplitude reaches its minimum. One such measurement is shown in figure 6.11.

A table of the machine parameters used during the experiment is shown in table 6.2.

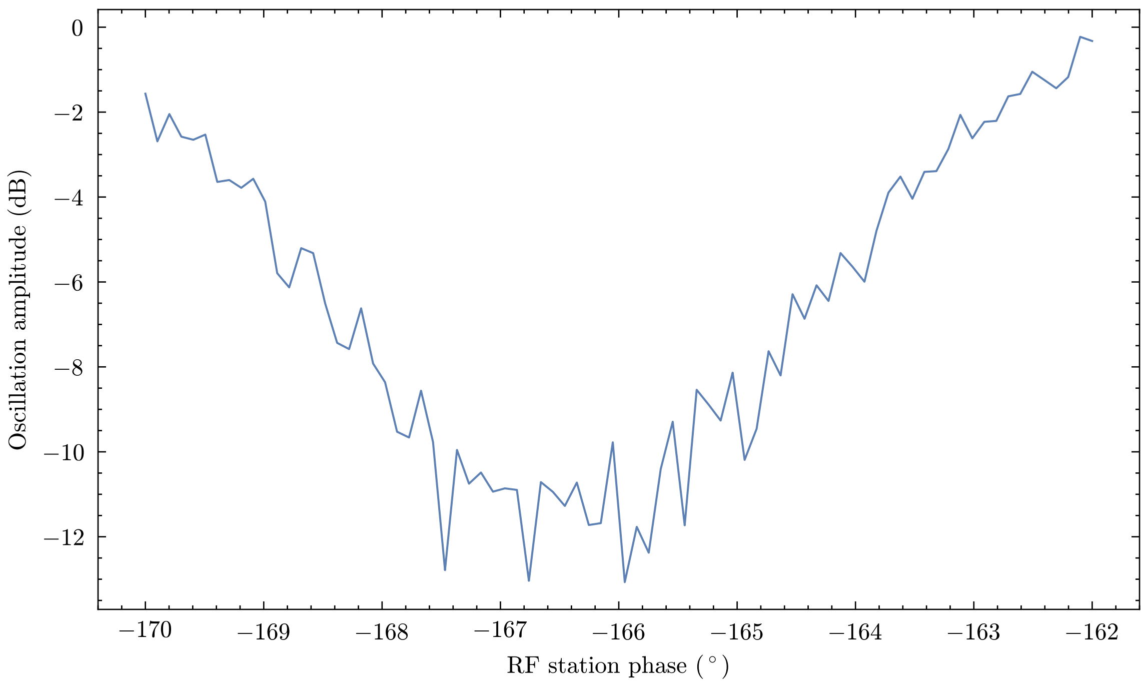

Figure 6.11.: Plot of the longitudinal bunch position oscillation amplitude as a function of the cavity RF phase. The minimum corresponds to the condition of zero crossing.

Energy	E	1.3 GeV
Total RF voltage	V_{RF}	766 kV
RF frequency	f_{RF}	499.753 MHz
Synchrotron frequency	f_{sync}	6.3 kHz
Momentum compaction factor	α_{c}	3.1×10^{-4}
Filling pattern	-	single bunch
MBI threshold (measured)	I_{thres}	0.120 mA

Table 6.2.: Machine parameters employed during the control experiments.

A series of control experiments were conducted using different controller gains and beam currents. The variance of the filtered CSR signal, $X(t)$, provides insight into the effectiveness of the controller, as a lower variance of $X(t)$ directly corresponds to reduced low-bursting behavior. To separate the effects of the controller from the natural evolution of the microbunching dynamics with changing beam current, we compared the signal variance with and without the controller, denoted as $X_{\mathrm{on}}(t)$ and $X_{\mathrm{off}}(t)$, respectively. The performance of the controller was quantified using the CSR signal attenuation, defined as

$$\text{CSR signal attenuation} = \frac{\operatorname{var}\left(X_{\mathrm{on}}(t)\right)}{\operatorname{var}\left(X_{\mathrm{off}}(t)\right)}. \tag{6.10}$$

Figure 6.12 shows the CSR signal attenuation as a function of gain and beam current. The results indicate that the effectiveness of a given gain is strongly dependent on the beam current, with distinct regions of high and low stability emerging. This pattern was consistently

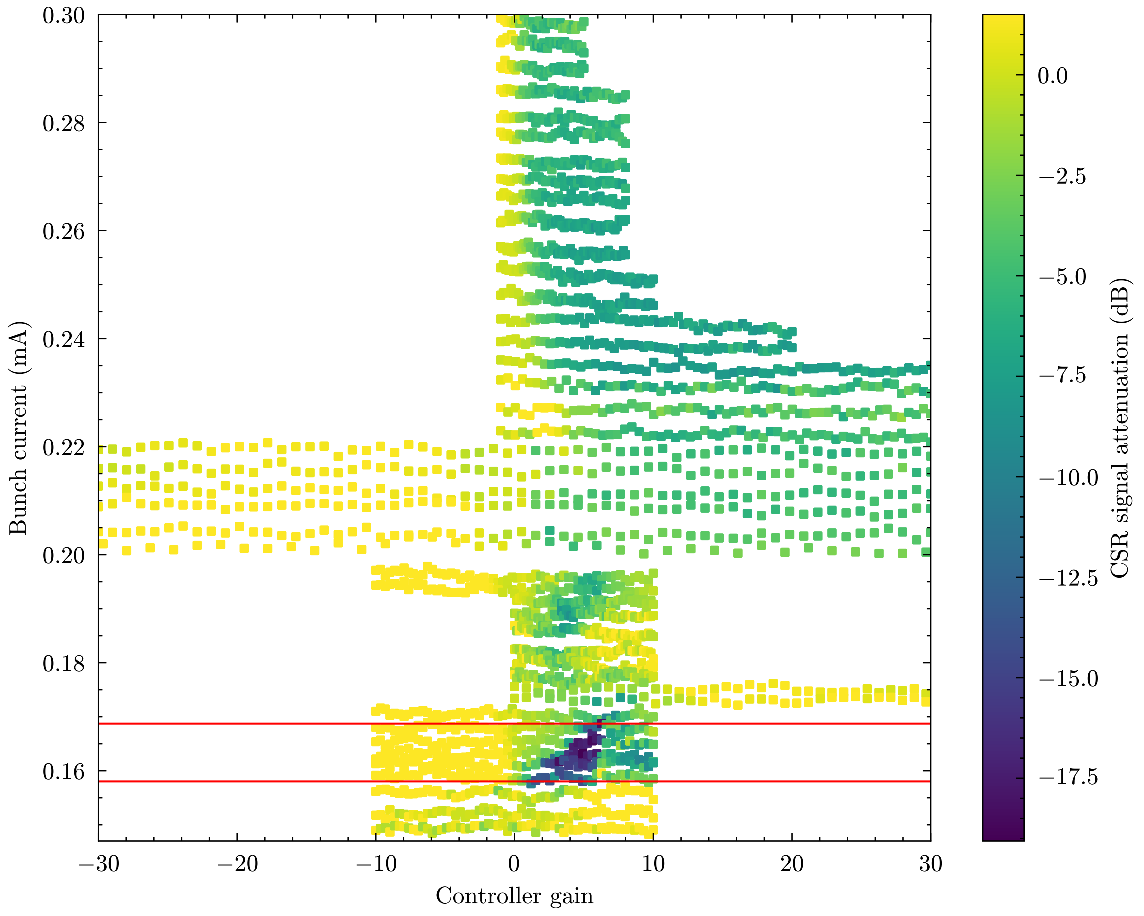

Figure 6.12.: CSR low-bursting signal variance as a function of the controller gain G and of the beam current. A current range where the controller could achieve increased stability is highlighted by the red horizontal lines.

reproduced under the same machine parameters. The current dependence of the control had not been investigated in previous control studies available in the literature.

A region where the controller achieves a high-degree of damping is found between 160 μA and 170 μA. A control experiment in this region is shown in figure 6.13. At time $t = 0$ the controller is switched on, strongly damping the low-bursting (top plot). In the bottom plot, the unfiltered signal is shown, highlighting how the bursting behavior at higher frequencies is still intact.

A comparison of the FFT of the CSR signal with and without controller is shown in figure 6.14. The low-frequency peaks, corresponding to the low-bursting, are almost fully eliminated. The bursting peak at $\approx$ 25 kHz is still present with the control, albeit being narrower. This effect is likely due to the missing modulation of the bursting by the low-bursting, leading to a narrowing of the peak.

When integrating the spectral power around the bursting peak (the region marked by grey lines in figure 6.14), the power emitted in the controlled case is 1.93 times that of the uncontrolled case. This increase can be explained by figure 6.13, which shows that in the absence of

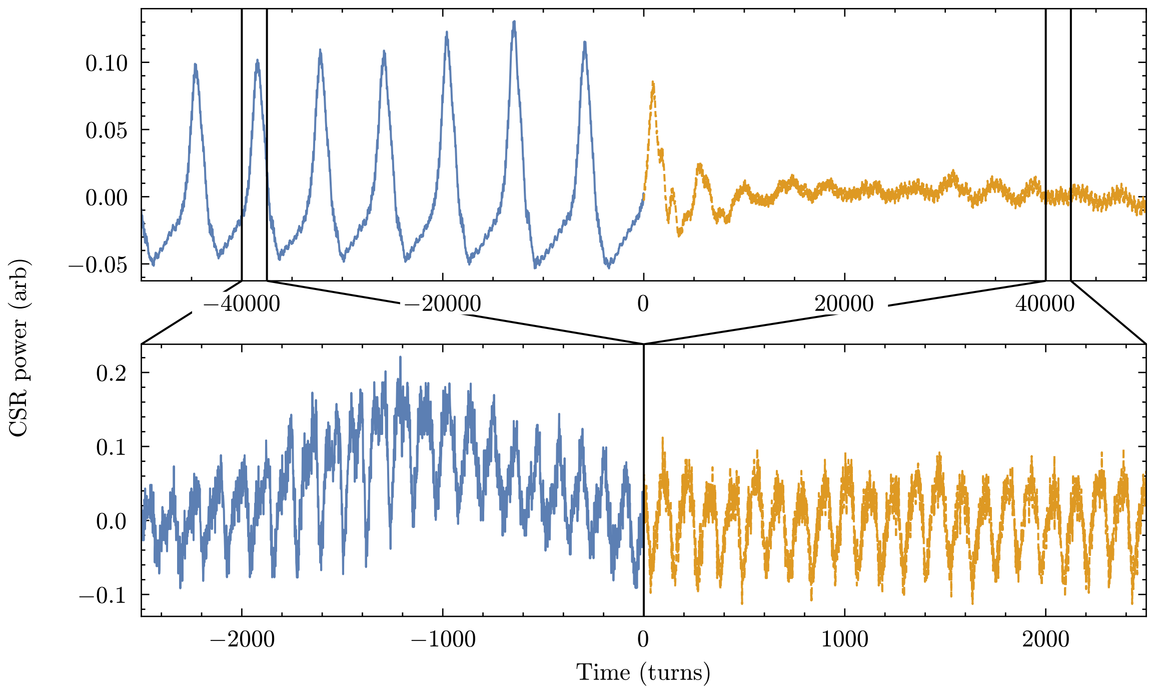

Figure 6.13.: CSR power signal as a function of time. The controller is switched on at time $t = 0$. In the top part, the low-pass filtered $X(t)$ signal is shown, highlighting the low-bursting oscillations. The bottom part shows the unfiltered signal for some time regions of the top plot, highlighting the bursting behavior.

