SAPIENZA
UNIVERSITÀ DI ROMA

Like Father Like Son

Automatically Generating Rewards For Reinforcement Learning

The Department of Computer Science

Corso di Laurea Magistrale in Informatica

Candidate

Raffaele Leo
ID number 2047688

Thesis Advisor	Co-Advisor
Prof. Velardi	–

Academic Year 2023/2024

Thesis defended on 21 October 2024
in front of a Board of Examiners composed by:

Prof. ... (chairman)
Prof. ...
Prof. ...
Prof. ...
Prof. ...
Prof. ...
Prof. ...

Like Father Like Son
Master's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: leo.2047688@studenti.uniroma1.it

Abstract

This thesis explores how auto-generating rewards can help enhance the Reinforcement Learning (RL) process for visualization recommendation systems. The primary goal is to develop an auto-reward generating system that does not have the overhead that the current state-of-the-art systems require. When using PPO learning to train a model, it gives updates and metrics for how the model is learning, how far it is straying from its original weight, and how well it understands the rewards it is getting. We build what we call a Stable Observer Network (SON) that uses these metrics to sit and observe how well the Proximal Policy Optimization (PPO) model understands the rewards it is getting back. After observing for a set amount of iterations, the SON takes over as the reward function; predicting the rewards for each output of the PPO model. We then show that while using our SON to predict rewards, our models consistently perform better than the baseline for visualization tasks. These results show that modifying feedback based on the PPO metrics can help visualization models when using RL, and that using an auto-reward generator does not equate to needing hours of overhead before even starting.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The need for data visualization has been intertwined with the growing complexity of human history. To make well-thought-out and organized decisions, we need well-laid-out and organized visualizations to accompany us. The first visualization ever recorded is a map from ancient Egypt dating back to 1160 B.C. which displays the distribution of minerals and quarrying information for these minerals spread through the Wadi Hammamat mountains[5]. As the ancient Egyptian empire grew and the issues of gathering resources became more complex, people needed to visualize the resources available to make informed decisions about the future.

We live in an era where the amount of data generated every day is staggering. From social media interactions to scientific research, the volume of data we produce and collect has grown exponentially. This phenomenon, often referred to as big data, makes the data collection of ancient Egypt feel like a bucket of sand in the desert. For the average user, attempting to sort through and effectively use this vast ocean of data can feel impossible. To solve this, data visualization becomes crucial as it takes raw data and transforms it into comprehensible graphics, enabling users to make more informed decisions.

However, traditional data visualization methods often struggle to keep up with the ever-changing nature of big data. These initial methods typically involved hours of manual tuning on pre-defined rules to systematically figure out the best way to generate a visualization. Although these laid the groundwork for future research these tools were too rigid and specific to a specific task to be useful on a larger scale. What users are looking for are visualization tools that are not only aligned to their preferences but also adaptable to new and different kinds of data.

This is where advancements in machine learning have come into the forefront of this area. With a boom in computer power and at the same time new ideas for interpreting text, we can now take natural language as an input parameter effectively for the first time in our history. Large Language Models have the ability to interoperate human language and output a sequence based on the input. Combining the rules and language that traditional systems started with the flexibility of the new LLMs created a boom in this field.

Natural Language to Visualization is what came out of this new wave of technology. Now the adaptability and flexibility issues of the traditional visualization systems are a thing of the past. This area of study is called LLM for VIS and it is where

this thesis resides, we use LLMs for creating visualizations through natural language queries. These large models however are great at general tasks but have a hard time learning the specifics of a user's needs. Here recent advancements have shown that using Reinforcement Learning (RL) to fine-tune these generic models can help specialize the model for our specific needs.

Reinforcement Learning is a type of machine learning where models learn optimal behaviors through trial and error, receiving rewards or penalties based on their actions. RL has been effectively used in well-defined environments, such as game playing and robotics, where the rules and objectives are clear.

However, applying RL to data visualization introduces unique challenges. In this context, the "environment" is less concrete, as the model must interpret and visualize textual and numerical data, learning to produce outputs that best meet user expectations. One of the core challenges in using RL for data visualization is defining an appropriate reward function. Unlike simpler tasks where success can be easily quantified, the quality of a single visualization is often subjective, relying on human preferences and the specific context of the data. An optimal reward function must therefore capture these nuanced criteria to guide the model effectively.

To enhance RL's applicability to data visualization, Reinforcement Learning with Human Feedback (RLHF) has emerged as a promising approach. RLHF enables models to iteratively receive and learn from human feedback, creating visualizations that evolve alongside the data and the needs of the user. This approach leverages the adaptability of RL while incorporating the critical insights and preferences provided by human users.

To address these challenges, I propose implementing an Observer model to analyze the responses of large language models (LLMs) to given rewards and to eventually generate these rewards autonomously. The Observer model leverages feedback from Proximal Policy Optimization (PPO) to refine the reward signals, promoting faster and smoother convergence of the LLM in generating high-quality visualizations.

This approach stems from advancements in self-learning reward functions, such as those facilitated by Hugging Face, which require extensive dataset restructuring to provide proper feedback. The Observer model we propose simplifies this process by taking in the ground truth, predicted output, and discrete reward, then generating a suggested reward based on PPO feedback. This model is particularly tested and applied within the scope of sequence-to-visualization tasks, aiming to improve the generation of visual outputs aligned with user preferences.

1.1 Thesis Overview

Here we will give a brief overview of how this thesis is laid out and what to expect from each section, for a generic overview see Figure 1.1. We have three main chapters starting with Related Works, we then lay out the problem we are trying to solve and how we plan to solve it, and finally lay out the methods used to solve this issue.

In Chapter 2 (Related Work) we present a brief history of Visual Recommendation Systems, showing how we got to where we are today and where our contribution fits.

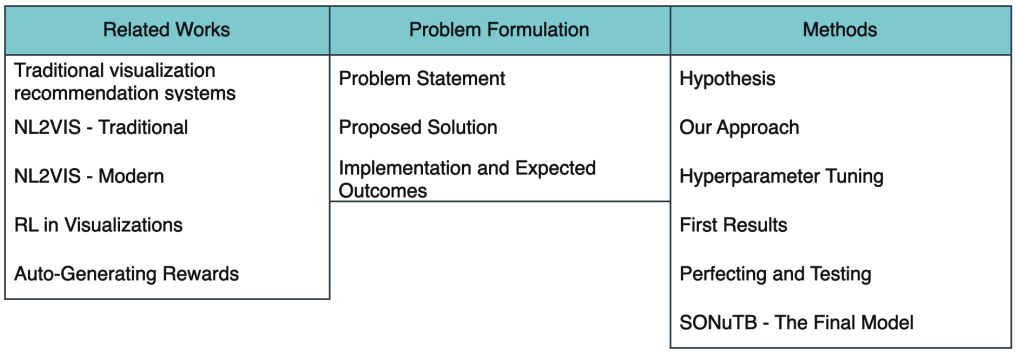

Figure 1.1. How this thesis is organized from a bird's eye view.

- **Traditional Visualization Recommendation Systems**: An overview of the beginnings of the visualization field. Focuses on three seminal works that attempted to automatically generate visualizations using a complex set of rules and regulations. These logical stipulations lay the foundation for visualization languages to come.
- **NL2VIS** The next section talks about Natural Language to Visualization. This was the goal for many years to achieve, as a model that can interoperate natural language can be used by anybody willing to try. These models used a variety of tactics for interpreting natural language from techniques like statistical natural language processing all the way to modern-day transformers
- **RL for LLM on vis tasks**: We then get to Reinforcement Learning for Large Language Models on Visualization tasks. This is the related works section that is most closely tied to this paper, and we go through different techniques for accomplishing this task and what areas are lacking in this field.

In Chapter 3 (Problem Formulation) we define the problem formulation along with how our solution resolves it.

- **Problem**: We first lay out the underlying problem we are trying to solve. We begin by stating the currently implemented solutions to this problem and the areas in which these solutions fall short.
- **Proposed Solution** We then introduce the Stable Observer Network which is our proposed solution to the above issues. It is discussed in a very high view, leaving the details to the implementation section and Chapter 3.
- **Implementation and Expected Outcomes**: Finally we discuss how we would like our observer network to be implemented and what the expected outcomes of this implementation would be. We discuss the dataset we are using, how the rewards are calculated, what is used for baseline, and the metrics we use to train our model.

In Chapter 4 (Methods) we go through our methodology and experiment results; how we arrived at our final solution and why it works.

- **Hypothesis**: Coming into this section we state our ultimate goal with this thesis and the assumptions coming into our issue. It lays the groundwork for this section.
- **Our Approach** Goes into our approach with more technical detail. We discuss our loss function and how we arrived at making it along with all of the parts and what they are made up of in detail.
- **Case Study 1**: Our first test of our Stable Observer Model and the promise it showed. Although we ended up making many adjustments to our original model this was a crucial step in our scientific process.
- **Case Study 2**: This section discusses the steps between our initial test and our final result. We go through the different trials of our SON and how we ended up arriving where we did.
- **Case Study 3: Stable Observer Network using TinyBert**: SONuTB is the final step of our scientific process. We ended up utilizing TinyBert for our observer model and optimizing our loss function for this new model. In this section, we also discuss the results of this final product and examples showing why it works.

Chapter 2

Related Work

As the use of big data has risen across various domains, the challenge of easily interpreting and utilizing this data has grown in parallel. The larger and more complex the data we are using becomes the harder it is for users without a technical background to grasp what it is telling us. Visualization Recommendation Systems (VRS) have emerged as crucial tools to bridge the gap between complex data sets and the users who need to interpret them. VRSs aim to automate the process of selecting the most appropriate visual representations for given data, thereby making data analysis more accessible and insightful for both experts and non-experts. These systems leverage a range of techniques, from rule-based approaches to advanced machine learning models, to recommend visualizations that enhance understanding and decision-making. A practical example of this is Tableau's "Show Me" feature, which suggests visualizations based on the data selected by the user. Such tools have become integral in various fields, providing tailored visual recommendations that facilitate deeper data insights and more informed decision-making.

VRSs can be categorized into several typologies based on their underlying methodologies:

- **Traditional Visualization Recommendation Systems:** These typically use predefined rules and heuristics to suggest visualizations. They rely on a set of established guidelines and best practices to match data types with appropriate visual formats.
- **NL2VIS (Natural Language to Visualization):** This category involves translating natural language queries into visual representations. Techniques in this area range from traditional methods to advanced models like sequence-to-sequence learning using Transformers or Large Language Models (LLMs).
- **Reinforcement Learning (RL) for LLMs on Visualization Tasks:** This approach applies reinforcement learning principles to fine-tune their models through interaction with an environment. This can include methods such as Reinforcement Learning from Human Feedback (RLHF) and auto-generating rewards to optimize the recommendation process.

In this section, we move through the evolution of VRSs over time, highlighting the most important breakthroughs and how these insights have led us to where we

are today.

2.1 Traditional visualization recommendation systems

Before the emergence of machine learning models, many early systems used complex chains of rules to generate visualizations based on natural language queries. One of the earliest papers written on predicting visualizations based on a user query is Mackinlay's APT (A Presentation Tool) (1986)[14]. Mackinlay used a rule-based approach that matched data attributes to a set of predefined visualization templates. This was one of the first papers to propose visualization as a language of its own, with rules that can be mathematically manipulated and optimized. Building on this foundation, Roth and Mattis (1990)[19] expanded on this idea of finding the correct visualizations using discrete rules, but they wanted to make more adaptive graphics presentations. Their work emphasized the importance of adapting visualizations to the specific needs of the dataset and user, taking in user feedback to dynamically improve the outputs. Although this work came well before the term VRS with human feedback was coined or popularised, a lot of the ideas discussed within this paper have strong parallels with the modern deep-learning technique.

Moving forward into the 2000s, new techniques emerged that moved beyond the foundational rule-based and heuristic approaches and introduced VRS that could handle more complex data and inputs. This made them much more useful in the mainstream instead of just theoretical work. These innovations were driven by enhanced computational power, more sophisticated algorithms, and a stronger focus on user experience and scalability.

One of the first innovations came from a 2002 paper written by Stolte et al[23] which introduced a system called Polaris to help users extract data from cluttered and messy databases. This paper came out in a time when large data handling was fairly new to the computing world and ways to handle it and visualise it were severely lacking. Polaris, which later evolved into Tableau, utilizes a declarative language for visualization specifications, allowing users to create visualizations through a drag-and-drop interface. The system uses these algebraic functions to arrange the visual elements, translating user interactions with the drag-and-drop table into SQL queries, allowing users to explore their data without needing knowledge of SQL.

The next step was taken in 2004 with a paper released by Viegas et al[26] which explored history flow visualizations. They developed a technique to analyze the collaborative editing process that takes place every day on online platforms like Wikipedia. They helped to visualize the evolution of a document over time, highlighting patterns of both cooperation and conflict among authors. The paper demonstrates the effectiveness of using history flow visualizations for analyzing author interactions, document stability, and the nature of conflicts, thus pushing the boundaries of how temporal data and collaborative processes are analyzed and understood in the field of data visualization.

