university of groningen / faculty of science and engineering / artificial intelligence

Low-Latency Language-Action Foundation Models via Upside-Down RL

Bachelor's Project Thesis

Lukasz Hubert Sawala, s5173019, l.h.sawala@student.rug.nl,
Supervisors: J.D. Cardenas Cartagena & M. Sabatelli

Abstract: This paper explores the Upside-Down Reinforcement Learning (UDRL) algorithm, an offline RL paradigm, introducing novel transformer-based architectures to create a scalable and controllable framework for efficient low-resource command-conditioned behavior in complex state-action spaces. Two architectures are proposed: UDRLt and UDRLt-MLP, both leveraging lightweight transformers for efficient control in continuous action spaces. Results show that UDRLt-MLP significantly outperforms the Decision Transformer baseline and achieves higher alignment with desired outcomes, even under out-of-distribution commands, while requiring only a fraction of computational resources. In more challenging transfer settings like AntMaze, fine-tuning and iterative self-improvement via rollout-based imitation partially recover performance, though limitations in dataset quality persist. A self-imitation algorithm is proposed to mitigate data scarcity issues. The findings highlight UDRL's potential for efficient and generalizable control, identifying issues and future research directions.

1 Introduction

Reinforcement Learning (**RL**) is a powerful framework for environment-based interaction, offering strong mathematical foundations for solving problems framed as Markov Decision Processes (MDPs). Despite its successes, RL often suffers from significant limitations, including poor generalization, sample inefficiency, training instability, and difficulty scaling to large action-state spaces (Henderson et al., 2019; Y. Li, 2022; Liu et al., 2024). A central challenge of RL is that this framework relies on emergent behavior, where agents learn to match reward functions indirectly. This dynamic leads to a notorious issue of *reward hacking*, making it difficult for developers to align learned behaviors with intended outcomes (Sutton & Barto, 2015).

Upside-Down Reinforcement Learning (**UDRL**) (Schmidhuber, 2019) is an offline RL paradigm that promises to address several of these issues by explicitly conditioning action prediction on control commands such as desired return and horizon. However, no research has yet attempted to scale UDRL to complex environments with highly challenging learning dynamics.

The motivations behind UDRL are biologically inspired: in natural intelligence, learning is largely driven by unsupervised or self-supervised processes, with reward maximization acting as a high-level signal rather than the core learning driver (Botvinick et al., 2020). UDRL embraces this view by learning diverse behaviors in a way that does not heavily depend on externally defined reward functions.

Transformers (Vaswani et al., 2017), originally developed to solve long-standing problems in natural language processing, have shown strong transferability and scalability capabilities in a broad range of supervised learning tasks (Dosovitskiy et al., 2021; Kamatala et al., 2025; Zhai et al., 2022). Their architecture, centered on attention mechanisms, excels at modeling contextual relationships. This makes transformers naturally well-suited to UDRL, where action selection must flexibly adapt to various conditioning inputs (Brohan et al., 2023). Nevertheless, their performance typically depends on large and diverse datasets.

The most prominent application of transformers in command-driven decision making is the Decision Transformer (**DT**) (Chen et al., 2021), which reframes policy learning as a supervised sequence modeling task. Instead of explicitly modeling MDPs, DT predicts the next action conditioned on a long history of past states, actions, and rewards-to-go. DT has demonstrated competitive

performance, matching or exceeding state-of-the-art offline RL methods on benchmarks such as Atari and OpenAI Gym (Chen et al., 2021). However, its dependence on dense reward trajectories and full return-to-go sequences makes it less effective in sparse reward or pure imitation learning (**IL**) scenarios. In addition, DT acts as a *passive imitator*, unable to incorporate flexible user-defined commands beyond the initial return signal.

In contrast, UDRL-based approaches train agents with various control signals, such as desired return and horizon. This formulation allows for learning behavior that is more directly aligned with human intent, while still allowing for some sort of exploration by generating new training data from the agent's own past experiences, addressing some of DT's core limitations. On top of that, UDRL enables a complex control mechanism while still ensuring its straightforwardness by supporting only a few rigorously defined commands.

Further support for transformer-based imitation comes from recent successes in vision-language models. For example, π_0 (Black et al., 2024) models dense action flows across multi-task robotic manipulation settings, demonstrating precise and generalizable behavior. Similarly, PerAct (Shridhar et al., 2022) achieves robust manipulation directly from raw point cloud data using transformer backbones. These works illustrate that transformers are capable of generalizing across diverse action spaces and goal structures-an essential property for frameworks like UDRL, which require robust, goal-conditioned decision-making.

Despite the complementary strengths, UDRL and transformer-based architectures have yet to been meaningfully combined. While Decision Transformers bring supervised learning to decision-making problems, they fall short of true command-driven control. No existing work has yet explored whether transformers can be trained under a fully UDRL-based framework, where control commands–not rewards–drive learning and behavior.

At deployment, specifying arbitrary return and horizon commands within the UDRL framework allows behavior to be flexibly aligned with the user's objectives (Cardenas-Cartagena et al., 2024), effectively sidestepping the uncontrollability of standard RL and the complexity of alternative control frameworks. This paper explores this potential across two environments, particularly in high-dimensional continuous control tasks - one fully deterministic and one stochastic. Agents are compared along three key axes: performance, controllability, and efficiency. While the original UDRL formulation is probabilistic, generating a distribution over possible actions, this work simplifies the architecture into a point-wise predictive model. This reduces training and inference complexity while preserving the benefits of command-conditioned behavior. The proposed UDRL Transformer supports both expert imitation and a lightweight self-improvement loop, making it broadly applicable and easy to deploy. Success in this setting could suggest that the UDRL-Transformer framework offers a superior solution for large-scale decision-making problems and pave the way for future research, potentially unlocking a new paradigm in control learning and advancing progress toward scalable, aligned AI systems.

The key contributions of this research are hence based on bringing scale to the UDRL algorithm, which can be concretized as follows:

- We show that a Transformer used as a perception component and trained under the proposed UDRL framework achieves superior controllability and performance compared to the Decision Transformer, while being significantly more efficient in terms of data handling and computation.

- We provide evidence that UDRL Transformers generalize more effectively to unseen goal-conditioned tasks-such as the AntMaze benchmark-than competing models. However, we also identify limitations: the method underperforms in stochastic environments unless given access to either exploration/self-imitation mechanisms or a large volume of high-quality data.

2 Theoretical Framework

2.1 Reinforcement Learning

The tasks that the agents are compared on are defined as RL problems in continuous action spaces, framed as finite-horizon non-discounted MDPs, defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ is the

state space, $\mathcal{A}$ is the action space, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition probability function, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1]$ is the discount factor.

The agent's objective is to learn an optimal policy over actions given states ($\pi^* : \mathcal{S} \times \mathcal{A} \to [0, 1]$) that maximizes the expected cumulative reward, defined as:

$$\pi^* \in \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right],$$

While widely applicable, RL's formulation lacks a native mechanism for specifying desired outcomes at inference time. UDRL addresses these problems by reframing learning as supervised behavior cloning from past experiences **conditioned on those desired outcomes**. This redefinition allows users to directly influence agent behavior, making trained policies adaptable and hence more applicable across real-world scenarios.

2.2 Upside-Down Reinforcement Learning

UDRL does not learn to map states to actions based on a reward mechanism. Instead, each decision is conditioned on a command specifying the desired return-to-go and remaining horizon to be obtained by the agent, alongside any extra information.