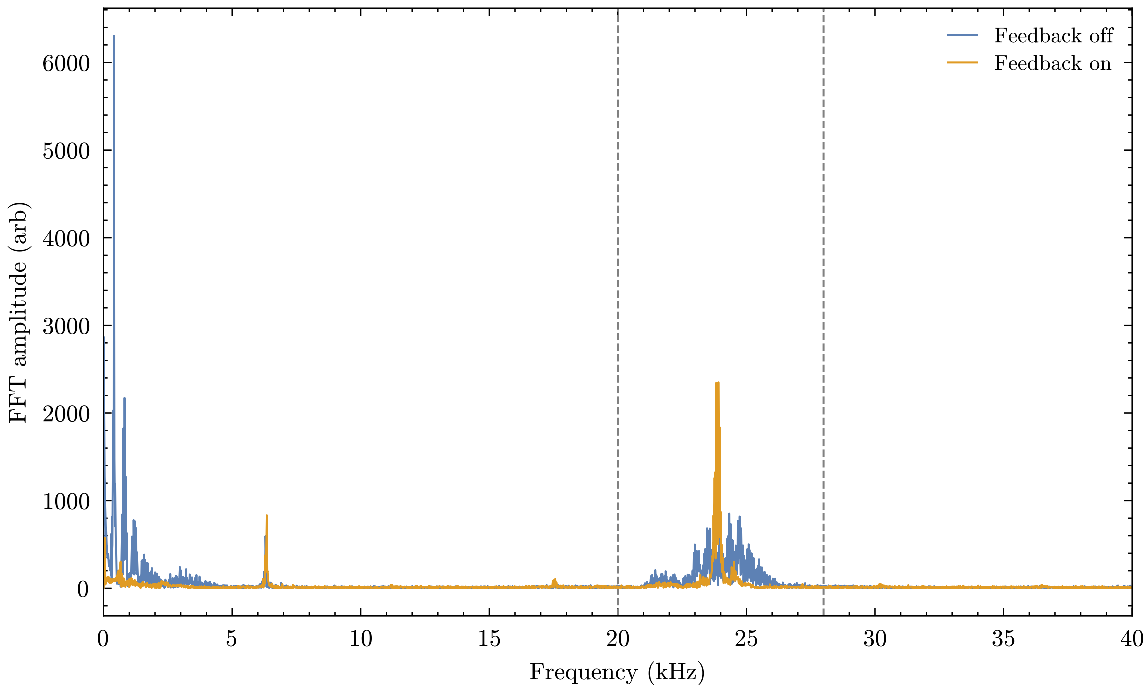

Figure 6.14.: FFT of the signal shown in the bottom part of figure 6.13. The range between zero and 5 kHz is due to the low-bursting oscillation. The region highlighted by the grey lines corresponds to the bursting behavior. The small peak around 6 kHz is due to the synchrotron oscillations.

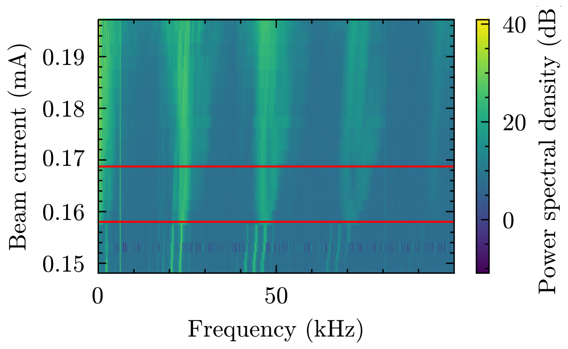

Figure 6.15.: FFT of the CSR power signal as a function of current, based on the experiments with no controller. The current interval of stability is highlighted in red.

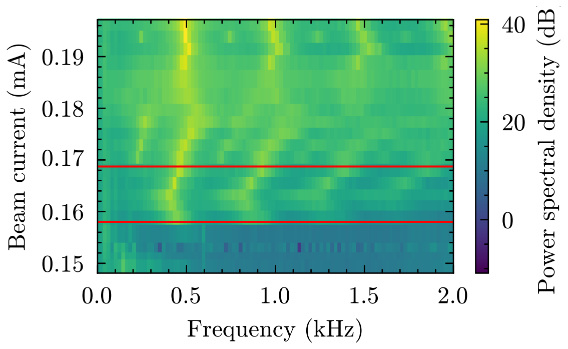

Figure 6.16.: Zoomed-in version of the FFT of the CSR power signal as a function of current, as in figure 6.15, in the low-bursting region.

the controller, low-bursting peaks exhibit an approximately 50 % duty cycle. As a result, THz radiation is not emitted for nearly half of the time. The controller does not affect the radiation intensity in the bursting region but eliminates the off-phase of the low-bursting cycle, effectively doubling the radiated power.

Figure 6.12 shows the existence of a current range, delimited by the red horizontal lines, were the controller achieves optimal performance. The FFT of the CSR signal was plotted as a function of current in figure 6.15, in complete analogy to figure 2.5. The lower limit of the stable region corresponds to a dramatic change in the microbunching instability dynamics. If the low-bursting region is examined figure 6.16, one can see how the stability region corresponds to the onset of the low-bursting. No feature is evident that can explain why the controller ceases to function at higher currents.

The functionality of the virtual top-up scheme described in the previous section, was verified, in order to perform tests in the neighborhood of the stable region. The employed filling pattern is shown in figure 6.17. A bunch current in the stable region was selected as target, and the parameter scan of figure 6.12 was repeated, leading to figure 6.18. In this case, the total beam current is shown on the y-axis. The system was capable of successfully switching between bunches while spending most of the experiments in the stability region. In some cases, no bunch was found in this region, so the next closest one is chosen, leading to the experiments with reduced stability. The bunch change boundaries are shown as horizontal dashed lines.

In conclusion a simple proportional controller can achieve a moderate level of control of the microbunching instability at KARA. Its functionality is reproducible for the same set of machine and controller parameters. The method, though, is successful only in a limited region just above the slow-bursting threshold, which motivates the use of more sophisticated methods. Furthermore, it does not affect the bursting oscillations.

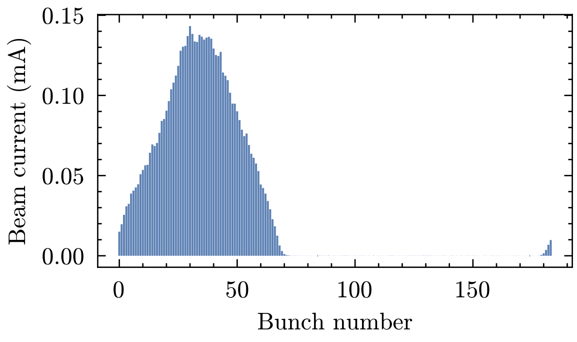

Figure 6.17.: Filling pattern used for the virtual top-up experiment. The discrepancy with the range shown in figure 6.12 is due to different offsets in the current measurement transformer, as these two experimental campaigns were carried out roughly one month apart.

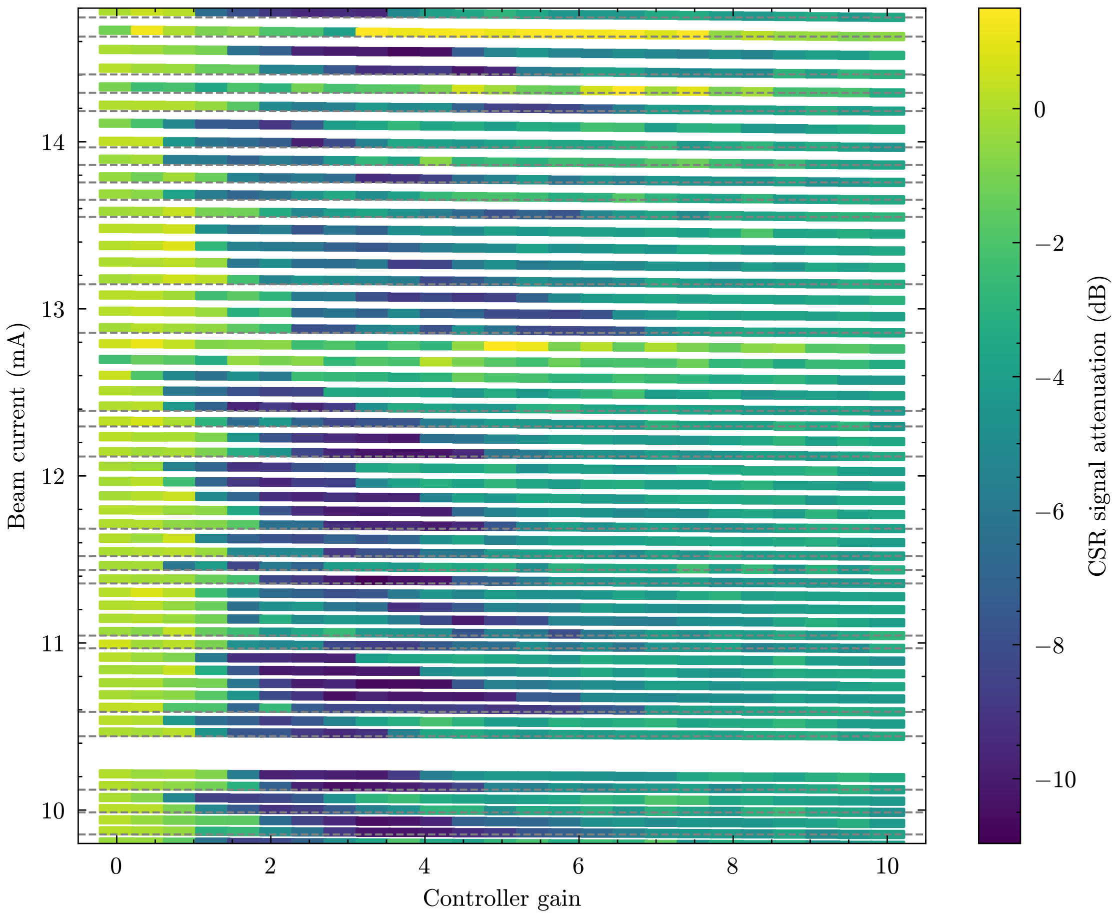

Figure 6.18.: Plot of the CSR low-bursting signal variance as a function of beam current and controller gain. In this case a target bunch current was chosen such that the controller would act on a bunch in the stable region. Compared to figure 6.12, a variation of beam current of 50 % still allows for control experiments in the stable region. The dashed lines show instances were the bunch was changed.

6.4. Formulation as a reinforcement learning task

As discussed in section 6.2, the microbunching instability is a complex phenomenon, that can be attacked as a broad range of control problems, depending on the specific aspect of the dynamics one wishes to tackle. RL-based approaches have been shown to be effective in tackling the bursting behavior, albeit in simulation. As shown in the previous section, the performance of a controller is strongly dependent on current, with current variations as low as 5% completely changing the behavior of the system. The intrinsic adaptability of the RL approach, thanks to its innate self-learning capability, makes it an ideal candidate to try and tackle the low-bursting dynamics.

In order to apply a KINGFISHER-based RL system to the microbunching control problem, the same actor structure described in section 5.4 for the control of the HBO was employed. An input layer containing the latest eight KAPTURE samples, though, would lead to an agent awareness time, as defined in section 3.4.1, in the order of 3 μs, meaning this time window is the only available information the agent can use to reconstruct the state of the system. This value is one order of magnitude smaller that the bursting period $1/25\,\text{kHz} = 40\,\mu$s. In order for a layer to fit within a single AIE tile, the maximum number of input neurons is 64. Even in this case, the awareness length would be barely sufficient to capture a single burst. As discussed in section 5.4, using several oscillations can be used to mitigate the effect of noise.

An alternative approach is to add a decimation stage as described in section 4.3.4, shown in figure 6.19. This allows to adapt the sample rate that is fed into the NN based on the timescale of the dynamic of interest. In this specific case, the interpolation stage applied to the actions has the additional benefit of smoothing the signal, avoiding reflected power spikes from triggering the safety system of the LLRF.

The output of the NN can be used to craft different action signals. The action can be directly applied as an amplitude modulation. Such an approach can be prone to fast oscillations in the action signal, specifically during the initial training parts were the coefficients of the NN are random. An alternative is to employ a so-called *delta action*. In this case, denoting the output of the NN as δ_t, the action a_t is

$$a_t = \text{clip}\left(\delta_t, +a_{t-1}, a_{\min}, a_{\max}\right), \tag{6.11}$$

where $a_{\min}$ and $a_{\max}$ are two values limiting the amount the action can grow. This approach is a commonly used in RL to avoid agents from learning absolute settings, making them more transferable to other environments. Furthermore, it reduces large fluctuations in the action if δ_t is smaller that the range $[a_{\min}, a_{\max}]$. As described in [106], this approach can lead to offsets in the modulation voltage that can have the effect of moving the threshold of the microbunching instability. As such, the method suggested in [106] and described in section 6.2 of applying a sinusoidal modulation of controllable amplitude and frequency was also implemented. The NN in this case outputs two values, one setting the amplitude and the other the frequency of a sinusoidal amplitude modulation. The system was designed such that the amplitude and frequency controls could also be set via a delta action.