In 2006 Heer et al.[6] presented a framework for improving interactive visualizations while taking advantage of established software design patterns. These patterns, taken from research and user studies, help to provide solutions to common visualization challenges. For example, they use an "Overview + Detail" pattern that

allows users to see both a broad overview of the data and detailed views simultaneously, allowing for an easier exploration of the data. Another pattern they use is called "Dynamic Query," which allows users to filter and query data in real-time, allowing easier exploration of the data. This is particularly useful in large datasets, where traditional static queries make it difficult to get a holistic impression of the data. Their work shows the importance of using design patterns to address usability issues, improve user interaction, and make complex data more understandable for the average user.

Despite their contributions, these initial systems faced several limitations, including difficulties in handling complex and high-dimensional data, limited adaptability, and challenges in scalability. These limitations displayed the need for more sophisticated and flexible approaches.

2.2 NL2VIS

2.2.1 Traditional techniques

As we move into the modern era, a type of model known as "Natural Language to Visualization" (NL2VIS) took over as the main focus of Visual Recommendation Systems. These models are much larger and more complex than what had been used in the past and can interpret Natural Language inputs with ease. The focus has now drifted away from strict rules into something that more resembles a personal assistant. Traditional techniques, which we will go over in the next section, primarily rely on statistical and natural language processing (NLP) methods to interpret user queries and generate visualizations. A generic example of what NL2VIS looks like can be seen in Figure 2.1.

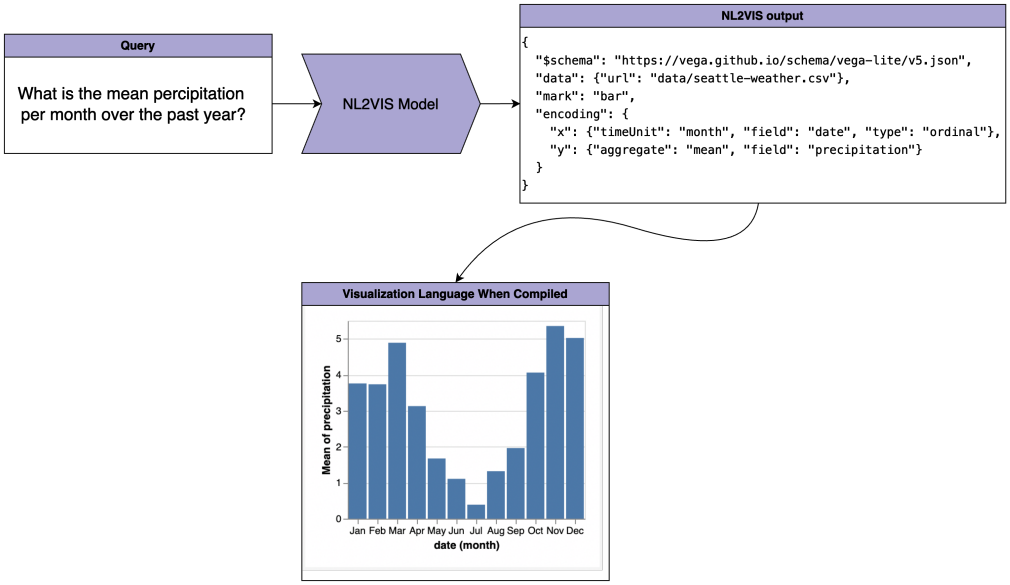

Figure 2.1. A generic example of an NL2VIS model. Note that the model outputs a visualization language instead of the graphic, in this case we used Vega-Light as the example visualization language.

The first example that comes to mind is BOLT, a system designed for multi-dashboard authoring using natural language. Introduced by Srinivasan and Setlur[22], BOLT uses NLP techniques to map user queries to common dashboard objectives, thus generating the appropriate visualizations. The system uses a pipeline that takes natural language inputs, extracts the relevant data attributes, and matches them to some predefined dashboard templates. By parsing natural language inputs and matching them with corresponding dashboard attributes, BOLT allows users to create and customize dashboards interactively, allowing for a more efficient and accessible data analysis.

Another contribution is NL4DV, a toolkit developed to generate analytics for data visualization from natural language queries. Presented by Narechania et al[18], NL4DV combines traditional NLP with rule-based approaches to help interpret the user queries and convert them into visualizations. The toolkit processes input queries using lexical and dependency parsing techniques, taking data attributes along with analytic tasks to generate a structured JSON object. This object includes data attributes, analytic tasks, and Vega-Lite specifications relevant to the input query, allowing developers to create visualization interfaces while keeping the number of parameters relatively low for an NLP task. NL4DV's capabilities can be seen through examples such as rendering visualizations in Jupyter notebooks and developing multimodal visualization systems.

These traditional techniques laid the groundwork for more sophisticated methods by introducing and developing the idea of transforming natural language into visual insights. By leveraging statistical models combined with rule-based systems, early NL2VIS tools showed the feasibility and utility of natural language-driven visualization, setting the stage for further advancements in the field.

2.2.2 Sequence to Visualization using Transformers

One of the first papers to utilise the recent advancement of transformer models is NcNet, written by Y. Luo et al[13]. They generate visualizations using a grammar they developed called "Vega-Zero", which is their own offshoot of A. Satyanarayan et. al's Vega-Lite[20]. NcNet is a Transformer-based sequence-to-sequence model designed for NL2VIS tasks. It introduces several novelties, such as attention-forcing layers to help the learning process and a visualization-aware rendering to improve output quality. The model accepts inputs such as tabular datasets and natural language queries, and it has the optional input of a chart template, which serves as a constraint to limit the types of visualizations generated. By leveraging a natural language to Vega-Zero dataset for training, NcNet achieves high accuracy and relevance in visualization generation, showing huge promise in pushing forward the field of generating visualizations for users with no programming experience.

After the release of NcNet, the interest in using larger transformer models for VRS peaked. This was reinforced by new and exciting Large Language Models (LLMs) being developed around the same time, most notably T.B. Brown et al's GPT-3[1] and H. Touvorn et al's LLaMA[25]. The promising results of NcNet and the excitement of having new tools to work with birthed a new sting of VRSs utilizing these new models to offer a new level of interaction and accessibility.

One such contribution is the work by Wang et al.[27]. This paper explores the

design and performance of sequence-to-sequence transformer-based models on NL to Vega-Light datasets. The authors employ BERT as an encoder and compare it with T5 sequence-to-sequence models. Their approach involves fine-tuning these models to translate natural language queries into visualization specifications. Through extensive experimentation, they demonstrate that BERT and T5 models are capable of capturing the semantic nuances of the queries and generating accurate visualizations. Their work highlights the robustness of transformer architectures in handling complex NL2VIS tasks and sets a new benchmark for future research in this area.

Another system in this field is Chat2VIS, developed by P. Maddigan et al[15]. Chat2VIS fine-tunes ChatGPT to map natural language queries to visualizations. The system extends ChatGPT by incorporating an optional chart template input like NcNET, allowing to specify the desired style and format of the visualizations. This feature ensures that the generated visualizations align with specific aesthetic or functional requirements. The fine-tuning process involves training the model on a large-scale dataset of natural language queries and corresponding visualizations, which helps the model's ability to generate relevant and accurate visual outputs. Chat2VIS demonstrates the potential of leveraging pre-trained language models and fine-tuning them with relevant data to improve the quality of visualization recommendations.

In summary, these advancements show the transformative impact of LLMs on visualization recommendation systems. By enabling the generation of visualizations from more generic natural language queries, these modern approaches make data analysis more intuitive and accessible, pushing the boundaries of what is possible in the field of data visualization. As advanced as these models have become, most lack the specificity required for visualization tasks, and the need for effective fine-tuning methods has become more apparent.

2.3 RL with LLM for VIS

The combination of Reinforcement Learning (RL) with Large Language Models (LLMs) has been the most promising method to fine-tune these larger, more general models. Traditional fine-tuning strategies for LLMs, while powerful, have limitations. These strategies often lack the ability to align generated outputs closely with human judgment, particularly when dealing with complex and subjective tasks such as data visualization. This misalignment can result in outputs that are technically correct but do not meet user expectations or preferences.

Reinforcement learning techniques offer a potential solution to these limitations by incorporating iterative feedback within the fine-tuning process. In RL, models learn to make a sequence of decisions by receiving rewards or penalties based on their actions, allowing them to improve over time. This approach has been successfully applied in various domains to enhance model performance and adaptability.

One of the pioneering works in this area is the Seq2SQL system by Zhonget al[32] in 2018, which employs reinforcement learning as a final step in training their natural language to SQL model. They observed that within the WHERE clause of SQL queries, multiple answers could produce the same or similar results. By integrating RL, Seq2SQL improves the model's robustness, allowing it to learn a

broader range of correct outputs rather than being constrained to a single ground truth per example. This was one of the first approaches to show the potential of RL in enhancing natural language processing tasks within data visualization.

Building on the advancements of RL in NLP, S. Zhang et. al built the Table2Chart [30] system that leverages Q-learning, a model-free reinforcement learning algorithm, to automate chart creation from tabular data. In Table2Chart, Q-learning iteratively refines its chart generation model based on user feedback. Starting with an initial set of rules, the system incorporates user feedback to iteratively improve the effectiveness of its visual representations. The reinforcement learning model evaluates charts using a reward function that considers readability, user satisfaction, and clarity. This user-centric approach demonstrates the adaptability and efficiency of RL techniques in visualization tasks.

Taking the application of RL to more complex data structures, the paper by G. Li and P. He et al.[10] introduces a method for visualizing hierarchical tables using deep reinforcement learning (DRL) called InsigHTable. This paper addresses the challenge of using sparse rewards in hierarchical data visualization by employing a novel DRL framework using auxiliary rewards. The agent in this framework learns actions such as transposing, linearizing, and swapping table elements to optimize the layout and readability of hierarchical tables. By tayloring its strategies based on user feedback and metrics, InsigHTable takes a new step in allowing users to more easily interpret and interact with complex hierarchical data, giving a strong case for the versatility of RL in enhancing data visualization.

Furthermore, the work by T. Tang and R. Li et al.[24] explores using DRL to improve the generation of visualizations with the model PlotThread. This model trains an RL agent to assist users in exploring the design space of storyline visualizations. The system optimizes visualizations' layout, style, and content based on a reward function using accuracy, clarity, and visual appeal as its reward metrics. By focusing on user engagement and iterative improvement, PlotThread finds a balance between data accuracy and visual appeal, again highlighting how using RL to create visualizations leads to a more precise and visually compelling output.

2.3.1 RLHF: Promise and Limitations

While RL techniques have shown great promise, they often require large amounts of feedback to fine-tune models effectively. This is where Reinforcement Learning with Human Feedback (RLHF) comes into play. RLHF combines the iterative optimization of RL with direct human input, allowing models to better align with human judgment. This approach has been successfully applied in various domains to help model performance by refining its outputs based on real human evaluations.

However, RLHF also presents several challenges. Obtaining enough human feedback for the model to get good insights is a huge bottleneck, as only large corporations have enough outreach to collect both a wide birth and large number of feedback. Additionally, the subjectivity that comes with using human judgment can lead to unpredictability in feedback, further complicating the learning process.

2.3.2 Auto-Generating Rewards for RLHF

To address these limitations, recent research has explored the concept of evolving reward models as an alternative to direct human feedback. Automating the reward generation process helps to mitigate the reliance on human input, allowing models to improve autonomously while still aligning to human preferences.

One of the first papers to experiment with this kind of auto-generated feedback was written by Faust et al[4] in 2019. They present a set of evolving reward functions that can be used instead of the traditional discrete reward, giving more dynamic and personal feedback to the model throughout its training process. This method involves using evolutionary algorithms to generate and optimize reward functions, eliminating the need for constant human input. The evolved rewards guide the learning process, enabling the system to improve autonomously, leading to higher quality outputs than if traditional techniques were to be used.

The workflow described by H. Dong et al[3] extends this concept by outlining a structured process for transitioning from reward modeling to online RLHF. This approach involves taking the initial reward models and slowly refining and evolving them over time, keeping the system adaptive and responsive to changing requirements without human intervention. They found that if the reward model was allowed to change its rewards slightly as the learning process continued based on feedback, it lead to a smoother convergence than using a discrete reward that is resistant to change.

The article by A. Khandelwal[9] fully automates a reward system to be given to large language models, a technique now available as a default plug-in for Hugging Face. It involves a process similar to a triplet loss function, trying to show the model what inputs are of higher or lower quality by separating them into different spaces. This process involves training a reward model that evaluates the quality of model outputs, enabling iterative improvements without constant human input. This approach streamlines the feedback loop, allowing for a more efficient and less resource-dependent learning process than traditional RLHF.

