Essay, WASP Software Engineering Course Module 2025

Kristina Levina

1 Introduction

In my PhD project, I tackle the problem of scaling symbolic reinforcement learning on two levels: (1) fundamental research and (2) real-world application. At the fundamental level, I research how to extend symbolic reinforcement learning towards scalability, efficiency, and applicability in complex domains. In symbolic reinforcement learning, high-level representations of tasks and goals can significantly enhance learning efficiency and safety in real-world scenarios. The symbolic part comes from reward machines—automata-based structures—that allow to encode reward structures of the environments and specifications for the agent's behaviours using domain knowledge.

Reward machines allow reinforcement learning agents to follow structured and interpretable guidance during learning, reducing the risk of harmful/unsafe behavior that can frequently arise in complex domains from solely trial-and-error learning. For example, without structured guidance, agents may discover undesired shortcuts that maximise rewards but violate safety or ethical constraints. By incorporating domain knowledge directly into the reward structure, reward machines ensure that the agent's learning process is aligned with the desired task specifications. Reinforcement learning agent can now use the available reward structure and not deviate form it to find harmful but more optimal from the perspective of the agent ways to achieve goals.

However, reward machines have scalability problems, and I actively research how to mitigate this in my research. As task complexity grows—especially in stochastic or partially observable environments—state-of-the-art reward machines struggle with state-space explosion and inefficient policy learning. I explore new formulations of reward machine states, transitions, and abstractions to allow for compact yet expressive representations that support efficient learning.

Next, the extended reward machines will be applied to energy optimisation in radio base stations. This domain is particularly relevant in the context of increasing demands for sustainable and autonomous communications infrastructure. This is of particular importance to enable automated decision making in communications networks with reinforcement learning agents because domain knowledge should guide the agents and disallow any harmful behavior. Therefore, enabling reinforcement learning with reward machines in safety-critical applications is another area of my PhD project.

In the end, I aim to achieve the following PhD project targets: 1) Enable the application of reward machines in stochastic, complex, large, and valuable domains; 2) Make reward machines more popular in industrial applications by 3) researching their application in automated energy optimisation.Through this work, I aim to bridge the gap between theoretical advances in reinforcement learning and their practical deployment in mission-critical systems.

2 Lecture principles

2.1 Idea from Lecture 1

Agility can be quite important in a PhD project as well. That is, research blocks that include ‘Idea generation‘, ‘Literature review‘, ‘Planning and Design‘, ‘Theoretical implementation‘, ‘Empirical Results‘, and ‘Paper‘ should not come one after another in a cemented manner. The process should rather be iterative, where it is completely fine to go back and redesign the idea or perform additional literature review. Alternatively, the researcher can stuck in one phase indefinitely. How frequent iterations are is up for the researcher. Some might even conduct several phases simultaneously.

2.2 Idea from Lecture 2

The normal empirical pipelines are usually implemented by one PhD student who is responsible for his/her project. Even though the researcher is motivated to make it right, introduction of bugs and bad practices is inevitable when working on the code implementation alone. We might think that these will be caught at some point by either supervisors or pier-reviewers. In reality though, nobody has time to get into your work, and increasing productivity with AI doesn’t help and even makes things worse in terms of quality.

The real code reviewer is the next researcher who is interested in the idea after reading already published paper. If building on the idea, the next researches takes the code and can potentially find inappropriate parts of the code. At this stage, the paper is already published, and nothing really can be done except for writing a new paper and exposing the existing paper. This takes time, and the feedback is too slow for the typical PhD cycle.

The proposal is to integrate normal code review into the PhD cycle. The idea on the theoretical level is typically already reviewed by supervisors; only the code review is missing. How to solve this? Allocating another PhD student to make the code review might help. Another solution would be to hire an independent researcher/software developer who will be responsible for code reviews/advice of all members of the lab. This of course spends resources of the lab but brings quality to the code, simplifies code writing because all members of the lab can get advice on coding from this expert. Ultimately, it results in more paper publications from the lab because code writing is now faster. Every PhD student knows how implementing experimental pipelines takes precious time and how annoying and labour-intensive is bug fixing. So, in the end, the lab wins in quality and number of papers, which is an important metric to push for getting grants.