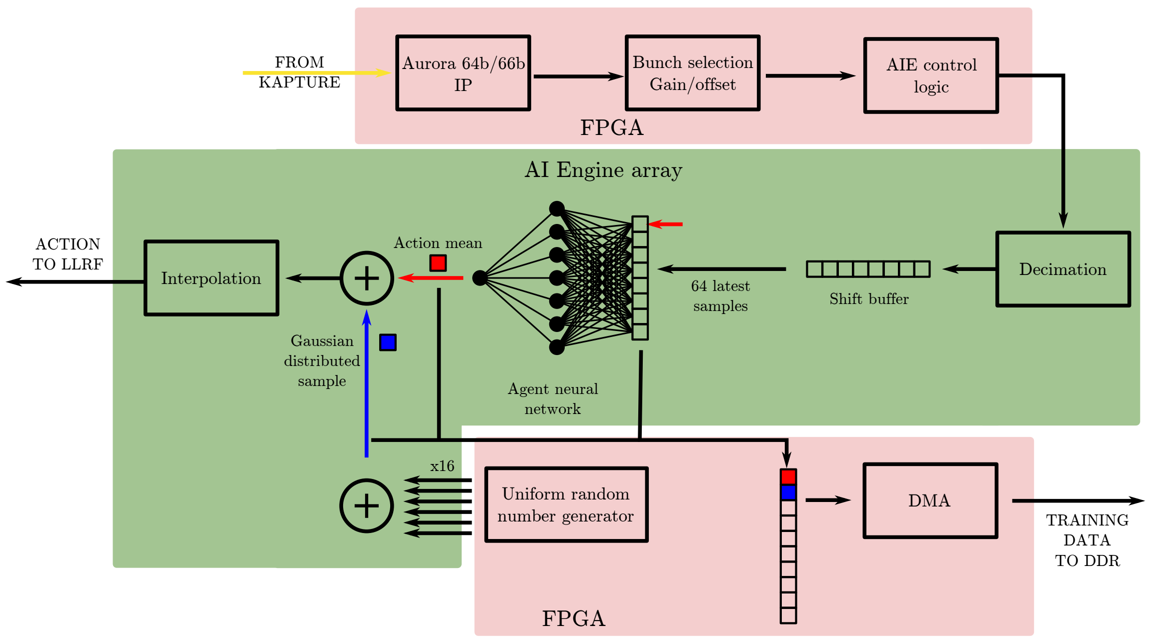

Figure 6.19.: Schematic of the firmware running on Versal™, as shown in figure 5.11. A decimation and interpolation stage are added to adapt the interaction rate with the timescale of the dynamics that needs to be controlled. Additionally, the core keeping track of the last observations was rewritten, now using a shift-register implementation instead of a circular buffer.

The training time reward definition paradigm allowed for on-the-fly reward engineering during the experimental shifts. These rewards were based on the observation vector being fed to the agent NN. Among the attempted reward functions, if x_t are the samples provided by the decimator, one can find

$$R(t) = -\left(x_t - \bar{x}\right)^2 , \tag{6.12}$$

$$R(t) = -\left|x_t - \bar{x}\right| , \tag{6.13}$$

$$R(t) = -\left(x_t - x^\star\right)^2 , \tag{6.14}$$

where $\bar{x}$ is the average of x_t during the episode, and $x^\star$ is a user-selected value. All of these function are penalizing the agent when the CSR power is fluctuating. Equations (6.12) and (6.13) are equivalent to a variance, that in the previous section was shown to capture the effectiveness of a controller. The squared version penalizes higher errors more than small errors, while the absolute value one does not. Under these definitions, though, an agent completely disrupting the CSR emission would obtain a perfect reward. To ensure this is not the case, the squared distance from a set-point, as in equation (6.14), can be used instead. In this case an agent would get penalized for completely disrupting the emission of CSR.

A more detailed discussion of the attempted reward functions is provided in section 6.6.

Hyper-parameter	Value
Learning rate η	0.0006
Discount rate γ	0.99
Number of steps	2048
Batch size	64
Number of epochs	10

Table 6.3.: Hyper-parameters used for the PPO algorithm.

6.5. Reinforcement learning based control

A first successful control experiment was carried out with the machine parameters shown in table 6.4. The decimation factor was chosen to be 96 such that the bursting high-frequency oscillations are not perceived by the agent. In this way, the agent would target the low-bursting. The reward function of equation (6.12) was employed, as it worked reliably for the HBO problem. An actor with 16 input neurons, 8 hidden neurons, and one output neuron used directly as a modulation signal were employed. The PPO algorithm was employed for training with the parameters shown in table 6.3. Due to the low beam lifetime in low-α_c operation mode, the beam current decays rapidly with a lifetime in the order of 50 min. No virtual top-up was employed in this first experiment, to understand the learning capabilities of the agent. Given the strong dependence of the microbunching instability dynamics with bunch current, a reference acquisition without agent action, i.e. with the feedback switched off, was taken after each training data acquisition. The decay of the bunch current leads to a decrease in CSR output, in turn increasing the reward. Thus, it is important to compare the agent reward with a baseline acquisitions with the feedback switched off, in order to disentangle the contribution to the reward increase due to the agent and the one due to the current decay. Three of these training episodes are shown in figure 6.20, with their corresponding reward shown in figure 6.21. In the left panel, the untrained agent is not affecting the low-bursting behavior, while the bursting is filtered out by the decimation filter, so it is present but not visible. In the no-action signal of the central panel, the bursting is still present, albeit with a lower amplitude due to the decay of the current. The RL agent managed to maintain the fluctuations at a lowered and stable level, after an initial transient. In the right panel, the current falls below the low-bursting threshold. The RL agent at this point did not train fast enough to adapt to the quickly changing dynamics, and is thus exciting the instability. Despite the varied results, it is important to notice that the RL agent consistently performs better than the no-action baseline, as shown in the accumulated reward over time (figure 6.21), except at the very end, precisely due to the transition below threshold.

The fact that the controller does not perform well at high currents might indicate a fundamental characteristic of the controllability of the microbunching instability, for which studies are so-far missing in literature. The proportional controller shown in section 6.3 also exhibits a similar behavior. Furthermore, if one considers the stability region to scale linearly with the bursting threshold, one would expect the controller stability region to fall roughly from

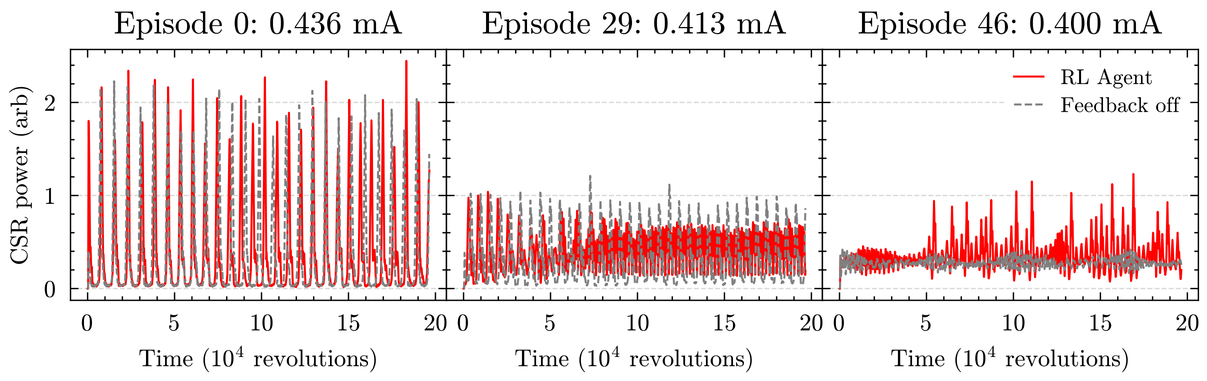

Figure 6.20.: CSR signal in three training episodes showing the strong dependence on beam current. An RL agent data acquisition (red) is compared with one without feedback (grey), in order to highlight the effect of the controller.

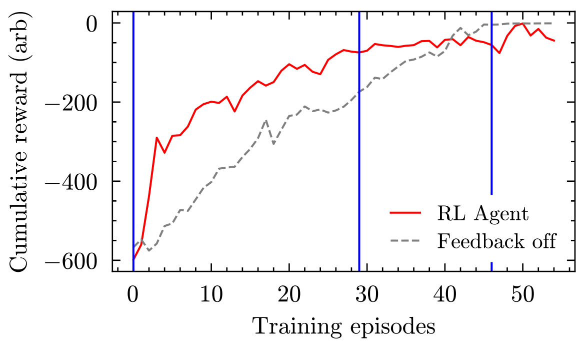

Figure 6.21.: Reward as a function of training step with and without feedback from the RL agent. The blue lines mark the episodes shown in figure 6.20.

0.425 mA and 0.390 mA for the parameters shown in table 6.4. This nicely fits with the values were RL can learn a functioning controller, as shown in figure 6.20.

Energy	E	1.3 GeV
Total RF voltage	V_{RF}	767 kV
RF frequency	f_{RF}	499.750 MHz
Synchrotron frequency	f_{sync}	9.3 kHz
Momentum compaction factor	α_c	6.7×10^{-4}
Filling pattern	-	single bunch
MBI threshold (measured)	I_{thres}	0.310 mA

Table 6.4.: Machine parameters employed during the RL control experiments.

The results of this section have been published in [14].

Delta action threshold	Amplitude frequency mode	Initial current	Final current	Is in stable region?	Reward function	Achieved control?
-	-	(mA)	(mA)	-	-	
1.0		0.715	0.669		Eq. 6.14	
1.0		0.649	0.608		Eq. 6.14	
1.0		0.584	0.548		Eq. 6.15	
1.0		0.530	0.499		Eq. 6.15	
1.0		0.493	0.463		Eq. 6.14	
10.0		0.456	0.432		Eq. 6.14	
10.0		0.425	0.402	Yes	Eq. 6.14	Yes
10.0		0.396	0.349	Yes	Eq. 6.14	
10.0		0.603	0.535		Eq. 6.14	
10.0		0.436	0.394	Yes	Eq. 6.12	Yes
10.0		0.358	0.330		Eq. 6.12	
10.0	Yes	0.754	0.521		Eq. 6.12	
10.0	Yes	0.494	0.389	Yes	Eq. 6.16	Yes

Table 6.5.: Setting of RL control experiments targeting the low-bursting oscillations with a decimation factor of 96.

6.6. Studies of reinforcement learning control

After the result of the previous section, a more systematic study of the RL control of the microbunching instability was carried out. For a total decimation factor of 96, as employed in the previous section, and the KARA machine parameters shown in table 6.4, a set of training experiments were carried out with the parameters of table 6.5. The chosen decimation factor means an action is taken by the agent every 96 revolutions, corresponding to roughly a quarter of a synchrotron period, one of the main timescales of the microbunching instability. This in turn means the hardware platform developed in this work is suited to control this instability.

The NN was composed of three layers of size 16, 8, and one or two depending on the use of the amplitude and frequency action or not. The stability region in these conditions can be assumed to be located between 0.425 mA and 0.390 mA. An attempt with a decimation factor of 192 also achieved control in this region. During the experiments two other reward functions were devised,

$$R(t) = -\left|x_t\right|, \tag{6.15}$$

$$R(t) = -\left(x_t - \bar{x}\right)^2 - \begin{cases} 1, & \text{if } a_t < 0 \\ 0, & \text{otherwise} \end{cases}. \tag{6.16}$$

The goal of the first one would be to obtain an agent capable of completely damp the microbunching instability. The second one was conceived in order to penalize variations applied to the action, in order to smooth the produced action signal, with potential benefits to the RF system.