2.4 Final Thoughts

In this section, we have explored the advancements in Visualization Recommendation Systems, from traditional rule-based techniques to modern transformer-based models for NL2VIS tasks. While large language models have significantly improved the capability of these systems, aligning model outputs with human feedback remains an open problem, particularly in visualization tasks. Reinforcement Learning with Human Feedback has shown promise in addressing this by incorporating iterative human feedback, yet it is limited by needing large amounts of consistent human input. To overcome these challenges, we propose our Stable Observer Network. This network aims to emulate the usefulness of evolving rewards without the need for real human feedback. By integrating this observer network, we hope to create a more efficient and accurate NL2VIS system, advancing the field of data visualization by enabling more adaptive and scalable solutions that anyone can use, not just large corporations with enough resources. For a quick overview of the history of visualization techniques see Figure 2.2

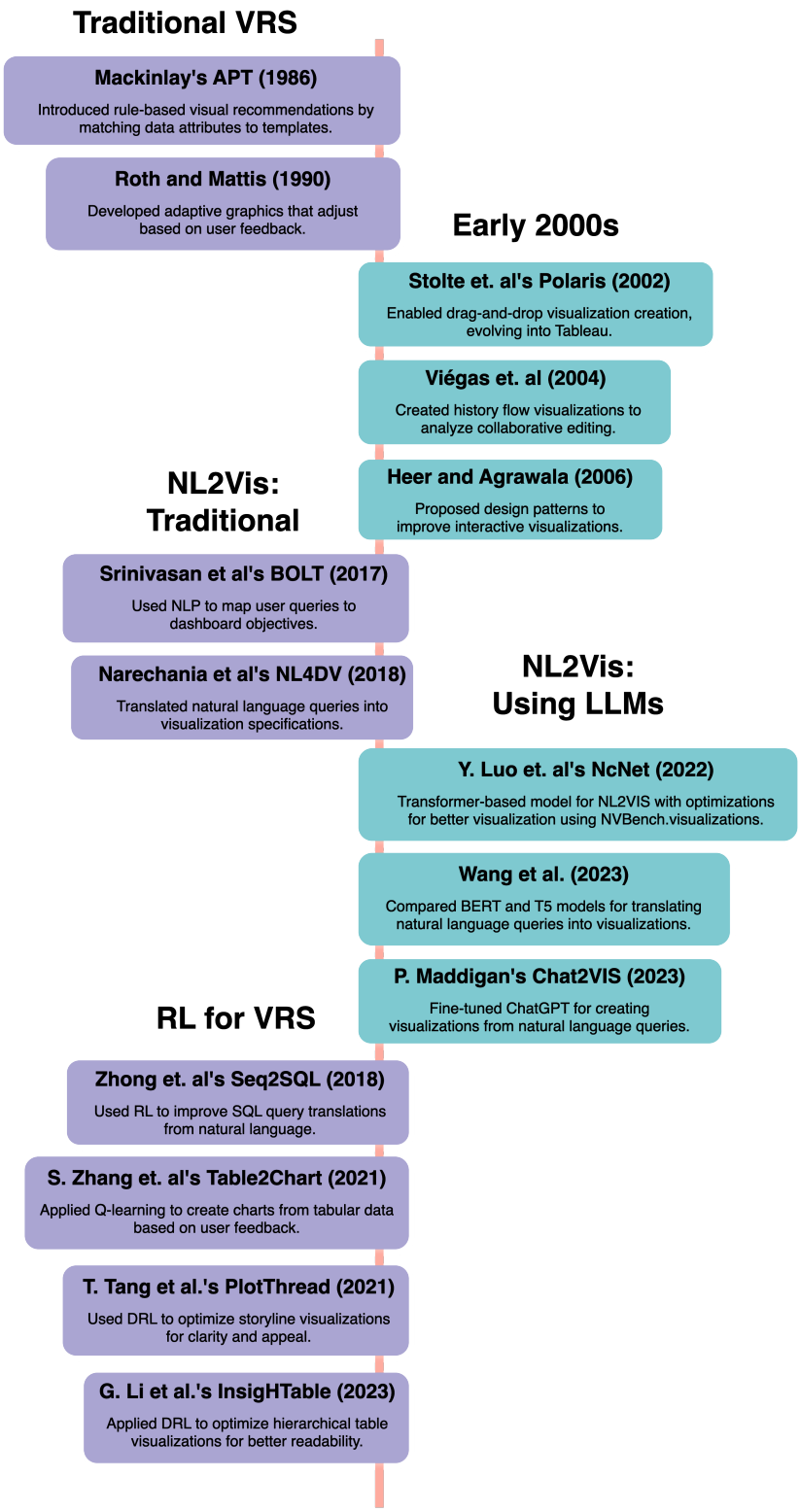

Figure 2.2. A visual timeline for the history of VRS.

Chapter 3

Problem Formulation

3.1 Background

When training a model to output visualizations most modern literature relies on a dataset called NvBench[12]. This was the first modern dataset within the visualization sphere to be large enough and varied enough to train large language models on, you can see two examples of the types of queries and visualizations in Figure 3.1. The release of this dataset in 2017 allowed a boom in research towards visual recommendation systems. Although NvBench is a fantastic starting point, utilizing solely supervised learning to train VRS comes with its drawbacks. When learning on a static dataset with correct/incorrect answers it leads to an inaccurate view of the problem at hand. Helping a user understand the data in front of him with a visual aid is a complex problem with sometimes many different correct answers for a given query. Learning these kinds of problems through only supervised learning leads to models with an inability to read context or any personalization from user queries, and are hard to generalize beyond the scope that they were trained in. These behaviors lead to model outputs that are technically correct but still not helpful for the user's needs. As an initial solution, researchers have turned to Reinforcement Learning (RL) to help strengthen the learning process. Although an improvement; RL only addresses the issue of multiple correct answers for a query, while the static rewards still have a hard time capturing the more subtle issues of context and personalization. This is why most modern LLMs such as ChatGPT rely on Reinforcement Learning with Human Feedback as the primary way to fine-tune a model to align with user preferences.

Traditionally when fine-tuning using RLHF, users will rate individual responses from the model according to how helpful they were. Using these scores as feedback allows the model to more accurately reflect its user's needs. Although this method has been proven to generate extremely dynamic and personalized responses, it is a slow process that requires a huge number of users to give feedback. Unfortunately, that means the best method for aligning models to human preferences can only be effectively used by large companies with access to a vast network of users. To mitigate this issue, a technique has been proposed that involves the use of a reward model. This smaller model generates rewards based on the outputs of the larger LLM that is being trained. This simulates RLHF without actually needing human

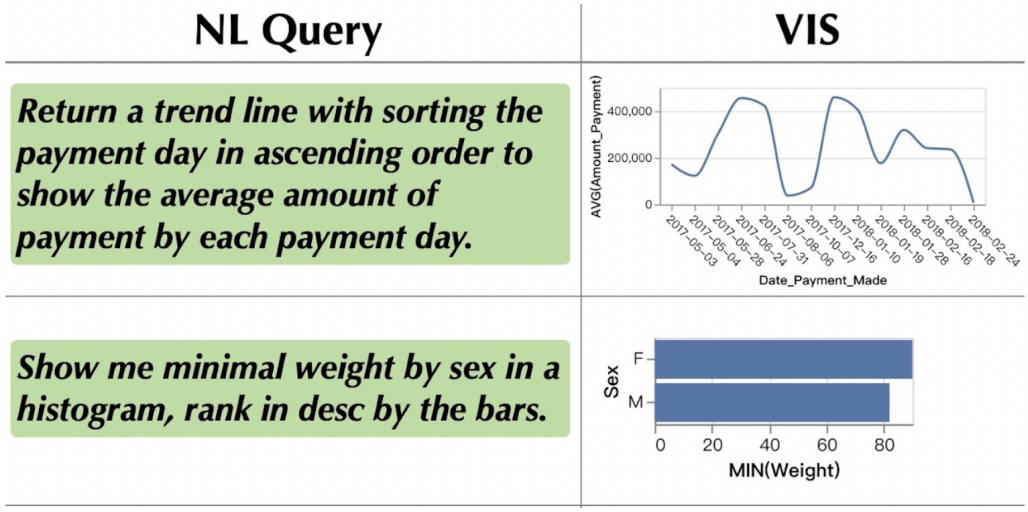

Figure 3.1. An example taken from the NVBench website showing two example queries and outputs of the dataset. The dataset comprises about 13,000 Natural Language to Visualization queries like these, making LLM research possible in this field.

feedback and can help train models to user preferences when you don't have access to a large network of users to give feedback themselves. There have been many different approaches to making this reward model, such as the ones we outline in section 2.3.2. Although the setup is easier than finding hundreds of users to give feedback, these reward models still take a large amount of time and effort to set up. This can be seen when looking at the strategy currently implemented into the HuggingFace PPO API[9]. Where their classifier model starts from a dataset that contains pairs of (question, correct_answer, incorrect_answer), the classifier is trained to produce higher scores for (question, correct_answer) pairs and lower scores for (question, incorrect_answer) pairs.

3.2 Problem

Despite the utility of reward models, several challenges persist:

- **Dataset Requirements**: The requirement of building a new dataset with explicit correct and incorrect answers for each question is time-consuming and costly. If these correct and incorrect outputs are not already given creating pairs that give good feedback to the model is a lengthy process that does not always end with success.
- **Lack of Generalization** After the dataset split has been completed, the next issue is a lack of generalization. There has now been time poured into tailoring this dataset to allow our auto-reward classifier to work, but what is left is a static dataset that is very hard to adjust to new requirements. New and different types of data or different needs that the users are asking for would require top-down renovations of our dataset each time.
- **Static Rules Limitation**: In some cases, datasets may only contain (question,

answer) pairs without explicit quality indicators. In this case, static rules can be defined to evaluate the quality of visualizations, but these rules can be arbitrary and can lead to inefficiencies while the model is converging. When the model is trying to learn a policy with static rules, we have observed that it often gets stuck in rivets or sup-optimal convergences. When this happens there is no flexibility in the system to recognize this and often leads to needing to re-run the learning process until the model reaches an optimal convergence.

In our case, we have a dataset with (question, answer) pairs but no information about the quality of the answers. However, we can define a set of rules to quantify the quality of the visualization generated by the model. Using these rules would allow us to create a discrete reward that can be used to train the model using simple RL, skipping the human feedback. The problem that then arises is these static rules do not consider their impact on helping the model find an optimal convergence, as reinforcement learning from the beginning has been plagued with these kinds of issues, especially in the Visualization field. Our objective is to analyze the model outputs alongside their assigned discrete rewards and ground truths to differentiate between high-quality rewards that facilitate effective learning and sub-optimal rewards that may lead to less optimal convergence.

3.3 Proposed Solution

To address these issues, we propose to implement a smaller model that will generate rewards to be used in the reinforcement learning process of our LLM; which we will call the **Stable Observer Network** or simply **SON**. SON is designed to have two main phases, the observation phase and the inference phase. During the observation phase, SON will observe the LLM undergoing a routine reinforcement learning training loop. During each iteration of this loop, our SON takes in the model's prediction, its corresponding ground truth and static reward, and metrics for how the model is learning. After taking in these metrics using a custom loss function, SON slowly learns how the model is learning, comparing the rewards given for each prediction against how the model responds to said rewards. Once it has observed enough iterations to get a feel for how the model is learning, we allow SON to take over and start feeding its predicted rewards to the LLM. Once SON takes over, the reinforcement learning process proceeds as normal, but with the LLM now taking our model's rewards as feedback instead of the static rewards that usually come with this learning process. In more formal terms we define these two phases as:

- **Training Phase**: Over n iterations, the SON learns to generate these dynamic scores, using the defined loss function to guide its learning process.
- **Inference Phase**: After $n + 1$ iterations, the Observer independently generates scores, eliminating the need for static rules and enhancing the model's adaptability and efficiency.

Through this process, our stable observer network aims to increase the flexibility and effectiveness of the reinforcement learning phase by:

- **Dynamic Score Generation**: SON generates scores based on a loss function that weighs the difference between predicted scores and rule-generated scores at a given time (t_i) and evaluates the model's response to these scores.
- **Automatic adaptability**: As our solution is extremely generic in terms of an approach, it is extremely resilient to changes in the dataset. If new data points are added or changes in the current dataset are made, the ad-hoc nature of our solution will automatically adapt., allowing for a smoother and less frustrating learning process.

Using this approach we end up with a solution more fitting for fine-tuning a visual recommendation system.

This approach provides a more elastic and responsive reward system, ensuring that the model not only learns from predefined rules but also adapts based on real-time feedback, leading to better convergence and overall performance. The Observer model addresses the limitations of static rules and the high costs associated with traditional reward model datasets, presenting a viable solution to improve the RL phase of LLMs, particularly in the context of generating high-quality visualizations.

3.4 Implementation and Expected Outcomes

Our implementation of the stable observer network was done in Amazon Sagemaker and we started our research with the following parameters:

- **Dataset**: The dataset we use is a modified version of NvBench created by the authors of NcNet. This dataset is made up of question-and-answer pairs transforming natural language into Vega-Zero, instead of Vega-Light. Vega-Zero is a modified version of Vega-Light that is formatted in a sentence structure instead of a hierarchical structure, making it easier for LLMs to formulate.
- **Rewards**: The static rewards we use for the learning phase of the observer model follow what has been conventional for visualization recommendation systems using RL. We give higher rewards if the model's response can compile, gets the type of chart correct, and queries the x and y-axis of the chart correctly.
- **Baseline**: For our baseline, we used the Llama 3.0 LLM fine-tuned on 200 instances of the modified NvBench dataset. Using Llama 3.0 keeps us on the cutting edge of LLMs being used and having it fine-tuned on a small number of instances leaves us with a model that has a lot of room to improve, allowing us to compare simple RL to SON.
- **PPO Metrics**: The learning metrics we will be using come from the HuggingFace API for PPO learning This API gives back statistics for how the model is learning, which are what we will use to determine good rewards from bad ones. In particular, we have chosen value-loss, policy-loss, and KL-Divergence as the statistics of particular interest. We will go into detail about what these statistics tell us and why they are important in the next section.