Concretely, given a finite-horizon MDP, each time step t in an environment is instead defined by the tuple $(\mathbf{s}_t, D_{r_t}, D_{h_t}, E_t)$, where:

- $\mathbf{s}_t \in \mathcal{S}$ is the environment state at time t,
- D_{r_t} is the desired reward-to-go at time t,
- D_{h_t} is the desired horizon during which the desired reward D_{r_t} is to be obtained,
- E_t is an *optional* argument containing any extra information needed to capture the full context of the decision.

This tuple is used to learn a behavior function $B_{\pi_\theta} : (\mathbf{s}, D_r, D_h, E) \to \mathbf{a}$, where π_θ denotes the underlying command-conditioned policy parameterized by θ (Srivastava et al., 2021). This behavior function is modeled as a Neural Network $\mathcal{N}_\theta$ and trained to predict the action $\mathbf{a}_t$ given the current state and control inputs. During training, the target label $\mathbf{a}_t^*$ stems from **previous experiences** and corresponds to the action actually taken in $\mathbf{s}_t$ in the environment under the same control conditions (computed retroactively as described later in the section).

Formally, the task of UDRL can be defined as:

$$\theta^* \in \arg\min_{\theta} \sum_{t} L\left(\mathcal{N}_\theta(\mathbf{s}_t, D_{r_t}, D_{h_t}, E_t),\ \mathbf{a}_t^*\right), \quad (1)$$

where L is an arbitrary loss function evaluated over all timesteps t in the dataset. In this research, the loss used is Mean Squared Error (MSE), which here can be defined as:

$$L_{\text{MSE}} = \frac{1}{t} \sum_{i=1}^{t} \left\| \mathcal{N}_\theta(\mathbf{s}_i, D_{r_i}, D_{h_i}, E_i) - \mathbf{a}_i^* \right\|^2,$$

This formulation defines the problem as a regression task over goal-conditioned actions, where the reward is treated as part of the input rather than as an external feedback mechanism.

Similarly to offline RL, the training tuples $(s_t, D_{r_t}, D_{h_t}, E_t, a_t^*)$ are constructed from **full trajectories collected during an interaction** with the environment. Since the entire return and horizon are accessible, the control signals (D_{r_t}, D_{h_t}) can be computed retroactively. Specifically, for a given timestep t within a trajectory, the desired return is defined as the undiscounted return-to-go:

$$D_{r_t} = R_t = \sum_{k=t}^{T} r_k, \quad (2)$$

where T denotes the terminal timestep of the episode. Correspondingly, the desired horizon is computed as the remaining number of steps in the trajectory:

$$D_{h_t} = T - t \quad (3)$$

This formulation provides a simple and scalable way to convert standard trajectories into command-conditioned training samples.

The additional context provided via the E_t input can enhance expressiveness by enabling users

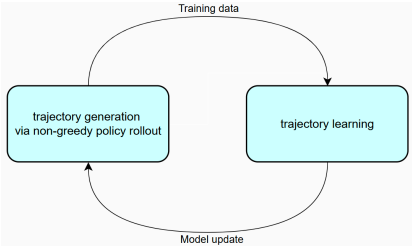

Figure 2.1: Self-imitation algorithm for UDRL. The agent alternates between exploration and supervised training based on its own past trajectories. Adapted from Schmidhuber, 2019.

to specify predefined information such as execution constraints. However, in simpler tasks like locomotion, control can often be specified by relying only on D_r and D_h, simplifying both data labeling and model training, therefore making the inclusion of E_t an optional but impactful design choice. In this research, both approaches (with and without E_t) will be explored.

Two main approaches exist for collecting the trajectories used for training:

- **Expert Imitation**: Trajectories are generated by an expert agent operating in the environment. The model is then optimized to minimize the discrepancy between its predicted actions and the expert actions.
- **Self-Imitation**: Inspired by policy iteration in RL, this method bootstraps training data from the agent's own experience. As illustrated in Figure 2.1, this process iterates: the newly trained policy is used to gather fresh data, which in turn is used to refine the model, supporting online learning.

While UDRL offers a promising alternative to standard RL, it is yet to be applied to complex, long-horizon tasks. Transformers address this by enabling large-scale adaptive conditional learning over sequences, making them well-suited for the controllable, command-driven behavior UDRL requires.

2.3 Transformers

A transformer is a neural network architecture introduced by Vaswani et al. (2017), consisting of multiple attention layers connected with MLPs, widely recognized for its ability to model complex dependencies within sequential data. Its core innovation lies in the use of multi-head self-attention blocks combined with positional encodings, enabling the model to capture contextual relationships between tokens regardless of their distance within the input.

An attention block uses three learnable weight matrices — Key(K), Query(Q), and Value(V) — to flexibly adjust the input embeddings. Attention scores are computed based on the similarity between Queries and Keys, and then used to weight the corresponding Values, following the formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V,$$

where d_k represents the first dimension of the Key matrix.

This mechanism allows each token to dynamically "attend" to other tokens in the sequence, increasing modeling capabilities of structural and conditional relationships, highly desired in UDRL.

A brief description of a standard encoder-decoder transformer architecture is given in Appendix E.

One of the most widely-used and highly-researched Transformer-based approaches is the Decision Transformer (Chen et al., 2021), which we discuss in the following subsection.

2.4 Decision Transformer

Decision Transformer (**DT**) (Chen et al., 2021) has been one of the most influential architectures using transformers to solve decision-making tasks. Similarly to UDRL, it reframes the RL problem as a form of supervised learning; however, it conditions behavior on a subset of desired outcomes, utilizing **only the D_r signal**. On top of that, the DT architecture changes policy learning into a sequence modeling problem, utilizing the enormous autoregressive capabilities of the decoder-based transformer (see Figure E.1). The model learns to predict the next action by attending to the previous states, actions, and rewards-to-go (seen in (2)), without explicitly using additional commands such as the desired horizon or extra instructions.

At each timestep, the model receives a trajectory of past states, actions, and rewards-to-go, and uses

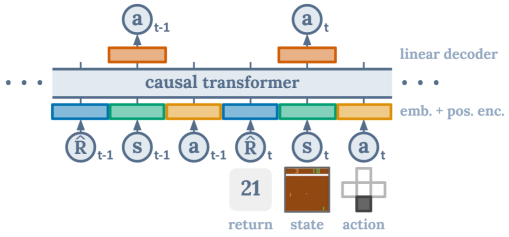

Figure 2.2: A high-level overview of the Decision Transformer architecture (Chen et al., 2021). The model predicts actions autoregressively based on past states, actions, and rewards-to-go.

that to predict the next action. This structure allows DT to flexibly condition its behavior on the expected future return while maintaining a simple and scalable input format. An overview of the DT architecture is shown in Figure 2.2.

This sequence modeling approach enables the DT to leverage the strengths of Transformers, such as modeling long-term dependencies, while simplifying the conditioning scheme compared to UDRL. As a result, DT has been shown to perform competitively across a variety of standard RL benchmarks, especially in offline RL settings such as Atari and OpenAI Gym (Chen et al., 2021).

Despite its empirical success, DT exhibits key limitations that challenge its applicability in goal-directed settings. Recent studies (Kim et al., 2024; Tanaka et al., 2025) show that the model often ignores the reward-to-go signal it is conditioned on, effectively behaving like a pure behavior cloning agent. This issue is fundamental and requires large changes to the architecture to mitigate it, as proposed by the authors. This undermines its ability to adapt behavior based on varying outcome specifications. Moreover, DT provides no explicit temporal control. Together, these limitations—combined with DT's autoregressive structure and sensitivity to distributional shifts—undermine its robustness and controllability in real-world deployment, especially when behavior must adapt to changing goals, distribution shifts, or time constraints.

3 Methodology

3.1 UDRL Transformer

As mentioned in the previous sections, despite the complementary strengths of the UDRL framework and Transformer architectures, no prior work has explored their combination. The proposed UDRL Transformer architecture addresses this gap by uniting the structured controllability of UDRL with the powerful modeling capabilities of Transformers.