Delta action threshold -	Amplitude frequency mode -	Initial current (mA)	Final current (mA)	Is in stable region? -	Reward function -	Achieved control?
No	No	0.184	0.176	No	Eq. 6.14	No
No	No	0.171	0.164	No	Eq. 6.14	No
No	No	0.154	0.145	Yes	Eq. 6.15	No

Table 6.6.: Setting of RL control experiments targeting the bursting with a decimation factor of 18.

A set of trainings, summarized in table table 6.6, were carried out with a total decimation factor of 18 in order to target the bursting dynamics. The machine parameters used were the one summarized in table 6.2. Both training attempts above and below the low-bursting threshold were attempted. No agent was successful in increasing the reward above the natural increase due to the current decay.

In conclusion, controlling the low-bursting was shown to be possible. The agent, though, would work reliably only in a specific current range that was also exhibiting higher controllability with a proportional controller. The control of the bursting was attempted in a limited amount but was not successful. In all cases discussed, a strong phenomenon was the improvement of reward due to the natural decay of the beam current. This leads to a reduction in CSR power output with the effect that can lead to a fictitious increase in reward, possibly biasing the RL algorithm.

6.7. Discussion and outlook

Compared to what was described for the HBO in chapter 5, the controllers described in the current chapter for the microbunching instability have only limited use during user operation. The degree of control that is achieved, albeit sufficient for some applications, only works in specific operating conditions dictated mainly by the bunch current. Albeit operating KARA with these parameters can be performed reliably, the natural decay of the beam current means that the operation in this condition can only last for a limited amount of time, in the order of a minute.

The reason why control of the instability outside of this region cannot be achieved is still an open problem, and its discussion lies outside the scope of this work. Nonetheless, there are several changes that can be applied to the setup that may lead to a definitive answer, and hopefully, a functioning controller.

Enhanced diagnostics could, in theory, improve the ability of an agent to achieve control. In this problem, the underlying state is the longitudinal phase space. The agent relies on its available observations to extract useful information about this state in pursuit of its goal. While the CSR power output is rich in information, it does not allow for a direct reconstruction of the underlying phase space. A more suitable approach could be the use of EO sampling setups, as discussed in previous sections. However, reconstructing the phase space from the

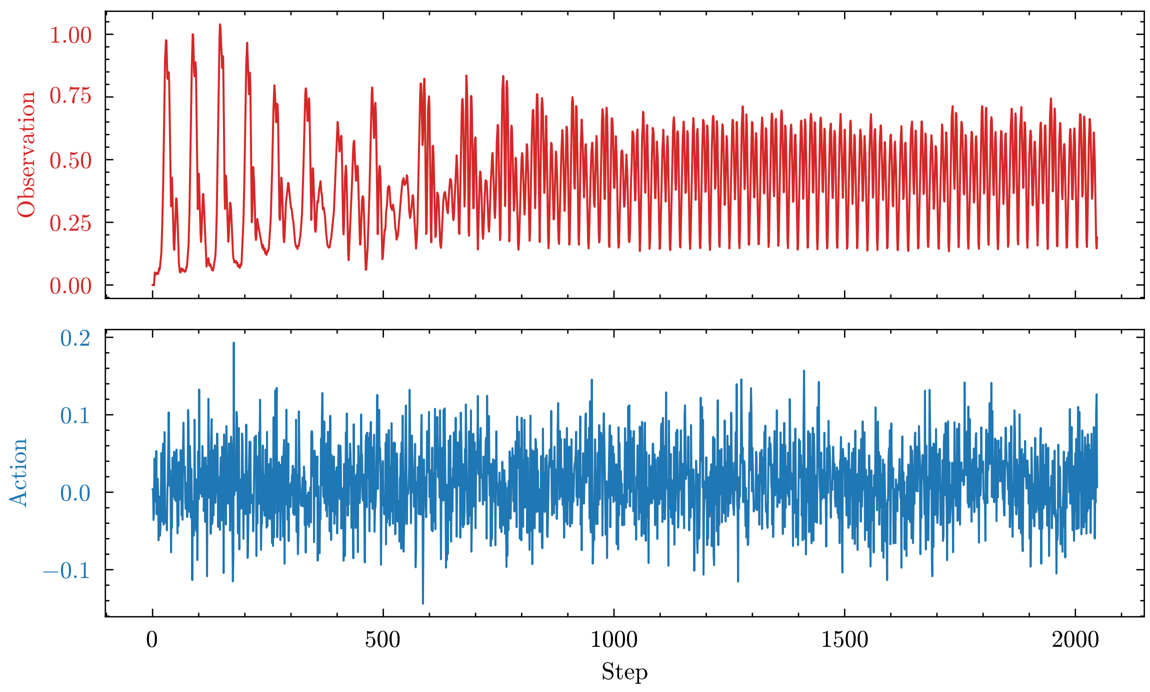

Figure 6.22.: Observation and action signals for episode 29 of figure 6.20. The applied action signal has a strong exploration noise component.

charge density with sufficient accuracy for real-time feedback remains an open challenge [149]. Additionally, this approach would require larger NNs to be implemented in hardware, further increasing computational demands.

Additional improvements could be applied to the actions that have been chosen. A graph of the employed action for the agent of section 6.5 is shown in figure 6.22. These fast swings are likely not being transferred to the beam, as the RF system behaves like a band-pass filter, attenuating part of the modulations. In fact, the LLRF interface described in section 6.1.3 can modify a set-point of the RF system, but it has no way to verify what is actually applied to the cavities. Having actions applied only to a variable extent can produce biases during the training process. Further instrumentation and integration of the RF system with KINGFISHER, such as an RF pick-up to monitor the field in the cavities, can in theory allow better diagnosis of this problem, for instance penalizing agent that produce actions that are filtered by the RF system. Furthermore, a longitudinal kicker cavity similar to the one described for the HBO feedback could in theory allow wide band actions to be applied. The currently available system, though, was shown in section 6.1.2 to not be powerful enough to affect the instability. A power upgrade could in theory allow further experiments.

Lastly, additional improvements can come from the algorithm side. The data being currently used has a strong time-series structure. Several NN architecture have been developed in order to better capture this kind of underlying structure. An example, LSTMs and GRUs, were discussed in section 2.2.2. Additionally, transformer models have been shown to further improve their capabilities with this kind of data [150]. Techniques as generalized state dependent exploration [151], allow smoother actions by randomizing the parameters of the NN instead of its output. This method allows for smoother actions to be produced, potentially leading to better behavior when interacting with the RF system.

6.8. Summary

To conduct control experiments targeting the microbunching instability with the KINGFISHER platform, a suitable set of observables and actions was necessary. At the time, the only option for low-latency, continuous data streaming was sampling the CSR power signal with the KAPTURE system, incorporating developments from the HBO research presented earlier. Based on systematic studies, amplitude modulation of the accelerating RF voltage through the LLRF system was chosen, as it effectively influenced the microbunching instability. Integrating KINGFISHER with the LLRF system to enable low-latency modulations required significant design and development efforts, undertaken during this work.

In order to test the functionality of the KINGFISHER integration with the LLRF, a proportional controller was used. This experiment demonstrated that low-bursting control was achievable at KARA, though only within a specific current range. Additionally, it highlighted the current dependence of the controller's performance, a phenomenon that has not yet been explored in the literature.

The first RL-based control experiments of the microbunching instability at a synchrotron light source were then carried out. The agent designed as part of this work was able to apply actions every 6.7 μs, in timescales comparable to the ones of the dynamics. Effective policies were reliably trained purely from interaction with the accelerator, showcasing the applicability of this technique to non-linear control problems.

7. Conclusions

This work studies and applies RL techniques to high-repetition rate control problems at large-scale facilities. A device successfully implementing these techniques has the potential of being a general turn-key controller, greatly aiding the commissioning and operation of these facilities. The use of edge devices can improve the performance of the system and thus broaden its range of applicability. To drive the work, some research question where conceived.

Question 1 *How can reinforcement learning algorithms be adapted so training on high-repetition rate environments can be performed?*

In chapter 3, the experience accumulator architecture was described. The underlying idea of this system is to monitor a high-performance real-time RL policy and use this data to perform training. Such a scheme allows to employ the current state of the art algorithm implementations. Furthermore, this approach was extended by allowing the computation of the reward function at the time of training, removing a component from the real-time domain, thus simplifying the design. In this way, the engineering of a reward function for the control problem at hand can be performed during the deployment of the controller, simplifying the installation of these class of systems.

Question 2 *How can reinforcement learning be deployed to an edge computing platform?*

In chapter 4, the KINGFISHER platform designed and developed in this work is described. Furthermore, the system was integrated and tested at the KARA accelerator test platform. KINGFISHER is based on the AMD/Xilinx Versal™ family of edge computing devices, employing a modular design that decouples the interfacing with the large-scale facility from the data processing path. Furthermore, the system is designed to employ the implementation of the RL algorithms provided by the Stable-baselines 3 library. The platform can be also remotely controlled through the EPICS control system, simplifying its integration within facilities such as KARA. Its versatility also enables performing experiments with standard controllers.

Question 3 *Is the edge-computing platform capable of learning online from interaction with an environment?*

Section 2.1 discusses some beam dynamics that can be encountered in synchrotron light sources, such as KARA, that can benefit from an external feedback control. The control of the HBO and microbunching instability are carried out respectively in chapter 5 and chapter 6. The HBO controller was shown to be effective, sometimes achieving better performance than the conventional controller, while exhibiting direct self-learning capabilities from interaction with the environment. In the case of the microbunching instability, a controller was produced, but its range of applicability was limited. Further studies on the novel field of the control of the microbunching instability will be required in order to fully understand the reason of these limitations.

Question 4 *How does online edge reinforcement learning compare to more traditional control approaches?*

As shown in chapters 5 and 6, the methodology described in this work is a promising solution in scenarios where simulation costs are prohibitive and data generation rates are high. Specifically, Chapter 5 shows that a controller can be reliably trained in the span of a few minutes, automatically adapting to the varying conditions of the accelerator. In this sense, the described system is at all effects a turn-key adaptive controller. Chapter 6 sheds light on another aspect of this approach. While the RL controller performed well in conditions were the traditional approaches were functional, a failure in training can be complex to debug. In the specific case under study, the cause is due to the underlying nature of the microbunching instability. Advancements in the field of explainable RL could in the future strongly mitigate this issue.

Contribution summary

This work described the first online-learning low-latency RL system deployed on an heterogeneous computing platform to be applied at a particle accelerator. In order to achieve this, several contributions were fundamental. First of all, the definition of the experience accumulator and training time reward definition RL training schemes were defined. These system allow previously impossible flexibility in the choice of training algorithm. The KINGFISHER system for the AMD/Xilinx Versal™ platform was then developed, managing both the data gathering required by the experience accumulator, together with the interfaces required to interact with KARA. Additionally, the reported design of low-latency NNs targeting the AIE array of Versal™ serve as a future reference for algorithm implementation targeting this platform.

Finally, the application of the system of the HBO shed light on the capabilities of RL in such challenging conditions. Meanwhile, the implementation of a controller targeting the microbunching instability at KARA allowed to analyze the current dependence of controllers targeting this phenomenon, and proved that high repetition rate RL is a viable option to solve this problem.

Outlook

The system developed in this work can be extended in several impactful ways to enhance its generalizability and overall performance. One critical challenge lies in transferring an agent design from its implementation in current NN libraries to the AIE array, as outlined in section 4.3. This process is complex, requiring specialized expertise and meticulous design. To address this, the open neural network exchange (ONNX) standard provides a robust framework for representing NN models as computation graphs [152]. A promising initiative in its early stages aims to enable the transpilation of ONNX representations directly into Versal™ AIE code [153]. Once mature, such a platform could dramatically simplify the deployment of new agent NNs, significantly enhance system generalization, and substantially reduce verification times by leveraging standardized implementations.