Starting from these conditions our goal is to systematically prove that using a generative reward model in the form that we propose will outperform relying solely on basic reinforcement learning when generating visualizations. We will compare model statistics to see if using the feedback actually helps the model converge smoother and faster. We will then compare validation results, hoping to show improvements in all areas possible. These areas include how many outputs for the validation set are comparable, and have the correct type of chart, x-axis, and y-axis. We hope that these results will inspire others in the field to experiment with auto-generating reward models and will make the lives of visualization recommendation system researchers easier.

Chapter 4

Methods

4.1 Hypothesis

Our ultimate goal is to create an auto-generating reward system that helps the model to converge both faster and more optimally. Our broad hypothesis to accomplish this goal is that if we take into account the model's statistics while generating rewards, we will learn how the model responds to certain rewards and thus be able to help it converge. The main area of focus is Natural Language to Vega-Zero, which is a language providing a combination of a database query and how to chart the queried data. Looking at it through this lens shows that we are combining two areas of study; one being NL to Visualization and the other being auto-reward generation. Both of these fields are very new and exciting to work in, meaning that although we follow established patterns and approaches, a lot of what is done here is novel as we tread into new territory. When looking at our specific problem we make a couple of assumptions:

Assumption 1: Your dataset consists of only query, response, and ground truth.

Assumption 2: Although the ground truth is considered the best option, there are multiple other outputs that will still help the user.

Assumption 3: You do not have the resources to make human annotation possible.

Using these assumptions we make the hypothesis that using a learned reward function will help the PPO function learn a more optimal policy, making the model both converge faster and make more accurate predictions.

4.2 Our approach

This approach was used in the seminal text to SQL paper SEQ2SQL [32], where Zhong et. all used reinforcement learning for the WHERE clause in generating SQL. They realized that there are multiple correct answers for a WHERE clause and that using a simple ground truth for training would not be enough. Therefore they set up a reward function that accounted for the SQL not compiling (-2) compiling but

incorrect (-1) and completely correct (+1). An issue that comes up when applying a discrete reward like this to seq2seq models is it becomes hard for the model to explore as the space that very slim. Even slight deviations from the correct policy can lead to long strings of negative feedback. This phenomenon of finding sub-optimal policies has been combated using evolving[**?**] or learned[**?**] reward functions.

As we started to look into different ways to fine-tune the Llamma model, we were inspired by approaches such as Seq2SLQ by Zhong et al[32] and InsiHTable by G. Li et al[10] to try a Reinforcement Learning approach. Both of these approaches talked about using RL as a tool to widen the scope of the model's outputs. With a task such as outputting visualizations, where there is no one correct output, fine-tuning using supervised learning can lead to punishing correct outputs. As we were already using HuggingFace and Amazon Sagemaker; PPO learning seemed to be the most promising route to take.

4.2.1 Proximal Policy Optimization (PPO)

PPO or Proximal Policy Optimization was first introduced in 2017 by Schulman et. all [21] as a more reliable way to train a model using RL. PPO updates the model's policy while not straying too far from the original, which allows for a more stable learning period. The objective function for calculating the PPO model's next step in the learning process is:

$$\mathrm{L}(\theta) = E_t\left[\min\left(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right) - c_1(V_\theta(s_t) - R_t)^2 + c_2\mathrm{S}[\pi_\theta](s_t)\right]$$

This formula is long and daunting, but it is only trying to accomplish three main objectives; improve the model without taking too large of a step, get a good estimate of how good a policy is, and keep some randomness so as not to overfit the problem. In more detail, the main variables of the formula can be categorized as follows:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio between the current policy and the old policy.
- $\hat{A}_t$ is how well the advantage action has performed, which indicates how much better an action is compared to the baseline.
- The "min" operation, including the clipping term $clip(r_t(\theta), 1-\epsilon, 1+\epsilon)$, makes certain that updates to the policy are small, preventing overly dramatic changes.
- $V_\theta(s_t)$ is the model's prediction for how well this next step will perform.
- R_t is the return value for the action the model had just taken.
- $\mathrm{S}[\pi_\theta](s_t)$ represents entropy, where S controls the entropy of the policy to encourage exploration.
- c_1 and c_2 are both hyperparameters to measure how much the model should focus on fitting to the data versus exploring new policies.

Through this function, PPO learning provides a Reinforcement Learning process that is more stable and less prone to sub-optimal convergences and has quickly become a standard practice within the RL community. PPO is not perfect, however, and still does not solve the problems we stated above. Often simple use of reinforcement learning when outputting complex language leads to sub-optimal convergences. As we started to work out the ideas stated in section 3.3, the PPO metrics became of interest. While training, the PPO model gives back data to show how well our model is responding and learning the new policy. In this section, I want to focus on three of these metrics; Policy Loss, Value Loss, and KL Divergence[31].

Policy Loss: Defined as $L^P(\theta) = r_t(\theta)\hat{A}_t$ and measures how well the policy network is able to select rewards that maximize the overall expected value by taking the probability ratio between policy and multiplying it by the action performance. It is a direct reflection of how well the model is learning the correct policy based on the current environment and rewards. Keeping track of this value shows how clear the environment is to the model exploring it, or how well the rewards express what makes a good vs bad output[21].

Value Loss: Defined as $L^{VF}(\theta) = (V_\theta(s_t) - R_t)^2$ and measures how accurately the value function is predicting the expected returns from a given state or state-action pair. It essentially reflects the model's ability to estimate future rewards accurately. This pairs directly with the Policy Loss to create the full picture of how the model is learning from our reward system, it is particularly important as if the model does not have an understanding of what will come next it cannot converge properly[7].

KL Divergence. Defined as $D_{KL} = \log \frac{\pi_{\theta_{old}}(a_t|s_t)}{\pi_\theta(a_t|s_t)}$ and measures the difference between the current policy distribution and the previous policy distribution. This is a key function of PPO learning as without this measure the model can spiral out of control into sub-optimal output states. When tracking the KL Divergence we want to see a steady divergence between the original and new policies, as we need the model to adapt to the new information without changing drastically[28].

4.2.2 Use of PPO metrics in our loss function

When playing with PPO learning and the different options it gave us we were inspired by the works by Aleksandra Faust et al[4] and Meier et al [17] who solved similar problems to ours using evolving reward functions. These papers along with many others claim that using reward functions that are able to adapt and evolve to their environment were able to consistently outperform baseline reinforcement learning techniques. Using this as inspiration we created a loss function that takes in these metrics as feedback for their corresponding rewards. This loss function became the centerpiece of our Stable Observer Network. As a starting point, we looked at the loss function implemented by the current HuggingFace auto-reward generator. This loss function pairs with the dual-output strategy discussed in section 2.3.2[9].

$$L = -log(\delta(R_{chosen} - R_{rejected}))$$

Where the R_{chosen} represents a good model output and the $R_{rejected}$ represents a bad model output. This loss function tries to separate the "good" from the "bad" outputs as much as possible. When designing our loss function, however, the main challenge was how to train the SON without putting time into creating a new database separated into "good" and "bad" outputs. When observing our model being trained using reinforcement learning, we saw that some rewards gave back very poor metrics such as high policy and value losses, or made the KL divergence change direction quickly. Using this information we decided to use these metrics in our loss function as a way to separate "good" rewards from the "bad." After many trials and experiments, we finally landed on a loss function that seemed to fit. This function requires users to start with a simple discrete reward that could be used for a reinforcement learning task.

$$L = \alpha(R' - R) + \beta\frac{L_{metrics}}{(R' - R) + \epsilon}$$

$$L_{metrics} = w_1\Delta PL + w_2\Delta VL + w_3\Delta KLD$$

The left side of the function compares the SON output (R') with the discrete reward (R). The right side then looks at how the discrete reward affected the metrics for the PPO function and if there should be some distance between the predicted and discrete reward. The $L_metrics$ constitute the three PPO metrics discussed above; the Policy Loss (PL), Value Loss (VL), and KL Divergence (KLD). These metrics are taken as a rate of change to indicate if the last reward increased or decreased these metrics. If the discrete reward decreased these values it indicates that it was "good" for the model's learning and "bad" if vice versa.

Keep in mind that at this point in time the SON is simply observing and the rewards that it is generating are not being used within the reinforcement learning process yet. In order to teach SON how to differentiate between the "good" and "bad" rewards we need to divide $L_{metrics}$ by the difference between the predicted and discrete reward (R' and R). This creates separation when the SON outputs a reward similar to the discrete reward but the metrics are high, meaning the discrete reward was not optimal for model convergence. Through this learning process, the SON learns what rewards allow the model to learn smoothly and what rewards are detrimental to a smooth convergence. Although, like the KL Divergence for PPO learning, we always keep the SON outputs close to the original discrete reward, ensuring the SON does not "cheat" and only gives rewards that decrease the metrics without thinking about what the outputs are trying to do. The overall implementation of the SON can be seen in Figure 4.1.

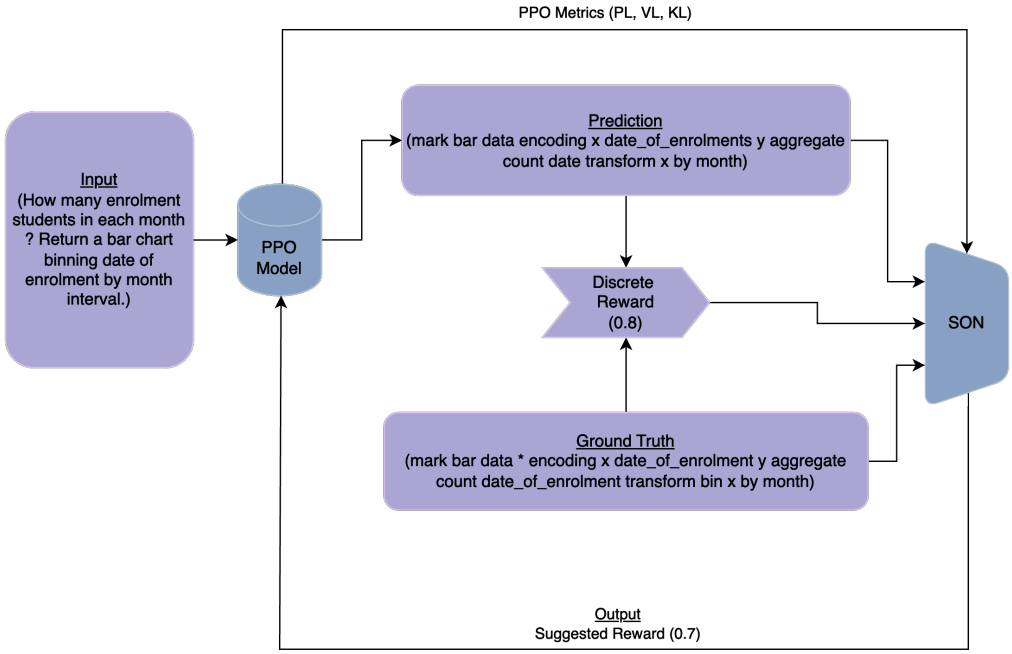

Figure 4.1. An overview of our Stable Observer Network (SON). The SON takes in the prediction, ground truth, and discrete reward, to generate a suggested reward, it then uses the PPO Metrics from that suggested reward to learn how it affected the model and adjusts its feedback accordingly.

4.3 Hyperparameter tuning

The two most important hyperparameters that were introduced are α and β, these metrics act as a guide for how important it is for the SON to follow the discrete rewards vs how much weight to give the PPO metrics. If we see SON outputting only what the discrete reward says we can nudge α lower and β higher, while if our network starts outputting rewards too far out of the original range we can tune it to follow the discrete reward more closely. These two parameters ended up being very model-dependent and would have to get adjusted for each network we ended up using, but in general, it was beneficial to give α about three times more importance than β. This not only ensured that SON did not stray too far out of context but allowed the adjustments made by β to act more like nudges than complete changes. We needed to add ϵ to account for divide by 0 errors or extremely small number errors, this allows the training to run more consistently and accurately. Finally, we have the three w_i weights attached to the PPO metrics. The first two weights (w_1 and w_2) are attached to the policy and value losses respectively; each of these weights is set to negative values to encourage a deduction for each of the two losses. The KL divergence weight (w_3) was positive as our goal is to have a divergence that is consistent and not grow out of control, the direction was not important. We did find, however, that w_3 needed to be set as a much lower value than the first two weights, as the Policy and Value losses each were consistently decimals while the KL Divergence could be much larger.

4.4 Case Study 1

4.4.1 Hyperparameters discover

For our first full experiment, we set α to 1.0 and β to 0.5. This was to ensure SON followed the discrete reward enough that the results would be comparable, but allowing the rewards to be pulled by the metrics. Keeping α a higher value was crucial as SON was being trained from scratch from only the first 200 instances, so a clear direction was necessary. ϵ was set to 0.001 while w_1 and w_2 (the weights for policy and value loss) were set to -1.0 to encourage a reduction in the losses while w_3 (the KL Divergence) was set to 0.1 to encourage a slight increase in its value.