This combination is motivated by several key observations:

- Many real-world environments involve high-dimensional, continuous state and action spaces where traditional function approximators struggle; transformers are well-suited to model such domains, making transformer-based architectures a desirable choice for developing scalable frameworks.
- The UDRL framework is naturally compatible with offline RL and expert imitation, as it can train directly from logged trajectories without requiring online interaction, making it an attractive choice for leveraging large datasets.
- Unlike Decision Transformers, which condition on fixed desired returns without explicit temporal control, UDRL provides a richer form of conditioning via both the extra(t) and the desired horizon, potentially offering a more fine-grained control over agent behavior.
- While many large-scale RL approaches drift toward sequence modeling, UDRL maintains a standard non-regressive approach, making it more efficient and modular when combined with a transformer that operates over short, structured context windows rather than full trajectories.

The two proposed architectures, UDRLt and UDRLt-MLP, can be seen in Figure 3.1

3.2 Architecture Design

Following the standard UDRL formulation, the inputs to the model consist of the desired reward, desired horizon, current state, and optional extra

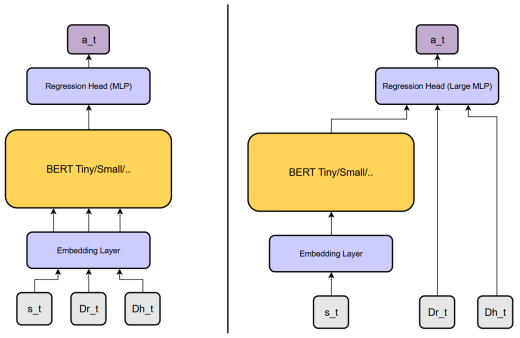

Figure 3.1: A high-level overview of the proposed UDRL Transformer architectures. Both models predict actions based on the current state and two control inputs, D_{r_t} and D_{h_t}. On the left, UDRLt uses the transformer end-to-end to process the full input context. On the right, UDRLt-MLP utilizes the transformer as a perception component, combining its output with the control inputs to achieve more direct action control.

information. Together, this tuple – combining environmental information and control commands – is referred to as the task **context**. The UDRLt architecture leverages the full context directly, enabling the transformer to contextualize the state with respect to the desired control signal. A small regression head then maps the resulting contextual embedding to the action space. In contrast, the UDRLt-MLP variant processes the state independently of the control signal, which is only concatenated later during action prediction. This design offers a more direct and modular way of influencing action generation.

As transformers operate on sequences of tokens, and the state input already contains rich, high-dimensional information, it can be embedded directly into the transformer's token space. However, the desired reward and horizon are scalar values. To align with common practice in related work (Chen et al., 2021; Shridhar et al., 2022), these scalars are projected into the model's embedding space using learnable linear layers.

Given that few tokens are used (most commonly state, D_r, and D_h), the model does not benefit from autoregressive processing. As a result, only the encoder part of the transformer is useful, as it leverages bidirectional attention blocks that model full contextual relationships among input tokens. This design choice is standard and consistent with practices in related work (W. Li et al., 2023).

Due to the relatively small size of the input sequence compared to typical NLP workloads, a lightweight transformer is employed - specifically, BERT Tiny (Devlin et al., 2019). This encoder-based model is widely recognized for its simplicity and adaptability. It contains approximately 4 million parameters, structured into 2 transformer blocks with 2 attention heads per block, and uses a hidden size of 128. While considered small in NLP, its capacity is suitable for RL tasks, which typically feature simpler temporal and contextual dependencies.

To produce actions, a regression head is added on top of the transformer encoder. In UDRLt, this head is shallow, as all conditioning is handled directly within the transformer. In UDRLt-MLP, however, the regression head is deeper to accommodate concatenated command information (D_r, D_h) not seen by the transformer itself.

As discussed in Section 2.2, training can be done via expert imitation or self-imitation. This work primarily focuses on expert data, but we also introduce a self-improvement loop in later experiments as a mechanism to combat low-quality data.

Overall, this architecture is designed to model fine-grained dependencies between control commands and environmental states while maintaining the simplicity, flexibility, and efficiency of the UDRL framework. This structure should enable dynamic adaptation of behavior based on subtle shifts in task context – even across similar states – improving both controllability and generalizability. As task complexity increases and richer forms of conditioning (such as compositional goals or soft constraints) become necessary, the transformer's ability to flexibly prioritize and contextualize inputs makes it a strong candidate for scalable goal-conditioned decision-making.

3.3 Data and Environment

Minari (Pitis et al., 2023) is a standardized dataset library developed by the Farama Foundation, designed to support offline and imitation-based RL workflows by providing high-quality, diverse datasets collected from a wide range of environments and agent types.

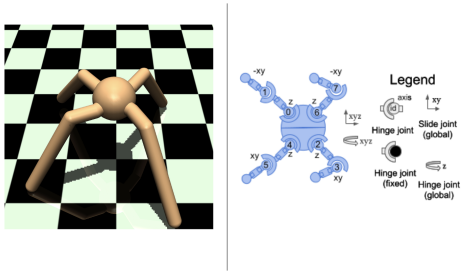

Figure 3.2: Illustration of the MuJoCo Ant environment. On the left, the 3D rendering of the quadrupedal ant agent is shown, featuring four legs attached to a central body. On the right, the kinematic structure is depicted, where each leg comprises two hinge joints, resulting in 8 actuated joints in total. Each joint allows rotation along a specific axis (z or xy), enabling complex locomotion. The legend describes the joint types and axes of movement.

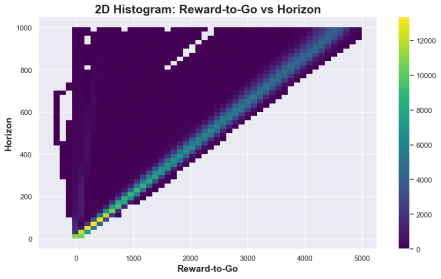

Figure 3.3: Frequency plot showing the desired return vs horizon distribution of the Ant dataset used in the research. The triangular shape of the graph is in line with the fact that higher returns require longer time to obtain, with the maximum reward per step capped at around 5.

The data used in this research has been fetched directly from the Minari HuggingFace page*. It is provided in HDF5 format via the h5py dictionary structure, where each key represents an episode. Each transition contains: the current state, terminal signal, taken action, obtained reward, and additional environment metadata. For a summary of the data and tasks chosen in this research, see table 3.1. The tasks mentioned in the table are as follows:

3.3.1 Ant-v5

Ant-v5 is a continuous control locomotion task in which a quadrupedal robot (ant) is trained to walk forward as efficiently as possible (see Figure 3.2). Once the agent falls to the ground, it cannot recover, which places a strong emphasis on maintaining stability throughout the episode. The state space encodes proprioceptive information, such as joint forces, consisting of 105 continuous variables. The action space comprises 8 continuous control signals corresponding to joint torques, all constrained within the range $[-1, 1]$. To ensure this constraint is satisfied, a standard practice is to apply a tanh activation to the model's outputs, used in this research as well (Chen et al., 2021; Tanaka et al., 2025).

This environment is commonly used to evaluate RL algorithms in high-dimensional locomotion settings. Minari provides three different dataset versions for this task: random, medium, and expert (corresponding to the skill level of the agent used to generate the trajectories), from which the medium is chosen due to its wide range of trajectory returns. For the distribution of desired return vs desired horizon, check Figure 3.3. The dataset was collected from an agent that achieves roughly 80% of expert-level performance, yet still makes occasional mistakes, providing valuable examples of both successful and recoverable behavior. A standard trajectory for this environment lasts for 1000 steps. The distribution of returns-to-go shown in Figure 3.4 reveals that similar states can correspond to both low and high cumulative rewards, highlighting the importance of the task **context** when learning policies in this environment.