Performance improvements can also be achieved by adopting RL algorithms that better align with the experience accumulator architecture and the edge platforms employed in this system. Notably, gradient-free methods offer a compelling alternative, eliminating the need for gradient and backpropagation computations, as discussed in section 2.2.2 [154]. These methods, combined with the system's high interaction rate, unlock the possibility of enabling agents to perform autonomous, real-time training directly on the AIE array. Similarly, an autoencoder-based feature extraction stage could be designed to generate an optimized observation vector for the agent, allowing fully online training and adaptation.

Looking ahead, real-time online learning RL agents have the potential to evolve into transformative tools that augment human operators in solving complex problems. The integration of large language models [2] could further streamline the deployment of high interaction rate control solutions, with far-reaching implications. These advancements could revolutionize not only large-scale research facilities dedicated to fundamental and applied science but also find critical applications in medicine and industrial automation.

A. Low-latency Fourier transform

The results of this appendix have been published in [13].

A discrete Fourier transform (FT)[1] maps a sequence of N complex samples equally spaced in time $\{x_i\}_{i=0,\dots,N-1}$ into a sequence of complex numbers $\{X_i\}_{i=0,\dots,N-1}$ equally spaced in frequency defined as

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi \frac{k}{N} n}. \tag{A.1}$$

This transformation is widely used because it converts time-domain sequences into the frequency domain, making it easier to analyze the behavior of sequences with periodic components. However, directly computing a FT using equation (A.1) is computationally expensive, with the number of operations increasing at a rate of $O(N^2)$.

A class of algorithms known as FFT allows efficient computation of the FT with complexity $O(N \log N)$. The main idea is that, if N is even, it can be split into two $N/2$-FT

$$\begin{aligned} X_k &= \sum_{n=0}^{N/2-1} x_{2n} e^{-i2\pi \frac{k}{N} 2n} + e^{-i2\pi \frac{k}{N}} \sum_{n=0}^{N/2-1} x_{2n+1} e^{-i2\pi \frac{k}{N} 2n} \\ &= \left(\frac{N}{2} \text{ FT of even } x_n\right) + e^{-i2\pi \frac{k}{N}} \left(\frac{N}{2} \text{ FT of odd } x_n\right). \end{aligned} \tag{A.2}$$

The Cooley-Tukey algorithm [155] uses this idea recursively to reduce the amount of computations as much as possible. For this reason, FTs with a power-of-two value for N are particularly widespread, as they allow the greatest number of reductions.

Notably, this technique requires to have all the samples before computing the final FFT. Equation (A.1), on the other hand, allows for progressive accumulation as each sample is x_n is provided. For streaming data, this means that after the last sample x_{N-1} is accumulated into $\{X_k\}$, the FT is fully computed, despite requiring a larger number of computations to be carried out. This approach, though, would require to store all the complex exponentials, as the AIE tiles are not able to natively compute sine and cosine functions with floating point precision in an efficient manner. There is not enough memory on the engines to store a complete lookup table of the discrete FT samples.

For the sake of computational optimization, an approach similar to equation (A.2) can be applied, dividing the calculation into *chunks* that are then summed up when a new FT output

[1] In the rest of this appendix, the discrete Fourier transform will be denoted as FT.

is needed. In the formula below the summation is divided into M chunks, each one needing a sequence of N/M samples for the update. The final FT bin is composed using the M chunks. The value of M can be chosen arbitrarily, provided it is a divisor of N.

$$
\begin{aligned}
X_k = \sum_{n=0}^{N-1} x_n\, e^{-i\frac{2\pi}{N}kn} &= \sum_{m=0}^{M-1} \sum_{j=0}^{N/M-1} x_{\frac{N}{M}m+j}\, e^{-i\frac{2\pi}{N}k(\frac{N}{M}m+j)} \\
&= \sum_{m=0}^{M-1} e^{-i\frac{2\pi}{M}mk} \sum_{j=0}^{N/M-1} x_{\frac{N}{M}m+j}\, e^{-i\frac{2\pi}{N}kj}
\end{aligned}
\tag{A.3}
$$

This version allows to dramatically reduce the size of the lookup table: each coefficient depends either on m or on j, never both, so instead of using N coefficients per FT bin, only $\frac{N}{M} + M$ are needed. Furthermore, this construct allows for the computation of one FT every $\frac{N}{M}$ samples by reusing the accumulated chunks.

An algorithm for the computation of the low-latency Fourier transform (LLFT) can be obtained by implementing equation (A.3). The problem can be mapped to the AIE array by using the ideas outlined in [81]. The computation of chunks is divided among different tiles. Each tile is also responsible, after accumulation, of computing the final $\{X_k\}$. The compute kernels were implemented as a C++ template class, having N and M as parameters, together with the range of k indices for which the LLFT is computed. The lookup tables containing the complex exponentials are computed during the compilation of the graph. A build function was developed, as shown in listing A.1, allowing recursive template-based construction of the LLFT kernel data-path at compilation time.

The communication among the tiles is structured as follows: the inputs $\{x_n\}$ are delivered via an AXI4-Stream to a tile, which reads batches of eight floating-point values. These values are then forwarded to the compute tiles through a cascade stream of `v8float`. The compute tiles propagate the vector containing the input values to all tiles in the cascade stream daisy chain. Once the resulting $\{X_k\}$ values are computed, they are also forwarded along the daisy chain. For a 1024-sample FT, when processing every 256 input samples, the measured latency is on the order of a few tens of microseconds. This latency is highly dependent on how the computation is distributed across the various AIE tiles.

```cpp
static int currentKernelPosition = 0;

template<int N, int M, int kstart, int totalLen, int len, int currentLen = totalLen, typename
    currentPort>
void buildLLFTGraph(currentPort cascadeOutput) {
    // Add current kernel to the list
    typedef LLFT<N, M, kstart + currentLen - len, len, (currentLen - len)/4, totalLen==
        currentLen> curLLFT;
    llftKernels[currentKernelPosition] =
        adf::kernel::create_object<curLLFT>(curLLFT::buildUlut(), curLLFT::buildVlut());

    adf::kernel &curKernel = llftKernels[currentKernelPosition];

    adf::source (curKernel) = "LLFT.cc";
    adf::runtime<adf::ratio>(curKernel) = 1.;

    // Make the necessary connections
    adf::connect<adf::cascade> (curKernel.out[0], cascadeOutput);

    // The below code does the following job: you need a way to stop the template recursion,
    // this is achieved by creating the previous function without executing it (thanks to the
    // if statement). This is necessary because partial template specialization, in this case
    // for currentLen == len to stop the recursion is not allowed in a class
    if (currentLen == len) return;

    currentKernelPosition++;

    buildLLFTGraph<N, M, kstart, totalLen, len, (currentLen != len) ? currentLen-len :
        currentLen>(curKernel.in[0]);
}
```

Listing A.1: Example of a function using C++ templates that recursively builds the LLFT cascade data-path.

Acronyms

ACAP	advanced compute and acceleration platform
ADC	analog-to-digital converter
ADF	adaptive data flow
AI	artificial intelligence
AIE	AI Engine
ALU	arithmetic logic unit
AMD	Advanced Micro Devices, Inc.
API	application programming interface
ASIC	application specific integrated circuit
AVX	advanced vector extension
AXI4	advanced extensible interface 4
BBB	bunch-by-bunch
BPM	beam position monitor
CISC	complex instruction set computer
CLB	configurable logic block
CNN	convolutional neural network
CPU	central processing unit
CSR	coherent synchrotron radiation
CWS	client workstation
DAC	digital-to-analog converter
DDPG	deep deterministic policy gradient
DDR	double data rate
DDS	direct digital synthesizer
DMA	direct memory access
DPU	deep processing unit
DQN	deep Q-learning
DSP	digital signal processing
EO	electro-optical
EPICS	experimental physics and industrial control system
FEL	free electron laser
FFT	fast Fourier transform
FIFO	first-in first-out
FIR	finite impulse response
FPGA	field programmable gate array
FT	Fourier transform
GEM	gigabit Ethernet MAC
GPGPU	general purpose GPU
GPU	graphics processing unit
GRU	gated recurrent unit
HBO	horizontal betatron oscillations
HDL	hadware description language
HLS	high-level synthesis
ILA	integrated logic analyzer
IOC	input/output controller
IOMMU	input/output memory management unit
IP	intellectual property
ISR	incoherent synchrotron radiation

JIT	just-in-time
KALYPSO	Karlsruhe linear array detector for MHz repetition rate spectroscopy
KAPTURE	Karlsruhe pulse taking ultra-fast readout electronics
KARA	Karlsruhe research accelerator
LLFT	low-latency Fourier transform
LLRF	low-level radiofrequency
LSTM	long-short term memory
LUT	look-up table
MAC	multiply and accumulate
MBI	microbunching instability
MDP	Markov decision process
ML	machine learning
MLP	multi-layer perceptron
NMU	NoC master unit
NN	neural network
NoC	network-on-chip
NPS	NoC packet switch
NSU	NoC slave unit
ONNX	open neural network exchange
OS	operating system
PCB	printed circuit board
PCG	permuted congruential generator
PCIe	peripheral component interconnect express
PL	programmable logic
PLL	phase-locked loop
POMDP	partially observable Markov decision process
PPO	proximal policy optimization
PS	processing system
PV	process variable
RAM	random access memory
ReLU	rectifying linear unit
RF	radiofrequency
RGMII	reduced gigabit media-independent interface
RISC	reduced instruction set computer
RL	reinforcement learning
RL4AA	reinforcement learning for autonomous accelerators
RNN	recurren neural network
RTP	run-time parameter
S2MM	stream to memory mapped
SAC	soft actor-critic
SCADA	supervisory control and data acquisition
SIMD	single instruction multiple data
SoC	system-on-chip
TD	time difference
TD3	twin delayed DDPG
THD+N	total harmonic distortion plus noise
THERESA	terahertz readout sampling
TPU	tensor processing unit
TRPO	trust-region policy optimization
VCO	voltage controller oscillator
VLIW	very large instruction width
VNNI	vector neural network instructions
XRT	Xilinx run time
XSA	Xilinx support archive

Bibliography

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Commun. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[2] OpenAI et al. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL]. URL: https://arxiv.org/abs/2303.08774.

[3] J. Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (July 2021), pp. 583–589. ISSN: 1476-4687. DOI: 10.1038/s41586-021-03819-2. URL: http://dx.doi.org/10.1038/s41586-021-03819-2.

[4] T. B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL]. URL: https://arxiv.org/abs/2005.14165.

[5] V. Mnih et al. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG]. URL: https://arxiv.org/abs/1312.5602.

[6] D. Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (Jan. 2016), pp. 484–489. ISSN: 1476-4687. DOI: 10.1038/nature16961. URL: http://dx.doi.org/10.1038/nature16961.

[7] J. Degrave et al. "Magnetic control of tokamak plasmas through deep reinforcement learning". In: *Nature* 602.7897 (Feb. 2022), pp. 414–419. DOI: 10.1038/s41586-021-04301-9. URL: https://doi.org/10.1038/s41586-021-04301-9.

[8] A. M. Annaswamy. "Adaptive Control and Intersections with Reinforcement Learning". In: *Annual Review of Control, Robotics, and Autonomous Systems* 6.1 (May 2023), pp. 65–93. ISSN: 2573-5144. DOI: 10.1146/annurev-control-062922-090153. URL: http://dx.doi.org/10.1146/annurev-control-062922-090153.

[9] S. Gu et al. *A Review of Safe Reinforcement Learning: Methods, Theory and Applications*. 2024. arXiv: 2205.10330 [cs.AI]. URL: https://arxiv.org/abs/2205.10330.

[10] Hirlaender, Simon et al. "Towards few-shot reinforcement learning in particle accelerator control". en. In: (2024). DOI: 10.18429/JACOW-IPAC2024-TUPS60. URL: https://jacow.org/ipac2024/doi/jacow-ipac2024-tups60.

[11] O. Weng et al. *Architectural Implications of Neural Network Inference for High Data-Rate, Low-Latency Scientific Applications*. 2024. arXiv: 2403.08980 [cs.LG]. URL: https://arxiv.org/abs/2403.08980.