3 Guest-Lecture Principles

3.1 Idea from Guest 1

Requirements engineering? As the number of stakeholders in a typical research lab is limited and the product being developed is a paper, almost all requirements are specified on the conversational level. However, documenting them helps not only the researchers to not forget specific details but also helps to structure the project.

3.2 Idea from Guest 2

It is obvious that politics within the research lab plays an important role in decision making. I am not sure what to say more about it. In relation to my research specifically, I can emphasize that politics at the industrial work place is more influential on decision making that in the research lab at university. Hierarchy can look differently from the inside that from the outside in a company.

If the researcher wants to be heard, relationships with team members is much more important than the work being done.

4 Data Scientists versus Software Engineers

I agree with the written text in the book regarding the differences in the two roles. machine-learning researchers are often data scientists in soul and are focused much on the model performance using some metrics rather than building a holistic system. Software engineers, in contrast, are focused on delivering a real-world product that is robust and resilient. However, with more and more awareness regarding this discrepancy, some research labs are increasingly discussing the specified traps on integrating the developed models into MLOps-like systems. Software engineers, on the other side, are getting more and more awareness and pressure from the users to get, for example, good accuracy on the models. Therefore, these roles might merge with time. However, it is really important to emphasize that pure researches in data science might still focus on components' improvement rather that the whole system. This is simply the nature of research. Another reason is of course the size of a holistic machine learning product in relation a specific research questions being asked.

5 Paper analysis

5.1 Paper 1: DDPT: Diffusion-Driven Prompt Tuning for Large Language Model Code Generation

5.1.1 Core ideas and their SE importance

In [1], the authors try to improve the performance of LLMs in code generation. For this, instead of fine-tuning the given LLM for code tasks, they keep the LLMs frozen and instead map inputted natural-language text into an embedding that is directed towards and optimal embedding. Optimal prompt embeddings are learnt from Gaussian noise, using a diffusion-based optimization process. The pipeline is called Diffusion-Driven Prompt Tuning for Large Language Model Code Generation (DDPT). They also want to reduce labor effort in prompt engineering. So, two main emphasised contributions of this paper that are relevant for SE are 1) autonomous prompt engineering and 2) making the prompts structured in the vectors space, which enables usage of all kind of ML for again automation. Remarkably, the authors use open LLM codet5p for their experiments, which seems overly restrictive, but they list this among many more limitations of their paper in the corresponding sections, and either frame them as future work or simply things that could be improved.

5.1.2 Relation to my research

No relation thus far because I don't use prompt engineering and LLMs. However, I could use some paper writing and structuring I encountered in this paper in my papers. For example, clearer discussion of limitations.

5.1.3 Integration into a larger AI-intensive project

Let's have a hypothetical RL system that takes high-level task instructions in natural language. The LLM part translates natural language to task specifications. DDPT could generate optimal prompt embeddings, for example. By automating mapping from language to task specifications, users don't need to think anymore and use their countless hours in defining tasks symbolically. That's right, my reward-machine speficications would be generated by this DDPT, and users would enjoy safe RL, or almost safe RL because LLMs are probabilistic, and there is always a margin for error.

5.1.4 Adaptation of my research

Hypothetically, to support such DDPT-style structured prompting and its engineering aims, I can do the following actions:

Code pipelines would need to be redesigned to be more holistic such that they can support LLM APIs in the first place. Without this compatibility, my code pipelines would not be capable of using DDPT. Right now, I use .txt files that contain either Pyhon-generated reward machines or manually constructed specifications in the form of reward machines. These .txt files are used as input to RL agents for guiding the learning process based on the reward-based feedback that abstracts the environment. While this approach works fine for controlled experimentation and testing new algorithms, it lacks the flexibility and automation needed for integration into real-world scenarios, where people don't want to spend hours on specifying tasks.