3.3.2 AntMaze

AntMaze uses the same ant robot as in Ant-v5, but places it in a maze environment with the objective to reach a specified goal location. AntMaze is designed to evaluate an agent's ability to perform goal-conditioned, long-range planning using high-dimensional continuous control. The action space is the same as for the Ant, while the state space has 4 extra variables: the goal position (x,y) and the current position (x,y). In this research, those vectors are combined in a single vector *goal direction*, referring to the difference between those two vectors.

*https://huggingface.co/farama-minari

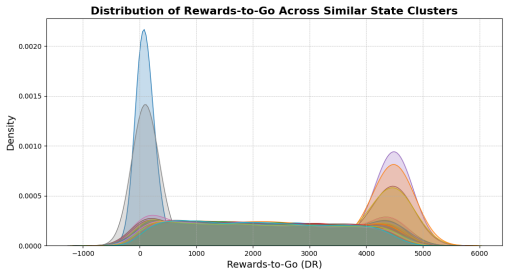

Figure 3.4: Distribution of desired returns across 100 clusters of similar states in the Ant-v5 environment. States were clustered using K-Means, and the corresponding return distributions per cluster were visualized. The overlap of low and high desired returns within clusters highlights the importance of goal conditioning.

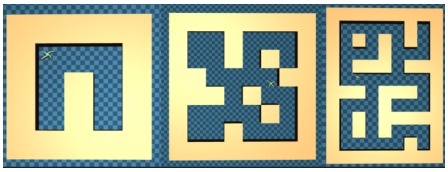

Figure 3.5: Three of the standard maze settings. Starting from left: Umaze, Medium, Large.

There are three types of mazes, with the same state and action space, differing only in complexity and size: umaze, medium maze, and large maze, seen in Figure 3.5 and summarized in Table 3.1.

This environment is **stochastic**: in the standard setup, a random location for both the agent and the goal is sampled at the beginning of the episode. The main challenge of this task lies in this high stochasticity, which requires the agent to adapt to widely varying start and goal configurations across episodes. Even for highly-trained RL experts, success rates peak at around 80%, highlighting the task's difficulty. This challenge intensifies when employing agents conditioned on additional control commands such as desired reward and horizon, demanding precise and context-sensitive behavior under uncertain and diverse conditions.

A significant compatibility issue exists between the two datasets: the Farama Foundation provides transitions collected from a simplified AntMaze environment variant that lacks contact force data applied to joints. This results in a drastic reduction in state dimensionality when transferring from the original Ant (105 dimensions) to the AntMaze dataset (27 dimensions), hindering transfer learning possibilities.

Using the formulas mentioned in Subsection 2.2, the control commands D_{r_t}, D_{h_t} are appended to each transition in the dataset. The goal location is used as the extra(t) input as defined under the UDRL framework. Due to the increased complexity of the task, the Neural Networks used in this part of the experiments (both for the action heads and for the full models) utilize per-layer normalization and residual connections every 2nd layer to stabilize gradient updates and allow for deeper architectures.

3.4 Ant Experiment

The performance of the UDRL Transformer models (UDRLt and UDRLt-MLP) will be assessed along three key axes: performance (measured by the reward obtained), efficiency (including model size and inference time), and alignment with control commands. These models will be compared against two baselines: an UDRL Neural Network - a simple model mapping desired reward (D_{r_t}), desired horizon (D_{h_t}), and state ($\mathbf{s}_t$) to action ($\mathbf{a}_t$), and an autoregressive Decision Transformer.

All models will be trained using grid search over hyperparameters with the Adam optimizer (Kingma & Ba, 2014). The loss function corresponds to the MSE formulation described in Section 2.2. MSE is used because the actions predicted by the models in all experiments are 8-dimensional continuous vectors, making it a standard and appropriate choice for regression-based control tasks.

Considering that standard trajectories in this environment span 1000 steps, the desired horizon (D_h) is fixed at 1000 during training, while the desired reward (D_r) varies across evenly spaced intervals between 0 and the maximum returns obtainable by an expert agent, determined by analyzing the available datasets constructed using such agents. To test for strong Out-Of-Distribution (**OOD**) behavior, this maximum will be expanded with a few extra values, indicated in the individual conditions.

To compare agent performance, plots of desired versus obtained rewards will be generated for various experimental conditions. These plots will show the mean and standard error of rewards obtained

Task Name	Action Dim	Observation Dim	Dataset Types	Chosen Dataset	No. Transitions
Ant-v5	8 (C)	105 (C)	expert, medium, simple	medium	∼1M
AntMaze-v5	8 (C)	27 + 2 (C)	umaze, medium, large (*diverse/play*)	umaze(both), medium(both)	∼1M

Table 3.1: List of datasets used in this research. "C" indicates the dimension is continuous. *Play* indicates an environment with fixed goal and target locations, while *diverse* means that both locations are sampled from a large set of positions.

over 10 independent runs under identical command inputs.

3.5 Ant to Antmaze Transfer Learning Experiment

This experiment evaluates the models' transferability by shifting the task from pure locomotion to goal-directed navigation within the AntMaze environment, maintaining the same locomotion mechanisms but requiring adaptation to goal-conditioned behavior.

Rewards in AntMaze can be sparse or dense, based on reaching or approaching the goal. While dense rewards demonstrate transfer learning by providing smoother feedback based on proximity to the goal, the available Farama datasets contain only sparse rewards–based solely on goal-reaching success–creating a mismatch between the training data and evaluation conditions. Due to mismatches in state dimensions between the Ant and AntMaze datasets, the observations are padded with 78 zeros to match the input size, representing a key bottleneck in this research.

As described previously, the goal direction vector is encoded in the extra context input (E_t) and concatenated with the outputs of the frozen base model (UDRLt-MLP). This combined vector passes through a fine-tuned external action head, which adjusts the base actions to suit the AntMaze environment (see Figure 3.6).

The medium-diverse dataset is selected here for its higher complexity among available Farama datasets.

These fine-tuned agents will be compared against models trained specifically on the AntMaze dataset, using the same architecture as in the Ant experiments. The comparison will focus on the UDRL-NN and UDRLt-MLP models, as motivated in the next Section.

A critical note is that the maximum obtainable

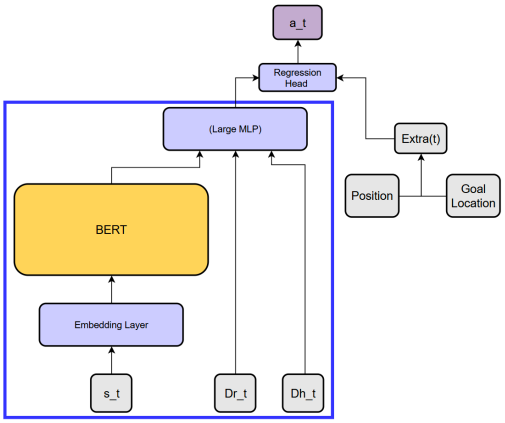

Figure 3.6: Proposed architecture on the transfer learning experiments. The UDRLt-MLP architecture's (bounded by the blue box indicating that the parameters are frozen) outputs are contacted with the constructed goal direction vector, which is then passed through a fine-tuned MLP head to produce new, adjusted actions.

reward per timestep is 1, achieved only if the agent is exactly at the goal. Given the 1000-step episode length, a desired reward (D_r) of 1000 is unachievable unless the agent spawns at the goal in every run - an impossible scenario in this stochastic environment. Following the same logic, values of $D_r > 800$ are thus practically unattainable, and $D_r > 600$ demands near-perfect expert performance. Therefore, model comparisons will focus on relative performance rather than proximity to the optimal $y = x$ line.