[12] L. Scomparin et al. *Microsecond-Latency Feedback at a Particle Accelerator by Online Reinforcement Learning on Hardware*. 2024. arXiv: 2409.16177 [physics.acc-ph]. URL: https://arxiv.org/abs/2409.16177.

[13] L. Scomparin et al. “KINGFISHER: A Framework for Fast Machine Learning Inference for Autonomous Accelerator Systems”. In: *Proc. 11th Int. Beam Instrum. Conf. (IBIC’22)* (Kraków, Poland). International Beam Instrumentation Conference 11. JACoW Publishing, Geneva, Switzerland, Dec. 2022, MOP42, pp. 151–155. ISBN: 978-3-95450-241-7. DOI: 10.18429/JACoW-IBIC2022-MOP42. URL: https://jacow.org/ibic2022/papers/mop42.pdf.

[14] L. Scomparin et al. “Preliminary results on the reinforcement learning-based control of the microbunching instability”. In: *Proc. 15th International Particle Accelerator Conference* (Nashville, TN). IPAC’24 - 15th International Particle Accelerator Conference 15. JACoW Publishing, Geneva, Switzerland, May 2024, pp. 1808–1811. ISBN: 978-3-95450-247-9. DOI: 10.18429/JACoW-IPAC2024-TUPS61. URL: https://indico.jacow.org/event/63/contributions/4057.

[15] R. P. Walker. “Synchrotron radiation”. In: (1994). DOI: 10.5170/CERN-1994-001.437. URL: https://cds.cern.ch/record/398429.

[16] S. Y. Lee. *Accelerator Physics*. 3rd. WORLD SCIENTIFIC, 2011. DOI: 10.1142/8335. eprint: https://www.worldscientific.com/doi/pdf/10.1142/8335. URL: https://www.worldscientific.com/doi/abs/10.1142/8335.

[17] M. Brosi. “In-Depth Analysis of the Micro-Bunching Characteristics in Single and Multi-Bunch Operation at KARA”. 54.01.01; LK 01. PhD thesis. Karlsruher Institut für Technologie (KIT), 2020. 198 pp. DOI: 10.5445/IR/1000120018.

[18] E. Blomley. “Investigation and Control of Beam Instabilities at the Karlsruhe Research Accelerator using a 3-D Digital Bunch-by-Bunch Feedback System”. 54.11.11; LK 01. PhD thesis. Karlsruher Institut für Technologie (KIT), 2021. 138 pp. DOI: 10.5445/IR/1000137621.

[19] P. Schreiber. “Negative Momentum Compaction Operation and its Effect on the Beam Dynamics at the Accelerator Test Facility KARA”. 54.11.11; LK 01. PhD thesis. Karlsruher Institut für Technologie (KIT), 2022. 131 pp. DOI: 10.5445/IR/1000148354.

[20] M. Caselle et al. “KAPTURE-2. A picosecond sampling system for individual THz pulses with high repetition rate”. In: *Journal of Instrumentation* 12.01 (Jan. 2017), p. C01040. DOI: 10.1088/1748-0221/12/01/C01040. URL: https://dx.doi.org/10.1088/1748-0221/12/01/C01040.

[21] L. Rota et al. “KALYPSO: Linear array detector for high-repetition rate and real-time beam diagnostics”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 936 (2019). Frontier Detectors for Frontier Physics: 14th Pisa Meeting on Advanced Detectors, pp. 10–13. ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2018.10.093. URL: https://www.sciencedirect.com/science/article/pii/S0168900218314074.

[22] L. Rota. “KALYPSO, a novel detector system for high-repetition rate and real-time beam diagnostics”. 54.02.02; LK 01. PhD thesis. Karlsruher Institut für Technologie (KIT), 2018. 166 pp. DOI: 10.5445/IR/1000082349.

[23] M. M. Patil. “Development and integration of high throughput detector systems for photon-based diagnostics at Karlsruhe Research Accelerator”. 54.11.11; LK 01. PhD thesis. Karlsruher Institut für Technologie (KIT), 2023. 150 pp. DOI: 10.5445/IR/1000163059.

[24] L. Rota et al. “Development of a Front-End ASIC for 1D Detectors with 12 MHz Frame-Rate”. In: *PoS* TWEPP-17 (2018), p. 033. DOI: 10.22323/1.313.0033.

[25] G. Niehues et al. “High Repetition Rate, Single-Shot Electro-Optical Monitoring of Longitudinal Electron Bunch Dynamics Using the Linear Array Detector KALYPSO”. en. In: *Proceedings of the 9th Int. Particle Accelerator Conf.* IPAC2018 (2018), Canada. DOI: 10.18429/JACOW-IPAC2018-WEPAL026. URL: http://jacow.org/ipac2018/doi/JACoW-IPAC2018-WEPAL026.html.

[26] S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning*. Cambridge, England: Cambridge University Press, May 2014.

[27] J. Ansel et al. “PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation”. In: *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. ASPLOS ’24. ACM, Apr. 2024. DOI: 10.1145/3620665.3640366. URL: http://dx.doi.org/10.1145/3620665.3640366.

[28] M. Abadi et al. *TensorFlow, Large-scale machine learning on heterogeneous systems*. Nov. 2015. DOI: 10.5281/zenodo.4724125.

[29] F. Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[30] Y. Jia et al. “Caffe: Convolutional Architecture for Fast Feature Embedding”. In: *arXiv preprint arXiv:1408.5093* (2014).

[31] A. Paszke et al. “Automatic differentiation in PyTorch”. In: (2017).

[32] A. Griewank and A. Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

[33] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018. ISBN: 0262039249.

[34] D. Silver. *Lectures on Reinforcement Learning*. URL: https://www.davidsilver.uk/teaching/. 2015.

[35] T. P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2015. DOI: 10.48550/ARXIV.1509.02971. URL: https://arxiv.org/abs/1509.02971.

[36] D. Silver et al. “Deterministic Policy Gradient Algorithms”. In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by E. P. Xing and T. Jebara. Vol. 32. Proceedings of Machine Learning Research 1. Bejing, China: PMLR, June 2014, pp. 387–395. URL: https://proceedings.mlr.press/v32/silver14.html.

[37] G. E. Uhlenbeck and L. S. Ornstein. “On the Theory of the Brownian Motion”. In: *Phys. Rev.* 36 (5 Oct. 1930), pp. 823–841. DOI: 10.1103/PhysRev.36.823. URL: https://link.aps.org/doi/10.1103/PhysRev.36.823.

[38] J. Achiam. “Spinning Up in Deep Reinforcement Learning”. In: (2018).

[39] S. Fujimoto, H. van Hoof, and D. Meger. *Addressing Function Approximation Error in Actor-Critic Methods*. 2018. DOI: 10.48550/ARXIV.1802.09477. URL: https://arxiv.org/abs/1802.09477.

[40] J. Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG]. URL: https://arxiv.org/abs/1707.06347.

[41] J. Schulman et al. *Trust Region Policy Optimization*. 2015. DOI: 10.48550/ARXIV.1502.05477. URL: https://arxiv.org/abs/1502.05477.

[42] J. Schulman et al. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2015. DOI: 10.48550/ARXIV.1506.02438. URL: https://arxiv.org/abs/1506.02438.

[43] T. Haarnoja et al. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. 2018. arXiv: 1801.01290 [cs.LG]. URL: https://arxiv.org/abs/1801.01290.

[44] J. Bradbury et al. *JAX: composable transformations of Python+NumPy programs*. Version 0.3.13. 2018. URL: http://github.com/google/jax.

[45] J. Kaiser et al. "Bridging the gap between machine learning and particle accelerator physics with high-speed, differentiable simulations". In: *Phys. Rev. Accel. Beams* 27 (5 May 2024), p. 054601. DOI: 10.1103/PhysRevAccelBeams.27.054601. URL: https://link.aps.org/doi/10.1103/PhysRevAccelBeams.27.054601.

[46] S. Huang et al. "The 37 Implementation Details of Proximal Policy Optimization". In: *ICLR Blog Track*. https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/. 2022. URL: https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/.

[47] A. Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[48] G. Brockman et al. *OpenAI Gym*. 2016. DOI: 10.48550/ARXIV.1606.01540. URL: https://arxiv.org/abs/1606.01540.

[49] M. Towers et al. *Gymnasium*. Mar. 2023. DOI: 10.5281/zenodo.8127026. URL: https://zenodo.org/record/8127025 (visited on 07/08/2023).

[50] J. Kaiser et al. "Reinforcement learning-trained optimisers and Bayesian optimisation for online particle accelerator tuning". In: *Scientific Reports* 14.1 (July 2024). ISSN: 2045-2322. DOI: 10.1038/s41598-024-66263-y. URL: http://dx.doi.org/10.1038/s41598-024-66263-y.

[51] A. Santamaria Garcia et al. "The reinforcement learning for autonomous accelerators collaboration". In: *Proc. IPAC'24* (Nashville, TN). IPAC'24 - 15th International Particle Accelerator Conference 15. JACoW Publishing, Geneva, Switzerland, May 2024, pp. 1812–1815. ISBN: 978-3-95450-247-9. DOI: 10.18429/JACoW-IPAC2024-TUPS62. URL: https://indico.jacow.org/event/63/contributions/4082.

[52] N. Bruchon et al. "Basic reinforcement learning techniques to control the intensity of a seeded free-electron laser". In: *Electronics* 9.5 (2020), p. 781.

[53] F. O'Shea, N. Bruchon, and G. Gaio. "Policy gradient methods for free-electron laser and terahertz source optimization and stabilization at the FERMI free-electron laser at Elettra". In: *Physical Review Accelerators and Beams* 23.12 (2020), p. 122802.

[54] J. St. John et al. "Real-time artificial intelligence for accelerator control: A study at the Fermilab Booster". In: *Phys. Rev. Accel. Beams* 24 (10 Oct. 2021), p. 104601. DOI: 10.1103/PhysRevAccelBeams.24.104601. URL: https://link.aps.org/doi/10.1103/PhysRevAccelBeams.24.104601.

[55] N. Madysa et al. "Automated Intensity Optimisation Using Reinforcement Learning at LEIR". In: *JACoW IPAC* 2022 (2022), pp. 941–944. DOI: 10.18429/JACoW-IPAC2022-TUPOST040. URL: https://cds.cern.ch/record/2845859.

[56] V. Kain et al. "Sample-efficient reinforcement learning for CERN accelerator control". In: *Phys. Rev. Accel. Beams* 23 (12 Dec. 2020), p. 124801. DOI: 10.1103/PhysRevAccelBeams.23.124801. URL: https://link.aps.org/doi/10.1103/PhysRevAccelBeams.23.124801.

[57] J. L. Hennessy and D. A. Patterson. *Computer Architecture, Fifth Edition: A Quantitative Approach*. 5th. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011. ISBN: 012383872X.

[58] C. Maxfield. *The Design Warrior's Guide to FPGAs: Devices, Tools and Flows*. 1st. USA: Newnes, 2004. ISBN: 0750676043.

[59] J. von Neumann. "First Draft of a Report on the EDVAC". In: *IEEE Annals of the History of Computing* 15.04 (Oct. 1993), pp. 27–75. ISSN: 1934-1547. DOI: 10.1109/85.238389.

[60] J. E. Stone, D. Gohara, and G. Shi. "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems". In: *Computing in Science and Engg.* 12.3 (May 2010), pp. 66–73. ISSN: 1521-9615.

[61] Y. Sun, T. Baruah, and D. Kaeli. *Accelerated Computing with HIP*. Dec. 2022. ISBN: 979-8218107444.

[62] S. Cook. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012. ISBN: 9780124159334.

[63] *PG203 - UltraScale+ Devices Integrated 100G Ethernet Subsystem LogiCORE IP Product Guide*. v3.1. Xilinx. URL: https://docs.amd.com/r/en-US/pg203-cmac-usplus.

[64] *IEEE 1800-2023 Standard for SystemVerilog - Unified Hardware Design, Specification, and Verification Language*. IEEE.

[65] *IEEE 1076-2019 Standard for VHDL Language Reference Manual*. IEEE.