Instead, I would need to change the input of my models and environments such that they get instructions from LLM-generated symbolic representations of the task—reward machines. That are in turn generated from natural language. In this way, the pipeline should be more end-to-end and include natural-language instruction from users. Therefore, instead of relying on static .txt files, the environment classes would directly consume symbolic task representations—reward machines—generated by LLMs based on high-level natural-language inputs. Prompt embeddings can be generated from these natural-language instructions by DDPT and directly map to RM specifications in this complex system.

Hence, the entire pipeline could be made end-to-end. That is, starting from a natural-language task description given by a user, the system would use DDPT to produce an optimal prompt embedding. This embedding would serve as an abstract representation in the vector space that is trained to map to a symbolic reward machine specification without manual specifications. Such a pipeline would lead to both automation and perhaps improved structure. Automation and structure are important characteristics of large AI projects.

I would be cautious in doing so however without proper verification of symbolic task representations in the form of reward machines. Do they really reflect what the user inputted in natural language? Does the user know what he/she needs while thinking in natural language and being abstracted from logic? Even if the user is sure that he/she knows how to prompt the required behaviour, does the LLM agent capture this behaviour correctly? There is always a margin of error when we are thinking in terms of probabilistic methods. The verification is thus very much needed, and I would need to figure this out how to support that. To this end, automated checks, human-in-the-loop validation, or interpretability tools could be of use. This can be a large research step in itself.

In the long run, my project could move toward permitting end-users to describe tasks in most common programming language—plain English. DDPT could automatically convert this plain English to symbolic reward machines. RL agent could reliably complete given goals. Everybody could be happy. This would be a research project that could be aligned with modern research directions of human-centric AI moving away from bulky and difficult to pronounce logic specifications.

5.2 Paper 2: Investigating Issues that Lead to Code Technical Debt in Machine Learning Systems

5.2.1 Core ideas and their SE importance

In [2], code-related technical debt is investigated with relation to machine learning systems. The authors analysed 24 highly relevant problems to technical debt in machine-learning systems. They used groups led by experts for investigations. The authors identified data preprocessing to be one of the most contributing problems to technical debt. In essence, inappropriate feature selection, missing value identification, and outlier detection, introduce bias or hinder generalisability. These problems propagate via the machine learning pipeline, meaning that correcting the models becomes challenging and effortful. Not only development time and maintenance costs are increased, but also “patch fixes” rather than holistic solutions further contributes to code technical debt. The paper presents maintenance challenges of combining machine learning and traditional code. In addition, the paper clarifies that poor engineering practices in machine learning pipelines propagate technical debt.

5.2.2 Relation to my research

My research is aimed at scaling symbolic reinforcement learning via reward machines. Therefore, this paper is highly relevant. Indeed, technical debt can potentially accumulate in several ways: 1) Pipeline Complexity: Reinforcement learning with reward machines involve layered code with multiple classes and interactions between them (specification, translation, environments, RL agents). This complexity contributes to technical debt, particularly in preprocessing and specification stages. 2) Also, currently source code is copied and then modified to fit into the desired functions, resulting in inability to upgrade to newer package versions. Therefore, maintainability is comprimised as my algorithms continue to be developed. The risk of accumulating technical debt grows.

5.2.3 Integration into a larger AI-intensive project

Let’s have a fully automated energy-optimizing network agent deployed in telecommunications infrastructure. Tasks are specified via reward machines. Real-time data pipelines preprocess metrics (for example, power, down-link power, sleep modes, usage). Learning modules operate as reiinforcement learning agents with given reward machines as integrated domain knowledge. The agents optimize base station behavior under constraints. Now, the paper can help to identify potential technical debt sources. Highly likely, one source is preprocessing modules that translate raw sensor data into symbolic rewar-machine-compatible states. In other words, events that govern transitions in the reward machines should be identified from raw sensor data, which may complicate things. Furthermore, the reward machine-based learning can be made robust and sustainable by taking looking into critical aspects of the paper’s ideas like doing proper hyperparameter tuning to decrease the development cost.