3.6 Data generation and extra tuning

The final experiment builds upon the transfer learning setup by retraining the UDRLt-MLP baseline model with three additional datasets, containing easier transitions (*medium-play*, *umaze-diverse*,

and *umaze-play*). On top of that, the model will be furtherly fine-tuned using additional data collected through *rollouts*. In this phase, the pretrained UDRLt-MLP agent is deployed in the environment to collect new trajectories across a full range of D_r values. This process resembles a simplified RL exploration mechanism; however, no random actions are taken, focusing on self-improvement rather than true exploration. The resulting data is used to fine-tune the same model that generated the data, which is then evaluated using the same experimental protocol as before, constituting a self-imitation algorithm as mentioned in Section 2.2.

This cycle of data collection, fine-tuning, and evaluation is repeated over several iterations, forming a self-improvement loop similar to the one introduced in the original UDRL paper (Schmidhuber, 2019), and illustrated in Figure 2.1. Although more selective data expansion strategies could be employed – such as filtering transitions by comparing UDRLt predictions to those of a pretrained RL expert for given (s_t, D_{r_t}, D_{h_t}) – the rollout-based method is chosen for its simplicity and reliability.

The effect of each condition will be evaluated by comparing the agent's behavior at different stages: training solely on the medium-diverse dataset, training on all four Farama datasets, and finally training on the rollout-augmented dataset. This progression allows for assessing the impact of increasing dataset diversity and iterative self-improvement on performance, alignment, and generalization.

4 Results

4.1 Ant-v5

Average episodic return of the different architectures in the Ant-v5 environment over a range of D_r can be observed in Figure 4.1. The data distribution in the training set ends slightly below 4800, so values beyond that point can be considered out-of-distribution (OOD) for all models.

To evaluate model efficiency, average inference times per model are reported in Table 4.1.

The Neural Network baseline achieves excellent alignment with the control commands and exhibits low variance, closely tracking the reward distribution of the dataset. However, its performance and stability gradually decline when evaluated on OOD commands, particularly when the desired reward (D_r) exceeds the maximum seen in training. This limitation will be revisited in the AntMaze results, where it becomes more pronounced.

Model	No Preprocessing (ms)	With Preprocessing (ms)
NeuralNet	0.41 (-91.4%)	0.42 (-93,7%)
DT	4.8 (baseline)	6.7 (baseline)
UDRLt	1.8 (-62.5%)	1.9 (-71,7%)
UDRLt-MLP	2.0 (-58.3%)	2.1 (-68,7%)

Table 4.1: Average inference time per 100 episodes (10 D_r conditions $\times$ 10 episodes). Device specification are reported in Appendix D

In contrast, the Decision Transformer shows no alignment with the control commands, despite being relatively resource-intensive. Its overall reward performance is also inferior to all other models tested, further underscoring its limitations in command-conditioned tasks.

The UDRLt architecture improves on DT in both inference speed and total rewards, but still struggles with controllability. Interestingly, this issue differs from the one observed in the DT: while DT tends to ignore the control signal entirely, UDRLt appears to process it inconsistently. This distinction is discussed further in Appendix C.

The UDRLt-MLP architecture appears to resolve this issue, demonstrating strong alignment with the control signals and slightly outperforming the Neural Network baseline, especially under OOD command conditions. This suggests improved model stability and better generalization to unseen targets. However, UDRLt-MLP requires slightly more computational resources than its simpler UDRLt counterpart. Therefore, an interesting research direction would be to apply various improvement methods (e.g., the self-imitation loop seen in Figure 2.1) and see whether similar alignment could be obtained, saving an additional 10% of the computational costs.

All UDRL-based agents achieve the highest overall returns, consistently surpassing the performance of the original dataset trajectories by up to 15%. The best returns were obtained by the UDRLt-MLP agent, indicating that the UDRL framework can combine elements from different successful trajectories to synthesize more efficient behavior than was explicitly demonstrated during training.

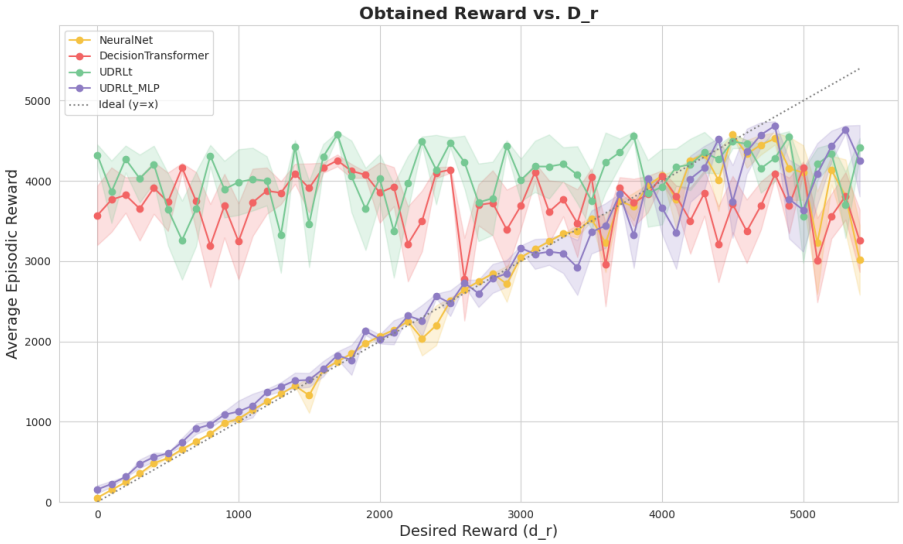

Figure 4.1: Performance of four agents in the Ant environment, showing average obtained reward as a function of the desired reward D_r, with D_h kept constant at the standard value of 1000. Each line represents the average episodic return over 10 evaluation runs, with shaded areas denoting the standard error. The dotted line indicates ideal command alignment (y=x), where the agent perfectly matches the desired reward.

Overall, these findings reinforce the potential of UDRL-based architectures for conditional control tasks, particularly in continuous, high-dimensional domains. The Discussion section will further expand upon this insight.

4.2 AntMaze-v5

The AntMaze experiments focus exclusively on the UDRLt-MLP and UDRL-NN agents. These models demonstrated the best trade-off between computational efficiency, controllability, and reward performance in the Ant-v5 environment, making them the most promising candidates for further experimentation. As a result, the Decision Transformer and UDRLt architectures are omitted from this section.

As discussed in the Data subsection, while the $y = x$ line represents ideal alignment, it is not a realistic benchmark for $D_r > 600$ due to the environment's setup and its inherent stochasticity. Therefore, this subsection focuses on comparing various conditions with each other, using the ideal alignment line only for reference.

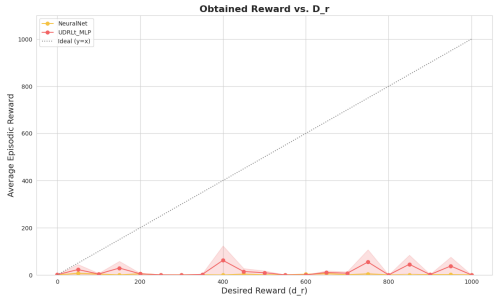

Figure 4.2: Performance of UDRL-NN and UDRLt-MLP agents transferred from Ant to AntMaze with no fine-tuning.

4.2.1 Ant to AntMaze (no fine-tuning)

A comparison of model performance when transferring from the Ant to the AntMaze without any additional training is shown in Figure 4.2.