[66] W. Meeus et al. "An overview of today's high-level synthesis tools". In: *Design Automation for Embedded Systems* 16.3 (Aug. 2012), pp. 31–51. ISSN: 1572-8080. DOI: 10.1007/s10617-012-9096-8. URL: http://dx.doi.org/10.1007/s10617-012-9096-8.

[67] *ARM IHI 0022H - AMBA AXI and ACE Protocol Specification*. September 2023. Arm Inc.

[68] *ARM IHI 0051B - AMBA AXI-Stream Protocol Specification*. 09 April 2021. Arm Inc.

[69] Xilinx Inc. “Versal: The First Adaptive Compute Acceleration Platform (ACAP)”. In: *White paper* (2020). Accessed in August 2024. URL: `https://docs.amd.com/v/u/en-US/wp505-versal-acap`.

[70] A. Zimpeck et al. “FinFET Technology”. In: *Mitigating Process Variability and Soft Errors at Circuit-Level for FinFETs*. Cham: Springer International Publishing, 2021, pp. 7–27. ISBN: 978-3-030-68368-9. DOI: `10.1007/978-3-030-68368-9_2`. URL: `https://doi.org/10.1007/978-3-030-68368-9_2`.

[71] *PG314 - Versal Devices Integrated 100G Multirate Ethernet MAC Subsystem Product Guide*. v2.3. Xilinx. URL: `https://docs.amd.com/r/en-US/pg314-versal-mrmac`.

[72] *PG313 - Versal Adaptive SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP Product Guide*. v1.1. Xilinx. URL: `https://docs.amd.com/r/en-US/pg313-network-on-chip/IP-Facts`.

[73] K. Shin and P. Ramanathan. “Real-time computing: a new discipline of computer science and engineering”. In: *Proceedings of the IEEE* 82.1 (1994), pp. 6–24. DOI: `10.1109/5.259423`.

[74] W. Wang. “Towards Intelligent Data Acquisition Systems with Embedded Deep Learning on MPSoC”. 54.12.02; LK 01. PhD thesis. Karlsruher Institut für Technologie (KIT), 2021. 166 pp. DOI: `10.5445/IR/1000133898`.

[75] *PG213 - UltraScale+ Devices Integrated Block for PCI Express Product Guide*. v1.3. Xilinx. URL: `https://docs.amd.com/r/en-US/pg213-pcie4-ultrascale-plus/Introduction`.

[76] A. R. Carneiro, M. S. Serpa, and P. O. A. Navaux. “Lightweight Deep Learning Applications on AVX-512”. In: *2021 IEEE Symposium on Computers and Communications (ISCC)*. 2021, pp. 1–6. DOI: `10.1109/ISCC53001.2021.9631464`.

[77] H. T. Kung and C. E. Leiserson. “Systolic arrays (for VLSI)”. In: *Sparse Matrix Proceedings 1978*. Vol. 1. Society for industrial and applied mathematics Philadelphia, PA, USA. 1979, pp. 256–282.

[78] *Introduction to Cloud TPU*. `https://cloud.google.com/tpu/docs/intro-to-tpu`. Accessed in August 2024.

[79] *Mythic AI*. `https://mythic.ai/`. Accessed in August 2024.

[80] *AM009 - Versal Adaptive SoC AI Engine Architecture Manual*. v1.3. Xilinx. URL: `https://docs.amd.com/r/en-US/am009-versal-ai-engine`.

[81] *UG1079 - AI Engine Kernel and Graph Programming Guide*. 2022.1. Xilinx. URL: `https://docs.amd.com/r/2022.1-English/ug1079-ai-engine-kernel-coding/`.

[82] *UG1366 - VCK190 Evaluation Board User Guide*. v1.1. Xilinx. URL: `https://docs.amd.com/r/en-US/ug1366-vck190-eval-bd`.

[83] *UG1393 - Vitis Unified Software Platform Documentation: Application Acceleration Development*. 2023.2. Xilinx. URL: `https://docs.amd.com/r/2023.2-English/ug1393-vitis-application-acceleration/`.

[84] *UG1076 - AI Engine Tools and Flows User Guide.* 2024.1. Xilinx. URL: https://docs.amd.com/r/en-US/ug1076-ai-engine-environment/Overview.

[85] M. Rothmann and M. Porrmann. "A Survey of Domain-Specific Architectures for Reinforcement Learning". In: *IEEE Access* 10 (2022), pp. 13753–13767. DOI: 10.1109/ACCESS.2022.3146518.

[86] A. R. Baranwal et al. "ReLAccS: A Multilevel Approach to Accelerator Design for Reinforcement Learning on FPGA-Based Systems". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.9 (2021), pp. 1754–1767. DOI: 10.1109/TCAD.2020.3028350.

[87] S. S. Sahoo et al. "MemOReL: A Memory-Oriented Optimization Approach to Reinforcement Learning on FPGA-Based Embedded Systems". In: *Proceedings of the 2021 on Great Lakes Symposium on VLSI.* GLSVLSI '21. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 339–346. ISBN: 9781450383936. DOI: 10.1145/3453688.3461533. URL: https://doi.org/10.1145/3453688.3461533.

[88] L. M. D. Da Silva, M. F. Torquato, and M. A. C. Fernandes. "Parallel Implementation of Reinforcement Learning Q-Learning Technique for FPGA". In: *IEEE Access* 7 (2019), pp. 2782–2798. DOI: 10.1109/ACCESS.2018.2885950.

[89] Y. Meng et al. "QTAccel: A Generic FPGA based Design for Q-Table based Reinforcement Learning Accelerators". In: *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW).* 2020, pp. 107–114. DOI: 10.1109/IPDPSW50202.2020.00024.

[90] S. Spanò et al. "An Efficient Hardware Implementation of Reinforcement Learning: The Q-Learning Algorithm". In: *IEEE Access* 7 (2019), pp. 186340–186351. DOI: 10.1109/ACCESS.2019.2961174.

[91] S. Shao et al. "Towards Hardware Accelerated Reinforcement Learning for Application-Specific Robotic Control". In: *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP).* 2018, pp. 1–8. DOI: 10.1109/ASAP.2018.8445099.

[92] C.-W. Hu, J. Hu, and S. P. Khatri. "TD3lite: FPGA Acceleration of Reinforcement Learning with Structural and Representation Optimizations". In: *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL).* 2022, pp. 79–85. DOI: 10.1109/FPL57034.2022.00023.

[93] H. Cho et al. "FA3C". In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems.* ACM, Apr. 2019. DOI: 10.1145/3297858.3304058. URL: https://doi.org/10.1145/3297858.3304058.

[94] J. Su et al. "Neural Network Based Reinforcement Learning Acceleration on FPGA Platforms". In: *ACM SIGARCH Computer Architecture News* 44.4 (Jan. 2017), pp. 68–73. DOI: 10.1145/3039902.3039915. URL: https://doi.org/10.1145/3039902.3039915.

[95] H. Watanabe, M. Tsukada, and H. Matsutani. “An FPGA-Based On-Device Reinforcement Learning Approach using Online Sequential Learning”. In: *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2021, pp. 96–103. DOI: 10.1109/IPDPSW52791.2021.00022.

[96] M.-J. Li et al. “Implementation of Deep Reinforcement Learning”. In: *Proceedings of the 2nd International Conference on Information Science and Systems*. ICISS ’19. Tokyo, Japan: Association for Computing Machinery, 2019, pp. 232–236. ISBN: 9781450361033. DOI: 10.1145/3322645.3322693. URL: https://doi.org/10.1145/3322645.3322693.

[97] Y. Meng, S. Kuppannagari, and V. Prasanna. “Accelerating Proximal Policy Optimization on CPU-FPGA Heterogeneous Platforms”. In: *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2020, pp. 19–27. DOI: 10.1109/FCCM48280.2020.00012.

[98] Y. Wei et al. “Low latency optical-based mode tracking with machine learning deployed on FPGAs on a tokamak”. In: *Review of Scientific Instruments* 95.7 (July 2024), p. 073509. ISSN: 0034-6748. DOI: 10.1063/5.0190354. eprint: https://pubs.aip.org/aip/rsi/article-pdf/doi/10.1063/5.0190354/20033792/073509_1_5.0190354.pdf. URL: https://doi.org/10.1063/5.0190354.

[99] J. Duarte et al. “Fast inference of deep neural networks in FPGAs for particle physics”. In: *JINST* 13.07 (2018), P07027. DOI: 10.1088/1748-0221/13/07/P07027. arXiv: 1804.06913 [physics.ins-det].

[100] FastML Team. *fastmachinelearning/hls4ml*. Version v0.8.1. 2023. DOI: 10.5281/zenodo.1201549. URL: https://github.com/fastmachinelearning/hls4ml.

[101] W. Wang et al. “Accelerated Deep Reinforcement Learning for Fast Feedback of Beam Dynamics at KARA”. In: *IEEE Transactions on Nuclear Science* 68.8 (2021), pp. 1794–1800. DOI: 10.1109/TNS.2021.3084515.

[102] G. Book et al. “Transferring Online Reinforcement Learning for Electric Motor Control From Simulation to Real-World Experiments”. In: *IEEE Open Journal of Power Electronics* 2 (2021), pp. 187–201. DOI: 10.1109/OJPEL.2021.3065877.

[103] C. Shannon. “Communication in the Presence of Noise”. In: *Proceedings of the IRE* 37.1 (1949), pp. 10–21. DOI: 10.1109/JRPROC.1949.232969.

[104] H. Nyquist. “Certain Topics in Telegraph Transmission Theory”. In: *Transactions of the American Institute of Electrical Engineers* 47.2 (1928), pp. 617–644. DOI: 10.1109/T-AIEE.1928.5055024.

[105] M. Sapkas. *Beam Control with Fast Reinforcement Learning Inference at the Edge*. 2024. URL: https://hdl.handle.net/20.500.12608/70130.

[106] T. Boltz. “Micro-Bunching Control at Electron Storage Rings with Reinforcement Learning”. PhD thesis. Karlsruher Institut für Technologie (KIT), 2021. DOI: 10.5445/IR/1000140271.

[107] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[108] M. Caselle et al. “A high-speed DAQ framework for future high-level trigger and event building clusters”. In: *Journal of Instrumentation* 12.03 (Mar. 2017), p. C03015. DOI: 10.1088/1748-0221/12/03/C03015. URL: https://dx.doi.org/10.1088/1748-0221/12/03/C03015.

[109] *Firefly Optical Transceivers*. https://www.samtec.com/optics/systems/firefly/. Accessed in September 2024.

[110] *UG1085 - Zynq UltraScale+ Device Technical Reference Manual*. v2.4. Xilinx. URL: https://docs.amd.com/r/en-US/ug1085-zynq-ultrascale-trm.

[111] *AM011 - Versal Adaptive SoC Technical Reference Manual*. v1.6. Xilinx. URL: https://docs.amd.com/r/en-US/am011-versal-acap-trm.

[112] *SP011 - Aurora 64B/66B Protocol Specification*. v1.3. Xilinx. URL: https://docs.amd.com/v/u/en-US/aurora_64b66b_protocol_spec_sp011.

[113] *PG074 - Aurora 64B/66B LogiCORE IP Product Guide*. v12.0. Xilinx. URL: https://docs.amd.com/r/en-US/pg074-aurora-64b66b.

[114] *PG021 - AXI DMA LogiCORE IP Product Guide*. v7.1. Xilinx. URL: https://docs.amd.com/r/en-US/pg021_axi_dma.

[115] *AMD Vitis AI Software*. https://www.amd.com/en/products/software/vitis-ai.html. Accessed in September 2024.

[116] *PG389 - DPUCVDX8G for Versal ACAPs Product Guide*. v1.3. Xilinx. URL: https://docs.amd.com/r/en-US/pg389-dpucvdx8g/Introduction.

[117] *AI Engine Intrinsics User Guide*. 2023.1. Xilinx. URL: https://www.xilinx.com/htmldocs/xilinx2023_1/aiengine_intrinsics/intrinsics/index.html.

[118] S. Oberman and M. Siu. “A high-performance area-efficient multifunction interpolator”. In: *17th IEEE Symposium on Computer Arithmetic (ARITH’05)*. 2005, pp. 272–279. DOI: 10.1109/ARITH.2005.7.