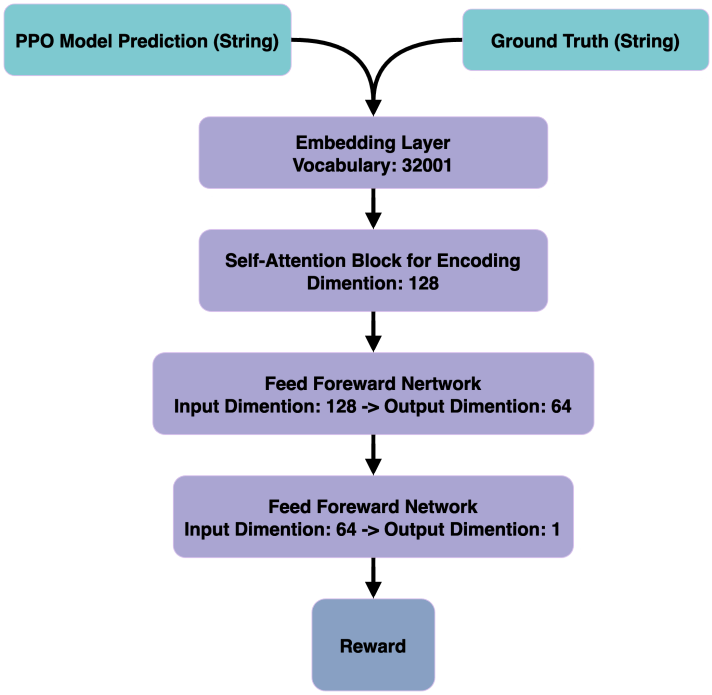

Figure 4.2. This flowchart shows the general outline of our first Stable Observer Network. We kept it simple at first to see if a visualization model would respond well to auto-generated rewards at all. After seeing success with this model we were able to expand the power of our observer network, as can be seen by the charts later on in this paper. For this first go we simply concatenated the incoming strings and then embedded that concatenation. After the embedding, we had three layers with the first being an encoding layer using a self-attention block followed by two feed-forward network layers.

4.4.2 Stable Observer Network architecture

Our SON architecture started out as a feed-forward network with self-attention and encoding layers. SON took in a string made up of concatenating Llama's prediction and its corresponding ground truth, which would then be embedded using the vocab size of 32001, which is the standard for PyTorch encoders. We used a simple self-attention layer as our encoding function as was suggested by Lin et al in

their seminal paper "A Structured Self-Attentive Sentence Embedding"[11], which we found worked best using a dimension of 128. We then passed this matrix through two feed-forward layers, squishing the dimensions of 128 to 64 and then to 1 for the output. This initial network was made simple on purpose as it was to serve as a feasibility test before trying more complex and time-consuming models. These parameters are outlined in Table 4.2 and a visual outline of the structure of the network can be seen in Figure 4.2.

Parameter	Value
Input	Concatenated String
Vocab Size	32,001
Embedding Dimension	128
Layer 1 Output Dimension	64
Layer 2 Output Dimension	1

Table 4.1. A quick overview of the parameters used in our initial Stable Observer Network, it essentially was a three-layer network with the first layer being an encoding layer using self-attention.

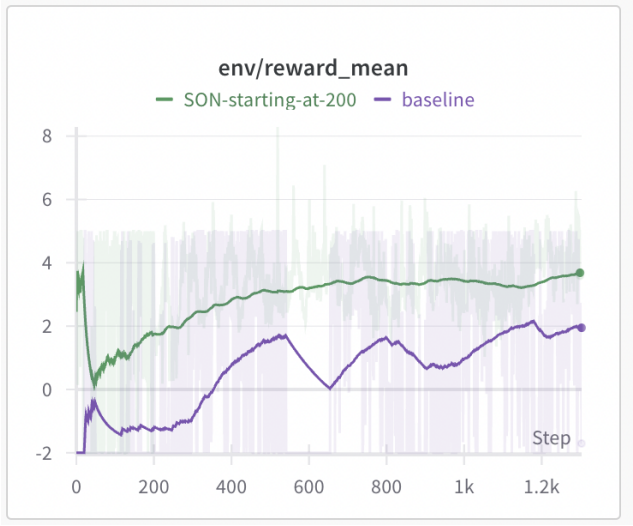

Figure 4.3. Case study 1: Initial Results. A graph comparing the rewards over time when using an observer model for learning the rewards vs a normal discrete reward. Although the model was able to improve the discrete reward overall it fell into many traps where it could not get appropriate feedback on how to get back on track for many iterations. Instead, the observer model was able to provide more nuanced feedback to the model to avoid these "plunges" while learning.

4.4.3 Initial Results

In our first full experiment of implementing a Stable Observer Network, we ran 1000 instances of PPO learning on a Llama model that had been fine-tuned to our

visualization task. We wanted to improve on the fine-tuning by using RL but we found that more often than not using traditional RL on a sequence to visualization task often leads to the model getting stuck and not converging properly. In this experiment, we hoped to prove that using a SON to auto-generate rewards would lead to a smoother and faster convergence. First, we ran the PPO learning with a simple discrete reward for 200 iterations, allowing the SON to observe how the model learns from these rewards. After those 200 instances, we stopped the SON training and implemented it in place of our discrete rewards. The effect was immediate, with our SON giving more nuanced rewards than a simple -1, 0, +1, the model was able to avoid the classic spirals that it would hit without the help of an observer. The reward distribution can be seen in Figure 4.3, where the SON reward output is in green and the discrete reward output is in purple. Although the discrete reward remains on an upward trend on average, it is easy to see the areas of confusion where what the model outputs returns a -2 reward no matter what. This harsh punishment of exploration is unhealthy for a model's learning process and does not give direct feedback for how to improve, only that the space being explored is not good.

The benefit of the kinder, more nuanced feedback given by the SON can be seen in plotting the PPO metrics that were used in the loss function. We ran two experiments right after one another, the first being a baseline by running a traditional reinforcement learning task, using our discrete reward discussed above. The second run was the same for the first 200 iterations, then SON took over after that and our model took the auto-generated rewards instead of the traditional discrete reward. Figure 4.4 shows the KL coefficient over time between these two runs. for each of the next couple of figures, the traditional reinforcement learning run is shown in purple and the experimental run using our SON is in green. As you can see in Figure 4.3 the dangers of running a traditional reinforcement learning task on a visualization recommendation system. Our baseline model gets stuck in loops of trying to explore using a KL coefficient and ends up not fully understanding the space that it's in. When it gets stuck in a rut like this it forces the model to explore and change in drastic ways and ends up converging at a point far away from where we started. When using our auto-generated rewards, however, it is easy to see how the model is now able to avoid these ruts and traps that it was so easily drawn to before. It is able to get better feedback on its responses and the KL coefficient declines at a nice steady pace. Similar results can be seen in the next two figures, 4.4 and 4.5, where the impact of implementing the Stable Observer Network at iteration 200 is immediate. Figure 4.6 shows the policy loss for both models' runs, in this area both techniques keep a similar loss value, although the baseline seems to start to veer downward at the end while our SON remains stable. Figure 4.5 shows a much clearer difference, not only between the SON and baseline but between before and after our SON took over. Once the SON starts to give its auto-generated rewards as feedback on iteration 200, the value loss consistently drops and converges earlier than the baseline. These three graphs showed us that using a network like the one we built to predict rewards is not only feasible but could provide a huge benefit for generating visualizations.

Figure 4.4. Case study 1: Initial Results. A comparison of the KL coefficient over time while learning from a discrete reward (purple) vs the learned reward from the observer model (green). The observer model takes over at iteration 200 and has an almost immediate impact. While the PPO model has a hard time keeping track of the discrete rewards and keeps exploring further and further outputs, the observer model helps to keep a steady convergence

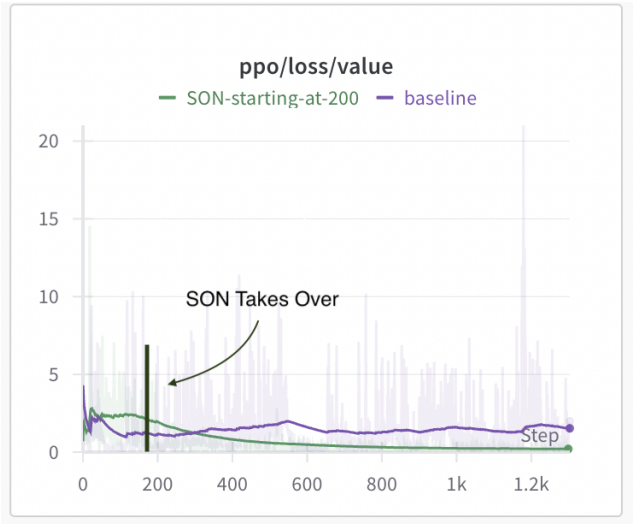

Figure 4.5. Case study 1: Initial Results. This graph shows the value loss of running a PPO fine-tuning using traditional reinforcement learning (purple) and our stable observer network (green). Here we can see the immediate effect of switching the PPO learner to using the rewards auto-generated by our Stable Observer Network. When the SON takes the wheel it is able to easily converge the value loss into a stable region, while our baseline has a consistently hard time keeping this balance. This means that when getting auto-generated rewards the PPO model is able to more accurately predict the expected returns from a given state, allowing for a more stable convergence.

Figure 4.6. Case study 1: Initial Results. This graph shows the policy loss of running a PPO fine-tuning using traditional reinforcement learning (purple) and our stable observer network (green). Contrary to the KL Divergence and the Value Loss graphs, our policy loss comparison shows very little difference between the baseline and our model, although the model using our model seems to be able to stay stable at the very end while the baseline started to dip.

4.4.4 Limitations for Case Study 1

Our first case study was a success and led us to continue perfecting a Stable Observer Network, which we will go into detail on in the next section. However, this first case study had its limitations that we attempt to address in the next experiments. The first thing that comes to mind when looking at these results is that there is no validation set, which would be the entire goal of improving the fine-tuning of a model. For these experiments, we were using a Llamma model fine-tuned on 2000 instances of our dataset, making it too powerful and over-fit to our task, so any changes made through reinforcement learning made little difference when testing on validation sets. The second is that reinforcement learning is extremely finicky and changing small things such as the reward or even the order that the model is fed data can change our "baseline" dramatically. In the next section, we address these issues and perfect our SON.

4.5 Case Study 2

After receiving the results from our first test run we decided to dig deeper into testing with the goal of proving that a PPO model fine-tuned with our auto-generator would output better visualizations than one fine-tuned with traditional reinforcement learning methods. Through this section we will go through the issues that needed to be solved, our solutions to these problems, and what worked versus what did not.

4.5.1 Weaker Base-Model and Harder Validation Set

During our first case study, we were running the PPO reinforcement learning on a Llamma model that had already been fine-tuned to the NVBench dataset with about 2000 instances using supervised learning. This was an issue as the model had become far too overfit on our data, and it became impossible to see what improvements the reinforcement learning was making. To isolate the results and put a spotlight on exactly what the PPO learning process was doing we decided to use a Llamma model fine-tuned on only 100 instances. We also decided to take it a step further and use a column of our dataset that labeled each instance as 'easy', 'medium', or 'hard'. We took only the easy and medium instances for both the supervised learning iterations and the reinforcement learning iterations. That meant for our validation set we would test our models on only the 'hard' instances within our dataset, this gave us much more room to see how the models were learning the space of visualizations, and avoided any danger of over-fitting.

4.5.2 PPO fine-tuning and rewards

The first issue we saw with changing our base Llama model to be far weaker than our first experiment was with the rewards. When giving a discrete reward a common trend is to have 'steps' where there are cut-and-dry discrete rewards that are given to the model. For example, in SEQ2SQL they had 3 different rewards that could be given: -2 if the SQL could not compile, -1 if the SQL could compile but did not produce the correct visualization, and 1 if it compiled and was correct. Our reward for the first case study followed a similar pattern by having -2 if the Llama output could not compile and 1 if it could compile. We then gave bonus points for getting the type of chart, x, and y-axis correct (+1 for each of the three). However, when applying the weaker model to this reward system it was not able to output answers that could be compiled, as PPO learning encourages models to explore the output space. Since the outputs we were getting were not compiling 90% of the time, the rewards the PPO model was receiving were all -2, giving no good feedback to either the baseline reinforcement learning model or the Stable Observer Network that was trying to learn which rewards were better or worse to give the model. To solve this we used a similar technique to Tianbao Xie et all[29] where they used a Jaccard similarly to compare the two outputted strings and return that as a reward. We still wanted to encourage the model to output vega zero that was able to compile more than just being close to the ground truth. So our reward ended up being a combination of the two. Our reward given was between 0 and 1, with 0.5 being given if the output could compile and the other 0.5 being a Jaccard similarity between the output and the ground truth. This new reward had the added benefit of being closer to what a regression model would output, evening the playing field between our SON and the baseline model. A visual for how different outputs will be rewarded can be seen in Figure 4.7.