5.2.4 Adaptation of my research

To incorporate the paper’s ideas on sources of technical debt in machine learning systems into my research, many steps could be done. First, pipeline should not be dependent on source code modification hindering further package updates. Next, logging and various checks can be implemented to find bad preprocessing steps contributing to technical debt. For example, reward machine specifications should be clean and not require ambiguous preprocessing functions. Second, modularised design is of utter importance. All parts of the pipeline should be isolated and be as independent as possible: task specification, preprocessing, reward machine translation, reward machine environment wrapping, agent training, etc.

Now, I mention specific steps to ensure minimal technical debt of my reinforcement learning system that uses reward machines for task specifications:

- Problem at the *data collection phase*: As reinforcement learning agent collects data via interaction with the environment on fly, the problem directly is not present in my system. However, reward machine-specifications if done manually in one language need to be translated automatically to reward machines that can serve as an input to a particular algorithm developed. This can accumulate unnecessary code with time as more algorithms are developed. The solution can be: one specification language that is fixed for all possible methods and algorithms during the development. The user always inputs tasks in this language, which means it has to be expressive enough. In our recent article, which is under review phase, we proposed numeric reward machines for this sake. For each algorithm that utilises a specific form of the reward machine then, there exist a translation algorithm—function that takes an input in the above-mentioned language and produces an output in the form required by a specific reinforcement learning algorithm. Let it be a hierarchical learning approach or plain Q-learning. These translation functions are independent and can be easily removed if some algorithms become unneeded as research progresses. This also will enable easier integration of the reinforcement learning system with reward machines to a larger industrial project.

- Problem at the *Data Pre-Processing Phase*: Directly this problem arises only when identifying which Boolean features are true in the environment. These are usually read from sensor data in real-world scenarios. The solution is automating the sensor-data processing such that it is easy to change the logic without revisiting too much code. In particular, this function of taking the raw sensor data as input and outputting true events in the environment that are relevant for traversing reward machines will need to be autonomous, aka modular, and independent of other code parts. This is critical for system integrity and reducing code technical debt.
- Problems at the *Model Creation and Training Phase*: The main problem here can be concerned missing hyperparameter tuning in my system. As the hyperparameter tuning is not performed, the chosen set of parameters may be very much sub-optimal, requiring more and more training time in the case if developed models fail to converge. Solution: instead of using the worst set of parameters that just works, a normal grid-search should be done to find the best parameters that ensure convergence of the developed algorithms. This can save much time during the development.

To sum up, quite many steps can be done to ensure minimal technical debt of my reinforcement learning system with reward machines. Special care should be given to early blocks in the pipeline and automating certain procedures like hyperparameter tuning.

6 Research Ethics & Synthesis Reflection

Public availability was one of my criterion for paper selection. Another criterion was that the long articles were published in CAIN 2025 to ensure novelty. My interest and applicability to my research were not applied sufficiently in the paper selection due to the limitations of public availability, which took higher priority. Arxiv.org was the paper provider (https://arxiv.org/abs/2504.04351 and https://arxiv.org/abs/2502.13011).

References

[1] Jinyang Li, Sangwon Hyun, and M Ali Babar. "DDPT: Diffusion-Driven Prompt Tuning for Large Language Model Code Generation". In: *arXiv e-prints* (2025), arXiv–2504.

[2] Rodrigo Ximenes et al. "Investigating Issues that Lead to Code Technical Debt in Machine Learning Systems". In: *arXiv e-prints* (2025), arXiv–2502.