[119] G. Marsaglia. “Xorshift RNGs”. In: *Journal of Statistical Software* 8.14 (2003), pp. 1–6. DOI: 10.18637/jss.v008.i14. URL: https://www.jstatsoft.org/index.php/jss/article/view/v008i14.

[120] M. E. O’Neill. *PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation*. Tech. rep. HMC-CS-2014-0905. Claremont, CA: Harvey Mudd College, Sept. 2014.

[121] *PG357 - Integrated Logic Analyzer (ILA) with AXI4-Stream Interface LogiCORE IP Product Guide*. v1.2. Xilinx. URL: https://docs.amd.com/r/en-US/pg357-axis-ila.

[122] *Experimental Physics and Industrial Control System*. https://epics.anl.gov/index.php. Accessed in October 2024.

[123] *gRPC: A high performance, open source universal RPC framework*. https://grpc.io/. Accessed in October 2024.

[124] *DOOCS: The Distributed Object-Oriented Control System Framework*. https://doocs.desy.de/. Accessed in October 2024.

[125] *TANGO controls*. https://www.tango-controls.org/. Accessed in October 2024.

[126] *caproto: a bring-your-own-IO implementation of the EPICS Channel Access protocol in pure Python*. https://github.com/caproto/caproto. Accessed in October 2024.

[127] L. Scomparin et al. "A low-latency feedback system for the control of horizontal betatron oscillations". en. In: (2023). DOI: 10.18429/JACOW-IPAC2023-THPL027. URL: https://jacow.org/ipac2023/doi/jacow-ipac2023-thpl027.

[128] M. Lonza and Presented By H. Schmickler. "Multi-bunch Feedback Systems". In: *CERN Yellow Reports: School Proceedings* (2017), Vol 3 (2017): Proceedings of the CAS–CERN Accelerator School on Intensity Limitations in Particle Beams. DOI: 10.23730/CYRSP-2017-003.471. URL: https://e-publishing.cern.ch/index.php/CYRSP/article/view/525.

[129] *Dimtel iGp12 bunch-by-bunch feedback system*. https://www.dimtel.com/products/igp12. Accessed in October 2024.

[130] M. Dehler et al. "Current status of the ELETTRA/SLS transverse multibunch feedback". In: *Proc. EPAC*. 2000, pp. 1894–1896.

[131] *Dimtel BPMH-20-2G hybrid*. https://www.dimtel.com/products/bpmh. Accessed in October 2024.

[132] D. Teytelman. *Experience with feedback systems in modern synchrotron light sources*. presented at Accelerator Performance and Concepts Workshop in Frankfurt, December 10-12, 2018.

[133] M. G. Minty and F. Zimmermann. *Measurement and Control of Charged Particle Beams*. Springer Berlin Heidelberg, 2003. DOI: 10.1007/978-3-662-08581-3. URL: https://doi.org/10.1007/978-3-662-08581-3.

[134] J. L. Steinmann. "Diagnostics of Short Electron Bunches with THz Detectors in Particle Accelerators". 54.01.01; LK 01. PhD thesis. Karlsruher Institut für Technologie (KIT), 2019. 226 pp. ISBN: 978-3-7315-0889-2. DOI: 10.5445/KSP/1000090017.

[135] C. Szwaj et al. "High sensitivity photonic time-stretch electro-optic sampling of terahertz pulses". In: *Review of Scientific Instruments* 87.10 (Oct. 2016), p. 103111. ISSN: 0034-6748. DOI: 10.1063/1.4964702. eprint: https://pubs.aip.org/aip/rsi/article-pdf/doi/10.1063/1.4964702/14055270/103111_1_online.pdf. URL: https://doi.org/10.1063/1.4964702.

[136] S. Maier. "Systematic Studies of the Micro-Bunching Instability with an Additional Corrugated Structure Impedance at KARA". 54.11.11; LK 01. PhD thesis. Karlsruher Institut für Technologie (KIT), 2024. 164 pp. DOI: 10.5445/IR/1000171277.

[137] P. Schönfeldt. "Simulation and measurement of the dynamics of ultra-short electron bunch profiles for the generation of coherent THz radiation". 54.01.01; LK 01. PhD thesis. Karlsruher Institut für Technologie (KIT), 2018. 142 pp. DOI: 10.5445/IR/1000084466.

[138] P. Schönfeldt et al. "Parallelized Vlasov-Fokker-Planck solver for desktop personal computers". In: *Phys. Rev. Accel. Beams* 20 (3 Mar. 2017), p. 030704. DOI: 10.1103/PhysRevAccelBeams.20.030704. URL: https://link.aps.org/doi/10.1103/PhysRevAccelBeams.20.030704.

[139] J. L. Steinmann et al. “Continuous bunch-by-bunch spectroscopic investigation of the microbunching instability”. In: *Phys. Rev. Accel. Beams* 21 (11 Nov. 2018), p. 110705. DOI: 10.1103/PhysRevAccelBeams.21.110705. URL: https://link.aps.org/doi/10.1103/PhysRevAccelBeams.21.110705.

[140] O. Manzhura et al. “Terahertz Sampling Rates with Photonic Time-Stretch for Electron Beam Diagnostics”. In: *Proc. IPAC'22* (Bangkok, Thailand). International Particle Accelerator Conference 13. JACoW Publishing, Geneva, Switzerland, July 2022, MOPOPT017, pp. 263–266. ISBN: 978-3-95450-227-1. DOI: 10.18429/JACoW-IPAC2022-MOPOPT017. URL: https://jacow.org/ipac2022/papers/mopopt017.pdf.

[141] S. Maier. “Systematic studies of RF phase modulation at KARA”. en. MA thesis. Karlsruher Institut für Technologie (KIT), 2020. 85 pp. DOI: 10.5445/IR/1000123900.

[142] S. Funkner et al. “Revealing the dynamics of ultrarelativistic non-equilibrium many-electron systems with phase space tomography”. In: *Scientific Reports* 13.1 (Mar. 2023). ISSN: 2045-2322. DOI: 10.1038/s41598-023-31196-5. URL: http://dx.doi.org/10.1038/s41598-023-31196-5.

[143] A. Santamaria Garcia et al. “Systematic study of longitudinal excitations to influence the microbunching instability at KARA”. en. In: (2023). DOI: 10.18429/JACOW-IPAC2023-WEPA018. URL: https://jacow.org/ipac2023/doi/jacow-ipac2023-wepa018.

[144] S. Simrock and Z. Geng. *Low-Level Radio Frequency Systems*. Springer International Publishing, 2022. ISBN: 9783030944193. DOI: 10.1007/978-3-030-94419-3. URL: http://dx.doi.org/10.1007/978-3-030-94419-3.

[145] *Dimtel LLRF9 low-level radio-frequency feedback system*. https://www.dimtel.com/products/llrf9. Accessed in October 2024.

[146] *UG1182 - ZCU102 Evaluation Board User Guide*. v1.7. Xilinx. URL: https://docs.amd.com/v/u/en-US/ug1182-zcu102-eval-bd.

[147] C. Evain et al. “Stable coherent terahertz synchrotron radiation from controlled relativistic electron bunches”. In: *Nature Physics* 15.7 (Apr. 2019), pp. 635–639. ISSN: 1745-2481. DOI: 10.1038/s41567-019-0488-6. URL: http://dx.doi.org/10.1038/s41567-019-0488-6.

[148] C. Evain et al. “Gain Switching of the Microbunching Instability to Produce Giant Bursts of Terahertz Coherent Synchrotron Radiation”. In: *Phys. Rev. Lett.* 133 (14 Oct. 2024), p. 145001. DOI: 10.1103/PhysRevLett.133.145001. URL: https://link.aps.org/doi/10.1103/PhysRevLett.133.145001.

[149] F. D. et al. “Longitudinal phase space density tomography constrained by the Vlasov-Fokker-Planck equation”. en. In: *Proc. IPAC'24* (Nashville, TN). IPAC'24 - 15th International Particle Accelerator Conference 15. JACoW Publishing, Geneva, Switzerland, May 2024, pp. 2350–2353. ISBN: 978-3-95450-247-9. DOI: 10.18429/JACoW-IPAC2024-WEPG55. URL: https://indico.jacow.org/event/63/contributions/4339.

[150] A. Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[151] A. Raffin, J. Kober, and F. Stulp. *Smooth Exploration for Robotic Reinforcement Learning*. 2021. arXiv: `2005.05719 [cs.LG]`. URL: `https://arxiv.org/abs/2005.05719`.

[152] *Open Neural Network Exchange repository*. `https://github.com/onnx/onnx`. Accessed in November 2024.

[153] *onnx2versal repository*. `https://github.com/rehohoho/onnx2versal`. Accessed in November 2024.

[154] H. Qian and Y. Yu. "Derivative-free reinforcement learning: a review". In: *Frontiers of Computer Science* 15.6 (Sept. 2021). ISSN: 2095-2236. DOI: `10.1007/s11704-020-0241-4`. URL: `http://dx.doi.org/10.1007/s11704-020-0241-4`.

[155] J. W. Cooley and J. W. Tukey. "An algorithm for the machine calculation of complex Fourier series". In: *Mathematics of Computation* 19.90 (1965), pp. 297–301. ISSN: 1088-6842. DOI: `10.1090/s0025-5718-1965-0178586-1`. URL: `http://dx.doi.org/10.1090/S0025-5718-1965-0178586-1`.

Acknowledgments

This thesis is the result of several years of work, sitting at the crossroads of three vast fields. As such, it would not have been possible without the support of many incredible people.

First and foremost, I want to thank my supervisors, Prof. Jürgen Becker and Prof. Frank Simon, for their guidance and feedback, which helped shape this thesis into what it is today.

A huge thank you goes to Michele Caselle for welcoming me into his group and giving me the opportunity to work with cutting-edge technology, an essential pillar of this research.

I am also deeply grateful to the people at the Institute for Data Processing and Electronics for their continuous support. Lars Eisenblätter, Alexander Bacher, Peter Schöck, and Jennifer Derschang, thank you for always lending a helping hand, often at the last minute. A big shout-out to the administration department for their support in navigating the many bureaucratic hurdles along the way. The PhD student group here was truly amazing, always ready to organize an outing or a fun activity, Fabian, Alexander, Olena, Meghana, and Nour, you made this journey infinitely more enjoyable. I also want to thank Suren and Michael for the amazing discussions. A special mention to the two people who not only gave me invaluable insights into my work but also patiently endured my endless rambling about the weather: Andreas Kopmann and Timo Dritschler.

This research would not have been possible without Andrea Santamaria Garcia. I will always be grateful for her introduction to reinforcement learning and particle accelerators.

The team at the Institute for Beam Physics and Technologies, operating KARA, was nothing short of fantastic, giving me the rare privilege of hands-on experience with a particle accelerator. I am especially thankful to Prof. Anke-Susanne Müller and Erik Bründermann for the insightful discussions throughout this work. To Marcel Schuh, Edmund (Eddy) Blomley, Akira Mochihashi, Michael Nasse, Julian Gethmann, and Johannes L. Steinmann, thank you for your endless patience in listening to my ideas for new experiments. And to Chenran, for all the fun memories at meetings and conferences, and for being my go-to person for all things reinforcement learning.

I also want to thank Dr. Dmitry Teytelman from Dimtel, for his help in implementing the custom LLRF control port and for the inspiring discussions together.

I acknowledge the support of the Doctoral School “Karlsruhe School of Elementary Particle and Astroparticle Physics: Science and Technology” (KSETA).

To my students, Michail Sapkas, Ziyu Xie, Sujeet Mehta, Florian Eisele, and Yannick Vollmer, thank you for being the environment where I could reinforcement-learn how to be a better supervisor and teacher.

Moving to a new country at the start of this journey was no small challenge, and I could not have done it without the unwavering support of my parents, Franco and Lucia, and my sister Emma. To the friends who kept me company through the ups and downs, Ekaterina, Felix, Simon, Angeline, Prashant, Praneeth, and Yasha thank you for being there. And finally, to Eleonora, who stuck with me even when I said, “Maybe I should accept this PhD offer in Germany...”, thank you for everything.