The limited performance observed in the plot can likely stem from several factors discussed above:

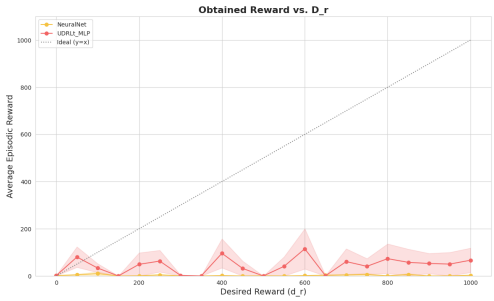

Figure 4.3: Performance in AntMaze after fine-tuning on the medium-diverse dataset.

- A mismatch in state dimensions - the AntMaze dataset lacks joint contact forces, reducing the state dimensionality from 105 to 27. To retain compatibility, 78 zeros were appended to the AntMaze observations, resulting in significant information loss.
- A substantially more complex task - unlike Ant-v5, based on deterministic forward locomotion, AntMaze demands stochastic-goal-directed navigation.

Despite these challenges, the pattern from Ant-v5 evaluation re-emerges: the transformer-based UDRL architecture (UDRLt-MLP) demonstrates superior adaptability to unseen scenarios, consistently achieving higher rewards across a range of D_r values compared to the Neural Network baseline. However, this performance is not satisfactory, showing that fine-tuning is necessary to transfer between those environments.

4.2.2 Ant to Antmaze (fine-tuned)

Figure 4.3 shows the performance of both models after fine-tuning on the medium-diverse AntMaze dataset, learning to follow the goal using the architecture outlined earlier in Figure 3.6.

The Neural Network baseline fails to show any notable improvement, continuing to obtain negligible rewards post-fine-tuning. This outcome reflects several compounding challenges that affect both models, but disproportionately hinder simpler architectures:

First, the fine-tuned setup freezes the original architecture while altering the input space - the base encoder receives only a reduced subset of the original features, limiting its adaptability.

Second, the inherent stochasticity of AntMaze, where both start and goal locations are randomized, introduces highly diverse episodes that cannot be comprehensively captured in a static offline dataset. This issue is particularly severe in IL settings and is further amplified by command conditioning through D_r and D_h, as discussed in the Data subsection.

Third, the reward structure used to generate the dataset is sparse: episodes contain no rewards until the goal is reached, followed by multiple non-zero terminal rewards. This sparsity undermines the core principle of UDRL, which relies on diverse reward patterns for conditioning and control, and instead induces unstable correlations between control signals and observable behavior.

These combined issues result in significant data noise - even training on small dataset subsets yields an average MSE greater than 0.02, suggesting an irreducible level of label inconsistency.

Despite these difficulties, the UDRLt-MLP model demonstrates a clear benefit from fine-tuning. It consistently achieves higher returns compared to its non-fine-tuned counterpart and occasionally aligns with the desired reward signal. However, it struggles with stability: across most conditions, individual trajectories vary widely, with some episodes completing high D_r conditions while others fail entirely.

Overall, this experiment reinforces the earlier observation that transformer-based UDRL models are more robust and transferable than simple feed-forward baselines. Yet, their performance remains highly sensitive to dataset quality and diversity - a known limitation of supervised learning approaches in complex, stochastic environments, where active exploration and interaction are often essential for stable and generalizable learning.

To put the performance of fine-tuned agents in context, Figure 4.4 shows results for models trained from scratch (without any frozen components) only on the same dataset used for fine-tuning in the previous section.

Those results reinforce the observations on the dataset limitations discussed earlier. Even full training on this dataset does not lead to fully stable or aligned behavior w.r.t. the control commands. Nonetheless, two key findings stand out:

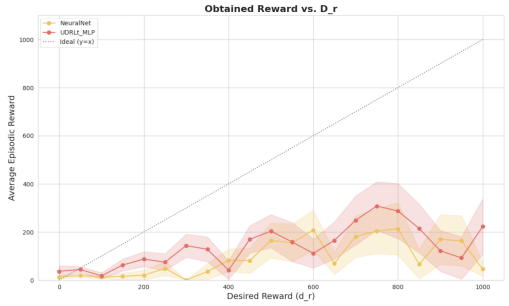

Figure 4.4: Performance in AntMaze of models trained on the medium-diverse dataset.

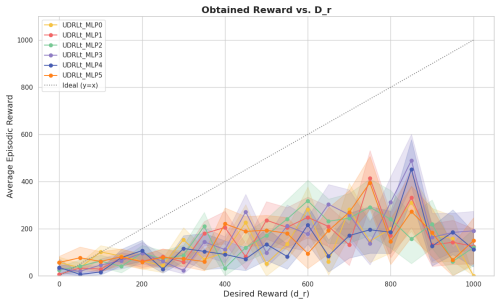

Figure 4.5: Performance of the UDRLt-MLP model across iterations of the self-imitation loop.

1. UDRLt-MLP again clearly outperforms the Neural Network, achieving significantly higher returns, although coming at the cost of increased computational demands.

2. Fine-tuning does improve performance, particularly for UDRLt-MLP, which recovers a non-negligible portion of the performance achieved by the fully trained model. However, the gap remains substantial, possibly due to the data and environment issues.

These findings suggest that fine-tuning alone is insufficient to achieve stable and fully controllable performance in complex, noisy environments, especially when using offline imitation datasets with sparse rewards and high stochasticity. This emphasizes the data-hungriness of the transformer-based UDRL framework.

To address the previously identified limitations, two additional tests were conducted. In the first test, instead of relying only on the most challenging AntMaze dataset (medium-diverse), a combination of four AntMaze datasets was used, as listed in Table 3.1. The evaluation plot for this condition is omitted, as it showed only minor differences compared to the previous condition. This suggests that incorporating more data of similar (but incompatible) quality marginally improves stability but does not significantly affect performance.

In the second test, the pre-trained UDRLt-MLP model (trained on the four datasets) was deployed in the environment to collect additional trajectories via active rollout. The plot comparing the different versions of the UDRLt-MLP model (depending on the iteration of the mentioned loop, with 0 denoting the base model trained on all 4 antamze datasets with no additional data collected) in this condition can be seen in Figures 4.5.

Applying this self-improvement loop with minimal adjustments–initially only pruning large portions of low-reward data– leads to *catastrophic forgetting*: the model gradually loses its ability to reproduce previously successful behaviors, with performance declining across successive iterations. To eliminate this, high-reward data points (defined here as $D_r > 500$) are preserved in each new dataset generated during the loop.

As seen in Figure 4.5, the main benefit of this approach is that the models trained for several iterations are the only ones in this research that consistently achieve $D_r > 600$, which is considered a near-expert-level performance in this environment. However, their alignment at lower reward levels ($D_r < 300$) remains poor. This likely stems from two factors: **bad initialization**–the base model was trained on incompatible data, creating a lasting distribution shift–and **persistent data scarcity** due to the stochastic nature of the environment.

Nevertheless, the loop offers a clear advantage: increased behavioral stability. As shown in Table 4.2, even a few iterations with relatively small datasets ($\sim$ 500k transitions) already lead to noticeable improvements in goal-reaching performance. These findings suggest that many of the limitations observed in the AntMaze setting could be mitigated by running additional loop iterations on larger, higher-quality datasets. While this study capped training at 8 short iterations due to time and computational constraints, a simple next step

Condition	Model	Goal Success Rate (%)
no-finetuning	NeuralNet	0
	UDRLt-MLP	**3-5**
finetuning	NeuralNet	0
	UDRLt-MLP	**6-9**
full training	NeuralNet	24-28
	UDRLt-MLP	**30-35**
full training (4)	NeuralNet	26-28
	UDRLt-MLP	**33-35**
full training (4) + rollout (5)	NeuralNet	Not Tested
	UDRLt-MLP	**38-44**

Table 4.2: Goal-reaching success rates of different models under various AntMaze conditions. The ranges come from the randomness of the environment, yielding various results even if multiple runs are run per task context. (4) implies that the conditions were on all 4 antmaze datasets as specified in Table 3.1. *Rollout* implies that the dataset has been expanded with rollouts of the fully-trained models, using the standard UDRL loop as explained in the text, with the number in brackets denoting the loop iterations done. A command-less, full RL expert achieves an 80% success rate.

would be to extend this setup with longer training cycles and more extensive datasets, unlocking the full potential of this refinement strategy.