4.5.3 Stable Observer Network 2.0

After making the changes to our baseline, we noticed the SON was now out of its depth for what it could learn. A time for change was needed and we turned to

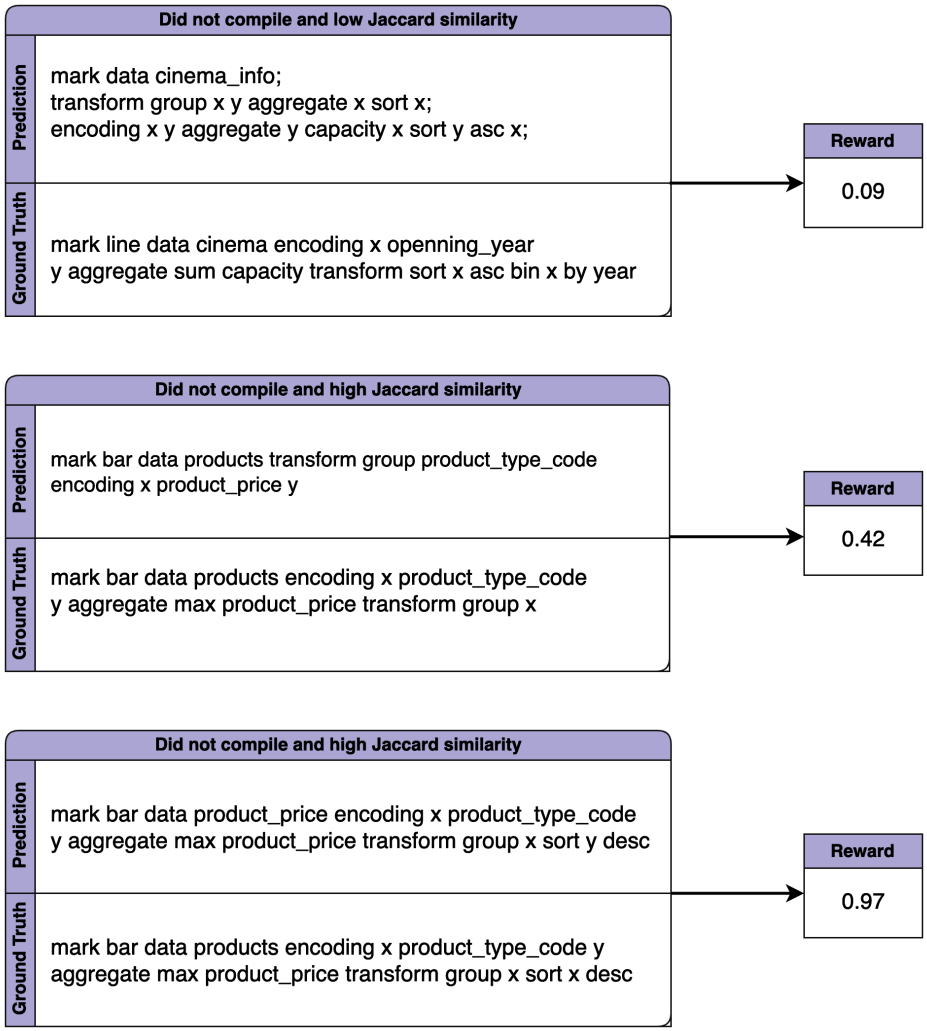

Figure 4.7. This figure shows our discrete reward function in three examples. In each box, you can see the PPO model's prediction labeled as Prediction, the corresponding ground truth labeled as Ground Truth, and the corresponding reward that our baseline reinforcement learning process gives. In the first example, the PPO model tries to explore in a direction that is not close to where we want it to go, it separated the output into three parts with a semicolon, almost like it was a coding language. Here it was not able to compile and was not close to the ground truth so it got an extremely low reward of 0.09 to discourage the model from going in this direction. In the second example, the model outputted something very close to the ground truth, however, it could not be compiled into a visualization. Here we give a reward in the middle since it is on the right path but still needs to give a prediction that will compile, for this specific example the prediction did not compile because our model failed to add anything for the y-axis. Lastly, we have the ideal output, where the prediction is both close to the ground truth and is able to compile into a visualization, here the reward is 0.97 to encourage the model to explore in this area.

the work of Cer et al[2] in their seminal paper "Universal Sentence Encoder" where among many ideas they talk about how it was found to be better to encode two

sentences separately when trying to compare them. In this way, we changed how our network took in the prediction and ground truth strings by embedding and encoding the two separately. We now had two heads that would encode each of the sentences to a dimension of 128, and we then concatenated these two to create a vector of dimension 256. After this we fed it through a slightly larger feed-forward network than before, now equipped with three layers, halving the dimensions down through 32. Here we decided to end with a sigmoid layer to ensure our outputs were squeezed between 0 and 1, with the hope that now the two strategies would be easier to compare with their outputs being within the same range. A graphic overview of SON 2.0 can be found in Figure 4.8, which shows the flow of information from the string inputs to the single integer output, with the Table 4.2 laying out all the hyperparameters stated above.

Parameter	**Value**
Input	Two Separate Strings
Vocab Size	32,001
Embedding Dimension 1	128
Embedding Dimension 2	128
After Concatenation	256
Layer 1 Output Dimension	128
Layer 2 Output Dimension	64
Layer 3 Output Dimension	32
Sidmoid Output Layer	1

Table 4.2. Here is an overview of the parameters used in our second iteration of the Stable Observer Network. This second iteration had about double the parameters of our first SON and stands apart by embedding the inputs separately and adding a sigmoid layer as the output function.

4.5.4 Hope and Despair

The new Stable Observer Network showed significant improvement over the old trials but still was not enough to show significant improvement over baseline. Starting the training from scratch simply did not give our models enough time to pick up on the intricacies necessary to guide the PPO model in an efficient manner. We hypothesize that the first experiment showed hope as the model and discrete reward were outputting relatively different outputs, where now both are squeezed between 0 and 1. Although writing a model from scratch has more romantic appeal as an aspiring researcher, we decided to get some help from the new LLM models that have been launched recently. And so we took the tools that worked best from the two past experiments and decided to customize our own Bert model.

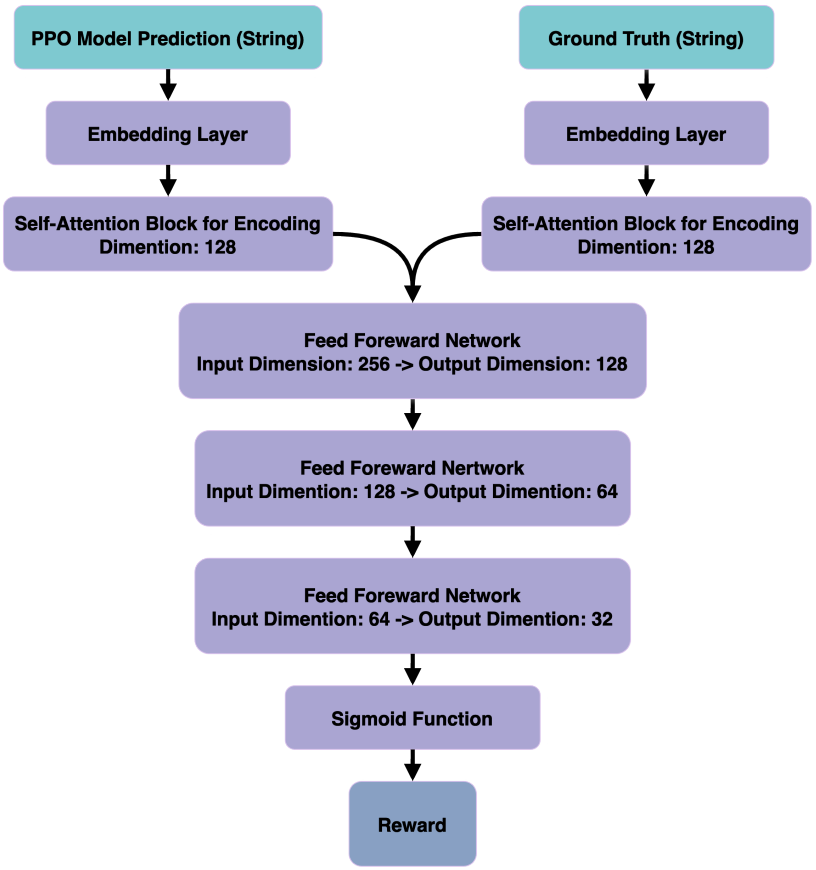

Figure 4.8. A visual representation of the upgrades made to the original Stable Observer Network. This flowchart emphasizes the difference in how our model processed the ground truth and prediction coming from the larger PPO model. Now the pipeline is split into two processes where we encode both strings separately and then concatenate them before the feed-forward layers. We then have the output layer of the model being a sigmoid function to ensure that the predicted reward is between 0 and 1. This allows for a better comparison between the auto-generated rewards and the baseline discrete reward.

4.6 Case Study 3: Stable Observer Network using Tiny-Bert

When experimenting with the weaker model, we found that using an observer model trained from scratch could not learn fast enough to be effective. Even with the new SON modifications we tested that it would need at least 600 iterations before it could produce any good feedback for the model. This is of course not practical at all for someone trying to auto-generate rewards during reinforcement learning, our goal through all of this is to keep the learning phase below 200 iterations, allowing the network to get enough time in to become effective and useful. The obvious answer to the dilemma was to use a model already trained for recognizing text and fine-tune it to our specific needs. We ended up choosing TinyBert which was developed by

Xiaoqi Jiao et al in 2019[8]. TinyBert was chosen from the rest of the plethora of LLM models that have been coming out recently for two reasons; customize-ability and efficiency. The TinyBert API allowed us to customize both its loss function and the final layer of the model which was necessary for us to continue the work that we had been doing. The second is the fact that did not want to over-complicate this process, adding an entire second LLM to this already convoluted fine-tuning process seemed like overkill. TinyBert allowed us to implement a model that was comparable to recognizing patterns in text to the state of the art while being 87% smaller and 7.5 times faster than the regular BERT model. We began to refer to this new model as Stable Observer Network using TinyBert, or SONuTB.

4.6.1 Customizing the TinyBert model

To customize the TinyBert model into our own SONuTB, we ended up taking the best parts of the network we built from scratch and placing them into the Bert model. First, for tokenization, we ended up tokenizing the two incoming sentences together instead of concatenating them first, this ensured the model places a [SEP] token between the two and treats them as two different phrases to then be compared. This borrows from the idea of encoding the sentences differently to ensure our previous SON model took each in as its own identity. Secondly, we were able to add a sigmoid function as the output layer for TinyBert, ensuring that both the discrete reward and our auto-generated rewards resided within the same bounds. This creates an environment where it is easier to compare how each reward system is affecting the PPO model and why one system is working better than the other. Finally, we added our own custom loss function to the TinyBert model. This of course was key as our specific task of predicting the best rewards requires observing how the PPO metrics change for each new reward given to the PPO model. After some testing, we ended up optimizing the loss function to be used along with TinyBert, which we will go over in detail in the next section. A graphic showcasing the new additions and flow of information using our new SONuTB can be seen in Figure 4.9.

4.6.2 Final Loss-Function

While testing on Tiny-Bert we found a few changes to our loss function helped improve the SONuTB's overall view of how the model is learning from the incoming rewards. The first thing we did was experiment with using different metrics, to try and see if making the metrics section less complicated would allow our observer network to gain more knowledge of the remaining metrics. While performing these experiments we found that dropping the kl divergence from our list of metrics allowed the SONuTB to gain a better knowledge of the Policy and Value loss and would be able to consistently keep those at lower values than when we included the KL Divergence. This most likely has to do with how the metrics are all combined into one parameter, and when there are too many metrics being combined the observer has a harder time deciphering which ones are important.

In the end, our final loss function ended up looking like this:

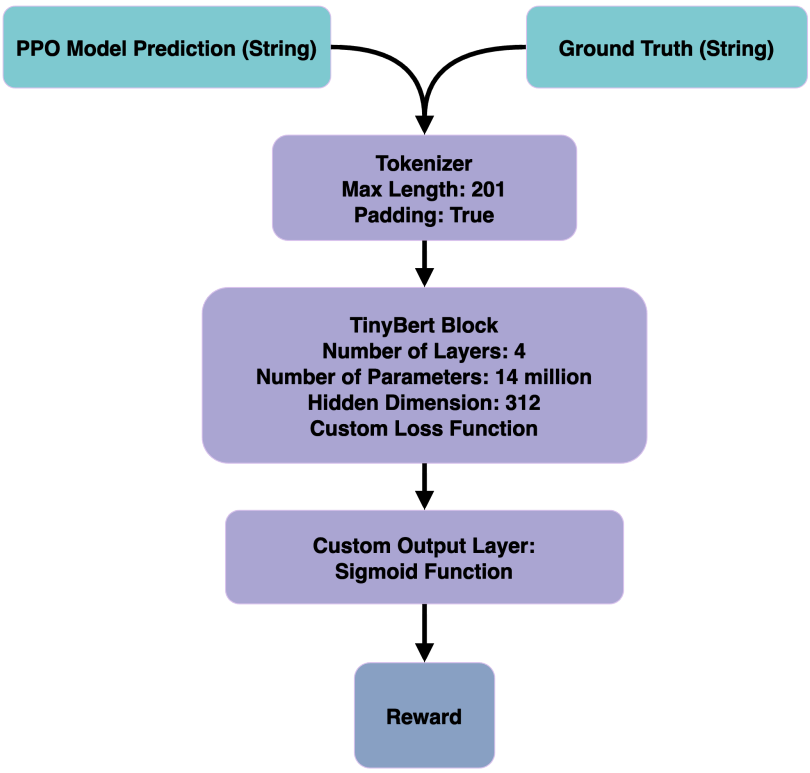

Figure 4.9. Our Stable Observer Newtork using TinyBert has a much simpler layout than the last two SONs since most of the logic is tied up in the TinyBert block. This uses the ideas tried in our second iteration of the SON with TinyBert taking in both strings separately instead of treating them as a single concatenated entity. It uses our custom loss function to be fine-tuned to our specific task and we were able to add the Sigmoid function as the output layer, again ensuring an output range that is the same as our baseline.

$$L = \alpha(R' - R) + \beta \frac{L_{metrics}}{(R' - R) + \epsilon}$$

$$L_{metrics} = w_1 PL + w_2 VL$$

As can be seen in Table 4.3 we set the parameter values to be slightly different than before, most notably we raised α by 0.5 and raised the metrics coefficients to 0.5 from 0.1. These changes allowed for a more balanced learning process, where even though it seems like we raised the importance of keeping the predicted value close to the discrete value, raising the weights on the PPO metrics was able to keep the balance between the two.