5 Discussion

This research has shown that the UDRL framework **can** be successfully applied to complex environments such as Ant-v5, enabling precise control through a simple architecture and minimal supervision. Compared to state-of-the-art models like π_0 or PerAct, the proposed approach maintains an exceptional simplicity of the framework design: it **requires no sophisticated components** such as the Action Flow model used in π_0 and uses only **a fraction of the parameters** typical in this domain. On top of that, it supports a fully automated labeling procedure, requiring no manual oversight to generate vast amounts of new training data on demand. Crucially, it **substantially outperforms the Decision Transformer** - a standard baseline in the field - **along all three axes: performance, alignment, and computational efficiency**.

The AntMaze experiments, while demonstrating the superiority of UDRLt-MLP over the simpler UDRL-NN baseline, reveal several persistent limitations. These range from dataset-related issues (such as distributional mismatch and data scarcity) to challenges intrinsic to the environment, mainly related to its combination of stochasticity and complexity. In this setting, the meaning of a given command heavily depends on the starting and goal locations, drastically increasing the complexity of the control task. These factors suggest that while the UDRL framework remains viable, the current environment setup may demand a more tailored dataset and larger models to reach full performance. Additional work is needed to fully adapt this setting to the UDRL paradigm.

Nevertheless, the rapid performance gains observed through the self-imitation loop indicate that the issues encountered in AntMaze are not fundamental flaws of the UDRL framework. Rather, they stem from limitations in the data and experimental scope. With cleaner datasets, more iterations of self-improvement, and better exploration mechanisms, it should be possible to eliminate many of the shortcomings and achieve performance comparable to what was seen in Ant-v5.

Overall, the methodology developed in this work shows highly promising results: a controllable, efficient, and scalable framework for command-conditioned behavior in RL. In deterministic yet high-dimensional settings like Ant-v5, the architecture reliably extracts meaningful patterns from data, remains aligned with control signals, and outperforms existing methods while being smaller, simpler, and of lower latency. These results position UDRL as a strong candidate for future research in efficient, interpretable, and robust command-conditioned control.

5.1 Future research

5.1.1 Data-related improvements

Future work could address the data issues discussed in the Methods section, contributing to the Farama foundation by expanding the current datasets with a higher environment variability. Generating data across varied environment configurations and types would allow more effective pretraining and transfer learning scenarios. A typical approach involves training a high-performing Soft Actor-Critic (SAC)

agent, tuning it to expert-level performance, and rolling it out in the environment to collect several million transitions. To ensure the resulting dataset supports robust training, it should capture a wide spectrum of trajectories, capturing both failed attempts and near-perfect executions. Such a dataset would support future research and allow for a clean re-evaluation of the methods proposed in this paper, resolving the fine-tuning limitations encountered in the AntMaze setting.

5.1.2 Textual Prompting

While the current (D_r, D_h) framework offers a simple and scalable way to label data and train models, it presents challenges for external stakeholders. In real-world applications (particularly in complex environments), it may be infeasible or unintuitive for users to specify values for D_r, D_h, or other contextual inputs required to obtain desired behavior.

To address this issue, future work could explore integrating support for textual prompts, allowing users to provide high-level, natural language instructions such as "complete the task as quickly as possible, regardless of precision." The model could then infer appropriate internal values for D_r, D_h, and $Extra(t)$, while continuing to train under the same framework. This abstraction layer could greatly enhance the accessibility and applicability of UDRL models in practical deployment scenarios, without compromising training efficiency.

5.1.3 Multi-modal learning

Transformers are well-known for their ability to handle rich latent space representations, making them particularly effective in multi-modal learning settings where input signals come from diverse sources such as cameras, depth sensors, or IMUs (Black et al., 2024; Brohan et al., 2023). Given that the transformer backbone constitutes the bulk of the UDRLt and UDRLt-MLP architectures, an interesting direction for future research would be to investigate whether the proposed UDRL framework generalizes to such multi-modal control tasks.

A success in this application could position UDRL as a lightweight, flexible alternative to currently dominant high-level models, especially in domains where explainability, command conditioning, low-latency, and efficiency are important. Extending UDRL to multi-modal input spaces would also align with recent trends in vision-language-action models, offering a streamlined alternative to current large-scale architectures without sacrificing versatility or control.

5.1.4 Action Tokenization

Another promising direction for UDRL would be to explore encoder–decoder architectures that generate actions as discrete tokens. Rather than predicting continuous actions directly from latent representations, the model could autoregressively output action tokens, similar to how language models generate text. This token-based formulation has been successfully applied in recent work, including the Decision Transformer used in this study, and more prominently in models like DeepMind's RT-2 (Brohan et al., 2023). RT-2 extends tokenization to a multi-modal setting, representing perception, control, and goals all as textual tokens, thereby unifying vision, language, and action within a single transformer architecture.

Adopting a similar framework in UDRL could enable the system to handle complex real-world environments better, especially those involving diverse sensory inputs and abstract, high-level instructions, increasing the impact of this framework.

5.2 Broader Impact Note

This research has shown that the UDRL framework enables learning conditional control with minimal overhead, particularly in labeling or supervision. Its simplicity allows for seamless integration of expert or human demonstrations, making it well-suited for large-scale, human-in-the-loop alignment scenarios. By allowing selective guidance with little effort, UDRL supports the combination of broad self-supervised exploration and targeted human feedback, a property that could significantly contribute to scalable and aligned AI systems of the future.

Moreover, the framework's lightweight architecture makes it feasible to deploy on-device, even in low-resource or embedded settings. This extends its applicability beyond high-performance labs to real-world, constrained environments, further broadening the potential societal impact of aligned and efficient learning systems.

References

Black, K., Brown, N., Driess, D., Esmail, A., Equi, M., Finn, C., Fusai, N., Groom, L., Hausman, K., Ichter, B., Jakubczak, S., Jones, T., Ke, L., Levine, S., Li-Bell, A., Mothukuri, M., Nair, S., Pertsch, K., Shi, L. X., ... Zhilinsky, U. (2024). $_0$: A vision-language-action flow model for general robot control. (arXiv:2410.24164). https://doi.org/10.48550/arXiv.2410.24164

Botvinick, M., Wang, J. X., Dabney, W., Miller, K. J., & Kurth-Nelson, Z. (2020). Deep reinforcement learning and its neuroscientific implications. https://arxiv.org/abs/2007.03750

Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Chen, X., Choromanski, K., Ding, T., Driess, D., Dubey, A., Finn, C., Florence, P., Fu, C., Arenas, M. G., Gopalakrishnan, K., Han, K., Hausman, K., Herzog, A., Hsu, J., Ichter, B., ... Zitkovich, B. (2023). Rt-2: Vision-language-action models transfer web knowledge to robotic control. https://arxiv.org/abs/2307.15818

Cardenas-Cartagena, J., Falzari, M., Zullich, M., & Sabatelli, M. (2024). Upside-down reinforcement learning for more interpretable optimal control [arXiv:2411.11457 [cs]], (arXiv:2411.11457). https://doi.org/10.48550/arXiv.2411.11457

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., & Mordatch, I. (2021). Decision transformer: Reinforcement learning via sequence modeling [arXiv:2106.01345 [cs]], (arXiv:2106.01345). https://doi.org/10.48550/arXiv.2106.01345

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. https://arxiv.org/abs/1810.04805

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. https://arxiv.org/abs/2010.11929

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2019). Deep reinforcement learning that matters. https://arxiv.org/abs/1709.06560

Kamatala, S., Jonnalagadda, A. K., & Naayini, P. (2025). Transformers beyond nlp: Expanding horizons in machine learning [Available at SSRN: https://ssrn.com/abstract=5110547]. *SSRN*.

Kim, J., Lee, S., Kim, W., & Sung, Y. (2024). Decision convformer: Local filtering in metaformer is sufficient for decision making. https://arxiv.org/abs/2310.03022

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. https://arxiv.org/abs/1412.6980

Li, W., Luo, H., Lin, Z., Zhang, C., Lu, Z., & Ye, D. (2023). A survey on transformers in reinforcement learning. https://arxiv.org/abs/2301.03044

Li, Y. (2022). Reinforcement learning in practice: Opportunities and challenges. https://arxiv.org/abs/2202.11296

Liu, Z., Liu, S., Zhang, Z., Cai, Q., Zhao, X., Zhao, K., Hu, L., Jiang, P., & Gai, K. (2024). Sequential recommendation for optimizing both immediate feedback and long-term retention. *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1872–1882. https://doi.org/10.1145/3626772.3657829

Pitis, S., Black, G., Hesse, C., Terry, J., Carroll, M., Ke, L., Gitiaux, X., Nair, A., Raffin, A., Lin, A., Dennler, M., Kumar, A., Agarwal, R., Klimov, O., O'Donoghue, B., Pineau, J., Raffin, A., et al. (2023). Minari: A standardized interface and repository of offline reinforcement learning datasets [Accessed: 2025-04-29]. https://minari.farama.org

Schmidhuber, J. (2019). Reinforcement learning upside down: Don't predict rewards – just map them to actions. (arXiv:1912.02875). https://doi.org/10.48550/arXiv.1912.02875

Shridhar, M., Manuelli, L., & Fox, D. (2022). Perceiver-actor: A multi-task transformer for robotic manipulation. https://arxiv.org/abs/2209.05451

Srivastava, R. K., Shyam, P., Mutz, F., Jaśkowski, W., & Schmidhuber, J. (2021). Training agents using upside-down reinforcement learning. https://arxiv.org/abs/1912.02877

Sutton, R. S., & Barto, A. G. (2015). *Reinforcement learning: An introduction* (2nd ed.). The MIT Press.

Tanaka, T., Abe, K., Ariu, K., Morimura, T., & Simo-Serra, E. (2025). Return-aligned decision transformer. https://arxiv.org/abs/2402.03923

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. https://arxiv.org/abs/1706.03762

Zhai, X., Kolesnikov, A., Houlsby, N., & Beyer, L. (2022). Scaling vision transformers. https://arxiv.org/abs/2106.04560

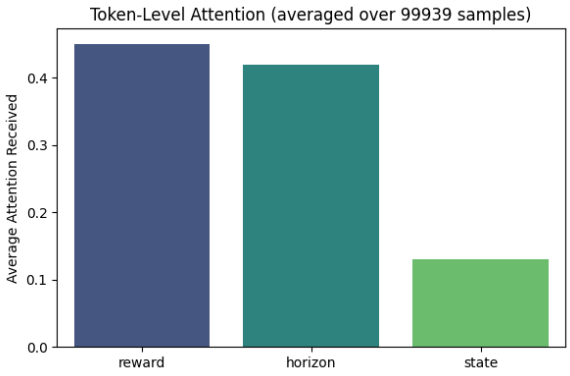

Figure C.1: Average attention scores obtained by the 3 input tokens of the UDRLt architecture, taken over 10% of the dataset ($\sim$ 100k samples)

A Model Hyperparameters

The hyperparameters used in this research can be seen in Table A.1.

B Code Source

All algorithms, pre-trained models, and dataset-making scripts have been published under the MIT license and are available in the following GitHub repository[†].

C UDRLt Attention Analysis

As seen in Figure C.1, the model seems to realize that the reward and horizon commands are highly necessary to produce desired behavior. However, its failure of alignment highlights that this knowledge is lost in the state processing, and the model does not know how to adjust its behavior based on those commands. This is supported by the performance of the UDRLt-MLP model, differing only by not feeding the commands directly to the transformer. This issue was targeted with two improvements: implementing complex D_r and D_h encoders and using huge BERT and Action Head variants, but yielded no results. On top of the suggestions in the Future Research section, an exploration method similar to the AntMaze self-imitation loop presented in this research could help to mitigate this issue on performance.

[†]https://github.com/LukaszSawala/Upside-Down-RL-Transformer

D Inference Times

The device used to generate Table 4.1 was CPU: 13th Gen Intel(R) Core(TM) i9-13900HX, Architecture: x86_64, 32-core.

E Transformer Architecture

A standard encoder-decoder transformer architecture is illustrated in Figure E.1. The encoder (shown on the left) is used to build a rich contextual representation of the entire input sequence by applying bidirectional transformations to the input sequence. It processes the tokens using a multi-head self-attention mechanism to weigh the importance of different inputs (e.g., words) and a position-wise feed-forward network to apply a non-linear transformation. The decoder, on the right, then autoregressively generates the output sequence one element at a time. In addition to the self-attention and feed-forward sub-layers, it includes a third sub-layer that performs masked (unidirectional) multi-head attention over the encoder's output, allowing it to focus on relevant parts of the source sequence.

Task Name	Model Name	no. Hidden Layers	Hidden Size	Batch Size	Learning Rate	Patience
Ant	NeuralNet	4	256	16	1e-4	3
Ant	DT	3(8-head)	128	8	1e-3	3
Ant	UDRLt	2(Bt)(2-head) + Lin. Proj	128(Bt)	8	1e-5	3
Ant	UDRLt-MLP	2(Bt)(2-head) + 6	128(Bt) + 256	8	5e-5	3
AntMaze-FT	UDRLt-MLP & NeuralNet	12	512	128	1e-4	10
AntMaze	UDRLt-MLP & NeuralNet	18	512	128	1e-4	15
AntMaze (RollOut)	UDRLt-MLP & NeuralNet	-	-	16	5e-5	10

Table A.1: List of hyperparameters of the models used in this research for all of the conditions. (Bt) stands for BERT-tiny. The Antmaze models are structured in the same way as the Ant models, and hence only the sizes of the action heads are given. The heads in hidden layers specify the number of attention heads of each transformer block. The "+" means that a model on the right has been put on top of the model on the left, i.e., for the Ant UDRLt-MLP, a 6-layer 256-wide MLP has been put on top of a BERT-tiny.

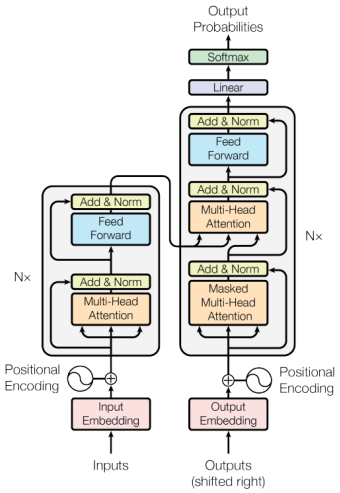

Figure E.1: A standard Transformer with an encoder (left) and decoder (right) architectures (Vaswani et al., 2017).