4.6.3 Results

After making these changes to our model and loss function we were finally able to see some concrete results. When collecting results for reinforcement learning it's

Hyper-Parameter	Symbol	Value
Alpha	α	1.5
Beta	β	0.5
Epsilon	ϵ	1×10^{-5}
Weight for PL	w_1	0.5
Weight for VL	w_2	0.5

Table 4.3. Hyper-parameter values for the final iteration of our loss function.

important to see how well the algorithm progresses through the learning process, so we had checkpoints starting at 400 iterations and ending at 800 iterations of PPO training. We stopped after 800 iterations because we noticed no improvements for either strategy after this point, it seems the model would over-fit to the training data and perform the same or worse on a validation set. When evaluating each checkpoint on the validation set, the model using our developed SONuTB for its reward system consistently outperformed the baseline of normal reinforcement learning training.

For these tests the easy and medium columns from the NVBench dataset for the training phase and the hard and extra hard columns for the validation phase. This served the dual purpose of being secure that there was no mixing of our training and validation data and also to see how the model performed on more complex data after each reward system was used. These results are shown in three charts; the first is Figure 4.10 which shows the percent of validation instances that the baseline model could compile vs the model trained with our observer network. Here the model trained using auto-generated rewards consistently outperformed the baseline by 5 to 10 percent for each checkpoint. The consistency of the out-performance was extremely exciting to see, and output compiling was the main goal of the reward that we implemented. This means that our reward model was able to outperform the goal that our discrete reward was made to achieve.

The next two figures, 4.11 and 4.12, go into more detail and show the percent correct of the x-axis and y-axis predictions respectively. In 4.11 we were able to outperform baseline as the PPO iterations went on for a longer period of time, with our auto-generated reward model performing 10 percent better when reaching 800 PPO iterations. Most impressively, in Figure 4.12, when using the observer model we performed almost 15 percent better when predicting the correct Y axis, which is a notoriously hard category for the NVBench dataset, as can be seen by the percent correct by our baseline.

Perhaps the most promising takeaway that can be drawn from these three graphs is that when using our observer model, the PPO model is able to consistently learn more when continuing to higher iterations. This can be seen particularly well when looking at figures 4.11 and 4.12, where when trying to predict the correct X axis the baseline stops learning at iteration 600, whereas our auto-generated rewards keep the model learning all the way to the 800th iteration. A more acute visual of this is the Y Correct chart where the baseline has a hard time learning anything past the 400th iteration while our model stays on a consistent upward trend until the 800th.

Getting these results after refining and testing these auto-generated reward models was incredibly satisfying and has made us excited to explore more in this space in the future.

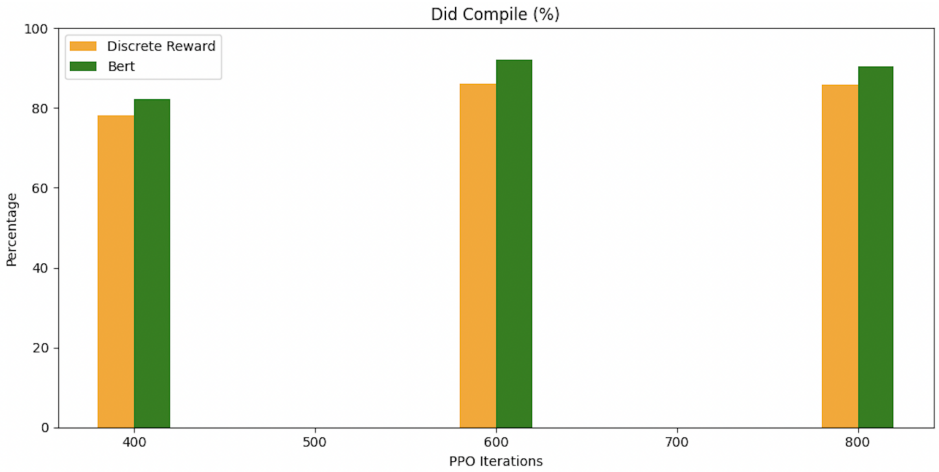

Figure 4.10. This figure shows the results from case study 3 using the custom Tiny-Bert model (SONuTB). The chart compares using PPO learning with a traditional discrete reward (shown in orange as the baseline) to a PPO model using SONuTB (shown in green). Our reward model takes over after iteration 200 and continues for the rest of the time. As can be seen, when tried on a validation set the model using our reward network outperformed the baseline by about 5 to 10 percent.

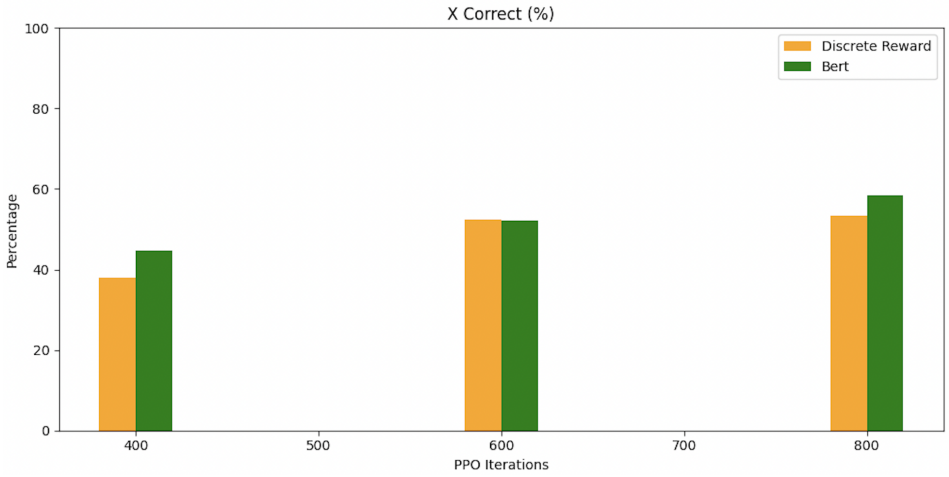

Figure 4.11. A chart showing results from case study 3 using Tiny-Bert for how well the fine-tuned models predicted the x-axis. This graph displays that around iteration 600 the baseline caps out for performance, as its predictions at 800 iterations stay the same as 600, while when the model uses our SONuTB for rewards, it continues to be able to learn through the 600th iteration and allows for a higher upper bound on growth. Here at iteration 800, our PPO model outperforms the baseline by about 7 percent.

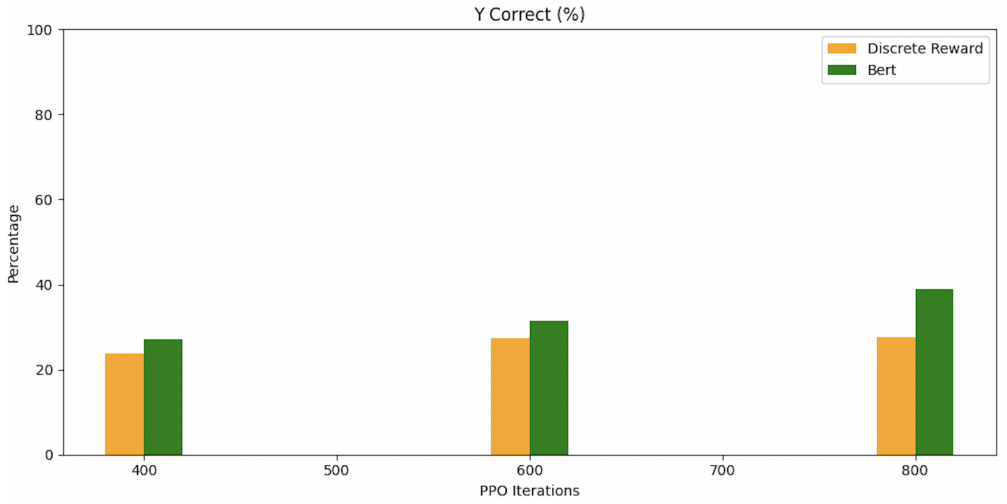

Figure 4.12. A chart showing results from case study 3 using Tiny-Bert for how well the fine-tuned models predicted the y-axis. Predicting the y-axis is a classically hard issue in visualization and the NVBench dataset in general, as can be seen by our baseline results. However here our baseline caps out at around iteration 400, improving only 4 percent from iterations 400 to 800. In contrast, when using the observer model for rewards, we start about five percent higher at iteration 400, then gain about thirteen percent from iteration 400 to 800. This leaves our model outperforming the baseline by 15 percent at the end of training, being the largest gap in our validation experiments.

4.6.4 Why and how does it work?

As shown in the reward distribution graph that is Figure 4.13, the discrete reward (shown in blue) tends to be stronger than the reward generated by our SONuTB (shown in brown). Although the size of the osculations are different, in general they follow a similar pattern for which predictions they give high or low scores. This makes sense as a large part of the loss function requires our observer model to follow the discrete reward, although it does deviate from the reward in some places. The smaller oscillation and the differences in some of the rewards is the key to understanding how our SONuTB outperforms traditional methods of fine-tuning.

When creating a discrete reward, the visualization community usually implements strict rules for when to give positive and negative rewards. For example, the simplest discrete reward that could be used is 0 if the output does not compile or 1 if it does compile. The issue with general overarching rules like this is they leave out a lot of edge cases that can lead the model to sub-optimal performance. For example, there is an edge case that kept popping up where the PPO model would output the visualization query followed by a long list of comments. Technically this output could compile, as the compiler would just ignore the list of comments after the visualization query, leading our discrete reward to give generally high values. This is a classic example of the PPO model finding edge cases and trying to "cheat" when finding the policy that grants the best reward. Obviously, this is an unwanted output, and scanning for edge cases like these in learning phases of 1000 or more iterations is extremely inefficient. However, the observer recognizes this as a bad output by watching the PPO metrics and automatically adjusts for situations like

this, giving much harsher rewards. Another more subtle example is Figure 4.14, where the PPO model added two "mark" tokens at the beginning of its output. Here the output does not compile so the discrete reward gives it a relatively low score, however with the push from the PPO metrics the observer model recognizes the error and punishes the model for this output more harshly than our discrete reward did. These small to large adjustments in the given rewards seem to be what leads the PPO model to perform better on validation tests. There is much work to be done in this area as we move forward, but we believe that these results are promising and logical enough to warrant further time and study, hopefully unlocking RL as a tool for visualization recommendation systems in the future.

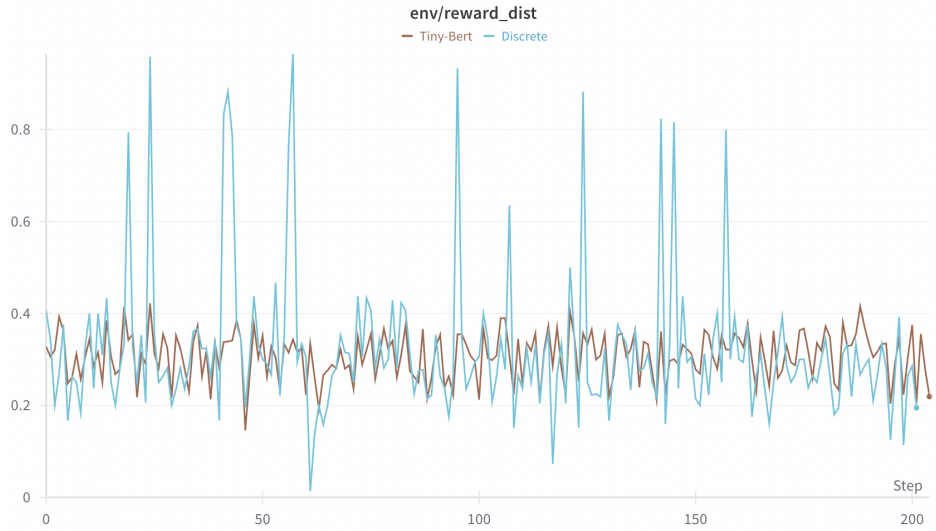

Figure 4.13. A reward distribution graph from case study 3, comparing a discrete reward output to our SONuTB predicted reward. The observer model's predictions can be seen in brown while the discrete reward is seen in light blue. In this study the SONuTB started to predict the rewards for our model after iteration 200, meaning that the first set of bars on iteration 400 used the discrete reward for the first 200 iterations then our SONuTB for the next 200. Testing models that used our observer network for less time showed us that the impact of using a predicted reward is almost immediate, and does not need long trials to see results. Here we are showing how our observer tends to be more reserved than the discrete reward system.

```
Prediction: mark mark bar data master_customer_id encoding x cmi_details transform sort cmi_details desc
Groundtruth: mark bar data customer_master_index encoding x cmi_details y aggregate none master_customer_id transform sort x desc
didnt compile
predicted reward: 0.2598533630371094, discrete reward: 0.35714285714285715
```

Figure 4.14. A single example from case study 3 of what our SONuTB outputs vs a discrete reward. Here "prediction" is our PPO model output, "Groundtruth" is the corresponding ground truth, "predicted reward" is our SONuTB output, and "discrete reward" is the baseline reward output. This example is much more subtle than in Figure **??** as both reward systems output harsh rewards (0.35 from the discrete rewards vs 0.25 from SONuTB), but our model recognized this prediction as worse than the discrete reward did.

Chapter 5

Conclusions and Future Works

5.1 Conclusions

As we demonstrated through this thesis, using an observer to keep track of how the PPO model reacts to different rewards helps lead to smoother and more optimal convergences in this area of visualization. We started with the issue of reinforcement learning being particularly hard to use with visualization models, as oftentimes the model has a hard time picking up on the best policy and can easily get lost during the process. When a model starts to get stuck during reinforcement learning we need to give it better feedback than what it is getting. Before the the only way to do this would be to stop the learning process, tweak the reward, and then start again. This process is long and tiring as oftentimes fine-tuning runs can take hours before completion, we wanted to create a technique that would be able to adapt to challenges on the fly, and not need multiple runs to get it right.

Auto-generating rewards is not a completely novel concept and when creating our reward model we needed to set ourselves apart from the rest of the competition. When looked at, the different methods that have been used in solving this problem have issues that accompany each of them. We focus on two state-of-the-art techniques that most closely match our area of work. The first technique discussed was Reinforcement Learning with Human Feedback, where we use humans to give direct feedback to the model, allowing for reinforcement learning models to get real-time feedback and be able to closely align with human preferences. The drawback here is that without a significant number of users giving feedback to properly train a model, it is impossible to have a robust training system, especially with users being biased and unreliable in general, a statistically large number is needed to drown out any bias that could be felt. Techniques that allow for a small team to succeed are necessary in any field, as larger companies already have a huge advantage over smaller startups. The second was a technique for auto-generating rewards by creating a contrasting dataset of good and bad outputs, teaching the model where to avoid and where to explore. It uses a loss function similar to a triplet loss, where it takes two contrasting instances and tries to set them as far apart as possible. Although this technique gets around the number of users issue, it still requires a large overhead of work to get moving, forcing users to create an entirely new dataset made up of contrasting outputs.

We then discussed our solution to the problem, where our goal was to create a simple plug-and-play reward model with little to no setup required. This came to fruition as we played with the idea of an "observer" model, something that can keep track of how the PPO model is learning and make sure it does not get off track. We accomplished this by developing a loss function that both took into account what the discrete reward thought about a prediction, and how the model reacted and learned from that reward. We found that some rewards were expressive and allowed the model to continue smoothly on its learning process, while others gave almost no positive information and would only confuse the PPO model, oftentimes leading to a sub-optimal result. Through our exploration of model types and loss functions, we learned that the most important metrics to look for when training our observer model are policy and value loss. These two metrics together tell us how well the model understood why it was given a particular reward (policy) and how well it can guess what the next reward will be (value). When looking at the difference in the predicted and baseline rewards, we can see that they follow similar trends. For the most part, SON follows the discrete baseline, with about 1 in 5 rewards being shifted either a little or significantly. It is these small shifts over time that allow our prediction to guide the model to optimal convergences. Using our loss function and getting some help from TinyBert we were able to successfully outperform traditional reinforcement learning techniques by five to ten percent, with a larger gap when predicting harder parameters like the Y axis for the visualization.

The main strength of our model is that it requires almost no setup to use, it simply needs to be plugged into the training loop of a PPO model and allowed to take over after a desired amount of iterations (we found 200 to be the sweet spot). With ease, however, always comes with drawbacks and while testing our model we noticed two main weaknesses in this approach compared to the other two discussed above. One is that we require users to create a discrete reward for our SON to observe and learn from, it is true that SON completely takes over after a certain amount of time, but before then SON has to observe something happening to get an idea for how the PPO model is learning. This means that the results of the training still rely on users to select a good discrete reward, unlike the other two techniques that get their rewards either from users or from comparing good and back outputs. Looking at it from this lens our contribution boosts and optimizes the discrete reward, rather than trying to replace it. The second weakness is that our model needs many iterations before it is effective. The SON needs to observe about 200 iterations to learn how the model is reacting to the discrete rewards, then another 200 iterations before the results of using the new rewards have a significant impact. This is not a problem for most PPO training's as fine-tuning normally takes about 800 to 1000 iterations. However, with the rise of LLMs and few-shot learning, this weakness must be discussed as other techniques kick in from iteration 0, allowing users to have more freedom to choose how long the fine-tuning will last. For most situations however using a model like ours shows great promise, if users have been trying to fine-tune using RL and are not using few-shot learning then implementing a reward generator like SON has few drawbacks. Of course, it is not perfect and there is much more to do, and there are many future works to be done.

5.2 Future Works

For this thesis, we focused on giving visualization recommendation systems a reliable way to use reinforcement learning without the headache of a setup that state-of-the-art techniques require. In this sense, we successfully pointed out a new technique that seems to have promise, although more work needs to be done to prove this theory. The obvious continuation is to provide more detailed and documented explorations of using different PPO metrics in the loss function. So far we have shown that using the loss function we made to predict new rewards makes the model perform better than the baseline, but we have not shown how using different metrics affects the model differently or any strengths and weaknesses of different combinations. An extensive guide on how to take the feedback from the learning process and use it would be a great extension to this thesis. The hope here would be that we can find ways to use different metrics to solve different problems, making specialized functions that can be swapped out depending on how the particular user model is struggling.

Another line of research that immediately comes to mind would be to expand the scope of using an observer model. Here we have discussed visualization models and how they struggle with reinforcement learning techniques, but the sequence-to-visualization field can be expanded out to the extremely broad field of sequence-to-sequence learning. Of course, we would have to take smaller steps before broadening our horizons that far out. When discussing sequence to visualization, what we mean by visualization is outputting a visualization language, implying languages like Vega and also subsets of coding languages like plotly or matplotlib. It would be interesting and groundbreaking if we could prove that using our Stable Observer Network on problems like Sequence to Code or Sequence to SQL would improve results from the baseline. This would help cement our research within the reinforcement learning sphere and greatly improve interest in this topic.

Lastly, we would need to test our trained models on the state-of-the-art LLM models when generating visualizations such as ChatGPT or Google's model Bard. While outperforming them would be the ideal goal, realistically these models have huge advantages over us, both in terms of size and amount of learning data. However, if we could prove that our technique allows a weaker model to perform even close to what a larger LLM can provide in terms of visualizations, it would be monumental in showing how our technique can help smaller models compete with big companies. Strategies such as ours that help smaller models compete with huge ones are key to keeping artificial intelligence research alive in smaller firms. If we can not find creative ways to keep up with larger companies, every academic paper that comes out will eventually be a fine-tuning of one larger model or another.

The course of science is never-ending, for research the future is comprised of almost infinite paths to test and perfect. We hope that through this paper we have demonstrated enough promise that while we work on perfecting this mode, others will take up the torch as well and we can help grow the field of visualizations together.

5.3 Final Thoughts

"Children have a remarkable capacity to guide us back to the essential" is a quote by the Irish poet David Whyte reminding us that no amount of experience can outweigh a pair of fresh eyes to help guide us along. When looking for new ways to fine-tune or teach a model how to learn oftentimes the hardest part is keeping the balance between forcing the model to stay on course, but also to keep it from over-fitting to the situation. When exploring a new area models will try any avenue possible to achieve the goal set out for them in the loss function, this can lead to optimal results but oftentimes can lead to weird edge cases or sub-optimal gradients as the models get lost. When discussing reinforcement learning the models can almost literally get lost while exploring to maximize rewards, particularly in the area of visualization where the correct answer is more complicated than a correct or incorrect output. When working with machine learning models it is hard to not personify them, as can be seen through the naming conventions all the way back to the beginning, when we called the process of matrix multiplication a "neural network"[16]. As a person begins to explore an unfamiliar area it is easy to get lost or walk in circles without something guiding us, whether that be a map for exploring a new area of land, or a child when thinking about our own directions in life. It was this line of thinking that led us to our Stable Observer Network, or SON, where we saw that our visualization model seemed lost and in need of a guide.

Bibliography

[1] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners, 2020.

[2] D. Cer, Y. Yang, S. yi Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, Y.-H. Sung, B. Strope, and R. Kurzweil. Universal sentence encoder, 2018.

[3] H. Dong, W. Xiong, B. Pang, H. Wang, H. Zhao, Y. Zhou, N. Jiang, D. Sahoo, C. Xiong, and T. Zhang. Rlhf workflow: From reward modeling to online rlhf. *arXiv preprint arXiv:2405.07863*, 2024.

[4] A. Faust, A. Francis, and D. Mehta. Evolving rewards to automate reinforcement learning. In *6th ICML Workshop on Automated Machine Learning*, 2019.

[5] J. Harrell and V. Brown. The world's oldest surviving geological map: The 1150 b.c. turin papyrus from egypt. *Journal of Geology - J GEOL*, 100:3–18, 01 1992.

[6] J. Heer and M. Agrawala. Software design patterns for information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):853–860, 2006.

[7] C. C.-Y. Hsu, C. Mendler-Dünner, and M. Hardt. Revisiting design choices in proximal policy optimization, 2020.

[8] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu. Tinybert: Distilling bert for natural language understanding, 2020.

[9] A. Khandelwal. Reward model training. *Medium*, 2023.

[10] G. Li, P. He, X. Wang, R. Li, C. H. Liu, C. Ou, D. He, and G. Wang. Insightable: Insight-driven hierarchical table visualization with reinforcement learning. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–18, 2024.

[11] Z. Lin, M. Feng, C. N. dos Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio. A structured self-attentive sentence embedding, 2017.

[12] Y. Luo, J. Tang, and G. Li. nvbench: A large-scale synthesized dataset for cross-domain natural language to visualization task, 2021.

[13] Y. Luo, N. Tang, G. Li, J. Tang, C. Chai, and X. Qin. Natural language to visualization by neural machine translation. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):217–226, 2022.

[14] J. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Trans. Graph.*, 5(2):110–141, apr 1986.

[15] P. Maddigan and T. Susnjak. Chat2vis: Generating data visualizations via natural language using chatgpt, codex and gpt-3 large language models. *IEEE Access*, 11:45181–45193, 2023.

[16] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[17] R. Meier and A. Mujika. Open-ended reinforcement learning with neural reward functions. *Advances in Neural Information Processing Systems*, 35:2465–2479, 2022.

[18] A. Narechania, A. Srinivasan, and J. Stasko. Nl4dv: A toolkit for generating analytic specifications for data visualization from natural language queries. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):369–379, 2021.

[19] S. F. Roth and J. Mattis. Data characterization for intelligent graphics presentation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 193–200, 1990.

[20] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, 2017.

[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[22] A. Srinivasan and V. Setlur. Bolt: A natural language interface for dashboard authoring. In *EuroVis (Short Papers)*, pages 7–11, 2023.

[23] C. Stolte, D. Tang, and P. Hanrahan. Polaris: a system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, 2002.

[24] T. Tang, R. Li, X. Wu, S. Liu, J. Knittel, S. Koch, T. Ertl, L. Yu, P. Ren, and Y. Wu. Plotthread: Creating expressive storyline visualizations using reinforcement learning. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):294–303, 2021.

[25] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. Llama: Open and efficient foundation language models, 2023.

[26] F. B. Viégas, M. Wattenberg, and K. Dave. Studying cooperation and conflict between authors with history flow visualizations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, page 575–582, New York, NY, USA, 2004. Association for Computing Machinery.

[27] S. Wang and C. Crespo-Quinones. Natural language models for data visualization utilizing nvbench dataset, 2023.

[28] Y. Wang, H. He, and X. Tan. Truly proximal policy optimization. In *Uncertainty in Artificial Intelligence*, pages 113–122. PMLR, 2020.

[29] T. Xie, S. Zhao, C. H. Wu, Y. Liu, Q. Luo, V. Zhong, Y. Yang, and T. Yu. Text2reward: Reward shaping with language models for reinforcement learning, 2024.

[30] S. Zhang, Z. Tang, Z. Chen, G. Li, J. Zhou, J. Liu, B. Dong, W. Dai, J. Han, H. Zhan, J. Li, Y. Su, C. Tang, C. Li, and J. X. Yu. Nvbench: A large-scale dataset for neural visual analysis of database queries. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, page 1792–1805, New York, NY, USA, 2021. Association for Computing Machinery.

[31] R. Zheng, S. Dou, S. Gao, Y. Hua, W. Shen, B. Wang, Y. Liu, S. Jin, Q. Liu, Y. Zhou, et al. Secrets of rlhf in large language models part i: Ppo. *arXiv preprint arXiv:2307.04964*, 2023.

[32] V. Zhong, C. Xiong, and R. Socher. Seq2SQL: Generating structured queries from natural language using reinforcement learning. 2018.